

# **THEOS C**

---

## **Standard Function Library Reference**

First Edition: May, 1997

Copyright © 1997 by THEOS Software Corporation.

All rights reserved.

The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used only in accordance with the terms of the agreement. Information in this document is subject to change. No part of this manual may be reproduced, transmitted, transcribed or translated into any language in any form by any means without the written permission of THEOS Software Corporation. Printed in the United States of America.

THEOS Software Corporation  
1801 Oakland Boulevard, Suite 315  
Walnut Creek, CA 94596-7000

Telephone: (510) 935-1118  
Fax: (510) 935-1177  
E-mail: [sales@theos-software.com](mailto:sales@theos-software.com)  
WWW: <http://www.theos-software.com>

THEOS® and the THEOS logo are registered trademarks of THEOS Software Corporation.

Documentation by Computer Printed Words.

# Index of Functions

## A

a64l	55
abort	56
abs	57
_absd	58
_absw	58
acc_access	59
accept	60
access	62
acc_maint	59
acos	64
acot	65
acsc	66
_addb	67
_addcsb	67
_addcsd	67
_addcsw	67
_addd	67
_addw	67
alarm	69
_alloca	70
_andb	71
_andcsb	71
_andcsd	71
_andcsw	71
_andd	71
_andw	71
asctime	73
asec	75
asin	76
_asm	77
assert	78
at	80
atan	82
atan2	82
atexit	83
_atexit_clear	83
_atexit_remove	83
atof	85
atoi	85
atol	85
atoll	85
atstr	80
_attach	87

## B

base64_decode	88
base64_encode	88
Basic2C	91
BasicL2C	91
_baud2cd	92
bcd2ieee	93
bcmp	94

bcopy	94
bell_oneshot	457
bind	95
bininsert	97
bsearch	98
bzero	94

## C

C2Basic	101
C2BasicL	101
c3tol	102
cabs	103
calloc	104
_cd2baud	92
ceil	105
cgets	106
chdir	108
_cld	593
clearerr	109
_cli	350
clock	110
close	111
closesocket	112
ClrBypass	560
cmd_abbrev	114
conmask	115
connect	117
_conrdy	119
conrdy	119
_con_tran	120
cos	121
cosh	121
cot	122
coth	122
cprintf	123
cpu_time	124
cputs	125
crc16	126
crc32	126
creat	127
crt	128
crtcolor	130
crt_oneshot	457
csc	132
csch	132
csi	133
ctime	134
CursorBlock	135
CursorHide	135
CursorLine	135
CursorShow	135
cuserid	136

**D**

_decb	138
_deccsb	138
_deccsd	138
_deccsw	138
_decdd	138
_decnucb	138
_decnucd	138
_decnucw	138
_decw	138
delay	140
deletex	141
_detach	142
devnames_close	143
devnames_open	143
devnames_parse	143
devnames_read	143
_devopen	148
difftime	149
dirclose	150
_dirfdb	150
_dir_mount	153
diropen	150
dirread	150
_dirucb	150
_disable	350
_disk_capacity	154
div	156
_dup	158
dup	158
dup2	158

**E**

ecvt	160
_enable	350
eof	162
erase	163
_errarg	168
_errbot	164
errmsg	166
errno	168
_errnum	168
execl	170
execlp	170
execv	170
execvp	170
exit	173
exp	175

**F**

fabs	176
fbuf	177
fclose	178
fcloseall	178
fcntl	180
fcreate	182

fcvt	184
FD_CLR	186
FD_ISSET	186
fdopen	187
FD_SET	186
FD_ZERO	186
feof	188
ferror	189
fflush	214
fgetc	190
fgetl	190
fgetpos	194
fgets	190
fgetsn	190
fgetw	190
fgrow	195
fifo_alloc	197
fifo_free	197
fifo_get	197
fifo_put	197
fifo_rdy	197
_file_alloc	199
_filechange	200
file_err	203
filelock	204
fileno	207
find_acc	208
find_first	210
find_next	210
floor	213
floppy_oneshot	457
flush	214
flushall	214
_fmemccpy	216
_fmemchr	216
_fmemcmp	216
_fmemcpy	216
_fmemdcmp	216
_fmemeq	216
_fmemicmp	216
_fmemieq	216
_fmemmove	216
_fmemrcpy	216
_fmemset	216
fmod	220
fopen	222
_force	227
fork	229
forktask	229
formask	231
formclear	231
formincr	231
formparm	231
fperror	233
_FP_OFF	443
FP_OFF	443
fprintf	234
_FP_SEG	443



FP_SEG	443	getenv	280
fputc	236	getexvar	282
fputl	236	getfddate	285
fputs	236	getfiledate	285
fputsn	236	_getflag	268
fputsnl	236	getflat	286
fputw	236	getfn	288
fread	243	_get_help_filename	289
free	244	gethostbyaddr	290
free_oneshot	457	gethostbyname	290
_free_sel	245	gethostname	291
free_sel	245	getkey	292
freopen	246	getlang	294
frexp	248	_getldt	268
fscanf	249	getlib	295
fseek	250	_getlimit	268
fsetpos	252	_getline	296
fsign	254	getll	298
_fstrcat	255	getlogin	299
_fstrchr	255	getlub	300
_fstrcmp	255	_getmem	301
_fstrcpy	255	_get_menu_filename	289
_fstrcspn	255	getmpid	304
_fstrdcmp	255	getmsec	306
_fstrend	255	getmsg	307
_fstreq	255	_getmsw	268
_fstricmp	255	get_oneshot	457
_fstrieq	255	_getosver	308
_fstristr	255	getpeername	309
_fstrlen	255	_getpid	268
_fstrpbrk	255	getpid	304
_fstrrchr	255	getpl	298
_fstrset	255	getppid	304
_fstrspn	255	getpriv	310
_fstrstr	255	getprotobyname	311
_fstrtok	255	getprotobynumber	311
ftell	261	gets	312
ftoa	263	GetScreenSize	313
fwrite	264	_get_sect	315
<b>G</b>			
gcd	265	getservbyname	317
gcv	266	getservbyport	317
get_acc_name	270	getsockname	318
_getar	268	getsockopt	319
getbaud	272	_getss	268
getc	190	_gettib	268
getch	273	gettime	321
getchar	190	GetUCB	322
getclass	274	getucb	322
_getcs	268	getuid	324
_getcsa	268	getver	325
getcwd	275	getvol	327
getdate	277	getw	328
getdev	279	gmtime	329
_getds	268	<b>H</b>	
_geteax	268	has	330
_getebp	268	hascolor	330
		HasMouse	332

hasprt .....	333
h_errno .....	641
htonl .....	334
htons .....	334
hypot .....	335

## I

i2tol .....	337
ieee2bcd .....	93
_inb .....	340
_incb .....	338
_inccsb .....	338
_inccsd .....	338
_inccsw .....	338
_incd .....	338
_incnucb .....	338
_incnucd .....	338
_incnucw .....	338
_incrmem .....	341
_incw .....	338
_ind .....	340
inet_addr .....	342
inet_ntoa .....	342
InitFileClose .....	344
InitFileOpen .....	344
InitFileRead .....	344
InitFileReadInt .....	344
InitFileReadNext .....	344
InitFileWrite .....	344
InitFileWriteInt .....	344
_insb .....	340
_insd .....	340
_insw .....	340
intoff .....	350
inton .....	350
_inw .....	340
_ioctl .....	351
ioctlsocket .....	360
_ioctl_ucb .....	351
ip .....	361
IsActive .....	366
isalnum .....	362
isalpha .....	362
isascii .....	362
isatty .....	367
IsBypass .....	368
iscntrl .....	362
iscon .....	369
iscsym .....	362
iscsymnum .....	362
isdigit .....	362
isdisp .....	362
isfnsym .....	362
isgraph .....	362
ishex .....	362
islower .....	362
isocal .....	362
ispcterm .....	371

isprint .....	362
isprtnum .....	372
ispunct .....	362
IsSession .....	373
isspace .....	362
issymbol .....	362
issymnum .....	362
isupper .....	362
isvga .....	374
isxdigit .....	362
itoa .....	375

## K

keyclose .....	376
killtask .....	377

## L

l3tol .....	378
l64a .....	379
labs .....	380
ldeletek .....	381
ldexp .....	383
ldiv .....	156
_leapyear .....	384
lfind .....	411
listen .....	385
ltoa .....	415
load .....	387
load_yn .....	389
localeconv .....	390
localtime .....	393
locate .....	395
lockf .....	396
locking .....	396
_lockres .....	398
_lockset .....	398
_lockshare .....	398
_locktest .....	398
_lockwait .....	398
log .....	400
log10 .....	400
log2 .....	400
logname .....	402
_logon .....	403
longfname .....	405
longjmp .....	407
lreadk .....	409
lreadn .....	409
lsearch .....	411
_lseek .....	413
lseek .....	413
ltoa .....	415
ltoc3 .....	415
ltol2 .....	415
ltol3 .....	415
_lubname .....	417
lwritelk .....	418

**M**

_make_alias . . . . .	420
make_alias . . . . .	420
_makelib . . . . .	421
makelib . . . . .	421
_make_sel . . . . .	423
make_sel . . . . .	423
malloc . . . . .	424
matcharg . . . . .	426
max . . . . .	427
max_alloc . . . . .	428
_maxd . . . . .	429
_maxw . . . . .	429
mblen . . . . .	430
mbstowcs . . . . .	430
mbtowc . . . . .	430
mem_avail . . . . .	432
memccpy . . . . .	433
memchr . . . . .	433
memcmp . . . . .	433
_memcpy . . . . .	433
memcpy . . . . .	433
memdcmp . . . . .	433
memeq . . . . .	433
_mem_grow . . . . .	437
memicmp . . . . .	433
memieq . . . . .	433
memmove . . . . .	433
memrcpy . . . . .	433
_memset . . . . .	433
memset . . . . .	433
menu . . . . .	439
menu2 . . . . .	439
min . . . . .	427
_mind . . . . .	429
_minw . . . . .	429
mkdir . . . . .	442
_MK_FP . . . . .	443
MK_FP . . . . .	443
_mklib . . . . .	421
mktemp . . . . .	444
mktime . . . . .	446
modf . . . . .	448
_mount_fs . . . . .	449
_movsb . . . . .	450
_movsd . . . . .	450
_movsw . . . . .	450
msalarm . . . . .	451
msgclose . . . . .	452
mvga_oneshot . . . . .	457

**N**

_norm_fn . . . . .	453
ntohl . . . . .	455
ntohs . . . . .	455

**O**

offsetof . . . . .	456
oneshot . . . . .	457
open . . . . .	459
openhlp . . . . .	461
openmenu . . . . .	461
_orb . . . . .	463
_orcsb . . . . .	463
_orcsd . . . . .	463
_orcsw . . . . .	463
_ord . . . . .	463
_orw . . . . .	463
_osmajor . . . . .	465
_osminor . . . . .	465
_osversion . . . . .	465
_outb . . . . .	466
_outd . . . . .	466
_outsb . . . . .	466
_outsd . . . . .	466
_outsw . . . . .	466
_outw . . . . .	466

**P**

pagewait . . . . .	467
pause . . . . .	469
_peekb . . . . .	470
_peekcsb . . . . .	470
_peekcsd . . . . .	470
_peekcsp . . . . .	470
_peekcsw . . . . .	470
_peekd . . . . .	470
_peeknucb . . . . .	470
_peeknucd . . . . .	470
_peeknucp . . . . .	470
_peeknucw . . . . .	470
_peekp . . . . .	470
_peekscrb . . . . .	470
_peekscrd . . . . .	470
_peekscrp . . . . .	470
_peekscrw . . . . .	470
_peekw . . . . .	470
perror . . . . .	472
_PhyAddr . . . . .	473
_phy_addr . . . . .	473
pipe . . . . .	475
_pokeb . . . . .	476
_pokecsb . . . . .	476
_pokecsd . . . . .	476
_pokecsp . . . . .	476
_pokecsw . . . . .	476
_poked . . . . .	476
_pokenucb . . . . .	476
_pokenucd . . . . .	476
_pokenucp . . . . .	476
_pokenucw . . . . .	476
_pokep . . . . .	476
_pokescrb . . . . .	476

_pokescrd . . . . .	476
_pokescrp . . . . .	476
_pokescrw . . . . .	476
_pokew . . . . .	476
popen . . . . .	478
_popf . . . . .	481
pow . . . . .	482
_pre_empt . . . . .	755
prime . . . . .	483
primel . . . . .	483
printf . . . . .	484
_pushf . . . . .	481
putc . . . . .	236
putch . . . . .	493
putchar . . . . .	236
_putenv . . . . .	495
putenv . . . . .	495
putexvar . . . . .	282
putfddate . . . . .	497
_putmem . . . . .	301
putmsg . . . . .	498
puts . . . . .	500
putw . . . . .	501
pwverify . . . . .	502

## Q

qsort . . . . .	504
_quittoff . . . . .	507
_quiton . . . . .	507

## R

raise . . . . .	508
rand . . . . .	509
_read . . . . .	511
read . . . . .	511
read_acc . . . . .	513
readk . . . . .	515
readn . . . . .	515
readp . . . . .	515
realloc . . . . .	518
reclock . . . . .	522
recunlock . . . . .	522
recv . . . . .	520
recvfrom . . . . .	520
regcmp . . . . .	524
regex . . . . .	524
release_dma . . . . .	529
_remote . . . . .	530
remove . . . . .	532
rename . . . . .	534
reserve_dma . . . . .	529
_restoreints . . . . .	539
rewind . . . . .	536
rmdir . . . . .	537
rsema . . . . .	550
rsemaname . . . . .	550
rsemares . . . . .	550

rsemaset . . . . .	550
rsemawait . . . . .	550
_rsvmem . . . . .	538

## S

_saveints . . . . .	539
scanf . . . . .	540
sch_acc . . . . .	513
schedint . . . . .	544
scsi_oneshot . . . . .	457
sec . . . . .	545
sech . . . . .	545
_seek . . . . .	546
seek . . . . .	546
select . . . . .	548
sema . . . . .	550
semaphore . . . . .	550
semares . . . . .	550
semaset . . . . .	550
semawait . . . . .	550
send . . . . .	555
sendto . . . . .	555
setbuf . . . . .	558
SetBypass . . . . .	560
setjmp . . . . .	561
setlocale . . . . .	562
_setprty . . . . .	566
SetScreenSize . . . . .	313
_set_slice . . . . .	566
setsockopt . . . . .	567
setvbuf . . . . .	558
sgetl . . . . .	570
shared . . . . .	571
signal . . . . .	573
sin . . . . .	575
sinh . . . . .	575
sizeof . . . . .	456
skipsp . . . . .	576
sleep . . . . .	577
sleep_msec . . . . .	577
sleep_sec . . . . .	577
sleep_until . . . . .	577
_snu . . . . .	755
socket . . . . .	579
sort . . . . .	586
spawnl . . . . .	587
spawnlp . . . . .	587
spawnv . . . . .	587
spawnvp . . . . .	587
sprintf . . . . .	590
sputl . . . . .	570
sqrt . . . . .	591
srand . . . . .	509
sscanf . . . . .	592
_std . . . . .	593
_sti . . . . .	350
_stosb . . . . .	594
_stosd . . . . .	594

_stosw . . . . .	.594
strcat . . . . .	.596
strchr . . . . .	.596
strcmp . . . . .	.596
strcoll . . . . .	.608
_strcpy . . . . .	.607
strcpy . . . . .	.596
strcspn . . . . .	.596
strdcmp . . . . .	.596
strdup . . . . .	.596
_strend . . . . .	.607
strend . . . . .	.596
streq . . . . .	.596
strerror . . . . .	.609
strftime . . . . .	.610
stricmp . . . . .	.596
strieq . . . . .	.596
stristr . . . . .	.596
_strlen . . . . .	.607
strlen . . . . .	.596
strlwr . . . . .	.596
strmake . . . . .	.596
strmatch . . . . .	.596
strncat . . . . .	.596
strncmp . . . . .	.596
strncpy . . . . .	.596
srneq . . . . .	.596
strnicmp . . . . .	.596
strnset . . . . .	.596
stpbrk . . . . .	.596
strchr . . . . .	.596
strset . . . . .	.596
strspn . . . . .	.596
strstr . . . . .	.596
strtod . . . . .	.613
strtok . . . . .	.596
strtol . . . . .	.613
strtoul . . . . .	.613
strtrim . . . . .	.596
strupr . . . . .	.596
strxfrm . . . . .	.615
_subb . . . . .	.616
_subcsb . . . . .	.616
_subcsd . . . . .	.616
_subcsw . . . . .	.616
_subd . . . . .	.616
_subw . . . . .	.616
swab . . . . .	.618
swapbits . . . . .	.619
_sync . . . . .	.620
syzerr . . . . .	.621
system . . . . .	.623
T	
ta_alloc . . . . .	.624
ta_free . . . . .	.624
ta_get . . . . .	.624
tanh . . . . .	.626
tan	
ta_put . . . . .	.624
ta_rdy . . . . .	.624
_tell . . . . .	.627
tell . . . . .	.627
tempnam . . . . .	.628
testarg . . . . .	.630
testhead . . . . .	.632
testwild . . . . .	.634
time . . . . .	.635
timer . . . . .	.550
tmpfile . . . . .	.628
tmpnam . . . . .	.628
_tmpnam_drive . . . . .	.628
toascii . . . . .	.636
_tolower . . . . .	.636
tolower . . . . .	.636
topen . . . . .	.638
tot_alloc . . . . .	.640
_toupper . . . . .	.636
toupper . . . . .	.636
TSAGetLastError . . . . .	.641
TSASetLastError . . . . .	.641
TSAStrError . . . . .	.641
TWS_disconnect . . . . .	.642
TWS_disconnect_now . . . . .	.642
TWS_execute . . . . .	.642
TWS_focus . . . . .	.642
TWS_maximize . . . . .	.642
TWS_minimize . . . . .	.642
TWS_ontop . . . . .	.642
TWS_receive . . . . .	.642
TWS_restore . . . . .	.642
TWS_send . . . . .	.642
TWS_title . . . . .	.642
TWS_user_focus . . . . .	.642
tzset . . . . .	.647
U	
ub_alloc . . . . .	.624
ub_free . . . . .	.624
_uncache . . . . .	.648
uncache . . . . .	.648
ungetc . . . . .	.649
ungetch . . . . .	.650
unlink . . . . .	.651
unload . . . . .	.387
unlock . . . . .	.653
_await . . . . .	.669
utoa . . . . .	.654
V	
va_alist . . . . .	.655
va_arg . . . . .	.655
va_dcl . . . . .	.655
va_end . . . . .	.655
va_start . . . . .	.655

vdi	657	wInvert	702
vdialpha	657	wLocate	704
vdia	657	wMenuBar	705
vdibar	657	wMouseDisable	710
vdicircle	657	wMouseEnable	710
vdiclear	657	wMouseHit	710
vdiclose	657	wMouseRead	710
vdigraph	657	wMouseState	710
vdiline	657	wMove	718
vdimark	657	wOnKey	719
vdio	657	wOpen	721
vdipie	657	wOrder	725
vdisetfc	657	wRefresh	728
vdisetfs	657	wRemove	729
vdisetlc	657	wRepaint	730
vdisetlh	657	wRestore	733
vdisetls	657	_write	731
vdisetmc	657	write	731
vdisetmh	657	writeln	732
vdisetms	657	wSave	733
vdisetta	657	wSaveSize	733
vdisettc	657	wScroll	736
vdisetth	657	wSelect	737
vdissetp	657	wStatus	739
vdissets	657	wSwitch	741
vditext	657	wSwitchTo	741
vfprintf	667	wTake	744
vprintf	667	wTitle	746
vsprintf	667	wVer	748

## W

_wait	669
wChoice	670
wChoiceFuncKeys	670
wChoiceSelect	670
wChoiceTimeout	670
wClear	676
wClip	677
wClose	678
wCloseAll	678
wColor	679
wCopy	681
wCount	683
wcstombs	684
wcstrlen	684
wctomb	684
wEdit	685
wEnter	688
wEnterFirst	688
wFinish	691
wFrame	692
wGetActive	695
wGetSess	696
wGetStat	697
wGetStr	698
wGetTitle	699
wGetWin	700
WhoAml	701

## X

xclose	749
xcreat	749
xdup	749
xfdopen	749
xfile	749
xflush	749
xfopen	749
xlocking	749
xlseek	749
xopen	749
_xorb	751
_xorcsb	751
_xorcsd	751
_xorcsw	751
_xord	751
_xorw	751
xread	749
xtell	749
xwrite	749

## Y

_yesno	753
yesno	753
_yesnoall	753
_yield	755
yield	755

# 1 Functions by Category

---

This section lists most of the THEOS C functions according to general usage. If you don't know the name of a function but you do know the general operation that you want to perform, this section can help you find the specific function that performs the operation desired.

The principal categories of library functions include:

- ▶ Account Name Management
- ▶ Buffer Manipulation
- ▶ Character Classification & Conversion
- ▶ Configuration File Access & Maintenance
- ▶ Console Input & Output
- ▶ Data Conversion
- ▶ Device-Driver Routines
- ▶ Device Names, Help, Keyword, Menu & Message File Routines
- ▶ Directory & Library Maintenance
- ▶ Directory Searching
- ▶ Disk Management & Maintenance
- ▶ Environment Access
- ▶ Error Handling & Exiting
- ▶ EXEC Variable Access
- ▶ File Handling
- ▶ Graphics
- ▶ International
- ▶ Math
- ▶ Memory Management
- ▶ Multitask Management
- ▶ TCP/IP Network Sockets
- ▶ Peek & Poke Far Memory
- ▶ Process Control
- ▶ Searching & Sorting
- ▶ Standard Input & Output
- ▶ String & Memory Manipulation
- ▶ Time & Date
- ▶ Signaling & Timing Control
- ▶ THEOS WorkStation Control
- ▶ Trigonometric Functions
- ▶ Variable-Argument Management
- ▶ Mouse, On Key, Session and Window Management

## 12 Functions by Category

---

### ■ Account Name Management

The following data and functions are declared in the header file ACB.H.

```
typedef structacb {           // account structure, in memory
    char name[8];             // account name
    short id;                 // account number
    long loc;                 // location in system.account
    char mail;                // set if have mail
    char rsvd[3];             // reserved
} ACB;
```

<code>acc_access,</code>	Lock or release the account system.
<code>acc_maint</code> ‡	
<code>find_acc</code> ‡	Find ACB information for an account name.
<code>get_acc_name</code> ‡	Find account name given an account number.
<code>pwverify</code> ‡	Verify password against account system.
<code>read_acc,</code>	Read environment information for an account.
<code>sch_acc</code> ‡	Find entry in account environment information.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.



## ■ Buffer Manipulation

### ■ FIFO Buffers

The following functions are declared in the header files `FIFO.H`.

```
typedef struct{
    unsigned short size;        // size of buffer
    unsigned short count;      // buffer count
    unsigned short store;      // next store index
    unsigned short fetch;      // next fetch index
    char buf[];                // the buffer
} FIFO;
```

<code>fifo_alloc</code>	Allocate a new first-in-first-out buffer.
<code>fifo_free</code>	Deallocate an existing first-in-first-out buffer.
<code>fifo_get</code>	Get next item from first-in-first-out buffer.
<code>fifo_put</code>	Add an item to first-in-first-out buffer.
<code>fifo_rdy</code>	Check first-in-first-out buffer for item availability.

### ■ Type-ahead Buffers

The following functions are declared in the header files `DRIVER.H`.

<code>ta_alloc</code>	Allocate a “type-ahead” buffer.
<code>ta_free</code>	Deallocate a type-ahead buffer.
<code>ta_get</code>	Get the next byte from a type-ahead buffer.
<code>ta_put</code>	Add a character to a type-ahead buffer.
<code>ta_rdy</code>	Test character availability for a type-ahead buffer.

## 14 Functions by Category

---

### ■ Other Buffers

The following functions are declared in the header files MEMORY.H and DRIVER.H.

<code>memcpy,</code> <code>_fmemcpy</code>	Copy one buffer to another buffer until a specific character is encountered or for a specific length.
<code>memchr,</code> <code>_fmemchr</code>	Locate the first occurrence of a character in a buffer.
<code>memcmp,</code> <code>_fmemcmp</code>	Compare two buffers.
<code>memcpy,</code> <code>_fmemcpy</code>	Copy one buffer to another buffer for a specific length.
<code>memdcmp,</code> <code>_fmemdcmp</code>	Compare two buffers using a case-insensitive dictionary order.
<code>memcmp,</code> <code>_fmemcmp</code>	Compare two buffers for equality.
<code>memicmp,</code> <code>_fmemicmp</code>	Compare two buffers, case-insensitive.
<code>memmove,</code> <code>_fmemmove</code>	Copy one buffer to another buffer “intelligently.”
<code>memrcpy,</code> <code>_fmemrcpy</code>	Copy one buffer to another buffer in reverse direction.
<code>memset,</code> <code>_fmemset</code>	Initialize buffer to a specific value.
<code>swab</code>	Copy one buffer to another, swapping adjacent bytes in the process.

## ■ Character Classification & Conversion

The following functions are declared in the header file `CTYPE.H`.

<code>isalnum</code>	Test for alphanumeric character.
<code>isalpha</code>	Test for alphabetic character.
<code>isascii</code>	Test for ASCII character.
<code>iscntrl</code>	Test for control character.
<code>iscsym</code>	Test for valid 1st character of C language symbol name.
<code>iscsymnum</code>	Test for valid character of C language symbol name.
<code>isdigit</code>	Test for decimal digit.
<code>isdisp</code>	Test if character is displayable, non-white space character or the space character. Includes isprint characters, line graphics and international characters.
<code>isfnsym</code>	Test for valid character of a file name.
<code>isgraph</code>	Test if character is displayable excluding white space.
<code>ishex</code>	Test if character is a hexadecimal digit.
<code>islower</code>	Test for lowercase character.
<code>isocal</code>	Test if character is an octal digit.
<code>isprint</code>	Test if character is a normal, displayable character.
<code>ispunct</code>	Test if character is punctuation.
<code>isspace</code>	Test if character is a white space character.
<code>issymbol</code>	Test if character can be used as 1st character of symbol or file name.
<code>issymnum</code>	Test if character can be used in symbol or file name.
<code>isupper</code>	Test if character is an uppercase letter.
<code>isxdigit</code>	Test if character is a hexadecimal digit.
<code>toascii</code>	Convert value to ASCII code.
<code>tolower,</code> <code>_tolower</code>	Convert letter character to lowercase.
<code>toupper,</code> <code>_toupper</code>	Convert letter character to uppercase.

### ■ Console Input & Output

The following functions are declared in the header files CONIO.H and STDIO.H.

<code>at, atstr</code>	Position text cursor on console or generate text cursor positioning string.
<code>cgets</code>	Get a character string from the console.
<code>conmask</code>	Set the console input conversion mask.
<code>conrdy, _conrdy</code>	Test if character is ready from console.
<code>cprintf</code>	Write formatted string to console.
<code>cputs</code>	Write string to console.
<code>crt</code>	Write video attribute, editing code to console.
<code>crtcolor</code>	Set color attributes for console.
<code>CursorBlock,</code> <code>CursorHide,</code> <code>CursorLine,</code> <code>CursorShow</code>	Change the display of the text cursor.
<code>getch</code>	Get a character from the console.
<code>_getline</code>	Accept a line of input from the console.
<code>getll, getpl</code>	Get line length or page length of console or printer file.
<code>has, hascolor</code>	Test if console has character attribute capability or if it supports text colors.
<code>menu, menu2</code>	Display menu on console and allow operator to select an item.
<code>pagewait</code>	Display page wait prompt and await operator release.
<code>putch</code>	Write a character to the console.
<code>ungetch</code>	“Unget” a character from the console.
<code>yesno, _yesno,</code> <code>yesnoall</code>	Accept a “Yes”/“No” response.

## ■ Data Conversion

The following functions are declared in the header file `STDLIB.H`.

<code>a64l</code>	Convert from base 64 ASCII string to long integer.
<code>atof</code>	Convert string to float.
<code>atoi</code>	Convert string to integer.
<code>atol, atol</code>	Convert string to long integer.
<code>base64_decode</code>	Translate binary to MIME base64.
<code>base64_encode</code>	Translate MIME base64 to binary.
<code>Basic2C</code>	Translate BASIC-language style short string to C string.
<code>BasicL2C</code>	Translate BASIC-language style long string to C string.
<code>bcd2ieee</code>	Convert BCD float to IEEE float.
<code>C2Basic</code>	Translate C string to BASIC-language style short string.
<code>C2BasicL</code>	Translate C string to BASIC-language style long string.
<code>c3tol</code>	Convert three-byte integer to long integer.
<code>crc16</code>	Generate a new 16-bit checksum.
<code>crc32</code>	Generate a new 16-bit checksum.
<code>ecvt</code>	Convert double to string.
<code>fcvt</code>	Convert double to string.
<code>_FP_OFF, _FP_SEG</code>	Get offset and memory segment from far pointer.
<code>ftoa</code>	Convert float to string.
<code>gcv</code>	Convert double to string.
<code>i2tol</code>	Convert two-byte integer to long integer.
<code>ieee2bcd</code>	Convert IEEE float to BCD float.
<code>itoa</code>	Convert integer to string.
<code>l3tol</code>	Convert three-byte integers to long integers.
<code>l64a</code>	Convert long integer to base 64 ASCII string.
<code>lltoa, ltoa</code>	Convert long integer to string.
<code>ltoc3</code>	Convert long integer to three-byte integer.
<code>ltoi2</code>	Convert long integer to two-byte integer.
<code>ltol3</code>	Convert long integers to three-byte integers.
<code>mblen</code>	Compute length of multi-byte character string.
<code>mbstowcs</code>	Convert multi-byte character string to wide character string.
<code>mbtowc</code>	Convert multi-byte character to wide character.
<code>_MK_FP</code>	Make far pointer from offset and memory segment.

## 18 Functions by Category

---

<a href="#">sgetl</a>	Convert from <a href="#">sputl</a> format to long integer.
<a href="#">sputl</a>	Convert long integer to machine-independent form.
<a href="#">strtod</a>	Convert string to double.
<a href="#">strtol</a>	Convert string to long integer.
<a href="#">strtoul</a>	Convert string to unsigned long integer.
<a href="#">swapbits</a>	Transpose all bits in an integer.
<a href="#">toascii</a>	Convert value to ASCII code.
<a href="#">tolower</a> , <a href="#">_tolower</a>	Convert letter character to lowercase.
<a href="#">toupper</a> , <a href="#">_toupper</a>	Convert letter character to uppercase.
<a href="#">utoa</a>	Convert unsigned long integer to string.
<a href="#">wcstombs</a>	Convert wide character string to multi-byte character string.
<a href="#">wctomb</a>	Convert wide character to multi-byte character.

## ■ Device-Driver Routines

The following functions are declared in the header files DRIVER.H, SC.H and STDLIB.H.

<code>_baud2cd</code>	Convert baud rate value to a code.
<code>bell_oneshot</code>	Initiate a “one-shot” timer for sounding the console bell.
<code>_cd2baud</code>	Convert baud rate code to baud rate value.
<code>_con_tran</code>	Translate an input character according to console class code specifications.
<code>crt_oneshot</code>	Initiate a “one-shot” timer for the console.
<code>_detach</code> ‡	Detach a device driver.
<code>_devopen</code>	Open or close a loaded device driver.
<code>floppy_oneshot</code>	Initiate a “one-shot” timer for floppy disk device.
<code>free_oneshot</code>	Terminates timer and frees handle for a programmed “one-shot” timer.
<code>free_sel, _free_sel</code>	Release a memory-segment selector.
<code>get_oneshot</code>	Allocate handle for “one-shot” timer.
<code>intoff, inton</code>	Disable or enable interrupts.
<code>_ioctl_uch</code> ‡	Perform control or status operation on a device.
<code>make_alias, _make_alias</code>	Make an alias to a memory-segment selector.
<code>make_sel, _make_sel</code>	Create a memory selector.
<code>mvga_oneshot</code>	Initiate a “one-shot” timer for VGA terminal usage.
<code>oneshot</code>	Use an allocated “one-shot” timer.
<code>_PhyAddr, _phy_addr</code>	Convert selector, offset, length to actual 32-bit address.
<code>_quitoff</code>	Disable break-Q ability.
<code>_quiton</code>	Re-enable break-Q ability.
<code>release_dma</code> ‡	Release DMA resource reserved with <code>reserve_dma</code> .
<code>_remote</code>	Invoke function located in another code segment.
<code>reserve_dma</code> ‡	Reserve DMA resource.
<code>_restoreints, _saveints</code>	Save or restore interrupt status.
<code>schedint</code>	Assign an interrupt service routine to an interrupt.
<code>scsi_oneshot</code>	Initiate a “one-shot” timer for the SCSI devices.
<code>yield, _pre_empty, _snu, _yield</code>	Select next user.
<code>ta_alloc</code>	Allocate a type-ahead buffer.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## 20 Functions by Category

---

<code>ta_free</code>	Deallocate a type-ahead buffer.
<code>ta_get</code>	Get the next byte from a type-ahead buffer.
<code>ta_put</code>	Add a character to a type-ahead buffer.
<code>ta_rdy</code>	Test character availability for a type-ahead buffer.
<code>ub_alloc</code>	Allocate a user buffer for driver.
<code>ub_free</code>	Deallocate a user buffer.
<code>uncache,</code> <code>_uncache</code>	Flushes memory cache for a memory selector.
<code>_unwait</code>	Awaken a process that was waiting for an interrupt.
<code>_wait</code>	Suspend operation until an interrupt occurs.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.



## ■ Directory & Library Maintenance

The following functions are declared in the header file SC.H, STDIO.H or DIRECT.H.

<code>chdir</code>	Change current working directory.
<code>getcwd</code>	Get current working directory.
<code>getftime</code> , <code>getfiledate</code>	Get last change date for a file.
<code>makelib</code> , <code>_makelib</code> , <code>_mklib</code>	Create a new library.
<code>mkdir</code>	Create new subdirectory.
<code>rmdir</code>	Erase existing subdirectory.

## ■ Directory Searching

The following functions are declared in the header file DISKFIND.H.

<code>dirclose</code> ‡	Close disk directory after searching.
<code>_dirfdb</code> ‡	Get fdb of file found with <code>dirread</code> .
<code>diropen</code> ‡	Open disk directory for searching.
<code>dirread</code> ‡	Get next directory entry matching original mask.
<code>_dir_uch</code> ‡	Get uch of file found with <code>dirread</code> .
<code>find_first</code> ‡	Find first directory entry with mask given.
<code>find_next</code> ‡	Find next directory entry.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

### ■ Disk Management & Maintenance

The following functions are declared in the header files DISKSIZE.H, FORMAT.H or DISKFIND.H.

<code>_dir_mount</code>	Allow change of removable disk.
<code>_mount_fs</code> ‡	
<code>_disk_capacity</code> ‡	Determine a disk's capacity and parameters.
<code>getvol</code>	Get disk volume name of device.
<code>_sync</code>	Disk cache synchronize.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## ■ Environment Access

The following functions are declared in the header files SC.H and STDLIB.H.

<code>cmd_abbrev</code>	Using <code>SYSTEM.TEOSnnn.SYNONYM</code> , determine full name of a command abbreviation.
<code>cuserid</code>	Get the current log-on name.
<code>getbaud</code>	Get the baud rate value for an attached device.
<code>getclass</code>	Get class code number for an attached device.
<code>getenv</code>	Get value associated with environment name.
<code>getlang</code>	Get current language code.
<code>getlib</code>	Get default library name.
<code>getlogin</code>	Get the current log-on name.
<code>getlub</code>	Get the ucb associated with lub.
<code>_getosver</code>	Get coded operating system version number.
<code>getpriv</code>	Get current user's privilege level.
<code>getucb, GetUCB</code>	Get pointer to unit control block (ucb) for device.
<code>getuid</code>	Get user's id.
<code>getver</code>	Get version number, name or serial number of operating system, or the version number or date of the program.
<code>ispcterm</code>	Determine if console is PC Term-compatible.
<code>isprnum</code>	Test and return logical unit number (lub) for printer, given an option keyword such as "PRT3."
<code>isvga</code>	Determine if console is VGA-compatible.
<code>logname</code>	Get the current log-on name.
<code>_lubname</code>	Get name for device number.
<code>_osmajor,</code> <code>_osminor,</code> <code>_osversion</code>	Get operating system version number.
<code>putenv, _putenv</code>	Set value for environment name.
<code>_setprty,</code> <code>_set_slice</code>	Change the priority or slice time of the program.
<code>WhoAml</code>	Get "Who Am I" information about the current console.

### ■ Error Handling & Exiting

The following functions are declared in the header files `IO.H`, `SC.H`, `STDIO.H` or `STDLIB.H`.

<code>abort</code>	Exit program with return code of 254.
<code>atexit</code> , <code>_atexit_clear</code> , <code>_atexit_remove</code>	Specify functions to execute when program exits.
<code>clearerr</code>	Clear file error status indicators.
<code>eof</code>	Test file for end-of-file status.
<code>_errbot</code>	Display message at bottom of screen and await reply.
<code>errmsg</code>	Display error message on <code>stderr</code> with parameter substitution; exit program.
<code>errno</code>	Variable name of ANSI function error number.
<code>_errno</code>	Variable name of THEOS error number.
<code>exit</code>	Exit program normally with return code of 0.
<code>feof</code>	Test file for end-of-file status.
<code>ferror</code>	Test open file for its error status.
<code>file_err</code>	Test open file for its error status and, if error exists, use <code>syserr</code> to exit program.
<code>f perror</code>	If file error exists, display related message on <code>stderr</code> and clear error status.
<code>getmsg</code>	Get message text for a system-defined message.
<code>perror</code>	Display user-defined message on <code>stderr</code> followed by file error message.
<code>putmsg</code>	Display system error message on <code>stdout</code> with parameter substitution.
<code>strerror</code>	Format system error message with parameter substitution.
<code>syserr</code>	Display system error message on <code>stderr</code> with parameter substitution and exit program.

## ■ File Handling

The file-handling functions are of two basic types: Standard or Unix-style.

### ■ Standard File Access

The following functions are declared in the header file `STDIO.H`.

<code>access</code>	Test file access permissions.
<code>_attach</code>	Attach a device or change attachment parameters.
<code>clearerr</code>	Clear file error indicators.
<code>creat</code>	Create a new file.
<code>deletek,</code> <code>ldeletek</code>	Delete a keyed-access file record.
<code>erase,</code> <code>remove,</code> <code>unlink</code>	Erase an existing file.
<code>fbuf</code>	Allocate a read/write buffer for a file.
<code>fclose</code>	Close an open file.
<code>fcloseall</code>	Close all open files.
<code>fcntl</code>	Change a file's attributes.
<code>fcreate</code>	Create a new file.
<code>feof</code>	Test file for end-of-file status.
<code>ferror</code>	Test open file for its error status.
<code>fflush</code>	Write a file's buffer to the actual file.
<code>fgetc</code>	Get a character from a stream file.
<code>fgetl</code>	Get a long integer from a stream file.
<code>fgetpos</code>	Get the current position pointer for a file.
<code>fgets</code>	Get a string from a stream file.
<code>fgetsn</code>	Get a string of fixed length from a stream file.
<code>fgetw</code>	Get an integer from a file.
<code>fgrow</code>	Change file's growth factor.
<code>_file_alloc</code> ‡	Get number of bytes of disk space currently allocated to file.
<code>_filechange</code>	Change a file's directory entry.
<code>file_err</code>	Test open file for its error status and, if error exists, use <code>syserr</code> to exit program.
<code>filelock</code>	Lock a region of a file to prevent other user's access.
<code>fileno,</code> <code>xfileno</code>	Get file handle for open file.
<code>flushall</code>	Flush all open file buffers to disk.
<code>fopen,</code> <code>xfopen</code>	Open a file for access.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## 26 Functions by Category

---

<code>ferror</code>	If file error exists, display related message on stderr and clear error status.
<code>fprintf</code>	Write formatted string to stream file.
<code>fputc</code>	Write character to stream file.
<code>fputl</code>	Write long integer to stream file.
<code>fputs</code>	Write string to stream file.
<code>fputsn</code>	Write fixed length string to stream file.
<code>fputsnl</code>	Write string to stream file and append new-line character.
<code>fputw</code>	Write integer to stream file.
<code>fread</code>	Read multiple, like-sized objects from stream file.
<code>freopen</code>	Open a file for access and assign to an existing file pointer.
<code>fscanf</code>	Read formatted strings, characters and numbers from a stream file.
<code>fseek</code>	Change the position of a stream file's read/write pointer.
<code>fsetpos</code>	Change the position of a stream file's read/write pointer.
<code>ftell</code>	Get file's current read/write position.
<code>fwrite</code>	Write multiple, like-sized objects to a stream file.
<code>getc</code>	Get a character from a stream file.
<code>getchar</code>	Get a character from stdin.
<code>getdev</code>	Get device number associated with an open file.
<code>getftime</code> , <code>getfiletime</code>	Get file's last change date.
<code>getflat</code>	Get the "flat file" portion of a file description.
<code>getfn</code>	Determine the full file name for a file.
<code>getll</code>	Get line length of console or printer file.
<code>getlub</code>	Get lub for device.
<code>getpl</code>	Get page length of console or printer file.
<code>gets</code>	Get a string from a stream file.
<code>_get_sect</code>	Get disk directory sector for a file.
<code>getw</code>	Get an integer from a stream file.
<code>hasprt</code>	Test if printer has character attribute.
<code>_ioctl</code>	Low-level control device associated with an open file.
<code>isatty</code>	Test file to see if it is a serial communications device.
<code>iscon</code>	Test file to see if it is the console.
<code>locate</code>	Get directory entry (fdb) for a file.
<code>lockf</code> , <code>locking</code> <code>xlocking</code>	Lock or unlock record at current position of stream file.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

<code>longfname</code>	Get and format complete file name, including path.
<code>mktemp</code>	Determine and create next unique work or temporary file name using specified mask.
<code>_norm_fn</code>	“Normalize” file name.
<code>perror</code>	Display user-defined message on <code>stderr</code> followed by file error message.
<code>pipe</code>	Open an interprocess input/output file.
<code>printf</code>	Display formatted characters, strings and numbers on <code>stdout</code> .
<code>popen</code>	Open a file for interprocess reads and writes.
<code>putchar</code>	Write a character to <code>stdout</code> .
<code>putdate</code>	Change file’s date in directory.
<code>puts</code>	Write a string to a stream file.
<code>putw</code>	Write an integer to a stream file.
<code>readk, lreadk</code>	Read a record from an indexed or keyed access file.
<code>readn, lreadn</code>	Read the next record from an indexed or keyed-access file.
<code>readp</code>	Read the prior record from an indexed or keyed-access file.
<code>reclck</code>	Lock a location in a stream or direct file.
<code>recunlock</code>	Release a locked location in a stream or direct file.
<code>rename</code>	Change the name of a file.
<code>rewind</code>	Change the position of a stream file’s read/write pointer back to the beginning of the file.
<code>scanf</code>	Read formatted characters, strings and numbers from a stream file.
<code>seek, _seek, lseek</code>	Change the position of a stream file’s read/write pointer.
<code>setbuf</code>	Allocate read/write buffer for an open, unused stream file.
<code>setvbuf</code>	Allocate read/write buffer for an open, unused stream file.
<code>tempnam</code>	Determine and create next unique work or temporary file name using specified mask.
<code>testwild</code>	Test file-name specification for presence of wild cards.
<code>tmpfile</code>	Open a temporary work file using <code>tmpnam</code> function.
<code>tmpnam</code>	Determine and create next unique work or temporary file name using default mask.
<code>_tmpnam_drive</code>	Set the drive used by the <code>tmpfile</code> and <code>tmpnam</code> functions.
<code>topen</code>	Open a stream file on a tape drive.
<code>ungetc</code>	“Unget” a character from a stream file.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## 28 Functions by Category

---

<code>unlock</code>	Unlock all records or locations in a file.
<code>vfprintf</code>	Variable argument function to write formatted characters, strings and numbers to a stream file.
<code>vprintf</code>	Variable argument function to write formatted characters, strings and numbers to stdout.
<code>writeln</code> , <code>lwriteln</code>	Write a record to an indexed or keyed file.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.



**■ “UNIX-Style” File Access**

The following functions are declared in the header files `HANDLE.H`, `LOCKING.H` or `UNISTD.H`.

<code>close</code> , <code>xclose</code>	Close an open file.
<code>creat</code> , <code>xcreat</code>	Create and open a new stream file.
<code>dup</code> , <code>dup2</code> , <code>xdup</code>	Create a second handle for an open file.
<code>eof</code>	Test file for end-of-file status.
<code>fdopen</code> , <code>xfdopen</code>	Get file pointer to file opened with <code>creat</code> , <code>open</code> or <code>pipe</code> .
<code>flush</code> , <code>xflush</code>	Write a file’s buffer to the actual file.
<code>lockf</code> , <code>locking</code>	Lock or unlock record at current position of stream file.
<code>lseek</code> , <code>xlseek</code>	Change file read/write pointer.
<code>open</code> <code>xopen</code>	Open a stream file.
<code>pipe</code>	Open an interprocess communications file.
<code>read</code> , <code>xread</code>	Read data from a file.
<code>tell</code> , <code>xtell</code>	Get current read/write position.
<code>write</code> , <code>xwrite</code>	Write data to a file.

### ■ Graphics

The following functions are declared in the header file VDI.H.

<code>vdi</code>	Perform VDI operation.
<code>vdialpha</code>	Change graphics device to normal, non-graphics mode.
<code>vdiaarc</code>	Draw an arc.
<code>vdibar</code>	Draw a rectangle.
<code>vdicircle</code>	Draw a circle.
<code>vdiclear</code>	Clear or eject graphics page.
<code>vdiclose</code>	Close the graphics device.
<code>vdigraph</code>	Change graphics device to graphics mode.
<code>vdiline</code>	Draw a straight line.
<code>vdimark</code>	Draw a marker.
<code>vdioopen</code>	Open graphics device.
<code>vdipie</code>	Draw a “pie slice.”
<code>vditext</code>	Draw text.

### ■ Define Graphic Attributes

<code>vdisetfc</code>	Set graphics fill color.
<code>vdisetfs</code>	Set graphics fill style.
<code>vdisetlc</code>	Set graphics line color.
<code>vdisetlh</code>	Set graphics line size.
<code>vdisetls</code>	Set graphics line style.
<code>vdisetmc</code>	Set graphics marker color.
<code>vdisetmh</code>	Set graphics marker size.
<code>vdisetms</code>	Set graphics marker style.
<code>vdisetta</code>	Set graphics text angle.
<code>vdisettc</code>	Set graphics text color.
<code>vdisetth</code>	Set graphics text size.
<code>vdisettp</code>	Set graphics text path.
<code>vdisetts</code>	Set graphics text style or font.

## ■ Configuration File Access & Maintenance

The following functions are declared in the header file INITFILE.H.

```
struct    INITFILE {
        unsigned short seg; // initfile image
        char mode;         // read or write mode
        char modified;     // set if modified
        long length;       // length
        long size;         // allocated size
        char filename[256]; // name of configuration file
    };
```

<code>InitFileClose</code> ‡	Close configuration/initialization file.
<code>InitFileOpen</code> ‡	Open configuration/initialization file.
<code>InitFileRead</code> ‡	Find and get item from configuration/initialization file.
<code>InitFileReadInt</code> ‡	Find and get integer item from configuration/initialization file.
<code>InitFileReadNext</code> ‡	Get next item from configuration/initialization file.
<code>InitFileWrite</code> ‡	Put item to configuration/initialization file.
<code>InitFileWriteInt</code> ‡	Put integer item to configuration/initialization file.

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

### ■ Device Names, Help, Keyword, Menu & Message File Routines

The following functions are declared in the header file `STDLIB.H`.

<code>devnames_close</code>	Close the device names file.
<code>devnames_open</code>	Open the device names file.
<code>devnames_parse</code>	Parse the current record read from the device names file.
<code>devnames_read</code>	Get the next non-blank, non-comment record from the device names file.
<code>_get_help_filename,</code> <code>_get_menu_filename</code>	Get full file name for help file or menu file associated with program.
<code>getkey</code>	Open keywords file if necessary and get keyword token.
<code>getmsg</code>	Open system messages file if necessary and get message text.
<code>keyclose</code>	Close the keywords file.
<code>load_yn</code>	Get “Yes”/“No” keywords.
<code>matcharg</code>	Test if two strings match using minimum spelling feature.
<code>msgclose</code>	Close the system messages file.
<code>openhelp, openmenu</code>	Open help file or menu file associated with program.
<code>putmsg</code>	Display system error message on <code>stdout</code> with parameter substitution.
<code>testarg</code>	Compare string to system keyword.
<code>yesno, _yesno,</code> <code>yesnoall</code>	Accept a “Yes”/“No” response.

## ■ International

The following functions are declared in the header file `LOCALE.H`.

<code>localeconv</code>	Get information about the current locale settings for numeric data formatting.
<code>setlocale</code>	Set environment for locale.
<code>strcoll</code>	Convert a string according to locale.
<code>strftime</code>	Format a date and time string using locale.

## ■ EXEC Variable Access

The following functions are declared in the header file `EXECVAR.H`.

<code>getexvar</code>	Get the value of the EXEC program variable.
<code>putexvar</code>	Set the value of the EXEC program variable.

### ■ Math

The following functions are declared in the header file `MATH.H`.

<code>abs</code>	Compute the absolute value of integer.
<code>acos</code>	Compute the arccosine.
<code>acot</code>	Compute the arccotangent.
<code>acsc</code>	Compute the arccosecant.
<code>asec</code>	Compute the arcsecant.
<code>asin</code>	Compute the arcsine.
<code>atan, atan2</code>	Compute the arctangent.
<code>bcd2ieee</code>	Convert BCD float to IEEE float.
<code>cabs</code>	Compute the absolute value of complex number.
<code>ceil</code>	Compute the integer ceiling of a number.
<code>cos, cosh</code>	Compute the cosine or hyperbolic cosine.
<code>cot, coth</code>	Compute the cotangent or hyperbolic cotangent.
<code>csc, csch</code>	Compute the cosecant or hyperbolic cosecant.
<code>div, ldiv</code>	Divide one integer by another.
<code>exp</code>	Compute the exponential value.
<code>fabs</code>	Compute the absolute value of a float.
<code>floor</code>	Compute the integral floor.
<code>fmod</code>	Compute the floating-point remainder.
<code>frexp</code>	Compute the exponential value.
<code>fsign</code>	Compute the integer sign of a float.
<code>gcd</code>	Compute the greatest common divisor of two floats.
<code>hypot</code>	Compute the length of the hypotenuse of a triangle.
<code>bcd2ieee</code>	Convert IEEE float to BCD float.
<code>ip</code>	Compute the integer or whole number portion of a float.
<code>labs</code>	Compute the absolute value of a long integer.
<code>ldexp</code>	Compute the product of a float times $2^{\text{exp}}$ .
<code>log, log2, log10</code>	Compute the natural, base 2 or common logarithm.
<code>max, _maxd, _maxw</code>	Determine the larger of two values or series of values.
<code>min, _mind, _minw</code>	Determine the smaller of two values or series of values.
<code>modf</code>	Determine the whole number and fractional portions of a float.
<code>pow</code>	Compute the value of a number raised to a power.
<code>prime, primel</code>	Compute the next prime number.
<code>rand, srand</code>	Compute the next pseudorandom number.

<code>sec, sech</code>	Compute the secant or the hyperbolic secant.
<code>sin, sinh</code>	Compute the sine or the hyperbolic sine.
<code>sqrt</code>	Compute the square root.
<code>tan, tanh</code>	Compute the tangent or the hyperbolic tangent.

The `MATH.H` header file also defines several constants that are useful when manipulating and evaluating mathematical expressions:

Constant name	Value	Comment
<code>HUGE_VAL</code>	1.18973149535723176e+4932 (ieee) 9.999999999999999e+126 (bcd)	Maximum value supported
<code>M_E</code>	2.71828182845904524	base of natural logs
<code>M_LOG2E</code>	1.44269504088896341	base 2 log of $e$
<code>M_LOG10E</code>	0.434294481903251828	base 10 log of $e$
<code>M_LN2</code>	0.693147180559945309	natural log of 2
<code>M_LN10</code>	2.302585809299404568	natural log of 10
<code>M_PI</code>	3.1415926535897932	$\pi$
<code>M_PI_2</code>	1.5707963267948966	$\pi \div 2$
<code>M_PI_4</code>	0.7853981633974483	$\pi \div 4$
<code>M_1_PI</code>	0.31830988618379067	$1 \div \pi$
<code>M_2_PI</code>	0.63661977236758135	$2 \div \pi$
<code>M_2_SQRTPI</code>	1.1283791670955126	$2 \div \sqrt{\pi}$
<code>M_SQRT2</code>	1.41421356237309505	$\sqrt{2}$
<code>M_SQRT1_2</code>	0.707106781186547524	$\sqrt{1/2}$

Table 1: Mathematical Constants

### ■ Memory Management

The following functions are declared in the header file GETMEM.H, MALLOC.H or STDLIB.H.

<code>_alloca</code>	Allocate a block of memory from the program stack.
<code>calloc</code>	Allocate and clear a block of memory from system resources.
<code>free</code>	Release memory allocated.
<code>_getmem</code>	Allocate a block of sharable memory from system resources.
<code>malloc</code>	Allocate a block of memory from system resources.
<code>max_alloc</code>	Determine maximum size of memory that can be allocated at one time.
<code>mem_avail</code>	Determine current amount of memory available for allocation from the system memory pool.
<code>_mem_grow</code>	Change size of allocated memory.
<code>_putmem</code>	Release a block of sharable memory.
<code>realloc</code>	Reallocate an existing chunk of memory.
<code>shared</code>	Access shared memory by name.
<code>tot_alloc</code>	Determine total amount of memory allocated in the heap.



## ■ Multitask Management

The following functions are declared in the header file `PROCESS.H` or `SC.H`.

<code>fork, forktask</code>	Start a copy of this program as a subtask to the current task.
<code>getmpid</code>	Get main process id for multitask programs.
<code>getpid</code>	Get the process id of the current task.
<code>getppid</code>	Get the process id of current task's parent.
<code>killtask</code>	Stop a subtask.
<code>sema, rsema</code>	Test the status of a semaphore flag.
<code>semaphore, rsemaname</code>	Catalogue a semaphore name.
<code>semares, rsemares</code>	Reset a semaphore flag.
<code>semaset, rsemaset</code>	Set a semaphore flag.
<code>semawait, rsemawait</code>	Suspend processing until semaphore flag is set.
<code>shared</code>	Access shared memory by name.
<code>spawnl, spawnlp, spawnv, spawnvp</code>	Start a subtask to the current program.
<code>timer</code>	Set semaphore at a time or upon elapsed time.
<code>yield, _pre_empty, _snu, _yield</code>	Select next user.

### ■ TCP/IP Network Sockets

The following functions are all declared in the header file `SOCKET.H`.

<code>accept</code> ‡	Accept a connection to a socket.
<code>bind</code> ‡	Associate a local address with a socket.
<code>closesocket</code> ‡	Close a socket.
<code>connect</code> ‡	Establish a connection to a peer.
<code>FD_CLR</code> ‡	Remove a socket number.
<code>FD_ISSET</code> ‡	Test for the existence of socket number.
<code>FD_SET</code> ‡	Add a socket number.
<code>FD_ZERO</code> ‡	Clear all socket numbers.
<code>gethostbyaddr,</code> <code>gethostbyname</code> ‡	Get host information corresponding to an address or a host name.
<code>gethostname</code> ‡	Get the standard host name for the local machine.
<code>getpeername</code> ‡	Get the address of the peer to which a socket is connected
<code>getprotobyname,</code> <code>getprotobynum-</code> <code>ber</code> ‡	Get protocol information corresponding to a protocol name or number.
<code>getservbyname,</code> <code>getservbyport</code> ‡	Get service information corresponding to a service name and protocol or a port and protocol.
<code>getsockname</code> ‡	Get the local name for a socket.
<code>getsockopt</code> ‡	Get a socket option for a specific socket.
<code>htonl, htons</code> ‡	Convert numbers from host-byte order to network-byte order.
<code>inet_addr,</code> <code>inet_ntoa</code> ‡	Converts internet address between the Internet standard “dotted” form and the struct <code>IN_ADDR</code> form.
<code>ioctlsocket</code> ‡	Control the mode of a socket.
<code>listen</code> ‡	Set a socket to listen for incoming connection.
<code>ntohl, ntohs</code> ‡	Convert numbers from network-byte order to host-byte order.
<code>recv, recvfrom</code> ‡	Receive data from a socket.
<code>select</code> ‡	Determine the status of one or more sockets.
<code>send, sendto</code> ‡	Send data to a socket.
<code>setsockopt</code> ‡	Set a socket option.
<code>socket</code> ‡	Create a socket.
<code>TSAGetLastError</code> ‡	Get the error code result of the last THEOS Socket function.
<code>TSASetLastError</code> ‡	Set the error code that is retrieved by <code>TSAGetLastError</code> .
<code>TSAStrError</code> ‡	Return a pointer to a message string matching the THEOS Sockets error number.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## ■ Peek & Poke Far Memory

The following functions are all declared in the header file PEEK.H.

<a href="#">_peekb</a> , <a href="#">_pokeb</a>	Get or put a byte in another memory segment.
<a href="#">_peekcsb</a> , <a href="#">_pokecsb</a>	Get or put a byte to the current program's code segment.
<a href="#">_peekcsd</a> , <a href="#">_pokecsd</a>	Get or put a long integer to the current program's code segment.
<a href="#">_peekcsp</a> , <a href="#">_pokecsp</a>	Get or put a pointer to the current program's code segment.
<a href="#">_peekcsw</a> , <a href="#">_pokecsw</a>	Get or put an integer to the current program's code segment.
<a href="#">_peekd</a> , <a href="#">_poked</a>	Get or put a long integer in another memory segment.
<a href="#">_peeknucb</a> , <a href="#">_pokenucb</a>	Get or put a byte in the nucleus code segment.
<a href="#">_peeknucd</a> , <a href="#">_pokenucd</a>	Get or put a long integer in the nucleus code segment.
<a href="#">_peeknucp</a> , <a href="#">_pokenucp</a>	Get or put a pointer in the nucleus code segment.
<a href="#">_peennucw</a> , <a href="#">_pokenucw</a>	Get or put an integer in the nucleus code segment.
<a href="#">_peekp</a> , <a href="#">_pokep</a>	Get or put a pointer in another memory segment.
<a href="#">_peekscrib</a> , <a href="#">_pokescrib</a>	Get or put a byte in the current user's system communication region (scr) segment.
<a href="#">_peekscrd</a> , <a href="#">_pokescrd</a>	Get or put a long integer in the current user's system communication region (scr) segment.
<a href="#">_peekscrip</a> , <a href="#">_pokescrip</a>	Get or put a pointer in the current user's system communication region (scr) segment.
<a href="#">_peekscriw</a> , <a href="#">_pokescriw</a>	Get or put an integer in the current user's system communication region (scr) segment.
<a href="#">_peektib</a>	Get a long integer from the task information block (tib) segment.
<a href="#">_peekp</a> , <a href="#">_pokew</a>	Get or put an integer in another memory segment.

### ■ Process Control

The following functions are declared in the header file `PROCESS.H` or `SC.H`.

<code>execl,</code> <code>execlp,</code> <code>execv,</code> <code>execvp</code>	Continue execution in another program.
<code>exit</code>	Normal program termination.
<code>fork, forktask</code>	Start a copy of this program as a subtask to the current task.
<code>killtask</code>	Stop a subtask.
<code>load, unload</code>	Load another program into memory.
<code>_lockres</code>	Clear lock on memory region.
<code>_lockset</code>	Set lock on memory region.
<code>_lockshare</code> ‡	Place a shared lock on the semaphore.
<code>_locktest</code> ‡	Test if memory region is locked.
<code>_lockwait</code> ‡	Wait for memory region to be unlocked.
<code>_logon</code>	Log onto another account.
<code>longjmp</code>	Return to <code>setjmp</code> location.
<code>setjmp</code>	Save current environment for subsequent return by <code>longjmp</code> .
<code>spawnl,</code> <code>spawnlp,</code> <code>spawnv,</code> <code>spawnvp</code>	Start a subtask to the current program.
<code>system</code>	Execute another program and, upon completion, return to this program.
<code>csi</code>	Exit this program and start another program.
<code>unload</code>	Unload a program from memory.
<code>yield, _pre_empty,</code> <code>_snu, _yield</code>	Select next user.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

**■ Searching & Sorting**

The following functions are declared in the header file `SEARCH.H`.

<code>binsert</code>	Perform a binary search and, if not found, insert item.
<code>bsearch</code>	Perform a binary search.
<code>lfind</code>	Perform a linear search.
<code>lsearch</code>	Perform a linear search and, if not found, add item.
<code>qsort</code>	Perform quick sort of an array.
<code>regcmp</code>	Prepare for a regular expression match.
<code>regex</code>	Perform regular expression search of a string.
<code>sort</code>	Perform a sort of an array.

### ■ Standard Input & Output

The following functions are declared in the header file `STDIO.H`.

<code>clearerr</code>	Clear file error status indicators.
<code>fbuf</code>	Allocate a read/write buffer for a file.
<code>fclose, fcloseall</code>	Close an open file or all open files.
<code>fdopen</code>	Get file pointer to file opened with <code>creat</code> , <code>open</code> or <code>pipe</code> .
<code>feof</code>	Test file for end-of-file status.
<code>ferror</code>	Test open file for its error status.
<code>fgetc, fgetl, fgets, fgetsn, fgetw, getc, getchar</code>	Get a character, long integer, character string or an unsigned integer from a stream file.
<code>fgrow</code>	Change file's growth factor.
<code>file_err</code>	Test open file for its error status and, if error exists, use <code>syserr</code> to exit program.
<code>filelock</code>	Lock a region of a file to prevent other user's access.
<code>fileno</code>	Get file handle for open file.
<code>flush, flushall, fflush</code>	Flush open file's buffers to disk.
<code>fpperror</code>	If file error exists, display related message on <code>stderr</code> and clear error status.
<code>fputc, fputl, fputs, fputsn, fputsnl, fputw, putc, putchar</code>	Write a character, long integer, character string or an unsigned integer to a stream file.
<code>getdev</code>	Get device number associated with an open file.
<code>getfn</code>	Get full name of file.
<code>getw</code>	Get an integer from a stream file.
<code>isatty</code>	Test file to see if it is a serial communications device.
<code>iscon</code>	Test file to see if it is the console.
<code>pipe</code>	Open an interprocess input/output file.
<code>popen</code>	Open a file for interprocess read and writes.
<code>putw</code>	Write an integer to a stream file.
<code>relock, recunlock</code>	Lock a location in a stream or direct file or release a locked location.
<code>scanf</code>	Accept a formatted string from <code>stdin</code> .
<code>setbuf, setvbuf</code>	Allocate a read/write buffer for a file.
<code>testhead</code>	Output page heading to <code>stdout</code> if required.
<code>unlock</code>	Unlock all records or locations in a file.

## ■ String & Memory Manipulation

The following functions are all declared in the header file `STRING.H`.

They all manipulate “strings” of characters or bytes, either null-terminated or not.

Functions that start with “str” or “\_fstr” operate on null-terminated strings. Functions starting with “mem” or “\_fmem” operate on strings that are not null-terminated.

Functions starting with “\_f” operate similarly to the functions that do not start with this letter sequence, but on far objects.

<code>matcharg</code>	Test if two strings match using minimum spelling feature.
<code>memcpy,</code> <code>_fmemcpy</code>	Copy bytes from one buffer to another until a specific character is encountered or until specific number of bytes copied.
<code>memchr,</code> <code>_fmemchr</code>	Point to next occurrence of a specific character in a buffer.
<code>memcmp,</code> <code>_fmemcmp</code>	Compare two buffers for a given length.
<code>memcpy,</code> <code>_fmemcpy</code>	Copy bytes from one buffer to another for a given length.
<code>memicmp,</code> <code>_fmemicmp</code>	Compare two buffers using a case-insensitive dictionary order.
<code>memcmp,</code> <code>_fmemcmp</code>	Compare two buffers for a given length for equality.
<code>memcmp,</code> <code>_fmemcmp</code>	Compare two buffers for a given length ignoring differences in case mode of characters.
<code>memcpy,</code> <code>_fmemcpy</code>	Copy bytes from one buffer to another for a given length, overlap allowed.
<code>memcpy,</code> <code>_fmemcpy</code>	Copy bytes from one buffer to another for a given length, in reverse sequence.
<code>memset,</code> <code>_fmemset</code>	Initialize a buffer for a given length to a specific value.
<code>regcmp</code>	Prepare for a regular expression match.
<code>regex</code>	Perform regular expression search of a string.
<code>skipsp</code>	Find next nonspace character in string.
<code>sprintf</code>	Create a string using formatting mask.
<code>sscanf</code>	Extract components of a formatted string.
<code>strcat,</code> <code>_fstrcat</code>	Append one string onto another.
<code>strchr,</code> <code>_fstrchr</code>	Find first occurrence of a character in a string.
<code>strcmp,</code> <code>_fstrcmp</code>	Compare two strings.

<code>strcoll</code>	Generate string according to current locale that is suitable for comparing with another <code>strcoll</code> string.
<code>strcpy,</code> <code>_fstrcpy</code>	Copy one string to another string location.
<code>strcspn,</code> <code>_fstrcspn</code>	Find first occurrence of a character from a character set in a string.
<code>strdcmp,</code> <code>_fstrdcmp</code>	Compare two strings that might contain decimal numbers.
<code>strdup</code>	Using <code>malloc</code> , create a new copy of a string in newly allocated memory.
<code>strend,</code> <code>_fstrend</code>	Point to end of a string.
<code>streq,</code> <code>_fstreq</code>	Compare two strings for equality.
<code>stricmp,</code> <code>_fstricmp</code>	Compare two strings without regard to case mode of characters.
<code>stristr,</code> <code>_fstristr</code>	Find first occurrence of a character sequence in a string without regard to the upper or lowercase mode of characters.
<code>strlen,</code> <code>_fstrlen</code>	Compute the length of a string.
<code>strlwr</code>	Convert characters in string to lowercase.
<code>strmake</code>	Copy bytes from a memory buffer to a string location for a given length, append the string terminator character.
<code>strmatch</code>	Test if string matches format mask.
<code>strncat</code>	Append fixed number of characters of one string or buffer to a string.
<code>strncmp</code>	Compare two strings for a fixed length.
<code>strncpy</code>	Copy fixed number of characters from one string to another, padding if necessary.
<code>strneq</code>	Compare two strings for a fixed length for equality.
<code>strnicmp</code>	Compare two strings for a fixed length without regard to the upper or lowercase mode of the characters.
<code>strnset</code>	Initialize a string to a given character value for a given length.
<code>strpbrk,</code> <code>_fstrpbrk</code>	Find first occurrence of any of a number of characters in a string.
<code>strrchr,</code> <code>_fstrchr</code>	Find last occurrence of a character in a string.
<code>strset,</code> <code>_fstrset</code>	Set all characters in a string to a given value.
<code>strspn,</code> <code>_fstrspn</code>	Find first occurrence of a substring in a string.
<code>strstr,</code> <code>_fstrstr</code>	Find first occurrence of a substring in a string.



`strtok,`  
`_fstok`

Find next token in a string.

`strtrim`

Remove extra spaces from string.

`strupr`

Convert characters in string to uppercase.

`vsprintf`

Variable argument function to write formatted characters, strings and numbers to a string.

### ■ Time & Date

The following functions and declarations are declared in the header file `TIME.H`.

```
typedef long clock_t;
typedef unsigned long size_t;
typedef unsigned long time_t;

struct tm {           // time structure
    short tm_sec;      // seconds after minute 0-59
    short tm_min;      // minutes after hour 0-59
    short tm_hour;     // hours since midnight 0-23
    short tm_mday;     // day of month 1-31
    short tm_mon;      // months since January 0-11
    short tm_year;     // years since 1900
    short tm_wday;     // days since Sunday 0-6
    short tm_yday;     // days since January 1 0-365
    short tm_isdst;    // Daylight Savings Time flag
};

extern int daylight;   // daylight time supported?
extern long timezone;  // seconds from UTC
extern char * tzname[2]; // Standard and Daylight names
```

<code>asctime</code>	Convert time in a time structure to a character string.
<code>clock</code>	Return the elapsed CPU time for the current process.
<code>ctime</code>	Convert time from long integer to character string.
<code>difftime</code>	Compare two times in time structures.
<code>gmtime</code>	Convert long integer UTC calendar time to time structure.
<code>_leapyear</code>	Determine if given year is a leap year.
<code>localtime</code>	Convert long integer UTC calendar time to local time in a time structure.
<code>mktime</code>	Convert time structure to long integer calendar time.
<code>strftime</code>	Convert and format time structure information into a string.
<code>time</code>	Get current date and time and convert to long integer UTC calendar time.
<code>tzset</code>	Initialize the time zone fields according to your TZ environment information.

The following functions are declared in the header file `TIMER.H`.

<code>getdate</code>	Get the current date as a character string.
<code>getmsec</code>	Get the current time in milliseconds; returns as a character string.
<code>gettime</code>	Get the current time as a character string.

## ■ Signaling & Timing Control

The following functions are declared in the header files SC.H, SIGNAL.H or TIMER.H.

<code>alarm</code>	Define timer for signal interrupt.
<code>delay</code>	Suspend program operation for a preset time period.
<code>msalarm</code>	Define timer for millisecond signal interrupt.
<code>raise</code>	Raise the signal interrupt.
<code>signal</code>	Define action to take when signal interrupt occurs.
<code>sleep,</code> <code>sleep_msec,</code> <code>sleep_sec,</code> <code>sleep_until</code>	Suspend program operation for a preset time period.
<code>timer</code>	Set semaphore at a specified time or upon elapsed time.

### ■ THEOS WorkStation Control

The following functions are all declared in the header file TWS.H.

<code>TWS_disconnect</code> ‡	Enables or disables the ability to disconnect the workstation manually.
<code>TWS_disconnect_now</code> ‡	Disconnects the workstation.
<code>TWS_execute</code> ‡	Forces the client workstation to execute another program in another window.
<code>TWS_focus</code> ‡	Selects the window used by this program as the active window.
<code>TWS_maximize</code> ‡	Maximizes the window used by the client workstation to its full size and selects it as the focus or active window.
<code>TWS_minimize</code> ‡	Minimizes the window used by the client workstation to an icon and selects another window as the focus.
<code>TWS_ontop</code> ‡	Enables or disables the window “on top” feature.
<code>TWS_receive</code> ‡	Transfers a file from the client workstation to the THEOS NetServer.
<code>TWS_restore</code> ‡	Restores the window used by the client workstation to the size used prior to the last maximize or minimize window operation.
<code>TWS_send</code> ‡	Transfers a file from the THEOS NetServer to the client workstation.
<code>TWS_title</code> ‡	Sets the title text displayed for the window on the client workstation.
<code>TWS_user_focus</code> ‡	Enables or disables the ability to select another window on the client workstation.

---

‡ Functions marked with this symbol can only be used with programs compiled with and executing on THEOS 32 Version 4 or greater.

## ■ Trigonometric Functions

The following functions are declared in the header file `MATH.H`.

<code>acos</code>	Compute the arccosine.
<code>acot</code>	Compute the arccotangent.
<code>acsc</code>	Compute the arccosecant.
<code>asec</code>	Compute the arcsecant.
<code>asin</code>	Compute the arcsine.
<code>atan, atan2</code>	Compute the arctangent.
<code>cos, cosh</code>	Compute the cosine or the hyperbolic cosine.
<code>cot, coth</code>	Compute the cotangent or the hyperbolic cotangent.
<code>csc, csch</code>	Compute the cosecant or the hyperbolic cosecant.
<code>hypot</code>	Compute the length of the hypotenuse of a triangle.
<code>log, log2, log10</code>	Compute the natural, base 2 or common logarithm.
<code>sec, sech</code>	Compute the secant or the hyperbolic secant.
<code>sin, sinh</code>	Compute the sine or the hyperbolic sine.
<code>tan, tanh</code>	Compute the tangent or the hyperbolic tangent.

### ■ Variable-Argument Management

The following functions are all declared in the header file `STDARG.H`.

```
typedef long  
*va_list[0];
```

<code>va_arg</code>	Retrieve next argument in variable argument list.
<code>va_end</code>	Terminate variable argument processing.
<code>va_start</code>	Initialize for new variable argument processing.

## ■ Mouse, On Key, Session and Window Management

The following functions are all declared in the header file WMAPI.H.

### ■ Mouse Management

This group of functions return information about the status of the mouse pointer.

<code>HasMouse</code>	Test if console has a mouse device.
<code>wMouseDisable</code>	Disable trapping for a specified mouse event.
<code>wMouseEnable</code>	Enable trapping for a specified mouse event.
<code>wMouseHit</code>	Report on last mouse event.
<code>wMouseRead</code>	Report on last mouse event.
<code>wMouseState</code>	Report on mouse event-trapping status.

### ■ On Key Control

The following function controls On Key trapping for the console.

<code>wOnKey</code>	Disable/enable On Key trapping for specified keys.
---------------------	--

### ■ Session Control

These functions get the status of and control the multiple sessions available on the console.

<code>ClrBypass</code>	Disable Session Manager/multisessioning bypass.
<code>wGetActive</code>	Get the console's currently active session number.
<code>GetScreenSize</code>	Get the attached screen size of the console and test if multiple sizes are supported.
<code>IsActive</code>	Test if program is using the current active session.
<code>IsBypass</code>	Test if in Session Manager/multisessioning bypass mode.
<code>IsSession</code>	Test if console is multisession-capable.
<code>SetBypass</code>	Enable Session Manager/multisessioning bypass.
<code>SetScreenSize</code>	Change the attached screen size of the console.
<code>wGetSess</code>	Get session number of current program.
<code>wSwitch</code>	Enable/disable session-switching.
<code>wSwitchTo</code>	Change active session.

### ■ Window Management

<code>wChoice</code> , <code>wChoiceFuncKeys</code> , <code>wChoiceSelect</code> , <code>wChoiceTimeout</code>	Using a window, allow operator to select an item for a list of choices.
<code>wClear</code>	Clear a window to a specific fill character.

## 52 Functions by Category

---

wClip	Set the clipping mode for a window.
wClose, wCloseAll	Close an open window.
wColor	Set the colors for subsequent interior text in a window.
wCopy	Copy text from one window to another.
wCount	Determine how many windows are supported and in use.
wEdit	Allow operator to edit the contents of a window.
wEnter, wEnterFirst	Accept and edit a text string in a window.
wFinish	Close all windows.
wFrame	Define frame attributes for a window.
wGetStat	Get current status of a window.
wGetStr	Read text from a window.
wGetTitle	Read the title of a window.
wGetWin	Get the currently active window number.
wInvert	Define the monochrome invert mode for a window.
wLocate	Report on the current cursor position for a window.
wMenuBar	Allow operator to select one item from a horizontal menu bar of items.
wMove	Move the location of a window.
wOpen	Open a new window.
wOrder	Change the display order for windows.
wRefresh	Redisplay a window on the screen.
wRemove	Remove a window from the display.
wRepaint	Redisplay all windows.
wRestore	Read a saved window.
wSave	Save a window.
wSaveSize	Compute the memory size required to save a window.
wScroll	Define the scroll bar for a window.
wSelect	Make a window active.
wStatus	Get the status and attributes of a window.
wTake	Copy text from one window to another, removing it from the source window.
wTitle	Define the title for a window.
wVer	Return the Window Manager version, if any.



## 2 THEOS C Standard Library

---

The remainder of this manual describes the functions provided in the THEOS C standard library. The functions are listed, for the most part, in alphabetical sequence. The exceptions to this sequence occur when function descriptions are grouped because several functions are either related (see semaphore functions on page 550) or perform a similar operation (see `has`, `hascolor` on page 330).

Use the “Function Name Index” at the beginning of this manual to find a list of the functions in strict alphabetical order. The prior chapter, “Functions by Category,” lists the functions in logical groups. For instance, all directory and library maintenance functions are listed together, all trigonometric functions are listed together, *etc.*

### ■ Format of Function Descriptions

All of the descriptions of these standard library functions use the same format. Not all the sections are provided for all functions because some do not apply to the function.

- ▶ At the top of each function description is a brief statement of the purpose or operation of the function.
- ▶ The formal declaration of the function is shown in a box, near the top of the page. This declaration lists the standard header file required for using the function. If the function is defined in more than one header file, all of the header file names are listed.

The formal arguments to the function are printed using an *italic* font. This same font is used every time that an argument is referred to in the description of the function.

The formal arguments are listed immediately following the formal declaration with a brief description of their meaning or value.

- ▶ **Operation:** A description of what the function or functions do. The complete operation might not be described here if it can be better described under one of the following headings.
- ▶ **Returns:** Lists the value or values returned by the function.
- ▶ **Errors:** Any errors detected and reported by the function are listed here.
- ▶ **Notes:** Comments about the function, its operation and descriptions of argument values are provided here. In some cases, suggested uses for the function are given.
- ▶ **Restrictions:** Any conditions that preclude the use of the function are listed or described here.
- ▶ **Defaults:** Some functions that have default conditions or operating parameters. These functions have these defaults listed or described here.

- ▶ **Conforms to:** This area uses check boxes to indicate if the function declaration and operation conforms to some of the common C language standards.
  - ▶ **ANSI** When checked, the function conforms to the ANSI/ISO/IEC 9899:1990 standard for the C language library functions.
  - ▶ **DOS** When checked, the function conforms to the Microsoft C standard function library.
  - ▶ **THEOS** This box is always checked because, of course, the function conforms to THEOS.
  - ▶ **POSIX** When this box is checked it means that the function conforms to the same function in the POSIX.1 standard function library. Most UNIX systems today are POSIX compliant. Microsoft's Windows NT, OpenVMS and MVS are POSIX compliant.

Note that, in general, when a function conforms to a non-THEOS standard, it means that a program using the function that is written for that other standard can be imported, compiled and executed successfully.

- ▶ **See also:** When other functions perform similar or related operations, or operate on the same object (such as a specific type of file), those function names are listed here.
- ▶ **Example:** When it is possible to provide a meaningful example of the function and its use, an example program or program segment is shown here.

## a64l

Convert a number from base-64 ASCII to a long integer.

```
#include <stdlib.h>
```

```
long a64l ( char * string )
```

---

*string*                   »   pointer to string containing base-64 ASCII value

**Operation:**       The base-64 or radix-64 value represented in *string* is converted to its long integer, base-10 equivalent value.

Leading white space is ignored in *string*. Conversion stops when six characters have been evaluated or when a non-base-64 character is encountered.

**Returns:**         The long integer value of the base-64 *string* value.

**Notes:**          In a base-64 ASCII number, the characters used to represent the base-10 digits are:

<i>base-10 value</i>	<i>base-64 character</i>
0	.
1	/
2–11	“0”–“9”
12–37	“A”–“Z”
38–63	“a”–“z”

**Conforms to:**                   a64l    q ANSI    q DOS    n THEOS    q POSIX

**See also:**           [base64\\_decode](#), [base64\\_encode](#), [l64a](#)

### **abort**

Terminate execution of the current program and exit with return code.

```
#include <stdlib.h>
void abort ( void )
```

**Operation:** Performs an abnormal program exit of the program. A message is displayed on `stderr` and the return code is set to 254. This is equivalent to performing:

```
fprintf(stderr, "Abnormal program termination!\n");
exit(254);
```

**Returns:** No value is returned as control does not return to the program.

The program's return code is set to 254.

**Defaults:** When no functions are registered with the [atexit](#) function, the `abort` function performs an [fcloseall](#) followed by the program termination.

**Conforms to:** `abort`    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [atexit](#), [\\_atexit\\_clear](#), [\\_atexit\\_remove](#), [exit](#), [syserr](#)

---

#### **Example:**

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    if (argc < 2) {
        puts("\nRequired parameters missing...");
        abort();
    }
    ...
}
```

---

## abs

Compute the absolute or unsigned value of an integer.

```
#include <stdlib.h>
int abs ( int i )
_____
i                » integer data value
```

**Operation:** The absolute or unsigned value of the integer is computed and returned.

**Returns:** The absolute value of the integer *i*.

**Conforms to:** **abs** n ANSI n DOS n THEOS n POSIX

**See also:** [cabs](#), [fabs](#), [labs](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main(void) {
    int    i;

    // Test abs
    printf("\nEnter an integer: "); // get value from operator
    scanf("%d",&i);                // convert to numeric
    printf("The absolute value of %i is %i\n",i,abs(i));
}
```

---

### Output:

>example

```
Enter an integer: -4321
The absolute value of -4321 is 4321
```

>

**absolute value assembly functions**

These two “functions” generate in-line code to compute the absolute value of a long or short integer.

```
#include <builtin.h>
long _absd ( long lval )
short _absw ( int ival )
```

---

*ival*                   »   short integer value  
*lval*                   »   long integer value

**Operation:**        **\_absd**        Compute the absolute value of the long integer *lval*.  
                      **\_absw**        Compute the absolute value of the short integer *ival*.

**Returns:**           **\_absd**        The absolute value of the long integer *lval*.  
                      **\_absw**        The absolute value of the short integer *ival*.

**Notes:**            These “built-in” functions generate in-line code, thus avoiding the expensive usage of the call and ret instructions.

**Restrictions:**    These functions are intended for device-driver authors.

**Conforms to:**                **\_absd**    q ANSI    q DOS    n THEOS   q POSIX  
                              **\_absw**    q ANSI    q DOS    n THEOS   q POSIX

**See also:**           [abs](#)

**acc\_access, acc\_maint**

Enable or disable access and maintenance mode of the accounting file.

```
#include <acb.h>
```

```
void acc_access ( int on_off )
```

```
void acc_maint ( int on_off )
```

---

*on\_off*                      »    Boolean value to enable or disable access or maintenance mode

**Operation:**        When *on\_off* is a true or non-zero value, the [\\_lockset](#) function is used to lock a system resource that restricts other user's access to the accounting information. When *on\_off* is a false or zero value, the [\\_lockres](#) function is used to remove the lock and allow other user's access.

**Returns:**            No value is returned by these functions.

**Errors:**            No errors are detected or reported. If another user already has exclusive use of the accounting information, then these functions merely wait until they release their lock.

**Notes:**            **acc\_access** Use this function when you want to access but not modify the account information file.

**acc\_maint** Use this function when you want to access and possibly modify the account information file.

These functions should only be used to prevent another user from performing the ACCOUNT, LOGON or LOGOFF commands.

Be sure to release the locks when the program is finished using the account information.

<b>Conforms to:</b>	<b>acc_access</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>acc_maint</b>	q ANSI	q DOS	n THEOS	q POSIX

**accept**

Accept a connection to a socket.

```
#include <socket.h>
```

```
SOCKET accept ( SOCKET s, SOCKADDR * addr, int * addrlen )
```

---

```
addr           »   Pointer to buffer for socket address
```

```
addrlen       »   Pointer to length of socket address buffer
```

```
s             »   Socket number
```

**Operation:** This routine extracts the first connection on the queue of pending connections on *s*, creates a new socket with the same properties as *s* and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, **accept** blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept** returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open.

**Returns:** If no error occurs, **accept** returns a value of type **SOCKET** which is a descriptor for the accepted packet. Otherwise, a value of **INVALID\_SOCKET** is returned, and a specific error code may be retrieved by calling [TSAGetLastError](#).

The integer referred to by *addrlen* initially contains the amount of space pointed to by *addr*. On return it contains the actual length in bytes of the address returned.

**Errors:** When the return is **INVALID\_SOCKET** the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEFAULT	The <i>addrlen</i> argument is too small (less than the size of a struct <b>SOCKADDR</b> .)
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAEINVAL	listen was not invoked prior to <b>accept</b> .
TSAEMFILE	The queue is empty upon entry to <b>accept</b> and there are no descriptors available.
TSAENOBUFS	No buffer space is available.
TSAENOTSOCK	The descriptor is not a socket.
TSAEOPNOTSUPP	The referenced socket is not a type that supports connection-oriented service.
TSAEWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.



- Notes:** The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the address family in which the communication is occurring. The *addrlen* is a value-result parameter. It should initially contain the amount of space pointed to by *addr*. On return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK\_STREAM. If *addr* or *addrlen* are equal to NULL, then no information about the remote address of the accepted socket is returned.
- Restrictions:** The socket *s* must refer to a socket that is listening for a connection after a [listen](#).
- Conforms to:** **accept**    q ANSI    q DOS    n THEOS    q POSIX
- See also:** [bind](#), [connect](#), [listen](#), [select](#), [socket](#)

## 62 access

### access

Tests for a file's existence and whether or not your requested access will be allowed.

```
#include <stdio.h> or <io.h>
int access ( char * name, int mode)
```

---

<i>mode</i>	»	requested access type
<i>name</i>	»	name of file

**Operation:** First the file *name* is located on disk and its directory is read. Private or shared ownership of the file is determined so that the appropriate protect codes can be analyzed. The file's protection codes are compared with the requested access *mode*. The *mode* values may be specified with the names defined in `STDIO.H`:

<i>mode</i>	Meaning
F_OK	Check file's existence
X_OK	Execution access
W_OK	Write access
R_OK	Read access

It is permissible for access modes to be combined by adding their mode values. For instance, to check for read and write access use mode 6.

**Returns:** A zero is returned when the file exists and your program is allowed the requested access; otherwise EOF is returned.

<i>errno</i>	Meaning
EACCESS	The file was found but it cannot be accessed as requested.
ENOENT	The file or path could not be found.

**Notes:** If the name argument does not include the full path name or if it is a simple file name without a file type, the current values for the environment variables "LIBRARY" and "SUBDIR" are used in the file search.

**Conforms to:**                    **access**    q ANSI    n DOS    n THEOS    q POSIX  
(Access mode X\_OK does not conform to the DOS access function.)

**See also:**                    [fperror](#), [strerror](#)

**Example:**

The following simple program asks the operator for a file name. Using the `access` function the file is tested for existence and for write access. The `ferror` function displays the appropriate error message if the requested access is denied.

```
#include <stdio.h>

void
main(void)
{
    char          fn[80]; // file name storage

    printf("Enter file name: ");
    gets(fn);             // accept file name from operator

    if (access(fn, W_OK)) { // does file exist with access?
        ferror();           // display error message
    }
}
```

---

**Output:**

```
>access
Enter file name: garbage.name
File "garbage.name" not found.

>access
Enter file name: system.cmd32.cc32
File "system.cmd32.cc32" is protected.
```

**acos**

Compute the arccosine of a value.

```
#include <math.h>
double acos ( double x )
```

---

*x*                      »   floating-point angle, in radians

**Operation:**        Compute the arccosine of the value *x*.

**Returns:**         The arccosine of *x* is returned, expressed in radians, in the range  $0 - \pi$ .

**Errors:**          The value of *x* must be in the range of  $-1$  to  $+1$ . Values outside of this range cause `acos` to return a zero and to set [errno](#) to EDOM.

**Notes:**           The arccosine is the inverse cosine function. That is, `acos(cos(x)) == x`.

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**        `acos`    n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

## acot

Compute the arccotangent of a value.

```
#include <math.h>
double acot ( double x )
```

---

*x*                                      »   floating-point angle, in radians

**Operation:**        Compute the arccotangent of the value *x*.

**Returns:**        The arccotangent of *x* is returned, expressed in radians, in the range  $0 - \pi$ .

**Notes:**        The arccotangent is the inverse cotangent function. That is,

$$\text{acot}(\text{cot}(x)) == x.$$

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                      **acot**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [acos](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

**acsc**

Compute the arccosecant of a value.

```
#include <math.h>
double acsc ( double x )
_____
x                » floating-point angle, in radians
```

**Operation:** Compute the arccosecant of the value  $x$ .

**Returns:** The arccosecant of  $x$  is returned, expressed in radians, in the range 0 to  $\pi/2$  for  $x \geq 1$  or the range  $-\pi$  to  $-\pi/2$  for  $x < 1$ .

**Notes:** The arccosecant is the inverse cosecant function. That is,

$$\text{acsc}(\text{csc}(x)) == x$$

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:** `acsc` q ANSI q DOS n THEOS q POSIX

**See also:** [acos](#), [acot](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

## add memory assembly functions

This group of “functions” generates in-line code to add a byte, short integer or long integer to a specified memory area or to your program’s code segment.

```
#include <builtin.h> or <peek.h>
void _addb ( void _far * far_offset, char bval )
void _addl ( void _far * far_offset, long lval )
void _addw ( void _far * far_offset, int wval )
```

```
void _addcsb ( void * offset, char bval )
void _addcsd ( void * offset, long lval )
void _addcsw ( void * offset, int wval )
```

---

<i>bval</i>	»	byte value to add
<i>far_offset</i>	»	pointer to remote data object
<i>lval</i>	»	long integer value to add
<i>offset</i>	»	offset within code segment
<i>wval</i>	»	word value (short integer) to add

**Operation:**

**\_addb** Adds the value of *bval* to the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_addl** Adds the value of *lval* to the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_addw** Adds the value of *wval* to the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_addcsb, \_addcsd, \_addcsw** These functions perform the same operation as **\_addb**, **\_addl** and **\_addw** except that the memory segment is predefined to be your program’s code segment alias. These three functions can only be used in device drivers because other programs do not have an alias to their code segment and they will generate a general protection error.

**Returns:** No value is returned by these functions. The values are added directly to the values in the locations specified.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can add whatever type of object desired to any location. For instance, an **\_addw** function can add a word value (double-byte value) to a location that is declared as a character string. Of course, the program would have to be coded such that it could handle this situation.

# 68 add memory assembly functions

---

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program’s code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

<b>Conforms to:</b>	<b>_addb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_addd</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_addcsb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_addcsd</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_addcsw</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_addw</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)



## alarm

Programs a timer to generate a signal interrupt after a specified number of seconds have elapsed.

```
#include <signal.h> or <signal.h> or <timer.h>
unsigned short alarm ( unsigned seconds )
```

---

*seconds*                    »    number of seconds to elapse

**Operation:**        A timer is programmed to raise SIGALRM after *seconds* amount of time has elapsed.

**Returns:**            If there was an alarm or [msalarm](#) timer already programmed, the time remaining for that timer is returned. Otherwise, zero is returned.

**Notes:**             The default action for SIGALRM is SIG\_IGN. If you do not use the [signal](#) function to define your own trap for this event, your program will not detect when the *seconds* have elapsed.

Both alarm and [msalarm](#) use the same SIGALRM mechanism and they both share the same timer. The difference between the two functions is in the unit of measure for the time period specified.

An alarm timer can be cleared by using a *seconds* value of zero. When this is done, any previous alarm timer is reset.

**Defaults:**         The default action for SIGALRM is SIG\_IGN.

**Restrictions:**     Only one alarm timer is allowed at any one time. Multiple calls to alarm or [msalarm](#) reset the prior timer to the new value.

**Conforms to:**                [alarm](#)    q ANSI    n DOS    n THEOS    n POSIX

**See also:**            [msalarm](#), [signal](#), [timer](#)

---

### Example:

```
#include <stdio.h>
#include <signal.h>
#include <timer.h>

void trigger(void)
{
    char buf[9];

    printf("\rTime is %s", gettime(buf));
    signal(SIGALRM, trigger);
    alarm(60);
}

main() {
    ...
    trigger();           // start one-minute timer
    ...
}
```

### `_alloca`

Allocates memory from the program stack.

```
#include <malloc.h> or <builtin.h>
```

```
void * _alloca ( size_t len )
```

---

*len*                      »    size of desired memory allocation, in bytes

**Operation:**       Reserve and allocate memory from the program stack.

**Returns:**        A pointer to the lowest byte of the allocated space. If the space could not be allocated, a NULL pointer is returned.

**Errors:**        No errors are detected except for insufficient memory indicated by the return of a NULL pointer.

**Notes:**        The memory allocated by this function is resident in the program's data segment. It is automatically released when the function using it performs a return. Do not use the `free` function to release this memory.

Use this allocation method only for small pieces of memory. If the stack does not have sufficient space available to honor this request, a memory access error occurs.

**Conforms to:**       `_alloca`    q ANSI    n DOS    n THEOS    q POSIX

**See also:**        [calloc](#), [free](#), [\\_getmem](#), [\\_putmem](#), [malloc](#), [realloc](#)

## and memory assembly functions

This group of “functions” generate in-line code to perform an AND operation on bytes, short integers or long integers in another memory area or in your program’s code segment.

```
#include <builtin.h> or <peek.h>
void _andb ( void _far * far_offset, char bmask )
void _andd ( void _far * far_offset, long lmask )
void _andw ( void _far * far_offset, int wmask )
```

```
void _andcsb ( void * offset, char bmask )
void _andcsd ( void * offset, long lmask )
void _andcsw ( void * offset, int wmask )
```

---

<i>bmask</i>	»	byte mask to and with
<i>far_offset</i>	»	pointer to remote data object
<i>lmask</i>	»	long integer mask to and with
<i>offset</i>	»	pointer to data item in code segment
<i>wmask</i>	»	integer mask to and with

**Operation:** These functions perform a bitwise, Boolean AND operation. That is, a bit position in the result is set to one if, and only if, the same bit position in the two objects is also one.

**\_andb** ANDs the value of *bval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_andd** ANDs the value of *lval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_andw** ANDs the value of *wval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_andcsb, \_andcsd, \_andcsw** These functions perform the same operation as **\_andb**, **\_andd** and **\_andw** except that the memory segment is predefined to be your program’s code segment alias. These three functions can only be used in device drivers because other programs do not have an alias to their code segment and they will generate a general protection error.

**Returns:** No value is returned by these functions. The values are ANDed directly to the values in the locations specified.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can AND whatever type of object desired to any location. For instance, an **\_andw** function can AND a word value (double-byte value) to a location that is declared as a character string. Of course, the program would have to be coded such that it could handle this situation.

## 72 and memory assembly functions

---

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

<b>Conforms to:</b>	<b>_andb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_andd</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_andcsb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_andcsd</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_andcsw</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_andw</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [add memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

## asctime

Convert a time structure to a character string.

```
#include <time.h>
```

```
char * asctime ( const struct tm * timeptr )
```

---

*timeptr*                    »    pointer to time structure containing date/time to convert.

**Operation:**        The date and time indicated by the `tm` structure pointed to by *timeptr* is converted to a string using the [strftime](#) function. A pointer to this string is returned.

**Returns:**            A pointer to the converted string. This string is always 26 characters long, including the null terminator.

**Errors:**            No errors are detected by this function.

**Notes:**            This function performs a special call to [strftime](#) with a mask of

```
"%a %b %d %H:%M:%S %Y\n"
```

Thus, if the `tm` structure contains a date/time for 8 a.m. on 1/31/2000, the resulting string will be:

```
"Mon Jan 31 08:00:00 2000\n"
```

Because the time is converted by the `%H` directive, the time is in 24-hour format. All fields have a constant width: Single-digit dates output as two characters, a zero and the digit.

**Restrictions:**    This function assumes that *timeptr* points to a time value returned by the [localtime](#) or [gmtime](#) function. That is, it is a broken-down time structure.

The string pointed to by the return value is local to the `asctime` function and will be reused the next time that `asctime` is called. Either use the string immediately or make a copy of it.

**Conforms to:**            `asctime`    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [ctime](#), [difftime](#), [gmtime](#), [localtime](#), [mktime](#), [strftime](#), [time](#)

## 74 asctime

---

### Example:

```
#include <stdio.h>
#include <time.h>

main()
{
    time_t t;

    time(&t);          // get current date/time

    printf("\nIt is now %s",asctime(localtime(&t)));
}
```

---

### Output:

>example

It is now Thu Jul 04 15:04:37 1996

>

## asec

Compute the arcsecant of a value.

```
#include <math.h>
double asec ( double x )
_____
x                » floating-point angle, in radians
```

**Operation:** Compute the arcsecant of the value  $x$ .

**Returns:** The arcsecant of  $x$  is returned, expressed in radians, in the range 0 to  $\pi/2$  for  $x \geq 1$  or the range  $-\pi$  to  $-\pi/2$  for  $x < 1$ .

**Notes:** The arcsecant is the inverse secant function. That is,

$$\text{asec}(\sec(x)) == x.$$

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:** `asec` q ANSI q DOS n THEOS q POSIX

**See also:** [acos](#), [acot](#), [acsc](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

**asin**

Compute the arcsine of a value.

```
#include <math.h>
double asin ( double x )
_____
x                » floating-point angle, in radians
```

**Operation:** Compute the arcsine of the value  $x$ .

**Returns:** The arcsine of  $x$  is returned, expressed in radians, in the range  $-\pi/2$  to  $\pi/2$ .

**Errors:** The value of  $x$  must be in the range of  $-1$  to  $+1$ . Values outside of this range cause `asin` to return a zero and set [errno](#) to EDOM.

**Notes:** The arcsine is the inverse sine function. That is,

`asin(sin(x)) == x.`

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                      **asin**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                      [acos](#), [acot](#), [acsc](#), [asec](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)



## **\_asm**

Insert assembly-language code into the program.

```
_asm ( "string-literal" ... )
```

---

```
string-literal      »  assembly-language instruction
```

**Operation:** This “function” inserts the *string-literal* as an assembly-language instruction into the compiled program.

```
_asm( "pop eax" )
```

is equivalent to:

```
#asm
    pop eax
#endasm
```

When multiple string-literals are specified, each is inserted as a separate instruction.

```
_asm( "move eax,(test)", "incd (eax)" )
```

is equivalent to:

```
#asm
    move eax,(test)
    incd (eax)
#endasm
```

**Returns:** The return of this “function” is dependent upon the assembly-language instruction in *string-literal*. Anything placed in the EAX register is treated as the return value from a function. You should always put a cast on this “function” call.

**Notes:** This keyword is defined in this function library reference, not because it is a function, but because it looks like a function. It is an intrinsic operator built into the C language compiler.

**Restrictions:** The assembler code in *string-literal* must be valid code and appropriate for the operating ring of the program.

**Conforms to:**            **\_asm**    q ANSI    q DOS    n THEOS    q POSIX

## 78 *assert*

---

### **assert**

The `assert` function inserts a diagnostic test into your program.

```
#include <assert.h>
void assert ( int exp )

_____
exp           »   expression to test
```

**Operation:** The value of *exp* is evaluated and tested for a non-zero result. When the value is non-zero, it is interpreted as meaning true and program execution continues.

An *exp* that evaluates to zero causes a diagnostic message to be displayed on the `stderr` device. This message consists of the expression used, the source file name and the line number.

```
assert: expression
sample.source:s(23)
```

After the message is displayed the [abort](#) function is used to exit the program.

**Notes:** This function is implemented as a macro.

**Options:** The action of the assertion statements entered into a program can be suppressed easily, without actually removing them from the source. The compiler name `NDEBUG` is tested during the macro expansion of `assert`. If `NDEBUG` is defined, then the `assert` is ignored and not included in the compiled program.

This provides a simple method of keeping the `assert` calls in the program source without having them in the production form of the program.

**Conforms to:** `assert`    n ANSI    q DOS    n THEOS    q POSIX

**Example:**

```
#include <stdio.h>
#include <assert.h>

void
main(void) {
    int    a,
           b,
           i;

    a = 3;
    b = 2;

    printf("\nIn for loop:\n");
    for (i=0; i<10; i++) {
        printf("%d\n", i);
        assert( (a+b) != i);
    }
}
```

---

**Output:**

>example

In for loop:

0

1

2

3

4

5

assert: (a+b) != i

EXAMPLE.C:S(16)

Abnormal program termination!

**at, atstr**

Positions the text cursor on the console display.

```
#include <stdio.h>
void at ( int col, int row )
char * atstr ( int col, int row )
```

---

*col*                   »   column number to position to, base 0

*row*                   »   row number to position to, base 0

**Operation:**       Both of these functions generate the character string that, when output to the console, positions the text cursor to the desired *row* and *col*.

**at**                The cursor positioning string is created and sent to the console output device.

**atstr**            The cursor positioning string is created and a pointer to the string is returned. The console's text cursor is not moved.

**Returns:**       **at**               No value is returned by this function. The text cursor positioning string is sent directly to the console output device.

**atstr**            A pointer to the text cursor positioning string is returned. This string will be at least three bytes in length plus the null terminator. This buffer is static and the contents are replaced with each call to *atstr*.

**Errors:**        No errors are reported by these functions. If invalid *row* or *col* values are given to the functions, the text cursor positioning string is still generated but, when the string is sent to the console display, the erroneous positioning values cause the request to be ignored.

**Notes:**        The values for *row* and *col* are base 0 and are relative to the upper left corner of the currently active window when the string is sent to the console display. When windows are not in use, they are relative to the upper-left corner of the display.

**Conforms to:**       **at**    q ANSI    q DOS    n THEOS   q POSIX  
                  **atstr** q ANSI    q DOS    n THEOS   q POSIX

**See also:**        [putch](#)

**Example:**

```
#include <stdio.h>
#include <conio.h>
#include <crt.h>
#include <string.h>
#include <lub.h>

void main()
{
    char msgtext[100];
    char atbottom[10];

    crt(CLEAR);
    at(0,0);
    cputs("Sample program output...");

    strcpy(atbottom, atstr(0,getp1(CONSOLE)-1));
    strcat(strcpy(msgtext, atbottom), "Standard message text...");

    cputs(msgtext);

    pagewait();
}
```

---

**Output:**

>example

Sample program output...

Standard message text...

### atan, atan2

Compute the arctangent of a value or ratio.

```
#include <math.h>
double atan ( double x )
double atan2 ( double x, double y )
```

---

*x*                   »   floating-point value  
*y*                   »   floating-point divisor

**Operation:**        **atan**            Compute the arctangent of the value *x*.

**atan2**          Compute the arctangent of the ratio *x* / *y*.

**Returns:**           **atan**            The arctangent of *x* is returned, expressed in radians, in the range  $-\pi/2$  to  $\pi/2$ .

**atan2**          The arctangent of the ratio of *x*/*y* is returned, expressed in radians, in the range  $-\pi$  to  $\pi$ .

**Errors:**            No errors are detected by atan. The atan2 function will set [errno](#) to EDOM when either *x* or *y* is zero.

**Notes:**            The arctangent is the inverse tangent function. That is,

`atan(tan(x)) == x.`

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**               **atan**    n ANSI    n DOS    n THEOS    n POSIX

**atan2**   n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

## **atexit, \_atexit\_clear, \_atexit\_remove**

These functions maintain the list of routines that are executed when the program exits.

```
#include <stdlib.h>
int atexit ( void (*function)(void) )
void _atexit_clear ( int codeseg )
void _atexit_remove ( void (*function)(void) )
```

---

*function*               »   name of function  
*codeseg*                »   memory code segment of function

**Operation:**       **atexit**       The location of *function* is added to a list of atexit routines. A maximum of 32 routines can be registered with this procedure. Duplicate calls to atexit with the same function name do not cause *function* to be added to the list twice.

**\_atexit\_clear**   The list of functions in the atexit list is cleared.

**\_atexit\_remove** The saved location of *function* in the atexit routines list is removed from the list.

**Returns:**       **atexit**       A zero is returned when *function* is successfully added to the list; otherwise, a non-zero value is returned.

**\_atexit\_clear**   No value is returned.

**\_atexit\_remove** No value is returned.

**Errors:**       No errors are reported by these functions.

**Notes:**       The function names registered with the atexit function are executed in a LIFO manner (last-in, first-out).

**Defaults:**    When no functions are registered with the atexit function, the abort and exit functions will perform only a [fcloseall](#) followed by program termination.

**Restrictions:** The function used with the atexit function must be a function that receives no arguments and returns no values.

## 84 *atexit, \_atexit\_clear, \_atexit\_remove*

---

Conforms to:	<b>atexit</b>	n ANSI	n DOS	n THEOS	q POSIX
	<b>_atexit_clear</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_atexit_remove</b>	q ANSI	q DOS	n THEOS	q POSIX

See also: [abort](#), [exit](#)

---

### Example:

```
#include <stdlib.h>
#include <stdio.h>

void
clear_screen(void) {           // an atexit function that clears
                               // the screen.
    putchar('\f');
}

int
main(int argc, char *argv[]) {
    if (argc < 2 ) {           // No cmd line arguments?
        puts("Required parameter missing.");
        abort();               // Exit with error message
    }

    atexit(clear_screen);      // on exit clear the screen
    ...
}                               // implies an exit(0)
```

---

### Output:

```
>example
Required parameter missing.
Abnormal program termination!

>
```



## atof, atoi, atol, atoll

## 86 *atof, atoi, atol, atoll*

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main(void) {
    char    *s;
    double  x;
    int     i;
    long    l;

    s = "  -1234.56E-8";           // Test for atof
    x = atof(s);
    printf("atof test: ASCII string: %s\tfloat:  %e\n",s,x);
    s = "  1234Apples";           // Test for atoi
    i = atoi(s);
    printf("atoi test: ASCII string: %s\tint:    %i\n",s,i);
    s = "  987654.32Lira";        // Test for atol
    l = atol(s);
    printf("atol test: ASCII string: %s\tlong:   %ld\n",s,l);
}
```

---

### Output:

```
>example
atof test: ASCII string:  -1234.56E-8  float:  -1.234560e-05
atoi test: ASCII string:  1234Apples  int:    1234
atol test: ASCII string:  987654.32Lira long:    987654
```

## **\_attach**

Attach a device-driver to a logical device.

```
#include <stdlib.h>
```

```
int _attach ( int lub, char * device, char ** options )
```

---

<i>device</i>	»	pointer to string specifying physical device name
---------------	---	---

<i>lub</i>	»	logical unit block number to attach
------------	---	-------------------------------------

<i>options</i>	»	pointer to array of pointers to option specifications, terminated by a null pointer
----------------	---	---

**Operation:** If *device* is already attached to *lub*, the *options* are used to reattach the device. When a different device is currently attached to *lub*, it is first detached using [\\_detach](#).

The SYSTEM.TEOS $nnn$ .DEVNAMES file is searched for an entry matching *device*. If found, the specified device-driver is loaded and attached to *lub*. The options specified by *options* are defined in the unit control block for the device and the device's initialization entry point is called to initialize the device with the requested options.

**Returns:** A success/fail indicator. A return of zero indicates success; a non-zero return indicates failure.

**Errors:** When an error is detected, the device is not attached. Although [errno](#) is not set nor is [\\_errnum](#), the return value is the error code.

**Notes:** This function is essentially the ATTACH command (mode 1) implemented as a function. All of the capabilities and option testing of this mode of the command are implemented in this function.

**Conforms to:**            [\\_attach](#)    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [\\_detach](#)

---

### **Example:**

```
#include <stdio.h>
```

```
#include <lub.h>
```

```
int
```

```
get_con(char * dev)
```

```
{
```

```
    char *option[4];
```

```
    option[0] = "B57600";           // baud rate 57,600
```

```
    option[1] = "E2";               // XON/XOFF flow control
```

```
    option[2] = "W8";               // 8 bit words
```

```
    option[3] = NULL;               // end of list
```

```
    return _attach(com1, dev, option);
```

```
}
```

### **base64\_decode, base64\_encode**

Translate MIME base64 data to or from binary data.

```
#include <stdlib.h>

int base64_decode ( int c, char * buffer, int len )
int base64_encode ( int c, char * buffer, int len )
```

---

<i>buffer</i>	»	pointer to storage location
<i>c</i>	»	byte value to translate
<i>len</i>	»	current length of data in <i>buffer</i>

**Operation:**      **base64\_decode**   Translate the MIME base64 byte *c* to binary and store the translated value in *buffer[*len*]*.

**base64\_encode**   Translate the binary byte *c* to MIME base64 and store the translated value in *buffer[*len*]*.

**Returns:**            Both functions return the new length or count of translated bytes in *buffer*.

**Notes:**            *MIME* is the acronym for Multipurpose Internet Mail Extensions and is a data encoding method used by many systems to transfer data to other computer systems, either via the Internet, fax or electronic mail programs. MIME encoding is not compatible with the encoding used by the functions [a64i](#) or [l64a](#).

After 76 MIME-encoded bytes are generated, the **base64\_encode** function adds a new-line character and a binary zero (0x0b, 0x00). These two codes are described as a “soft return” and are not counted in the length. Generally, when **base64\_encode** returns with a length of 76, it is best to write the encoded data to the output file. This is consistent with the MIME encoding standard RFC #2045.

**Restrictions:**    **base64\_decode**   You must terminate the decoding of the data by using the **base64\_decode** with an equal sign (=) character.

**base64\_encode**   You must terminate the encoding of a “record” by using **base64\_encode** with a byte value of -1.

<b>Conforms to:</b>	<b>base64_decode</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>base64_encode</b>	q ANSI	q DOS	n THEOS	q POSIX

**Example:**

To encode from binary to base64:

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE * infile,
        * outfile;
    char buffer[80];
    int ch,
        len;

    infile = fopen("data.file", "r");
    outfile = fopen("mime.data", "w");

    len = 0;
    while ((ch = fgetc(infile)) != EOF) { // for each character
        len = base64_encode(ch, buffer, len); // encode to base64
        if (len >= 76) { // buffer line full?
            len = base64_encode(-1, buffer, len);
            fputs(buffer, outfile); // write to MIME file
            len = 0; // reset buffer
        }
    }
    if (len) { // buffer has data?
        len = base64_encode(-1, buffer, len);
        fputs(buffer, outfile);
    }
    fcloseall();
}
```

\_\_\_\_\_

Basic2C, BasicL2C

These functions convert a BASIC-language style string to a C-language style string.

```
#include <string.h>
char * Basic2C ( char * cstr, const void * bstr )
char * BasicL2C ( char * cstr, const void * bstr )
```

---

*bstr* » pointer to the BASIC-language string  
*cstr* » pointer to C-language string storage

**Operation:** A C-language style string is a vector array of characters terminated by a binary zero. A BASIC-language style string is a vector array of characters whose first element(s) is a value that specifies the number of characters following that make up the string.

C style	<table><tr><td>E</td><td>x</td><td>a</td><td>m</td><td>p</td><td>l</td><td>e</td><td>0x0</td></tr></table>							E	x	a	m	p	l	e	0x0
E	x	a	m	p	l	e	0x0								
BASIC style	<table><tr><td>0x7</td><td>E</td><td>x</td><td>a</td><td>m</td><td>p</td><td>l</td><td>e</td></tr></table>							0x7	E	x	a	m	p	l	e
0x7	E	x	a	m	p	l	e								
BASIC long style	<table><tr><td>0x7</td><td>0x0</td><td>0x0</td><td>0x0</td><td>E</td><td>x</td><td>a</td><td>m</td><td>p</td><td>l</td><td>e</td></tr></table>	0x7	0x0	0x0	0x0	E	x	a	m	p	l	e			
0x7	0x0	0x0	0x0	E	x	a	m	p	l	e					

- Basic2C** The BASIC-language style short string pointed to by *bstr* is converted to a C-language style string and copied to the location pointed to by *cstr*.
- BasicL2C** The BASIC-language style long string pointed to by *bstr* is converted to a C-language style string and copied to the location pointed to by *cstr*.

**Returns:** Both functions return the value *cstr*, which is a pointer to the resulting string.

**Restrictions:** *cstr* must point to a buffer allocated sufficiently large enough to contain the resulting string plus the NULL terminator.

Conforms to:	<b>Basic2C</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>BasicL2C</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [C2Basic](#), [C2BasicL](#)

*\_baud2cd, \_cd2baud*

Converts a baud rate value to a character code.

#include <stdlib.h>  
char *\_baud2cd* ( long *baud*)  
long *\_cd2baud* ( char *code* )  
  
-----  
*baud*                   »   baud rate value  
*code*                   »   baud rate code

**Operation:**       *\_baud2cd*   The *baud* rate is encoded to a single character, suitable for usage in a byte i/o device unit control block (ucb).

*\_cd2baud*   The *code* is decoded to the baud rate value.

**Returns:**        The character code representing *baud* or *code*.

**Errors:**         Invalid *baud* rates or *code* cause a NULL to be returned.

**Notes:**         Valid values for *baud* and *code* include:

Baud	Code	Baud	Code
0	0	4800	12
110	3	7200	13
300	6	9600	14
600	7	19200	15
1200	8	38400	2
1800	9	57600	1
2400	10	76800	4
3600	11	115200	5

Table 2: Baud Rate Codes

These functions are normally used by a device-driver’s initialization routine.

**Conforms to:**       *\_baud2cd*   q ANSI     q DOS     n THEOS   q POSIX  
                  *\_cd2baud*   q ANSI     q DOS     n THEOS   q POSIX



**bcd2ieee, ieee2bcd**

Convert a BCD-encoded floating-point value to an IEEE-encoded floating-point value.

```
#include <stdlib.h>
void bcd2ieee ( void * float )
void ieee2bcd ( void * float )
```

---

*float*                   »   pointer to floating-point value to convert

**Operation:**       **bcd2ieee**   The BCD value pointed to by float is converted to its equivalent IEEE value.

**ieee2bcd**   The IEEE value pointed to by float is converted to its equivalent BCD value.

**Returns:**           No value is returned. The data value is converted “in-place.”

**Notes:**            These functions are normally used to convert data that is read from or written to a file. BCD format is more portable than IEEE format because it does not require the presence of a IEEE-compliant floating-point processor on the computer system.

**Restrictions:**    These functions require the presence of a floating-point processor.

<b>Conforms to:</b>	<b>bcd2ieee</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>ieee2bcd</b>	q ANSI	q DOS	n THEOS	q POSIX

**Example:**

```
#include <stdlib.h>
#include <stdio.h>

void
main()
{
    double  value;
    FILE    * datafile;

    if (!(datafile = fopen("basic.data", "dr+l")))
        exit(fperror());

    lreadk(datafile, 1L, &value);           // read 1st record;

    bcd2ieee(&value);                       // convert to ieee

    ...                                     // process data

    ieee2bcd(&value);                       // convert back to bcd

    lwritek(datafile, 1L, &value);          // write it back

    fclose(datafile);
}
```

**bcmp, bcopy, bzero**

These macro functions are provided to complete the socket function declarations in `SOCKET.H`.

```
#include <socket.h>
int bcmp ( const void buffer1, const void buffer2, size_t count )
void * bcopy ( const void * source, void * destination, size_t count )
void * bzero ( void * destination, size_t count )
```

<i>buffer<sub>1</sub></i>	»	Pointer to buffer to compare
<i>buffer<sub>2</sub></i>	»	Pointer to buffer to compare
<i>count</i>	»	Length of buffer to compare, copy or zero
<i>destination</i>	»	Pointer to destination buffer
<i>source</i>	»	Pointer to source buffer

- Operation:**
- bcmp**      Synonym to the [memcmp](#) function. The two buffers are compared for a length of *count* bytes.
  - bcopy**      Synonym to the [memcpy](#) function except that the sequence of the *source* and *destination* arguments are reversed. *count* number of bytes are copied from the *source* buffer to the *destination* buffer.
  - bzero**      Synonym to the [memset](#) function with a constant set-value of zero being supplied. The contents of *destination* are set to binary zeros for a length of *count* bytes.
- Returns:**
- bcmp**      A signed, integer value.

Comparison	Return value
<i>buffer<sub>1</sub></i> < <i>buffer<sub>2</sub></i>	< 0
<i>buffer<sub>1</sub></i> = <i>buffer<sub>2</sub></i>	0
<i>buffer<sub>1</sub></i> > <i>buffer<sub>2</sub></i>	> 0

- bcopy**      A pointer to *destination*.
- bzero**      A pointer to *destination*.

**Restrictions:**

- bcopy**      If *source* and *destination* overlap, the resulting *destination* may be erroneous. When *source* is greater than *destination*, the copy is performed correctly; when *destination* is greater than *source*, the process of copying the bytes in *source* may destroy subsequent bytes in *source*.

<b>Conforms to:</b>	<b>bcmp</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>bcopy</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>bzero</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**      [memory functions](#)

**Example:**      See the example for the function [socket](#).

## bind

Associate a local address with a socket.

```
#include <socket.h>
int bind ( SOCKET s, const SOCKADDR * addr, int addrlen )
```

---

<i>addr</i>	»	Pointer to buffer for socket address
<i>addrlen</i>	»	Pointer to length of socket address buffer
<i>s</i>	»	Socket number

**Operation:** This routine is used on an unconnected stream socket, before subsequent [connect](#) or [listen](#). When a socket is created with [socket](#), it exists in a name space (address family), but it has no name assigned. `bind` establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket.

**Returns:** If no error occurs, `bind` returns 0. Otherwise, it returns `SOCKET_ERROR`, and a specific error code may be retrieved by calling [TSAGetLastError](#).

**Errors:** When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEADDRINUSE	The specified address is already in use. See the <code>SO_REUSEADDR</code> socket option under <a href="#">setsockopt</a> .
TSAEFAULT	The <i>addrlen</i> argument is too small (less than the size of a struct <code>SOCKADDR</code> .)
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAEAFNOSUPPORT	The specified address family is not supported by this protocol.
TSAEINVAL	The socket is already bound to an address.
TSAENOBUFS	Not enough buffers available or too many connections.
TSAENOTSOCK	The descriptor is not a socket.

**Notes:** In the Internet address family, a name consists of several components. For `SOCK_DGRAM` and `SOCK_STREAM`, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to `INADDR_ANY`, a port equal to 0, or both. If the Internet address is equal to `INADDR_ANY`, any appropriate network interface will be used; this simplifies application programming in the presence of multi-homed hosts. If the port is specified as 0, the THEOS Sockets implementation will assign a unique port to the application with a value between 1024

and 5000. The application may use [getsockname](#) after `bind` to learn the address that has been assigned to it.

**Notes:** [getsockname](#) will not necessarily fill in the Internet address until the socket is connected because several Internet addresses may be valid if the host is multi-homed.

**Conforms to:** `bind` q ANSI q DOS n THEOS q POSIX

**See also:** [connect](#), [getsockname](#), [listen](#), [setsockopt](#), [socket](#), [TSAGetLastError](#)

**Example:** See the example for the function [socket](#).

## binsert

Locate the position in a sorted array where a new item should be added.

```
#include <search.h>
```

```
int binsert ( const void * key, const void * array, size_t nmemb, size_t size,
              int (*compar)())
```

---

<i>array</i>	»	pointer to area of array to begin search
--------------	---	--

<i>compar</i>	»	name of comparison routine
---------------	---	----------------------------

<i>key</i>	»	pointer to key to add
------------	---	-----------------------

<i>nmemb</i>	»	number of members remaining in array
--------------	---	--------------------------------------

<i>size</i>	»	size of each member in array
-------------	---	------------------------------

**Operation:** A binary search is performed on *array* looking for a member that matches *key*. This routine makes no presumptions about the content of the members of *array*. The *compar* routine must know how to compare two elements of the array.

If *key* is not found in *array*, *binsert* determines the position that the *key* should be in if it had existed in the *array*.

**Returns:** The index of *array* where *key* does exist or should be inserted. Use either the [bsearch](#) function prior to calling *binsert* or use the *compar* function to determine if *key* already exists in *array*.

**Notes:** Refer to the [bsearch](#) function notes for information about the *compar* routine.

The [bsearch](#) routine is normally used prior to calling this function to determine whether the *key* already exists in *array*.

**Restrictions:** The contents of *array* must be in sorted, ascending order, as defined by the comparison function *compar*. When it isn't, the operation and return value of this function are meaningless.

**Conforms to:** **binsert** q ANSI q DOS n THEOS q POSIX

**See also:** [bsearch](#), [lsearch](#), [lfind](#), [qsort](#)

---

**Example:** See [bsearch](#) example.

**bsearch**

Search a sorted array for an item.

```
#include <stdlib.h> or <search.h>
void * bsearch ( const void * key, const void * array, size_t nmemb, size_t size,
  int (*compar)() )
```

---

<i>array</i>	»	pointer to area of array to begin search
<i>compar</i>	»	name of comparison routine to use
<i>key</i>	»	pointer to key to find
<i>nmemb</i>	»	number of members remaining in array
<i>size</i>	»	size of each member in array

**Operation:** A binary search is performed on *array* looking for a member that matches *key*. This routine makes no presumptions about the content of the members of *array*. The *compar* routine must know how to compare two elements of the array.

**Returns:** A pointer to the *array* member matching *key*. If two or more members match *key*, the first one encountered by the search algorithm is the one returned.

A null pointer return value indicates no member was found matching *key*.

**Notes:** The *compar* routine must be a function that uses a declaration similar to:

```
int compar(const void *memb1, const void *memb2)
```

*memb1* and *memb2* are two pointers to elements of *array*. *compar* must return an integer that is:

```
< 0 when memb1 < memb2
= 0 when memb1 = memb2
> 0 when memb1 > memb2
```

**Restrictions:** The contents of *array* must be in sorted, ascending order, as defined by the comparison function *compar*. When it isn't, the operation and return value of this function are meaningless.

**Conforms to:**                **bsearch**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                [binsert](#), [lsearch](#), [lfind](#), [qsort](#)

**Example:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data {           // array for books data
    char  title[20],
          author[20];
} data[10];

int
keycomp(struct data *a, char *b) {
    return memcmp(a->title, b, strlen(b));
}

void
main() {
    FILE  *books;           // book catalogue file
    int    nmemb = 0,
           index,
           i;
    char   key[20],
           answer[20];
    struct data *item;

    if (!(books = fopen("my.books", "r"))) // open catalogue
        exit(f perror());
    for (nmemb=0; nmemb<10 &&
         fgets((char *)&data[nmemb], sizeof(struct data), books);
         ++nmemb) {
        data[nmemb].title[19] = '\0';
        data[nmemb].author[19] = '\0';
        trim(data[nmemb].title);
        trim(data[nmemb].author);
    }
    fclose(books);           // close the file

    qsort(data, nmemb, sizeof data[0], strcmp); // sort it

    printf("Enter title: ");
    gets(key);               // accept title

    item = bsearch(key, data, nmemb, sizeof data[0], keycomp);

    if (!item) {
        puts("Title not found in catalogue.\n");
        puts("Do you wish to add it? ");
        gets(answer);
        upcase(trim(answer));
        if (answer[0]=='Y') {           // insert new item?
            printf("\n\tNew author: ");
            gets(answer);
            index = binsert(key, data, nmemb, sizeof data[0],
                             keycomp);
            for ( i=nmemb; i>=index; i--) { // shift array
                memcpy(&data[i+1], &data[i],
                    sizeof data[0]);
            }
        }
    }
}

```

## 100 bsearch

---

```
        }
        strcpy(data[index].title, key);
        strcpy(data[index].author, answer);
    }
}

else {
    printf("The complete title is: \"%s\".\n",item->title);
    printf("                author: %s.\n",item->author);
}
}
```

---

### Output:

>example

```
Enter title: Of Mice
The complete title is: Of Mice and Men
                author: John Steinbeck
```

>example

```
Enter title: Grapes of Wrath, The
Title not found in catalogue.
Do you wish to add it? Y
    New author: John Steinbeck
```



## C2Basic, C2BasicL

These functions convert a C-language style string to a BASIC-language style string.

```
#include <string.h>
```

```
void * C2Basic ( char * bstr, const void * cstr )
```

```
void * C2BasicL ( char * bstr, const void * cstr )
```

---

*bstr* » pointer to the BASIC-language string

*cstr* » pointer to C-language string storage

**Operation:** A C-language style string is a vector array of characters terminated by a binary zero. A BASIC-language style string is a vector array of characters whose first element(s) is a value that specifies the number of characters following that make up the string.

C style

E	x	a	m	p	l	e	0x0
---	---	---	---	---	---	---	-----

BASIC style

0x7	E	x	a	m	p	l	e
-----	---	---	---	---	---	---	---

BASIC long style

0x7	0x0	0x0	0x0	E	x	a	m	p	l	e
-----	-----	-----	-----	---	---	---	---	---	---	---

**C2Basic** The C-language style string pointed to by *cstr* is converted to a BASIC-language style short string and copied to the location pointed to by *bstr*.

**C2BasicL** The C-language style string pointed to by *cstr* is converted to a BASIC-language style long string and copied to the location pointed to by *bstr*.

**Returns:** Both functions return the value *bstr*, which is a pointer to the resulting string.

**Restrictions:** *bstr* must point to a buffer allocated sufficiently large enough to contain the resulting string plus the leading length codes.

**Conforms to:**

<b>C2Basic</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>C2BasicL</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [Basic2C](#), [BasicL2C](#)

**c3tol**

Convert a three-byte integer to a long integer.

```
#include <stdlib.h>
long c3tol ( const void * c )
_____
c                »   pointer to three-byte integer
```

**Operation:** Convert the three-byte integer value *c* to a normal long integer value.

**Returns:** The long integer value.

**Notes:** The c3tol function is the complementary function to [ltoc3](#).

There are no standard functions that can manipulate or use the three-byte integer value. This function must be used to convert a three-byte integer back to a long integer for usage. The sole purpose of the three-byte integer format is more compact external storage of integer values that can fit in three bytes.

**Conforms to:** **c3tol**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [lltoa](#), [ltoa](#), [ltoc3](#), [ltoi2](#), [ltoi3](#)

## cabs

Compute the absolute or unsigned value of a complex value.

```
#include <math.h>
double cabs ( struct complex z )
```

---

*z*                      »    complex data value

**Operation:**        The absolute or unsigned value of the complex argument *z* is computed and returned.

**Returns:**            The absolute value of the complex argument *z* is returned as a double.

**Notes:**            The definition of a complex data type is:

```
struct complex {
    double x;
    double y;
};
```

A call to cabs is equivalent to:

```
sqrt(z.x * z.x + z.y * z.y)
```

**Conforms to:**                **cabs**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [abs](#), [fabs](#), [labs](#)

### **calloc**

Allocate and initialize memory for an array of objects.

```
#include <stdlib.h>
void * calloc ( size_t nmemb, size_t memb_size )
```

---

<i>memb_size</i>	»	size of each unit to allocate
<i>nmemb</i>	»	number of units to allocate

**Operation:** Allocates and initializes memory for an array of same-sized object. The allocated memory is initialized to zeros. The size of the allocated memory is the computed value  $nmemb \times memb\_size$ .

**Returns:** A pointer to the lowest byte of the allocated memory. If the space could not be allocated, a NULL pointer is returned.

**Errors:** No errors are detected by these functions except for insufficient memory, which is indicated by the return of a NULL pointer.

**Notes:** The memory allocated is guaranteed to be suitably aligned for storage of any type of object.

Always test the return value from the allocation routines to make sure that memory was actually allocated. If the return is not tested and a null pointer is returned, using that null pointer will cause a memory access error.

Use the [free](#) function to release memory allocated with this function or the [malloc](#) or the [realloc](#) functions to reuse the memory.

**Conforms to:** **calloc**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [\\_alloca](#), [free](#), [\\_getmem](#), [\\_putmem](#), [max\\_alloc](#), [malloc](#), [mem\\_avail](#), [realloc](#), [tot\\_alloc](#)

---

#### **Example:**

```
#include <stdlib.h>

void
main(void) {
    double * data;

    if (!(data=calloc(200, sizeof *data))) {
        errmsg(3,NULL);           // insufficient memory
        exit(3);
    }

    ...

    free(data);                   // release memory
}
```

## ceil

Compute the smallest whole number that is greater than or equal to a value.

```
#include <math.h>
double ceil ( double x )

_____
x                » floating-point value
```

**Operation:** The ceiling of  $x$  is computed and returned.

**Notes:** The ceiling of a value is the smallest whole number that is larger than or equal to the value.

For instance, the `ceil(3.5) == 4.0`; the `ceil(-5.6) == -5.0`.

**Returns:** The ceiling of  $x$ .

**Conforms to:** `ceil` n ANSI n DOS n THEOS n POSIX

**See also:** [floor](#), [fmod](#)

**cgets**

Accept a character string from the console.

```
#include <conio.h>
```

```
char * cgets ( char _far * buffer, int len )
```

---

*buffer*                   »   pointer to string storage

*len*                       »   one greater than the maximum number of characters to accept

**Operation:**       The [getch](#) function is used to accept a string of characters from the console input device.

Input is terminated when: *len* minus one characters are accepted, a carriage return is entered or the first character is a control character and the [conmask](#) for control character termination is set.

Certain control keys cause special actions to occur. The backspace key ( [Backspace](#) ) removes the prior character from *buffer* and the display. The line cancel key ( [Ctrl](#)+[X](#) ) acts as multiple backspace characters by erasing all of the characters from *buffer* and removing them from the display.

The tab character ( [Tab](#) ) is saved as a tab in *buffer* but is displayed as multiple spaces according to the tab set block in the SCR.

All other control characters that are entered are ignored and are not saved in *buffer*.

**Returns:**       A pointer to *buffer* is returned.

**Errors:**       No error is detected.

**Notes:**       This function is not the same as the DOS *cgets* function.

This function operates on the console input or output devices, not the *stdin* or *stdout* devices. Therefore redirection of *stdin* and *stdout* does not apply to these functions.

In addition, the console input and output controls provided by the THEOS operating system apply to these functions. Thus the input case mode and echo controlled by the [conmask](#) attributes and the exec stack may provide the input stream, if present. The console echo may be suppressed by the status of the [conmask](#) suppression bit in the SCR.

If there is a console echo file enabled with the system utility ECHO, all characters displayed or echoed by this function will also be output to the console echo file. Note that output to the console echo file may occur even though actual output to the console is suppressed.

All output to the console output device will be processed by the console's class code for screen attributes, cursor controls and screen editing functions.

**Conforms to:**                   **cgets**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**           [gets](#), [conmask](#)

**Example:**

```
#include <stdio.h>
#include <crt.h>           // screen control char defs
#include <conio.h>

void
main(void) {
    char  name[35],
          fi, mi, li;

    conmask("em");
    crtcolor(7,1,7,4);    // white on blue, white on red
    putch(CLEAR);        // clear the screen

    at(5,3); cputs("Enter your name: ");
    cgets(name, 35);      // accept name, max of 34 chars

    at(5,5); cputs("What are your initials: ");
    fi = getch();         // get first initial
    mi = getch();         // get middle initial
    li = getch();         // get last initial

    at(5,7); cputs("Thank you ");
    cputs(name);

    at(1,8);
}
```

## chdir

Change the current working directory.

```
#include <stdio.h> or <direct.h>
```

```
int chdir ( char * dirname )
```

---

*dirname*                   »   pointer to new working directory

**Operation:**       The current working directory is changed to the directory specified in *dirname*.

**Returns:**         A zero is returned to indicate a successful change to the cwd. A non-zero value indicates that the directory specified in *dirname* could not be found.

**Errors:**          When the directory specified in *dirname* can not be found, [errno](#) is set to ENOENT.

**Defaults:**       When *path* is only a partial path name to a directory, the current working directory is prepended to the path specified. For instance:

current cwd	<i>dirname</i>	new cwd
/aaa/bbb:s	ccc	/aaa/bbb/ccc:s
/aaa/bbb:s	ccc/ddd	/aaa/bbb/ccc/ddd:s
/aaa/bbb:s	/eee:c	/eee:c
/aaa/bbb:s	ccc:f	/ccc:f

**Restrictions:**   The directory specified in *dirname* must be owned by the current account or the public system account.

**Conforms to:**       **chdir**   q ANSI    n DOS    n THEOS   n POSIX

**See also:**         [getcwd](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Directory specification required!.\n");
        exit(16);
    }
    if (chdir(argv[1])!=0) {
        perror(argv[1]);
        exit(16);
    }
    // directory changed to directory specified on cmd line remainder
    // of program can use this as the default path to files.
    ...
}
```



## clearerr

Reset a file's error status.

```
#include <stdio.h>
void clearerr ( FILE * file )
```

---

*file*                   »   pointer to open file control block

**Operation:**       Clears the error status indicators for an open file. Specifically, the `_errno` and `_errarg` values are reset and the disk full and end-of-file flags are reset.

**Returns:**         No value is returned.

**Notes:**           Error indicators are not automatically reset. When an error occurs on a specific *file*, further operations on *file* will continue to report an error unless the error indicators are cleared with this function.

The `_errno` indicator is also cleared by the `ferror` function.

Although this function clears the error status indicators for a file, it does not change the cause of any error related to the file. For instance, if the file was at end-of-file, the file pointer will still be at the end of the file after this function executes.

**Conforms to:**       **clearerr**   n ANSI    n DOS    n THEOS   n POSIX

**See also:**         [eof](#), [feof](#), [ferror](#), [perror](#)

---

### Example:

```
#include <stdio.h>

void
main()
{
    FILE *outfile;
    char c;

    c = 'X';
    outfile = fopen("my.file", "w");
    if (outfile != NULL)
    {
        fputc(c, outfile);
        if (ferror(outfile))                 // error on write?
        {
            printf("Failure on write\n");
            clearerr(outfile);                // clear error status
            fputc(c, outfile);                // retry once
        }
        fclose(outfile);
    }
}
```

### clock

Compute the processor time used by the program.

```
#include <time.h>
clock_t clock ( void )
```

- Operation:** The amount of cpu processor time used by the current program is computed.
- Returns:** The number of “clocks” or ticks used during this program’s execution.
- Notes:** The processor time returned can be converted to seconds by dividing the return value by the constant `CLOCKS_PER_SEC`, which is defined in the `TIME.H` header file.
- Restrictions:** Only the amount of processor time used is returned, not true elapsed time. If execution of the program is delayed due to your program waiting for an interrupt to occur (most input and output operations are interrupt driven) or it must delay while other users/processes are receiving time, the processor time does not increase.
- Conforms to:** `clock`    n ANSI    n DOS    n THEOS    n POSIX
- See also:** [difftime](#), [getmsec](#), [gettime](#), [time](#)
- 

#### Example:

```
#include <stdio.h>
#include <time.h>

main()
{
    clock_t  start_time,
            end_time;

    start_time = clock();
    callcsi("show who");
    end_time = clock();

    printf("\nCPU time to perform SHOW WHO was %ld ticks
           or %g seconds",end_time-start_time,
           (end_time-start_time)/CLOCKS_PER_SEC);
}
```

#### Output:

```
>example
ACCOUNT  = SAMPLES
USERNUM  = 2
PORT     = 5
PRIVLEV  = 5
LOGON    = 10:27:58 07/04/96
```

CPU time to perform SHOW WHO was 340 ticks or 0 seconds.

---

## close

Close an open file.

```
#include <io.h> or <handle.h>
```

```
int close ( int fhandle )
```

---

*fhandle*                    »    file handle of an open file

**Operation:**        The file associated with *fhandle* is closed. Any data remaining in its write buffer is flushed to the file prior to closing it.

**Returns:**            A zero return indicates success; a non-zero return indicates an error.

**Errors:**             When an error is detected, [errno](#) is set to EBADF.

**Conforms to:**                **close**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**            [fileno](#)

**closesocket**

Close a socket.

```
#include <socket.h>
int closesocket ( SOCKET s )
```

---

*s*                      »    Socket number

**Operation:** This function closes a socket. Specifically, it releases the socket descriptor *s* so that further references to *s* will fail with the error `TSAENOTSOCK`. If this is the last reference to the underlying socket, the associated naming information and queued data are discarded.

**Returns:** A zero. When an error is detected, `SOCKET_ERROR` is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
<code>TSAENETDOWN</code>	The THEOS Sockets implementation has detected that the network subsystem has failed.
<code>TSAENOTSOCK</code>	The descriptor <i>s</i> is not a socket.
<code>TSAEINPROGRESS</code>	A blocking THEOS Sockets call is in progress.

**Notes:** The semantics of `closesocket` are affected by the socket options `SO_LINGER` and `SO_DONTLINGER` as follows:

<i>Option</i>	<i>Interval</i>	<i>Type of close</i>	<i>Wait for close?</i>
<code>SO_DONTLINGER</code>	Don't care	Graceful	No
<code>SO_LINGER</code>	Zero	Hard	No
<code>SO_LINGER</code>	Non-zero	Graceful	Yes

If `SO_LINGER` is set (i.e. the *l\_onoff* field of the linger structure is non-zero) with a zero timeout interval (*l\_linger* is zero), `closesocket` is not blocked even if queued data has not yet been sent or acknowledged. This is called a “hard” close because the socket is closed immediately and any unsent data is lost. Any `recv` call on the remote side of the circuit can fail with `TSAECONNRESET`.

If `SO_LINGER` is set with a non-zero timeout interval, the `closesocket` call blocks until the remaining data has been sent or until the timeout expires. This is called a graceful disconnect. Note that if the socket is set to non-blocking and `SO_LINGER` is set to a non-zero timeout, the call to `closesocket` will fail with an error of `TSAEWOULDBLOCK`. If `SO_DONTLINGER` is set on a stream socket (i.e. the *l\_onoff* field of the linger structure is zero), the `closesocket` call will return immediately. However, any data queued for transmission will be sent if possible before the underlying socket is closed. This is also called a graceful disconnect. Note

that in this case the THEOS Sockets implementation may not release the socket and other resources for an arbitrary period, which may affect applications that expect to use all available sockets.

**Conforms to:**        **closesocket**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**        [accept](#), [ioctlsocket](#), [setsockopt](#), [socket](#)

**Example:**        See the example for the function [socket](#).

### **cmd\_abbrev**

Determine the full file name of a program given only its abbreviation.

```
#include <stdlib.h>
char * cmd_abbrev ( char * name )
```

---

*name*                    »   pointer to command name or abbreviation

**Operation:**        Using the currently defined synonym file, look up any abbreviations or synonyms for *name* and find the program name on disk using the current drive search sequence.

**Returns:**           A pointer to the full file name of the program indicated by *name*. A null pointer indicates that the program could not be found.

**Errors:**            No errors are detected.

**Conforms to:**        **cmd\_abbrev**    q ANSI      q DOS      n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    char  name[9];

    printf("\nEnter a command abbreviation: ");
    gets(name);

    printf("\nThe full name of \"%s\" is \"%s\"\n",
           name, cmd_abbrev(name));
}
```

---

**Output:**

>example

Enter a command abbreviation: l

The full name of "l" is "SYSTEM.CMD32.LOGON"

>

## conmask

This function sets your user's console input mask. This input mask controls the case mode and input echo mode for your console.

```
#include <stdio.h>
void conmask ( char * mask )
```

---

*mask*                    »    pointer to new mask specification

**Operation:**        The *mask* is evaluated and the appropriate bits are set or reset in the con-mask field of your SCR.

Mask char	Meaning	Mask char	Meaning
n or E	No input echo	0	1st char control OFF
e	Input echo	1	1st char control ON
u	Fold to uppercase	s	Suppress output ON
m	Accept as typed	S	Suppress output OFF
l	Invert the input case	w	Screen page wait ON
k	EXEC stack suppress output ON	W	Screen page wait OFF
K	EXEC stack suppress output OFF		

**Returns:**            No value is returned by this function.

**Errors:**            Invalid characters in the *mask* are ignored and do not generate an error. If there are two conflicting *mask* characters specified (for instance: “en”), the last or right-most one will be the one used.

**Notes:**            It is not necessary to specify all parameters. For instance, if you are only interested in turning echo on, a mask setting of “e” is sufficient. The other parameters (case mode folding, 1st character control action, *etc.*) will remain as they are currently set.

**Defaults:**        The initial mask set by the CSI is “em0”.

**Conforms to:**        conmask    q ANSI    q DOS    n THEOS    q POSIX

### Example:

```
#include <stdio.h>
#include <ctype.h>

int
getans(void)                // get single key response
{
    int  ans;                // temp field for response

    conmask("nu");           // set input to uppercase, no echo

    do      {
        ans = getchar();    // accept operator input
    } while (iscntrl(ans));  // response unacceptable?

    conmask("em");           // reset to normal operation
    return ans;
}
```



## connect

Establish a connection to a peer.

```
#include <socket.h>
int connect ( SOCKET s, SOCKADDR * addr, int addrlen )
```

---

<i>addr</i>	»	Pointer to buffer for socket address
<i>addrlen</i>	»	Pointer to length of socket address buffer
<i>s</i>	»	Socket number

**Operation:** This function is used to create a connection to the specified foreign association. The parameter *s* specifies an unconnected datagram or stream socket. If the socket is unbound, unique values are assigned to the local association by the system, and the socket is marked as bound. Note that if the address field of the *addr* structure is all zeroes, connect will return the error TSAEADDRNOTAVAIL.

For stream sockets (type SOCK\_STREAM), an active connection is initiated to the foreign host using *addr* (an address in the name space of the socket). When the socket call completes successfully, the socket is ready to send or receive data.

For a datagram socket (type SOCK\_DGRAM), a default destination is set, which will be used on subsequent [send](#) and [recv](#) calls.

On a non-blocking socket, if the return value is SOCKET\_ERROR, an application should call [TSAGetLastError](#). If this indicates an error code of TSAEWOULDBLOCK, then your application can use [select](#) to determine the completion of the connection request by checking if the socket is writable.

**Returns:** A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is SOCKET\_ERROR, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEADDRINUSE	The specified address is already in use.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAEADDRNOTAVAIL	The specified address is not available from the local machine.
TSAEAFNOSUPPORT	Addresses in the specified family cannot be used with this socket.
TSAECONNREFUSED	The attempt to connect was forcefully rejected.
TSAEDESTADDRREQ	A destination address is required.

<i>Code</i>	<i>Meaning</i>
TSAEFAULT	The <i>addrlen</i> argument is incorrect.
TSAEINVAL	The socket is not already bound to an address.
TSAEISCONN	The socket is already connected.
TSAEMFILE	No more file descriptors are available.
TSAENETUNREACH	The network can't be reached from this host at this time.
TSAENOBUFFS	No buffer space is available. The socket cannot be connected.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.
TSAETIMEDOUT	Attempt to connect timed out without establishing a connection.
TSAEWOULDBLOCK	The socket is marked as non-blocking and the connection cannot be completed immediately. It is possible to select the socket while it is connecting by using <a href="#">select</a> for writing.

**Conforms to:**                    **connect**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [accept](#), [bind](#), [getsockname](#), [select](#), [socket](#)

**Example:**                    See the example for the function [socket](#).

## conrdy, \_conrdy

Test if character available for input from the console.

```
#include <conio.h>
unsigned short conrdy ( void )

#include <sc.h>
unsigned short _conrdy ( void )
```

**Operation:** Tests the console input stream to determine if there is a character ready to be read. No character is actually read by this function.

**Returns:** A true/false indicator: non-zero indicates that there is a character available, zero that there is not.

**Errors:** No error is detected.

**Notes:** This function tests the console input device, not stdin. Redirection is not possible.

The console is “ready” if there is data in the EXEC stack buffer, if characters have been “pushed” onto the type-ahead buffer with the [ungetch](#) function, if a key is pressed on the console keyboard, or if an On Key or mouse event has occurred.

<b>Conforms to:</b>	<b>conrdy</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_conrdy</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [cgets](#), [getch](#)

---

### Example:

```
#include <stdio.h>
#include <conio.h>

main()
{
    char c;

    cputs("\nPress any key to interrupt processing...");
    while (!conrdy()) {
        // do some process
    }

    c = getch();                // get character typed
    ...
}
```

*\_con\_tran*

The *\_con\_tran* function is used only by device drivers to test and translate system control key input from a console.

#include <driver.h>  
unsigned short *\_con\_tran* ( UCB \* *ucb*, char *c*, void \_far \* *buffer* )  
  
-----  
*c*                   »   character to be translated  
*buffer*               »   pointer to storage area for translation  
*ucb*                  »   pointer to the device unit control block

**Operation:**       The *ucb* is checked to determine if it is a console input device. If it isn't, the function is exited.

Next, the *ucb* is checked to see if the system escape character pending flag is set. If it isn't and this character matches the currently defined system escape character, the flag is set and the function is exited.

When the device is the console and the system escape character pending flag is set, the character *c* is checked to see if it matches any of the defined system control keys:

<i>c</i>	Meaning	<i>c</i>	Meaning
c	Program cancel	q	System quit
d	Debug break	s	System suspend toggle
o	Console display toggle	t	Clear type-ahead
p	Console echo toggle	w	Console page-wait toggle

If *c* matches any of the characters, the appropriate action is taken.

The system escape character pending flag is cleared.

**Returns:**       When the character is acted upon by this function, a zero is returned. Otherwise, the character is translated with the translation stored as a string in *buffer*. The length of the translated string is the return value of the function.

**Notes:**        A device-driver that might be attached as a console should test to see if it is attached as the console and, if so, process every character it receives from the device through this function.

**Restrictions:**   Can be used only by device drivers.

**Conforms to:**        *\_con\_tran*    q ANSI    q DOS    n THEOS    q POSIX

**cos, cosh**

Compute the cosine or hyperbolic cosine of an angle.

```
#include <math.h>
double cos ( double x )
double cosh ( double x )
```

---

$x$                       »   floating-point angle, in radians

**Operation:**      **cos**              Compute the cosine of the angle  $x$ .

**cosh**            Compute the hyperbolic cosine of the angle  $x$ .

**Returns:**            The cosine or hyperbolic cosine value is returned.

**Errors:**            When  $x$  is too large to produce meaningful results, a zero is returned and [errno](#) is set to ERANGE.

**Notes:**            The cosine of an angle is always in the range of  $-1$  to  $+1$ .

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                      **cos**      n ANSI      n DOS      n THEOS      n POSIX  
  **cosh**      n ANSI      n DOS      n THEOS      n POSIX

**See also:**            [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

### **cot, coth**

Compute the cotangent or hyperbolic cotangent of an angle.

```
#include <math.h>
double cot ( double x )
double coth ( double x )
```

---

*x*                      »   floating-point angle, in radians

**Operation:**        **cot**            Compute the cotangent of the angle *x*.

**coth**            Compute the hyperbolic cotangent of the angle *x*.

**Returns:**            The cotangent or hyperbolic cotangent value is returned.

**Errors:**            There is no error return for these functions.

**Notes:**            This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

<b>Conforms to:</b>	<b>cot</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>coth</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**        [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

## cprintf

Format characters, strings, literals and numbers, and display on the console.

```
#include <conio.h>
```

```
int cprintf ( char * format [, argument] ...)
```

---

*argument*           »   optional arguments to format and print

*format*             »   formatting control mask

**Operation:**       Format and print the *arguments* using the *format* mask. Output is to the console display.

**Returns:**         The number of characters displayed.

**Notes:**           This function is implemented as a system call and is intended for device drivers. As a system call, it adds very little overhead to a program.

This function operates like the [printf](#) function described on page 484, with some exceptions. The primary difference is that output is to the console display, not stdout.

The syntax and usage of the format field in cprintf is identical to the format field used in [printf](#). However, not all of the codes are available in the format specifications for cprintf. Specifically:

*flags*            Supports space, #, 0, ,, +, - and \* codes.

*modifier*        Supports h, l and L.

*type*            Supports %, b, c, d, i, o, p, s, u, x, and X, not e, E, f, g, G, or n.

**Version 4:**       In addition to the above features, cprintf used on a THEOS 32 Version 4 or above operating system also supports the b type code, flags of -, + and comma, and the flags may be specified in any order.

**Conforms to:**       cprintf   q ANSI    n DOS    n THEOS   q POSIX

**See also:**         [fprintf](#), [printf](#), [sprintf](#), [vfprintf](#), [vprintf](#), [vsprintf](#)

---

**Example:**         See the [printf](#) example.

### **cpu\_time**

Get the current time-of-day in milliseconds.

```
#include <time.h>
unsigned long cpu_time ( void )
```

**Operation:** The software clock maintained in the operating system nucleus is returned.

**Returns:** The number of milliseconds since midnight of the current day.

**Conforms to:** **cpu\_time** q ANSI q DOS n THEOS q POSIX

**See also:** [clock](#), [time](#)

---

**Example:** See also the [rand](#), [srand](#) functions.

```
#include <stdio.h>
#include <time.h>

void
main(void) {
    unsigned long t;

    t = cpu_time();           // get time-of-day
    printf("\nThe current time in milliseconds is %,lu.\n", t);
}
```

---

**Output:**

>example

The current time in milliseconds is 297,283,720.

>





**crc16, crc32**

Generate a 16-bit or 32-bit CRC (Cyclic Redundancy Code) checksum code.

```
#include <crc.h>
unsigned short crc16 ( char c, unsigned short crc )
unsigned long crc32 ( char c, unsigned long crc )
```

---

<i>c</i>	»	character (8-bit) value
<i>crc</i>	»	current crc checksum

**Operation:**      **crc16**      Generates and returns a new 16-bit checksum using the existing checksum of *crc* and the new byte value *c*.

**crc32**      Generates and returns a new 32-bit checksum using the existing checksum of *crc* and the new byte value *c*.

**Returns:**        **crc16**        The new 16-bit checksum.

**crc32**        The new 32-bit checksum.

**Notes:**                These functions use a table lookup algorithm to provide consistent and fast generation of checksums.

**Conforms to:**                **crc16**    q ANSI    q DOS    n THEOS    q POSIX  
                                 **crc32**    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
#include <crc.h>

unsigned long
checkit(void * buffer, int len)      // generate checksum for buffer
{
    unsigned long crc = 0;            // initialize checksum

    while (len--) {                   // repeat for entire buffer
        crc = crc16(*buffer++, crc);
    }
    return crc;
}
```

## creat

These functions create new files on disk or tape.

```
#include <handle.h> or <io.h>
int creat ( char * fname, int mode )
```

---

<i>fname</i>	»	pointer to complete file name
<i>mode</i>	»	file organization mode

- Operation:** A new file is created. After the file is created, it is opened and a handle to the file is returned.
- Returns:** Returns a non-negative file descriptor for the new file. A return of -1 indicates an error, in which case [errno](#) will be set.
- Errors:** There are many errors that might be detected by `creat`. Use the [ferror](#) function to determine the specific cause of the error.
- Defaults:** The file created is always owned by the current account id. The protection codes set for the new file are `XWR.X..` (Version 4 systems will use the `CREATE` environment variable).
- Newly created stream files are empty (zero length).
- Restrictions:** The *mode* argument is not used. Only stream files are created with this function.
- Conforms to:** `creat`    q ANSI    n DOS    n THEOS    n POSIX
- See also:** [fcreate](#), [fopen](#), [makelib](#), [\\_makelib](#), [\\_mklib](#), [mkdir](#)

**crt**

Output console cursor control or screen-editing command, or display a special character on the console.

```
#include <crt.h>
void crt ( int code )
```

---

*code*                   »   code for attribute or character to output

**Operation:**       The character specified by *code* is output to the console.

**Notes:**           This function is actually a synonym to the [putch](#) function. It is provided as a more symbolic name for screen editing operations and for consistency with other languages (e.g., BASIC CRT\$ function).

The CRT.H header file defines many of the common cursor-control and screen-editing codes. For instance, to clear the screen merely specify a *code* of CLEAR (do not enclose it in quotes). In addition, the header file defines names for the line-drawing graphics characters, such as ULC for the upper-left-corner symbol, *etc.* Although no names are defined for international characters, they are listed as a comment for quick-reference purposes.

If a console echo file is enabled, the character is first written to that file.

If console output is suppressed, no character is displayed on the console display device (but it might still be output to the console echo file).

Control characters are translated by the console's class code.

This function outputs to the console device, not stdout. Redirecting stdout has no affect on this function.

**Conforms to:**                   **crt**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**           [at](#), [atstr](#), [has](#), [hascolor](#), [putch](#), [wClear](#)

**Example:**

```
#include <conio.h>
#include <crt.h>

void dsp_config(void)
{
    crt(CLEAR);
    at(0,2);
    cputs("Company name:");
    at(0,3);
    cputs("Address:");
    at(0,4);
    cputs("City-state-zip:");
    at(0,6);
    cputs("Telephone:");
    at(0,7);
    cputs("Fax:");
    dsp_config_info();
}

void dsp_config_info(void)
{
    at(16,2); crt(EOL);
    crt(RVON);
    cputs(config.coname);
    crt(RVOFF);
    at(16,3); crt(EOL);
    crt(RVON);
    cputs(config.coaddr);
    crt(RVOFF);
    at(16,4); crt(EOL);
    crt(RVON);
    cprintf("%s, %s %s",config.cocity,config.costate,config.cozip);
    crt(RVOFF);
    at(16,6); crt(EOL);
    crt(RVON);
    cputs(config.cophone);
    crt(RVOFF);
    at(16,7); crt(EOL);
    crt(RVON);
    cputs(config.cofax);
    crt(RVOFF);
}
```

**crtcolor**

Set the colors for subsequent display of text on the console display device.

```
#include <stdio.h> or <conio.h>
void crtcolor ( int fg, int bg, int rvfg, int rvbg )
```

---

<i>bg</i>	»	background color code
<i>fg</i>	»	foreground color code
<i>rvbg</i>	»	reverse video background color code
<i>rvfg</i>	»	reverse video foreground color code

**Operation:** Sets the colors for subsequent output to the console output device.

**Errors:** No error is detected.

**Notes:** The color codes are:

Code	Color	Code	Color
0	Black	4	Red
1	Blue	5	Magenta
2	Green	6	Yellow
3	Cyan	7	White

The actual capabilities and procedure for changing colors on a console are determined by the console's class code.

An *rvfg* code of -1 means that the reverse video colors are the inverse of the normal video colors. For instance:

```
crtcolor(7,1,-1,-1)
```

is the same as:

```
crtcolor(7,1,1,7)
```

**Conforms to:** **crtcolor** q ANSI q DOS n THEOS q POSIX

**See also:** [has](#), [hascolor](#)

**Example:**

```
#include <stdio.h>
#include <conio.h>
#include <crt.h>
#include <colors.h>           // define color names

void dsp_config_info(void)
{
    if (hascolor())
        crtcolor(WHITE, BLUE, YELLOW, BLUE);
    at(16,2); crt(EOL);
    crt(RVON);
    cputs(config.coname);
    crt(RVOFF);
    at(16,3); crt(EOL);
    crt(RVON);
    cputs(config.coaddr);
    crt(RVOFF);
    at(16,4); crt(EOL);
    crt(RVON);
    cprintf("%s, %s %s",config.cocity,config.costate, config.cozip);
    crt(RVOFF);
    at(16,6); crt(EOL);
    crt(RVON);
    cputs(config.cophone);
    crt(RVOFF);
    at(16,7); crt(EOL);
    crt(RVON);
    cputs(config.cofax);
    crt(RVOFF);
}
```

### **csc, csch**

Compute the cosecant or the hyperbolic cosecant of an angle.

```
#include <math.h>
double csc ( double x )
double csch ( double x )
```

---

*x*                      »   floating-point angle, in radians

**Operation:**        **csc**            Compute the cosecant of the angle *x*.  
                     **csch**          Compute the hyperbolic cosecant of the angle *x*.

**Returns:**          **csc**            The cosecant of the angle *x*.  
                     **csch**          The hyperbolic cosecant of the angle *x*.

**Notes:**            The angle *x* is specified with radians, not degrees.

These functions use BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**            **csc**      q ANSI      q DOS      n THEOS      q POSIX  
                             **csch**      q ANSI      q DOS      n THEOS      q POSIX

**See also:**            [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [sec](#), [sech](#), [sin](#), [sinh](#), [tan](#), [tanh](#)



---

**csi**

This function executes a compiled command program without returning to this program.

```
#include <process.h> or <sc.h>
void csi ( const char _far * cmdstr )
```

---

*cmdstr*                   »   command string to execute

- Operation:** All of the open files in the current program are closed and all subtasks of this program are killed. The current program is exited by passing *cmdstr* to the command string interpreter for execution.
- Returns:** There is no return from this function call. After *cmdstr* is executed, control returns to the operating system.
- Errors:** If the *cmdstr* cannot be executed for some reason, the command string interpreter will handle the error in the same manner as if the command had been typed by the operator and a CSI prompt.
- Notes:** Other than command-line arguments specified in *cmdstr*, there is no communication between the current program and the called program.
- If the current program was using multiple code-segments, all of those code segments are unloaded.
- This function is similar to the [system](#) function except the csi function does not return to the current program and the [system](#) function does.
- Conforms to:**                   csi   q ANSI    q DOS    n THEOS   q POSIX
- See also:**                   [system](#), [execl](#)
- 

**Example:**

```
#include <process.h>

main()
{
    ...
    csi("filelist *.command *.exe (print");
}
```

**ctime**

Converts the date and time in a calendar time value to a string.

```
#include <time.h>
char * ctime ( const time_t * timer )
```

---

*timer*                    »   pointer to calendar time value

**Operation:**        The date and time indicated by the *timer* value are converted to a string using the [strftime](#) function.

When time is a NULL pointer, the current date and time are used.

**Returns:**           A pointer to the converted string. This string is always 26 characters long, including the null terminator.

**Errors:**            No errors are detected by this function.

**Notes:**            This function is equivalent to:

```
asctime(localtime(timer));
```

**Restrictions:**    The string pointed to by the return value is local to the [asctime](#) function and will be reused the next time that *ctime* or [asctime](#) is called. Either use the string immediately or make a copy of it.

**Conforms to:**                **ctime**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [asctime](#), [difftime](#), [gmtime](#), [localtime](#), [mktime](#), [strftime](#), [time](#)

---

**Example:**

```
#include <stdio.h>
#include <time.h>

main()
{
    time_t t;

    time(&t);                    // get current date/time

    printf("\nIt is now %s", ctime(&t));
}
```

---

**Output:**

>example

It is now Thu Jul 04 15:04:37 1996

>

## CursorBlock, CursorHide, CursorLine, CursorShow

Change the console cursor display attribute.

```
#include <stdio.h> or <conio.h>
void CursorBlock ( void )
void CursorHide ( void )
void CursorLine ( void )
void CursorShow ( void )
```

**Operation:**      **CursorBlock** Change the cursor to a block.

**CursorHide** Hide the cursor.

**CursorLine** Change the cursor to an underline.

**CursorShow** Display the cursor.

**Notes:**            Some functions, such as [wEdit](#), change the shape and the display of the cursor when invoked.

The main usage of the CursorHide function is to disable the cursor display when the program is not requesting or expecting any input from the operator. Remember to use the CursorShow function prior to requesting input because it will remain hidden if you don't.

**Defaults:**        When a program starts, the cursor is displayed as a blinking block (CursorShow and CursorBlock).

**Restrictions:**    The cursor is changed by these functions only when the console is a class 90 series, class 170 series, class 180 series, class 190 series , class 200 series or a class 210 series display (main console, PC Terminals, Windows workstations),

<b>Conforms to:</b>	<b>CursorBlock</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>CursorHide</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>CursorLine</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>CursorShow</b>	q ANSI	q DOS	n THEOS	q POSIX

**cuserid**

Retrieve the current user or account name that you are logged on to.

```
#include <stdlib.h>
char * cuserid ( char * user )

_____
user          »   pointer to storage area
```

**Operation:** The [getlogin](#) function is used to make a local copy of the currently logged-on account name. This copy has all trailing spaces removed. The resulting string is copied to *user*. If *user* is NULL, then the name is not copied.

**Returns:** A pointer to the currently logged-on account name.

When the program is operating as a background user, a pointer to a NULL string is returned.

**Notes:** If you do not want the account name copied to your data space, use the NULL argument.

**Restrictions:** When *user* is not NULL, it must be a pointer to an area of at least nine bytes (eight characters plus the null terminator).

When *user* is NULL, the copy of the account name is local to the *cuserid* function and is reused each time that *cuserid* is called.

The copy of the trimmed account name is local to the *cuserid* function and will be reset each time that *cuserid* is called.

**Conforms to:** **cuserid**    q ANSI    q DOS    n THEOS    n POSIX

**See also:** [getlogin](#), [getuid](#), [logname](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
    FILE * logfile;
    char account[9];

    logfile = fopen("log.file","wa");
    cuserid(account);
    if (account[0]==0)
        strcpy(account,"Phantom");
    fprintf(logfile, "User name = %s (%i)\n",account,getuid());
    fclose(logfile);
}
```

**Output:**

```
>example
```

```
>start example
```

```
>list log.file
```

```
User name = SAMPLES (2)
```

```
User name = Phantom (2)
```

```
>
```

### decrement memory assembly functions

This group of “functions” generates in-line code to decrement a byte, short integer or long integer in a specified memory area, your program’s code segment or in the nucleus segment.

```
#include <builtin.h> or <peek.h>
void _decb ( void _far * far_offset )
void _decld ( void _far * far_offset )
void _decw ( void _far * far_offset )
```

```
void _deccsb ( void * offset )
void _deccsd ( void * offset )
void _deccsw ( void * offset )
```

```
void _decnucb ( void * offset )
void _decnucld ( void * offset )
void _decnucw ( void * offset )
```

---

*far\_offset*           »   pointer to remote data object

*offset*               »   pointer to data item in code segment or nucleus segment

**Operation:**       To decrement means to subtract one from a value.

- \_decb**           Decrements the character or byte value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_decld**          Decrements the long integer value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_decw**           Decrements the short integer value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_deccsb, \_deccsd, \_deccsw**   These functions perform the same operation as **\_decb**, **\_decld**, and **\_decw** except that the memory segment is predefined to be your program’s code segment alias. These three functions can only be used in device-drivers because other programs do not have an alias to their code segment and they will generate a general protection error.
- \_decnucb, \_decnucld, \_decnucw**   These functions perform the same operation as **\_decb**, **\_decld**, and **\_decw** except that the memory segment is predefined to be the nucleus memory segment.

**Returns:**       No value is returned by these functions. The memory locations are decremented in place.

**Notes:**        These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can decrement whatever type of object desired in any location.

For instance, a `_decw` function can decrement a word value (double-byte value) in a location that is declared as a character string. Of course, the program would have to be coded such that it could handle this situation.

**Restrictions:** These functions are intended for usage by device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

**Conforms to:**

<code>_decb</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_decd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_decnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_decnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_decnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_deccsb</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_deccsd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_deccsw</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_decw</code>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**

[add memory assembly functions](#), [and memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

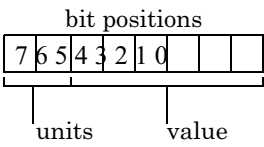
**delay**

This function suspends operation of your program for a specified amount of time.

#include <timer.h> or <sc.h>  
void **delay** ( char *time*)  
  
\_\_\_\_\_ *time*                    »    coded period of time

**Operation:** Execution of the program is suspended for the time length specified by *time*. The program may be awakened early by an interrupt event, such as an [alarm](#), multi-tasking semaphore signal, *etc.*

**Notes:** The value of *time* is coded: The two high-order bits specify the unit of measurement and the low-order six bits specify the length.



The units of measurement are:

Bits 76	Unit of time
00	milliseconds
01	hundredths of second
10	tenths of second
11	seconds

After interpreting the *time* value, *delay* calls the [sleep](#) function to actually perform the delay.

A program that is “delaying” may be awakened early by an interrupt from one of the functions [alarm](#), [clock](#), *delay*, [msalarm](#), [signal](#) or a semaphore awakening action.

**Conforms to:**                    **delay**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [pause](#), [sleep](#), [sleep\\_msec](#), [sleep\\_sec](#), [sleep\\_until](#)



delettek

Deletes a record from a keyed or indexed access file.

```
#include <stdio.h>
unsigned short delettek ( FILE * file, char _far * key )
```

---

<i>file</i>	»	pointer to file's fcb
<i>key</i>	»	pointer to record key

- Operation:** The record whose key matches the value in *key* is deleted from the file.
- Returns:** A non-zero return indicates the record was successfully deleted or that it does not exist; a zero indicates that it could not be deleted.
- Notes:** The key field of a keyed or indexed file is not a C string. That is, it is not necessarily a null-terminated sequence of characters. It is a fixed-length sequence of characters that may have null characters embedded within.

If *file* has been opened with access “r+” and without specifying the access modifier “m,” then deleting a record releases any other record locks set by this user for this file.

If *file* is in use by another user and that user has the desired record locked, the *delettek* function will wait for that lock to be released before deleting the record. If the delete is not immediately successful because the record is locked by another user, the function may wait for awhile to see if the lock is cleared by another user. The *scr.lock\_wait* item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in *scr.lock\_wait* can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the delete request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in *scr.lock\_pid*.

See also “Automatic Record-Locking:” on page 224.

- Restrictions:** This function may only be used with a file opened with keyed or indexed organization, and write or update access.

**Conforms to:** delettek q ANSI q DOS n THEOS q POSIX

**See also:** ldelettek, readk, readn, readp, writtek

**\_detach**

Detach a device.

```
#include <sc.h>
unsigned short _detach ( int lub )
```

---

*lub*                      »    logical unit number

**Operation:**        The device-driver for the device currently attached to *lub* is logically disconnected from the physical device. When no other device is using the device-driver, it is unloaded from memory.

**Returns:**            A success/fail indicator. A zero return indicates the device was successfully detached; a non-zero indicates that it could not be detached.

**Notes:**             This function performs the same operation as the ATTACH command when the detach mode is used.

**Restrictions:**      Neither the console nor disk drive s can be detached.

                      This function is usable only when the program is executing on THEOS 32 Version 4 or above.

**Conforms to:**            \_detach    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>
#include <sc.h>

void
main()
{
    if (_detach(32))                // detach COMM1
        printf("\nCOMM1 cannot be detached.");
    else
        printf("\nCOMM1 detached");
}
```

## devnames functions

Open, close or access the SYSTEM.TEOS32.DEVNAMEs file.

```
#include <stdlib.h>
void devnames_close ( FILE * devnamefile )
FILE * devnames_open ( void )
void devnames_parse ( char * line, char * name, int * ucb, int * device,
    int * unit, char * type, char * options )
char * devnames_read ( FILE * devnamefile )
```

---

<i>device</i>	»	pointer to string storage for device number
<i>devnamefile</i>	»	pointer to open device-names file's fcb
<i>line</i>	»	pointer to string containing record to parse
<i>name</i>	»	pointer to string storage for name of device definition
<i>options</i>	»	pointer to string storage for options for device
<i>type</i>	»	pointer to string storage for type specifications
<i>ucb</i>	»	pointer to string storage for unit control block number
<i>unit</i>	»	pointer to string storage for unit number within ucb

**Operation:**     **devnames\_close**   Close the file specified by *devnamefile*. This should be the same value that was returned by the *devnames\_open* function.

**devnames\_open**   Open the SYSTEM.TEOS32.DEVNAMEs file or the memory image if the device names file is loaded.

**devnames\_parse**   Analyze and parse the device-names record specified by *line*. The individual components of *line* are extracted as separate strings and stored in the locations pointed to by *name*, *ucb*, *device*, *unit*, *type* and *options*.

**devnames\_read**   Read the next non-blank and non-comment record from the device names file opened as *devnamefile*. Any comment on the line and the new-line character at the end of the record are removed and the string is null-terminated.

**Returns:**     **devnames\_open**   A pointer to the opened device-names file's fcb.

**devnames\_read**   A pointer to the record read from the device-names file. When there are no more non-blank, non-comment records in the file, a NULL pointer is returned.

**Notes:**     The SYSTEM.TEOS32.DEVNAMEs file is used by ATTACH, SYSGEN and other commands to translate the mnemonic names of devices, class codes and VDI types into the numbers used by THEOS.

Although there are many types of records in this file, they all use the same syntax:

SI01	6:5:0	CPSIO	W8	; first com port
_____	_____	_____	_____	_____
Name	Number	Type	Options	Remark/Comment

- Name

This is the mnemonic name used when attaching a device. It is also the name used when ATTACH displays the current attachments.
- Number

This is the number used by THEOS for the device and it is composed of three parts: *ucb:device:unit*. These components are separated into three fields: *ucb*, *device* and *unit*.  
  
'N' and 'V' type records only have a single number in this field. It specifies the number of the class code or VDI routine.
- Type

This field identifies the device or entry type. Multiple codes may be used on each device.

Type	Meaning
D	Disk device (implied I and O).
T	Tape device (implied I and O).
C	Console device.
I	Device accepts input.
O	Device performs output.
P	Device may be used as a printer.
S	Device uses serial communications.
N	Class code name record.
V	VDI name record.

- Options

The default options applied when this name is used for a new device attachment. For instance: B38400,W8,E2.
- Comment

The semicolon marks the beginning of a comment. All characters from the semicolon to the end of the line are ignored.

A record may be composed of only a comment if it starts with a semicolon character.

The `devnames_read` function reads the next record from the file ignoring any records that are blank or contain only a comment. The `devnames_parse` function parses that record into the individual components. If there are components of the record that you are not interested in, merely use a `NULL` pointer as an argument for that component.

Conforms to:	<code>devnames_close</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>devnames_open</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>devnames_parse</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>devnames_read</code>	q ANSI	q DOS	n THEOS	q POSIX

See also: [fclose](#), [fcloseall](#), [openhelp](#), [openmenu](#)

**Example:** The following example shows the usage of several functions including the devnames functions, [WhoAmI](#), [\\_peekscrb](#), [\\_peekscrd](#), [\\_peekscrw](#), and [time](#) functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <whoami.h>
#include <scr.h>

int bcd2int(char); // convert packed bcd to integer value

void
main(int argc, char *argv[])
{
    WHOAMI    whoami;
    FILE      *devnfile;
    char      *dev_record,
              dev_name[9],
              dev_type[10],
              dev_opt[100];

    int       dev_uch,
              dev_dev,
              dev_unit;

    int       netid1, netid2, netid3, netid4,
              found,
              class;

    char      logintime[100],
              logindur[100],
              con_options[20],
              work[5], work1[5];

    struct tm  logtime;
    struct {
        int days;
        int hours;
        int minutes;
        int seconds;
    } logdur;

    time_t     cur_utc,
              log_utc,
              logondur;

    time(&cur_utc);
    logtime = * localtime(&cur_utc); // fill in struct with cur time
    // get login time from scr
    logtime.tm_mon = bcd2int(Scr->logdate[0])-1; // base 0
    logtime.tm_mday = bcd2int(Scr->logdate[1]);
    logtime.tm_year = bcd2int(Scr->logdate[2]);
    logtime.tm_hour = bcd2int(Scr->logtime[0]);
    logtime.tm_min = bcd2int(Scr->logtime[1]);
    logtime.tm_sec = bcd2int(Scr->logtime[2]);

    // compute login duration, in seconds
    log_utc = mktime(&logtime);
    logondur = cur_utc - log_utc;

    WhoAmI(&whoami); // get whoami information
```

```
devnfile = devnames_open(); // open device names file

found = 0;
while (dev_record=devnames_read(devnfile)) {
    devnames_parse(dev_record, dev_name, &dev_uch, &dev_dev,
        &dev_unit, dev_type, dev_opt);
    if (strcmp(dev_name, whoami.device_name)==0) {
        found = -1;
        break;
    }
}
if (found) {
    whoami.uch_num = dev_uch;
    whoami.dev_num = dev_dev;
    whoami.sess_num = dev_unit;
    found = 0;
}

rewind(devnfile);          // rewind to start of file
class = getclass(27);      // get console class code number

while (dev_record=devnames_read(devnfile)) {
    devnames_parse(dev_record, dev_name, &dev_uch, &dev_dev,
        &dev_unit, dev_type, dev_opt);

    if (*dev_type=='N' && dev_uch==class) {
        found = -1;
        break;
    }
}
sprintf(con_options, "%i x %i, %s", getll(27), getpl(27),
    found ? dev_name : strcat(strcpy(work, "C"),
        itoa(work1, class)));

strftime(logintime, 100, "%A, %B %d, %Y, %I:%M %p", &logtime);

// convert login duration to days, hours, minutes, seconds
logdur.days = logondur/86400;
logondur -= logdur.days * 86400;
logdur.hours = logondur/3600;
logondur -= logdur.hours * 3600;
logdur.minutes = logondur/60;
logondur -= logdur.minutes * 60;
logdur.seconds = logondur;

// format login duration display
if (logdur.days)
    sprintf(logindur, "%i days, %i hours, %i minutes, %i seconds",
        logdur.days, logdur.hours, logdur.minutes, logdur.seconds);
else if (logdur.hours)
    sprintf(logindur, "%i hours, %i minutes, %i seconds",
        logdur.hours, logdur.minutes, logdur.seconds);
else
    sprintf(logindur, "%i minutes, %i seconds",
        logdur.minutes, logdur.seconds);

printf("\nUser:           %s (%i)", getlogin(), getuid());
```

---

```

printf("\nLogon date:      %s", logintime);
printf("\nLogon duration:  %s", logindur);
printf("\nPid:            %i", whoami.pid+1); // adjust to base 1
printf("\nConsole:       %s [%i:%i:%i] %s", whoami.device_name,
      whoami.ucb_num, whoami.dev_num, whoami.sess_num, con_options);
if (whoami.net_client_type) {
    netid1 = (whoami.net_client_address & 0xff000000)>>24;
    netid2 = (whoami.net_client_address & 0x00ff0000)>>16;
    netid3 = (whoami.net_client_address & 0x0000ff00)>>8;
    netid4 = whoami.net_client_address & 0x000000ff;
    printf("\nNetwork id:      %s [%s client]",
          whoami.net_client_name,
          whoami.net_client_type==1 ? "Windows" : "Theos");
    printf("\nNetwork address: %i.%i.%i.%i",
          netid1, netid2, netid3, netid4);
}
devnames_close(devnfile); // close device names file
}

int
bcd2int(char bcd) // convert packed bcd to integer value
{
    return (bcd>>4)*10 + (bcd & 0x0f);
}

```

---

**Output:**

>example

```

User:          SAMPLES (2)
Logon date:    Saturday, December 07, 1996, 02:31 pm
Logon duration: 27 minutes, 0 seconds
Pid:          15
Console:      NET2 [152:104:2] 80 x 30, COLORPC
Network id:    CPWDoc [Windows client]
Network address: 192.9.200.8

```

>

***\_devopen***

Open or close a device-driver.

```
#include <sc.h>
void _devopen ( UCB * ucb, int cmd )
```

---

<i>ucb</i>	»	pointer to unit control block
<i>cmd</i>	»	device open or close code

**Operation:** This function is used by utility programs that attach or detach device-drivers, such as ATTACH, LOGOFF, START, *etc.*

When *cmd* is a zero, the device-driver associated with *ucb* is invoked at its device open entry. When *cmd* is not zero, the device-driver is invoked at its device close entry.

**Restrictions:** This system call is intended to be used by “trusted programs” only. That is, there is little or no checking performed by the system call because it assumes that it is proper to do the request.

**Conforms to:** *\_devopen* q ANSI q DOS n THEOS q POSIX

**See also:** [\\_detach](#)



---

## difftime

Compute the difference between two calendar time values.

```
#include <time.h>
double difftime ( time_t time1, time_t time0 )
```

---

*time0*               »   oldest or smaller time value  
*time1*               »   newest or largest time value

**Operation:**       The difference between the two values *time0* and *time1* are computed by subtracting *time1* from *time0*.

**Returns:**         The difference, in seconds, between *time0* and *time1*.

**Restrictions:**   Calendar time values are accurate to the nearest second. Any fractional seconds are not available with the [time](#) function or this function.

**Conforms to:**       **difftime**    n ANSI       n DOS       n THEOS    n POSIX

**See also:**         [ctime](#), [time](#)

---

### Example:

```
#include <stdio.h>
#include <time.h>

main()
{
    time_t      start,
               end;
    int         i;
    double      result;

    time(&start);                      // get starting time

    // perform some calculation or process
    for (i = 1; i < 50000; ++i) {
        result = 1.23 * 4.56;
    }

    time(&end);                        // get ending time

    printf("Elapsed time to perform process: %g seconds.\n",
           difftime(end, start));
}
```

---

### Output:

```
>test
Elapsed time to perform process: 1 seconds.

>
```

### directory search functions

These functions provide directory search capabilities.

```
#include <stdio.h> or <diskfind.h>
void dirclose ( void )
FDB * _dirfdb ( void )
int diropen ( char * fname_mask )
char * dirread ( void )

#include <diskfind.h>
DISK_UCB * _dirucb ( void )
```

---

*fname\_mask*        »    string containing file name specifications

- Operation:** Three of these functions are used as a set: **diropen**, **dirread** and **dirclose**. The other two functions are used after a **dirread** is performed to access additional information about the file entry.
- dirclose** Close and terminate the current directory search initialized by the **diropen** function.
- \_dirfdb** Return a pointer to the file directory block (FDB) for the directory entry found with the last **dirread** function call.
- diropen** Using the *fname\_mask*, a directory search is initialized. No actual reading of directory entries is performed by this function. The **dirread** function is used after a **diropen** to read each of the directory entries that match the *fname\_mask* specification.
- dirread** Using the *fname\_mask* defined by with **diropen** function, the complete file name of the next file matching that specification is located.
- \_dir\_ucb** Return a pointer to the unit control block (UCB) for the directory entry found with the last **dirread** function call.
- Returns:**
- \_dirfdb** A pointer to the FDB for the file.
- diropen** Returns a zero when the directory open was successful. When an error is encountered, a non-zero value is returned and the *\_errnum* and *\_errarg* are assigned values that reflect the nature of the error.
- dirread** Returns a pointer to a string containing the complete file description of the next file that matched the specification.
- A NULL is returned when no more files match.
- \_dir\_ucb** A pointer to the UCB for the file.
- Errors:**
- diropen** When an error is detected the return value is non-zero and *\_errnum* and *\_errarg* are assigned values that reflect the specific error encountered. These global variables are used by several functions that allow you to determine or report the nature

of the error. Refer to [errno](#), [\\_errno](#), [\\_errarg](#) variables on page 168 for information about these error reporting functions.

**Notes:** The *fname\_mask* variable used by the `diropen` function normally is a file name description containing wild cards. The types of wild cards allowed are the same as used by the FILELIST system utility command (@, #, ? and \*).

The *fname\_mask* specification may be a simple name without any directory path specifications in which case only files in the current working directory are searched. It may be a path specification relative to the current working directory, or it may be a full path specification relative to the root directory, in which case only those directory locations are searched.

**Restrictions:** Only one directory search may be open at any one time. To perform a second search, the first search is terminated with the `dirclose` function and then the second search is initiated with the `diropen` function.

<b>Conforms to:</b>	<b>dirclose</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_dirfdb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>diropen</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>dirread</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_dirucb</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [find\\_first](#), [find\\_next](#)

---

**Example:** Compare the following example with the example used in the [find\\_first](#), [find\\_next](#) function description.

```
#include <stdio.h>
#include <diskfind.h>
#include <string.h>

main()
{
    char    filename[256];
    char    *fn,
            type[20];
    FDB     *fdb;

    if (diropen("dos\\dos/*..*"))
        perror();
    putchar('\n');

    while (fn = dirread())
    {
        strcpy(filename, fn);
        fdb = _dirfdb();
        if (fdb->filestat & _FDB_STAT_DIRECTORY)
            strcpy(type, "(subdirectory)");
        else if (fdb->filestat & _FDB_STAT_LIBRARY)
            strcpy(type, "(library)");
        else if (fdb->filestat & _FDB_STAT_STREAM)
            strcpy(type, "(stream)");
        else if (fdb->filestat & _FDB_STAT_RELATIVE)
            strcpy(type, "(relative)");
        else if (fdb->filestat & _FDB_STAT_KEYED)
```

## 152 directory search functions

---

```
        strcpy(type, "(keyed)");
    else if (fdb->filestat & _FDB_STAT_INDEXED)
        strcpy(type, "(indexed)");
    else if (fdb->filestat & _FDB_STAT_RANDOM)
        strcpy(type, "(random)");
    else if (fdb->filestat & _FDB_STAT_PROGRAM)
        strcpy(type, "(program)");
    else
        strcpy(type, "");
    printf("%s %s\n",filename, type);
}
dirclose();
}
```

---

### Output:

>testdir

```
DOS\dos/AUTOEXEC.01_:S (stream)
DOS\dos/AUTOEXEC.02_:S (stream)
DOS\dos/AUTOEXEC.04_:S (stream)
DOS\dos/AUTOEXEC.06_:S (stream)
DOS\dos/AUTOEXEC.BAT:S (stream)
DOS\dos/COMMAND.COM:S (stream)
DOS\dos/CONFIG.01_:S (stream)
DOS\dos/CONFIG.02_:S (stream)
DOS\dos/CONFIG.04_:S (stream)
DOS\dos/CONFIG.06_:S (stream)
DOS\dos/IO.SYS:S (stream)
DOS\dos/MSDOS.SYS:S (stream)
DOS\dos/DOS.:S (subdirectory)
DOS\dos/RECEIVE.:S (subdirectory)
DOS\dos/THEOS.:S (subdirectory)
```

>

## ***\_dir\_mount***

Flush the memory cache for a disk drive in preparation for changing the disk volume.

```
#include <sc.h>
void _dir_mount ( DISK_UCB * ucb )
```

---

*ucb*                      »    pointer to disk unit control block

**Operation:**        The disk i/o buffer for the disk drive specified with *ucb* is flushed to the disk.

**Notes:**            The *\_dir\_mount* is used by the MOUNT command in preparation for a change in disk volume. This is done prior to a call to [\\_mount\\_fs](#), which mounts the new file system from the drive.

**Conforms to:**        *\_dir\_mount*    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [\\_mount\\_fs](#)

---

### **Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>

void
main()
{
    DISK_UCB *drive_A;

    drive_A = (DISK_UCB *) getucb(0); // get drive A ucb
    _dir_mount(drive_A);

    printf("\nPlace the diskette in drive A.");
    printf("\nPress RETURN when ready...");
    getchar();

    _mount_fs(drive_A);
}
```

### `_disk_capacity`

Get the current capacity and space availability of a disk drive.

```
#include <sc.h> or <disksize.h>
unsigned short _disk_capacity ( int lub, DISKSIZE _far * disksize )
```

---

<i>disksize</i>	»	pointer to storage for disk capacity values
<i>lub</i>	»	logical unit number of drive (0–25)

**Operation:** The disk volume label sector for the disk identified with *lub* is read and the current capacity and space availability for the drive are computed. The members of the *disksize* structure are filled in with this information.

**Returns:** A success/fail indicator. A zero value indicates success; a non-zero value indicates failure.

**Errors:** The most likely cause of a failure is that *lub* is not the lub for a disk device or that it is not currently attached.

**Notes:** The DISKSIZE structure is defined in the DISKSIZE.H file.

```
typedef struct DISKSIZE {
    short cylinders;    // nbr of cylinders
    short heads;        // nbr of heads per cylinder
    short sects;        // nbr of 256 byte sectors/track
    short den;          // density code
    unsigned long bytes; // total number of bytes
    unsigned long bytes_unused; // total unused bytes
    unsigned long bytes_contig; // largest contiguous area
    unsigned long bytes_bad;    // bad sector bytes
} DISKSIZE;
```

**Restrictions:** *lub* must refer to a disk lub and it must be attached.

This function may only be used in a program executing on a system using THEOS 32 Version 4 or later.

**Conforms to:** `_disk_capacity`    q ANSI    q DOS    n THEOS    q POSIX

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <disksize.h>

void
main()
{
    int lub;
    char search[27],
        *sptr;
    DISKSIZE diskspace;

    strcpy(search, getenv("SEARCH")); // get drive search sequence
    sptr = search;

    while (*sptr) {
        lub = *sptr++ - 65;
        if (disk_capacity(lub, &diskspace)==0)
            printf("Drive %c capacity = %,15li, available %,15li\n",
                lub+65, diskspace.bytes, diskspace.bytes_unused);
    }
}
```

---

**Output:**

```
>example
Drive S capacity =      209,715,200, available      101,659,392
Drive P capacity =       27,262,976, available       27,197,440

>
```

**div, ldiv**

These functions compute the quotient and remainder produced by the division of two integers or two longs.

```
#include <stdlib.h>
div_t div ( int numer, int denom )
ldiv_t ldiv ( long numer, long denom )
```

---

<i>denom</i>	»	value to divide by
<i>numer</i>	»	value to be divided

**Operation:** The *numer* is divided by *denom*, giving a quotient and a remainder. The sign of the quotient and the remainder generated will be the proper sign, mathematically.

**Returns:** The quotient and remainder is returned in type `div_t` (for `div`) or `ldiv_t` (for `ldiv`), which are declared as:

```
typedef struct {
    int quot;
    int rem;
} div_t;

typedef struct {
    long int quot;
    long int rem;
} ldiv_t;
```

**Errors:** No errors are detected or reported.

<b>Conforms to:</b>	<b>div</b>	n ANSI	q DOS	n THEOS	q POSIX
	<b>ldiv</b>	n ANSI	q DOS	n THEOS	q POSIX



**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main(void) {
    int        i,
               j;
    long       l,
               k;
    div_t      ans;
    ldiv_t     lans;

    i = 4;
    j = 3;
    ans = div(i,j);    // divide 4 by 3

    printf("/n%i divided by %i equals %i with remainder %i.\n",
           i, j, ans.quot, ans.rem);

    l = 25;
    k = 7;
    lans = ldiv(l, k); // divide 25 by 7

    printf("%li divided by %li equals %li %li/%li.\n",
           l, k, lans.quot, lans.rem, k);
}
```

---

**Output:**

>test

4 divided by 3 equals 1 with remainder 1.  
25 divided by 7 equals 3 4/7.

dup, dup2

These functions duplicate an open file handle or pointer.

```
#include <handle.h>
int dup ( int fhandle )
int dup2 ( int fhandle1, int fhandle2 )

#include <sc.h>
FILE * _dup ( FILE * file)
```

---

<i>fhandle</i>	»	file handle or number
<i>fhandle1</i>	»	file handle or number
<i>fhandle2</i>	»	file handle or number
<i>file</i>	»	pointer to an open file control block

Operation:	dup	A copy of the file control block (fcb) for the file indicated by <i>fhandle</i> is created.
	_dup	A copy of the file control block pointed to by <i>file</i> is created.
	dup2	Force <i>fhandle2</i> to be a file handle to the same file as <i>fhandle1</i> . If <i>fhandle2</i> was associated with an open file, that file is first closed.

Returns:	dup	A new file handle. A return of -1 indicates failure and <a href="#">errno</a> is set to either EBADF or EMFILE.
	_dup	A pointer to the duplicated fcb. A return of NULL indicates failure.
	dup2	An integer success code is returned: zero means success; EOF indicates failure and <a href="#">errno</a> is set to either EBADF or EMFILE.

**Notes:** The copied or duplicated file control block is an exact copy of the original. However, after the copy is made, the two control blocks are treated separately. For instance, reading some data with one file control block does not affect the file position pointer of the other.

**Restrictions:** These functions can only duplicate an open stream file.

Conforms to:	dup	q ANSI	n DOS	n THEOS	n POSIX
	_dup	q ANSI	q DOS	n THEOS	q POSIX
	dup2	q ANSI	n DOS	n THEOS	n POSIX

**See also:** [fileno](#)

**Example:**

```
#include <stdio.h>
#include <sc.h>

void
main() {
    FILE          *orig_stdout = 0;

    if (!iscon(stdout)) {           // stdout redirected?
        orig_stdout = _dup(stdout); // duplicate
        freopen(":con", "w", stdout); // redirect to default
        puts("stdout redirected back to console\n");
    }
    else orig_stdout = stdout;

    puts("This is output on stdout which is the console.\n");
    fputs("Normal output to stdout, possibly redirected.\n",
          orig_stdout);
}
```

---

**Output:**

```
>test
This is output on stdout which is the console.

Normal output to stdout, possibly redirected.

>test > my.file
stdout redirected back to console

This is output on stdout which is the console.

>list my.file

Normal output to stdout, possibly redirected.
```

**ecvt**

Convert a floating-point value to a string.

```
#include <stdlib.h>
char * ecvt ( double value, int count, int * dec, int * sign )
```

---

<i>count</i>	»	number of digits
<i>dec</i>	»	decimal-point position
<i>sign</i>	»	sign of converted number
<i>value</i>	»	number to convert

**Operation:** Convert *value*, keeping *count* number of significant digits. The decimal point is omitted from the resulting string and *dec* is set to its position from the left end of the string. *sign* is set to the sign of *value* (either 0 or -1).

**Returns:** A pointer to the converted string.

**Errors:** No error is detected.

**Notes:** Both *ecvt* and *fcvt* use a common, internal buffer for the converted string. Subsequent calls to either function destroy the results from any prior call to either of these functions.

**Restrictions:** *count* must be  $\geq 0$ .

**Conforms to:** **ecvt** q ANSI n DOS n THEOS n POSIX

**See also:** [fcvt](#), [ftoa](#), [gcvt](#), [sprintf](#)

**Example:**

```
#include <stdio.h>           // needed for printf function
#include <stdlib.h>          // needed for ecvt function
#include <string.h>          // needed for strcpy function

void
main() {
    double    fvalue = 1234.5678;
    int       dec,
             sign;
    char      buffer[20];

    printf("\nThe original value is %g.\n",fvalue);

    strcpy(buffer,ecvt(fvalue, 2, &dec, &sign));
    printf("\nThe ecvt string with 2 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,ecvt(fvalue, 5, &dec, &sign));
    printf("\nThe ecvt string with 5 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,ecvt(fvalue, 8, &dec, &sign));
    printf("\nThe ecvt string with 8 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,ecvt(fvalue, 10, &dec, &sign));
    printf("\nThe ecvt string with 10 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
}
```

---

**Output:**

>test

The original value is 1234.5678

The ecvt string with 2 digits is 12, dec = 2, sign = 0

The ecvt string with 5 digits is 123456780, dec = 4, sign = 0

The ecvt string with 8 digits is 123456780000, dec = 4, sign = 0

The ecvt string with 10 digits is 12345678000000, dec = 4, sign = 0

**eof**

Test an open file for end-of-file condition.

```
#include <io.h>
int eof ( int fhandle )



---


fhandle          »   file handle or number
```

**Operation:** The file's current position pointer is tested to determine if it is at end-of-file.

**Returns:** A true/false value indicating the end-of-file status for the file. A non-zero value indicates that it is at end-of-file.

**Errors:** When an error is detected, the return value is set to -1.

**Notes:** This function differs from `feof` in that it uses *fhandle* rather than a pointer to a file control block.

This function is implemented as a macro call to [feof](#).

**Conforms to:** `eof` q ANSI n DOS n THEOS n POSIX

**See also:** [clearerr](#), [feof](#), [ferror](#), [file\\_err](#), [fileno](#), [perror](#)

---

**Example:**

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

void
main()
{
    int    file_handle,
           len;
    char   buffer[100];

    file_handle = open("my.file", O_RDWR);
    if (file_handle != -1)
    {
        while (!eof(file_handle))
        {
            len = read(file_handle, buffer, 99);
            buffer[len] = 0;
            printf("%s", buffer);
        }
        close(file_handle);
    }
}
```

---

**erase**

Erases a file from disk. This function name is a synonym name to the true function named [remove](#).

```
#include <stdio.h>
unsigned short erase ( char * filename )
```

---

*filename*                    »    file description of file to erase

**Operation:**        The file specified by *filename* is erased: All disk space previously used by the file is returned to the disk's free space map. The directory entry for the file is marked as deleted.

**Returns:**            A success/fail indicator. A return value of zero indicates success; a non-zero return value indicates failure and is the reason code.

**Errors:**            When the return value is non-zero, *filename* is not erased. The return value is the code indicating the failure reason. Both [errno](#) and [\\_ernum](#) are set to this reason code and [\\_errarg](#) is set to *filename*.

**Notes:**            The contents of *filename* should be complete and explicit. If *filename* does not specify a path, then the current working directory is assumed. If *filename* does not specify a drive, then all drives specified in the current drive search sequence are used until the file is either found or all drives in the search sequence are examined. If *filename* does not specify a file type, then only the current default library is searched.

The `erase` function name is the THEOS name for the file erase operation and is provided for compatibility with prior versions of THEOS C. The [remove](#) function is the ANSI name for the operation. For portability purposes use the [remove](#) function name.

**Restrictions:**    You may not erase a file that is currently open by any program or task, including your own.

**Conforms to:**                `erase`    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [close](#), [remove](#), [unlink](#)

---

**Example:**            See example on page 533.

**\_errbot**

Display a system message at the bottom of the console screen.

```
#include <sc.h>
```

```
unsigned short _errbot ( int num, void * args, int c1, int c2, int c3, int c4 )
```

---

<i>args</i>	»	pointer to array of string pointers
<i>c1</i>	»	valid operator response character
<i>c2</i>	»	valid operator response character
<i>c3</i>	»	valid operator response character
<i>c4</i>	»	valid operator response character
<i>num</i>	»	number of message from SYSTEM.TEOS <i>nnn</i> .MESSAGE file

**Operation:** Displays a system message on the bottom line of the console display with parameter substitution and awaits a reply from the operator.

After the message displays, the cursor is turned on and the function waits for the operator to acknowledge the message using one of the characters specified by *c1*, *c2*, *c3* or *c4*. All input other than one of these characters is ignored. All four response codes must be specified. If you want to allow fewer than four response characters, use NULL or zero values for the unused codes.

After the operator responds to the message correctly, the message is erased from the screen, the cursor location and status are restored and the operator's response is returned as the value of the function.

**Returns:** The character value (*c1*, *c2*, *c3* or *c4*) used to respond to this message display is returned as the value of the function.

**Errors:** No errors are detected.

**Notes:** The message from the SYSTEM.TEOS*nnn*.MESSAGE*n* file is displayed using parameter substitution. Familiarize yourself with the message because the usage of the *args* argument depends upon the number and type of parameter substitution codes in the message text.

The usage of the *args* parameter and the rules for substitution are the same as described on page 498 in the description of the [putmsg](#) function.

This function uses the console display and keyboard, not stdin or stdout. Redirection of stdin or stdout does not affect the operation of this function.

**Conforms to:** **\_errbot**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [cputs](#), [errmsg](#), [fperror](#), [getch](#), [perror](#), [putmsg](#), [strerror](#), [syserr](#)



**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>

void
main()
{
    char  filename[256];

    printf("\nEnter name of file to erase: ");
    gets(filename);

    if (!access(filename, 0))
    {
        if ('Y' == errbot(48, filename, 'Y', 'N', 0, 0,))
            erase(filename);
    }
    else
        printf("File not found.");
}
```

---

**Output:**

>test

Enter name of file to erase: sample.file

>

The following appears on the bottom of the screen and is erased when you respond with a Y or an N.

Ok to erase "SAMPLE.FILE" (Yes,No,All) Y

**errmsg**

Display a system message on `stderr` with parameter substitution.

```
#include <stdio.h>
void errmsg ( int num, void * args )
```

---

<i>args</i>	»	pointer to array of string pointers
<i>num</i>	»	number of message from <code>SYSTEM.TEOSnnn.MESSAGES</code> file

**Operation:** The text of the message from the `SYSTEM.TEOSnnn.MESSAGES` file is written to `stderr`, using parameter substitution.

**Notes:** Familiarize yourself with the message text because the usage of the *args* argument depends upon the number and type of parameter substitution codes in the message text.

The usage of the *args* parameter and the rules for substitution are the same as described on page 498 in the description of the [putmsg](#) function.

This function operates similar to `errbot` with the following differences:

- ▶ Text is written to `stderr` with no cursor controls.
- ▶ No operator response is requested.
- ▶ The message is not erased.
- ▶ No value is returned.

**Conforms to:** `errmsg`    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [\\_errbot](#), [ferror](#), [file\\_err](#), [fperror](#), [perror](#), [putmsg](#), [strerror](#), [syserr](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    char  filename[256];

    printf("\nEnter name of file to erase: ");
    gets(filename);

    if (!access(filename, 0))
    {
        errmsg(48, filename);
        if (yesno())
        {
            remove(filename);
            errmsg(49, filename);
        }
    }
    else
        errmsg(19, filename);
}
```

---

**Output:**

>test

Enter name of file to erase: bad.filename  
File "bad.filename" not found.

>test

Enter name of file to erase: sample.file  
Ok to erase "sample.file" (Yes,No,All) Y  
"sample.file" erased.

>

### **errno, \_errno, \_errarg variables**

These variables are set by and used by many different functions to identify the specific error or type of error detected.

```
#include <errno.h>
extern int errno;

#include <stdio.h>
extern char * _errarg;
extern int _errno;
```

**Notes:** The **\_errno** and **errno** variables serve similar functions but they cannot be used interchangeably. Both variables are used to indicate the specific error that was detected by some function. However, some functions set **\_errno** and others set **errno**. In general, the **errno** function is used by ANSI conforming functions, and **\_errno** and **\_errarg** are used by THEOS-specific functions.

**\_errno** The **\_errno** and **\_errarg** variables are set by many functions, including:

access	fcreate	open	xopen
chdir	fgetpos	remove	
clearerr	fopen	rename	
diropen	ftell	rmdir	
fclose	getmsg	topen	
fcloseall	mkdir	unlink	

**errno** The **errno** variable is set by many functions, including:

access	fgetc	mkdir	spawnvp
acos	<a href="#">fgetpos</a>	open	sqrt
acsc	fputc	pow	stat
asec	<a href="#">fsetpos</a>	putfddate	unlink
asin	<a href="#">ftell</a>	remove	xopen
atan2	getcwd	rename	
atoi	getfddate	rmdir	
atol	lockf	signal	
chdir	locking	sin	
cos	log	spawnl	
cot	log2	spawnlp	
dup	log10	spawnv	

The `errno` variable may have one of the following values, depending upon the function detecting the error and the specific error detected.

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EACCES	18	Protected file
EAGAIN	1025	
EBADF	24	Invalid file description or file not open
EDEADLOCK	404	Record lockout
EDOM	426	Domain error
EEXIST	44	File already exists
EINVAL	428	Invalid argument
ENODEV	13	Disk not attached
ENOENT	19	File not found
ENOMEM	3	Insufficient memory
ENOSPC	42	Disk full
EMFILE	15	Too many open files
ERANGE	427	Range error

`errno` may have other, THEOS-specific error codes.

**Conforms to:**

<code>errno</code>	n ANSI	n DOS	n THEOS	n POSIX
<code>_errarg</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_errnum</code>	q ANSI	q DOS	n THEOS	q POSIX

**exec functions**

These functions start and transfer control to a new program.

```
#include <process.h>
void execl ( char * program, char * arg0, ... char * argn, NULL )
void execlp ( char * program, char * arg0, ... char * argn, NULL )
void execv ( char * program, char * arg[ ] )
void execvp ( char * program, char * arg[ ] )
```

---

<i>arg</i>	»	pointer to an array of character pointers of command-line arguments
<i>arg<sub>0</sub></i> , ... <i>arg<sub>n</sub></i>	»	list of character pointers to command-line arguments
<i>program</i>	»	character pointer to program name

**Operation:** Each of these functions loads the program indicated by *program* into memory and continues execution at the start of that program.

Files that are currently open are not closed and subtasks are not killed.

**execl** *program* must be a complete specification for the program file. It must specify a file-name, file-type and, if appropriate, member-name. The filedrive need not be specified if it can be found in the normal drive search sequence.

Command-line arguments to *program* are specified as a list of pointers, one to each argument in sequence. The list is terminated with a NULL pointer.

**execlp** *program* may be a complete specification or a simple program name. When *program* is a simple name, the normal search sequence for programs is used, first on the system drive and then on all other drives in the drive search sequence. The search sequence used is similar to that used by the CSI when searching for command names. It differs in that the synonym/abbreviation file is not used and no search is made for EXEC programs.

Similar to **execl**, command-line arguments to *program* are specified as a list of pointers, one to each argument in sequence. The list is terminated with a NULL pointer.

**execv** Similar to **execl**, *program* must be a complete specification for the program file. It must specify a file-name, file-type and, if appropriate, member-name. The filedrive need not be specified if it can be found in the normal drive search sequence.

Command-line arguments to *program* are specified as a pointer to an array of character pointers to each argument. The array is terminated with a NULL pointer.

**execvp** Similar to `execlp`, *program* may be a complete specification or a simple program name. When *program* is a simple name, the normal search sequence for programs is used, first on the system drive and then on all other drives in the drive search sequence.

Similar to `execv`, command-line arguments to *program* are specified as a pointer to an array of character pointers to each argument. The array is terminated with a NULL pointer.

Thus, `execl` and `execlp` are similar in that they both use a list of pointers to the command-line arguments; `execv` and `execvp` are alike in that they both use a pointer to an array of pointers to the command-line arguments; `execl` and `execv` both require an explicit specification of the program name; and `execlp` and `execvp` use the current command search sequence for the program name.

**Returns:** There is no return from these functions.

**Errors:** If there is an error encountered in locating *program*, or in loading and starting *program*, the appropriate error message is displayed on `stderr` and your program is exited.

**Notes:** The current program is exited during the execution of these functions but the `atexit` routines are not performed.

**Restrictions:** The list of command-line arguments or the array of command-line arguments must be terminated with a NULL pointer.

<b>Conforms to:</b>	<b>exec</b>	q ANSI	n DOS	n THEOS	n POSIX
	<b>execlp</b>	q ANSI	n DOS	n THEOS	n POSIX
	<b>execv</b>	q ANSI	n DOS	n THEOS	n POSIX
	<b>execvp</b>	q ANSI	n DOS	n THEOS	n POSIX

**See also:** [abort](#), [atexit](#), [\\_atexit\\_clear](#), [\\_atexit\\_remove](#), [exit](#), [spawn functions](#), [system](#)

---

#### Example:

```
#include <stdio.h>           // needed for printf
#include <stdlib.h>          // needed for exit
#include <process.h>         // needed for exec functions
#include <string.h>          // needed for strcmp

void
main(int argc, char *argv[])
{
    char          *args[20];

    args[0] = "zero";
    args[1] = "one";
    args[2] = "two";
    args[3] = "three";
    args[4] = NULL;

    if (argc<2)
    {
        // cmd-line arg passed?
        printf("Required parameter missing.\n");
        exit(256);
    }
}
```

```
    }

    if (strcmp(argv[1], "execl")==0)
        execl("c32.cmd386.testarg",args[0],args[1],NULL);
    if (strcmp(argv[1],"execlp")==0)
        execlp("testarg",args[0],args[1],args[2],NULL);
    if (strcmp(argv[1],"execv")==0)
        execv("c32.cmd386.testarg",args);
    if (strcmp(argv[1],"execvp")==0)
        execvp("testarg",args);
}
```

TESTARG.C:

```
#include <stdio.h>

void
main(int argc, char *argv[])
{
    int        i;

    printf("\nNumber of arguments passed = %d\n\n",argc);

    for (i=0; i<=argc; i++)
        printf("Argument[%d] passed = %s\n", argv[i]);
}
```

---

### Output:

These two programs show the differences in the calling sequence and results obtained when the four exec functions are used. Note in the resulting output that the TESTARG program always has its command-line argument[0] equal to its program name. The other arguments defined in the exec functions follow this initial argument.

```
>test
Required parameter missing.
```

```
>test "execl"
```

```
Number of arguments passed = 3
```

```
Argument[0] passed = C32.CMD386.TESTARG:S
Argument[1] passed = zero
Argument[2] passed = one
Argument[3] passed = (null)
```

```
>test "execlp"
```

```
Number of arguments passed = 4
```

```
Argument[0] passed = C32.CMD386.TESTARG:S
Argument[1] passed = zero
Argument[2] passed = one
Argument[3] passed = two
Argument[4] passed = (null)
```



## exit

Exit the program, setting the program return code.

```
#include <stdlib.h> or <process.h>
void exit ( int retcode )
```

---

*retcode*                    »    program return code

**Operation:**        The program is exited. Each of the routines registered with the [atexit](#) function is called in LIFO sequence. No arguments are given to those functions so only global variables and files are accessible to them.

After the last [atexit](#) routine is performed, [fcloseall](#) is performed to flush and close all open files. Control is returned to the host or calling environment for the program with the program's return code set to *retcode*.

**Returns:**            No return from this function: The program is exited.

**Defaults:**        An implied `exit(0)` is performed when a program's main function is exited or returned from.

When no functions are registered with the [atexit](#) function, the [fcloseall](#) function is called before exiting the program.

**Conforms to:**                `exit`    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [abort](#), [atexit](#), [\\_atexit\\_clear](#), [\\_atexit\\_remove](#), [exec functions](#), [fclose](#), [fcloseall](#), [spawn functions](#), [syserr](#), [system](#)

## 174 *exit*

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    if (argc==1)
    {
        puts("\nRequired parameter missing.");
        exit(253);
    }
    ...
}
```

---

### Output:

```
>set rdymsg on
```

```
RC = 0, 11:01:06, ET = 0:00, CPU = 0.020
```

```
>test
```

```
Required parameter missing.
```

```
RC = 253, 11:01:09, ET = 0:03, CPU = 0.320
```

```
>
```

## exp

The exp function calculates the exponential value.

```
#include <math.h>
double exp ( double value )
```

---

*value*                   »   floating-point value

- Operation:**       The natural logarithm base  $e$  is raised to the power of *value*.
- Returns:**        The value of  $e^{value}$ . HUGE\_VAL is returned on overflow; zero on underflow.
- Errors:**        When the result causes overflow, HUGE\_VAL is returned and [errno](#) is set to ERANGE. On underflow, zero is returned but [errno](#) is not set.
- Notes:**         The value of  $e$  can be easily found by using the function with the argument 1.
- Conforms to:**        **exp**    n ANSI    n DOS    n THEOS    n POSIX
- See also:**        [log](#), [log2](#), [log10](#)

---

### Example:

```
#include <stdlib.h>                   // needed for exit function
#include <stdio.h>                    // needed for printf function
#include <math.h>                     // needed for exp function

void
main(int argc, char *argv[]) {
    double        value;

    if (argc>1) {                     // cmd line argument?
        value = atof(argv[1]); // convert 1st cmdline argument
    }
    else {
        printf("Required value missing.\n");
        exit(255);
    }

    printf("The exponential of %g is %g.\n",value,exp(value));
}
```

---

### Output:

```
>test 1
The exponential of 1 is 2.718282.

>test 300.4
The exponential of 300.4 is 2.89775966977112192e+130.
```

**fabs**

Compute the absolute or unsigned value of a floating-point value.

```
#include <math.h>
double fabs ( double d )
_____
d                » floating-point value
```

**Operation:** The absolute or unsigned value of the floating-point argument *d* is computed and returned.

**Returns:** The floating-point absolute value of *d* is returned.

**Conforms to:** fabs n ANSI n DOS n THEOS n POSIX

**See also:** [abs](#), [cabs](#), [labs](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main(void) {
    double d;

    // Test abs
    printf("\nEnter a float: "); // get value from operator
    scanf("%f",&d);              // convert to numeric
    printf("The absolute value of %f is %f\n",d,fabs(d));
}
```

---

**Output:**

>example

```
Enter a float: -43.21
The absolute value of -43.21 is 43.21
```

## fbuf

Allocate an input/output buffer for a recently opened but unaccessed file.

```
#include <stdio.h>
void fbuf ( FILE * file, void * buffer, size_t len )
```

---

<i>buffer</i>	»	pointer to buffer
<i>file</i>	»	pointer to file's fcb
<i>len</i>	»	size of buffer to allocate

**Operation:** An input/output buffer of size *len* is assigned to or allocated for the open file specified by *file*.

When *buffer* is a NULL pointer, a buffer is allocated and assigned to *file*.

When *buffer* is not a NULL pointer, it is assumed to be a pointer to a buffer allocated by your program and that it is at least *len* bytes in size. In this situation, *buffer* is merely assigned to *file*.

**Notes:** When a file is used solely for input or solely for output, the access performance for the file can be significantly increased by using this function to allocate a large buffer (1K to 4K or larger). This reduces the number of disk accesses your program will require to read data from or write data to the file.

The [setbuf](#), [setvbuf](#) functions operate similar to this function but provide a little more control over the buffering operation.

**Defaults:** If the [fbuf](#), [setbuf](#) or [setvbuf](#) functions are not used before a file is accessed, a buffer size of 256 bytes will automatically be allocated.

**Restrictions:** An input/output buffer can only be allocated between the time that a file is opened and the first access to the file.

Once a buffer is allocated for an open file, it cannot be reallocated without first closing the file and reopening it.

**Conforms to:** [fbuf](#) q ANSI q DOS n THEOS q POSIX

**See also:** [setbuf](#), [setvbuf](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE * in;

    if (!(in=fopen("my.file", "r")))
        exit(ferror());
    fbuf(in, NULL, 4*1024); // allocate a 4K buffer
    ...
}
```

**fclose, fcloseall**

Close a specific open file or all open files.

```
#include <stdio.h>
int fclose ( FILE * file )
int fcloseall ( void )
```

---

*file*                      »    pointer to file's fcb

- Operation:**        **fclose**        *file* is flushed and closed.
- fcloseall**    All open files are flushed and closed.
- Returns:**           **fclose**        A zero is returned when *file* is closed successfully. An EOF is returned if *file* cannot be closed.
- fcloseall**    The number of files successfully closed. An EOF is returned if any open file cannot be closed.
- Errors:**            When a file cannot be closed, the `_errno` variable is set to indicate the specific reason why it cannot be closed.
- Notes:**            Closing a file that was opened as a “temporary file” causes the file to be erased.
- Unless a file's growth factor has been set, closing a file whose allocation was extended will have the unused portion of the additional allocation trimmed and deallocated.
- The input/output buffers that were automatically allocated for each file closed are freed. This includes buffers allocated by the `fbuf`, `setbuf` and `setvbuf` functions. Buffers allocated by your program and used by the `fbuf`, `setbuf` and `setvbuf` functions are not freed by these functions.
- Defaults:**        The `fcloseall` function is called when a program is exited.
- Conforms to:**        **fclose**        n ANSI        n DOS        n THEOS        n POSIX
- fcloseall**    q ANSI        n DOS        n THEOS        q POSIX
- See also:**           [close](#), [exit](#), [fbuf](#), [InitFileClose](#), [keyclose](#), [msgclose](#), [setbuf](#), [setvbuf](#), [tempnam](#), [tmpfile](#), [tmpnam](#), [\\_tmpnam\\_drive](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE *    in,
             out;

    if (!(in = fopen("input.file", "r")))
        exit(fperror());
    if (!(out = fopen("some.output", "w")))
        exit(fperror());

    ...

    fcloseall();           // close all files
}
```

**fcntl**

This function provides access to the information in a file's File Control Block (FCB).

```
#include <stdio.h>
unsigned long fcntl ( FILE * file, int type, size_t value )
```

---

<i>file</i>	»	pointer to file's fcb
<i>type</i>	»	coded type of access request
<i>value</i>	»	data for writes

**Operation:** The file control block for the file specified by *file* is accessed according to *type* and *value*. The *type* argument is a coded parameter specifying both the access type (read or write) and the field of the fcb to access. The *value* argument is used when the type is a write access.

**Returns:** When *type* specifies a read access, the return value is the value read from the fcb. When *type* specifies a write access, the return value is undefined.

**Errors:** When *file* is not open, *fcntl* always returns a zero.

**Notes:** The *type* argument specifies the type of access (read or write) and identifies the specific element of the fcb to read or write:

Name	Meaning	Offset	read	write
_fst	File status	0x00	1	0
_flub	LUB, 255 = all	0x01	5	4
_faccess	Access mode	0x02	11	10
_freclen	Record length	0x04	19	18
_fkeylen	Key length	0x06	27	26
_fbufbase	Buffer location	0x08	801	800
_fbufsize	Buffer size	0x0C	817	816
_fbufptr	Next byte position	0x10	833	832
_fcount	Number of bytes left	0x14	849	848
_floc	Location of next read/write	0x18	865	864
_fsa	FSA slot number	0x1C	371	n/a
_fnxtkey	IX key storage location	0x1E	889	888
_fnxt	Type of IX key	0x22	395	394
_fixnxt	Disk address of IX key	0x24	913	912
_ftapefn	Tape file name location	0x28	929	928

Refer to the FCB.H file for descriptions of the meanings of the fields *\_fst* and *\_faccess*.

Care must be taken when using this function to change information in a file's fcb. For instance, when there are records in a file and *fcntl* is used to change the record length or key length, followed by a write to the file, portions of the file will become meaningless.



**Restrictions:** *file* must be a pointer to an open file.

**Conforms to:** **fcntl** q ANSI q DOS n THEOS q POSIX

**See also:** [feof](#), [fgetpos](#), [fopen](#), [fsetpos](#), [ftell](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE *    file;

    if (!file = fopen("my.file", "irl"))
        exit(f perror());
    printf("\nAccess ..... x%04x", fcntl(file, 11, 0));
    printf("\nRecord length . %d", fcntl(file, 19, 0));
    printf("\nKey length .... %d", fcntl(file, 27, 0));

    fclose(file);
}
```

---

**Output:**

```
>test
Access ..... 0x0045
Record length . 22
Key length .... 10
>
```

**fcreate**

Create a new direct, indexed or keyed access file.

```
#include <stdio.h>
FILE * fcreate ( char * fname, char * mode, int reclen, int keylen, long size )
```

---

<i>fname</i>	»	pointer to complete file name
<i>keylen</i>	»	allocated key length
<i>mode</i>	»	file organization mode
<i>reclen</i>	»	allocated record length
<i>size</i>	»	number of records desired.

**Operation:** A new random-access file is created and opened.

**Returns:** A pointer to the opened file's file control block. A zero or NULL return value indicates that an error was detected during the file creation.

**Errors:** There are many errors that might be detected by these functions. [\\_errno](#) is set to reflect the specific cause of the error. Use the [ferror](#) function to display the message associated with the error.

**Notes:** The *mode* field is coded with one of the following characters:

<i>mode</i>	<i>Meaning</i>
i	Create indexed file
k	Create keyed file
d	Create direct file

The mode must also specify the type of open mode used on the file. Refer to [fopen](#) function description on page 222. You may use any of the codes described there except "b," "t" and "p."

The *keylen* field is used for indexed and keyed files only; it is ignored for direct files.

The *reclen* value is exclusive of the *keylen* value. That is, the total allocated file record length will be *reclen+keylen* plus any overhead associated with the file structure.

The *size* value specifies the minimum number of records that must be allocated to the file. The actual size of the file created may be larger.

**Defaults:** When the file is created it has the following default permissions: shared read and write-protected, owner-execute-protected and not hidden. On THEOS 32 Version 4 systems, this default can be changed by defining an environment variable named CREATE whose value is the protection codes desired. Refer to the CREATE environment variable description in the *THEOS System Reference* manual.

The file created is always owned by the current account id.

Newly created indexed, keyed and direct files are filled with binary zeros.

**Restrictions:** The *reclen* field must be a positive value less than or equal to 65,535; the *keylen* field must be a positive value less than or equal to 128.

This function cannot create directories or libraries. Use the [makelib](#), [\\_makelib](#), [\\_mklib](#) or [mkdir](#) functions instead.

**Conforms to:** **fcreate** q ANSI q DOS n THEOS q POSIX

**See also:** [creat](#), [fopen](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
main(int argc, char **argv)
{
    FILE *f;
    char mode[3], key[80], rec[80];

    if (argc != 4) {
        puts("Syntax: MAKEFILE <fn> <type> <reclen> <keylen>");
        exit(1);
    }

    strcpy(mode, argv[2]);
    strcat(mode, "w");
    strlwr(mode);

    if (!(f=fcreate(argv[1],mode,atol(argv[3]),atol(argv[4]),0)))
        exit(fperror());

    while (1) {
        printf("key: ");
        gets(key);
        if (!(key[0]))
            break;
        printf("Rec: ");
        gets(rec);
        writek(f, key, rec);
    }
    fclose(f);
}
```

**fcvt**

Convert a floating-point value to a string.

```
#include <stdlib.h>
char * fcvt ( double value, int dcount, int * dec, int * sign )
```

---

<i>dcount</i>	»	number of digits after decimal point
<i>dec</i>	»	decimal-point position
<i>sign</i>	»	sign of converted number
<i>value</i>	»	number to convert

**Operation:** Convert *value*, keeping *dcount* number of digits to the right of the decimal point. The decimal point is omitted from the resulting string and *dec* is set to its position from the left end of the string. *sign* is set to the sign of *value* (either 0 or -1).

**Returns:** A pointer to the converted string.

**Errors:** No error is detected.

**Notes:** Both [ecvt](#) and [fcvt](#) use a common, internal buffer for the converted string. Subsequent calls to either function destroy the results from any prior call to either of these functions.

**Restrictions:** *count* must be  $\geq 0$ .

**Conforms to:** **fcvt**    q ANSI    n DOS    n THEOS    n POSIX

**See also:** [ecvt](#), [ftoa](#), [gcvt](#), [sprintf](#)

**Example:**

```
#include <stdio.h>           // needed for printf function
#include <stdlib.h>          // needed for fcvt function
#include <string.h>          // needed for strcpy function

void
main() {
    double    fvalue = 1234.5678;
    int       dec,
             sign;
    char      buffer[20];

    printf("\nThe original value is %g.\n",fvalue);

    strcpy(buffer,fcvt(fvalue, 2, &dec, &sign));
    printf("\nThe fcvt string with 2 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,fcvt(fvalue, 5, &dec, &sign));
    printf("\nThe fcvt string with 5 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,fcvt(fvalue, 8, &dec, &sign));
    printf("\nThe fcvt string with 8 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
    strcpy(buffer,fcvt(fvalue, 10, &dec, &sign));
    printf("\nThe fcvt string with 10 digits is %s, dec = %i,
           sign = %i\n",buffer,dec,sign);
}
```

---

**Output:**

>test

The original value is 1234.5678

The fcvt string with 2 digits is 123457, dec = 4, sign = 0

The fcvt string with 5 digits is 123456780, dec = 4, sign = 0

The fcvt string with 8 digits is 123456780000, dec = 4, sign = 0

The fcvt string with 10 digits is 12345678000000, dec = 4, sign = 0

**FD\_CLR, FD\_ISSET, FD\_SET, FD\_ZERO**

These macros maintain socket set arrays used by the [select](#) function.

```
#include <socket.h>

void FD_CLR ( SOCKET s, fd_set * set )
int  FD_ISSET ( SOCKET s, fd_set * set )
void FD_SET ( SOCKET s, fd_set * set )
void FD_ZERO ( fd_set * set )
```

---

*s*                               »   socket number

*set*                            »   pointer to array of socket numbers in a special structure

- Operation:

FD\_CLR

Removes the socket number *s* from the array *set*.

FD\_ISSET

Tests the array *set* for the existence of socket number *s*.

FD\_SET

Adds the socket number *s* to the array *set*.

FD\_ZERO

Clears all socket numbers from the array *set*.
- Returns:

FD\_ISSET

A true/false value. A non-zero return value indicates that *s* is a member of the array *set*; a zero return value indicates that it is not.
- Notes:

These functions are implemented as macros and are intended to be used to maintain the parameters used in the [select](#) function.
- Conforms to:

FD_CLR	q ANSI	q DOS	n THEOS	q POSIX
FD_ISSET	q ANSI	q DOS	n THEOS	q POSIX
FD_SET	q ANSI	q DOS	n THEOS	q POSIX
FD_ZERO	q ANSI	q DOS	n THEOS	q POSIX
- See also:

[select](#)

## fdopen

Get a pointer to a file FCB for a file opened with a handle.

```
#include <stdio.h>
FILE * fdopen ( int handle, char * type )
```

---

<i>handle</i>	»	file's file number
<i>type</i>	»	coded type of open

**Operation:** The file pointer associated with *handle* is returned.

**Returns:** The file pointer associated with *handle*.

**Errors:** If *handle* is not the handle for an open file, NULL is returned.

**Notes:** Although the *type* parameter is ignored, portability considerations and good programming practice dictate that the code used should specify the same access type as used when the file was originally opened.

**Restrictions:** When used on a system with a version of THEOS prior to Version 4, fdopen only allows 50 files to be open at any one time. Use the xfdopen function name (see extended file functions on page 749) if your program needs more than 50 files open. When fdopen is used on a system with THEOS 32 Version 4 and above, it can have as many as 1000 files at one time.

You should not mix usage of fdopen and xfdopen in the same program.

**Conforms to:**                    **fdopen**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**                    [creat](#), [open](#), [pipe](#)

**feof**

Test a file's current position pointer for end-of-file condition.

```
#include <stdio.h>
int feof ( FILE * file )
```

---

*file*                      »    pointer to file's fcb

**Operation:**            The file's current position pointer is tested to determine if it is at the end of the file.

**Returns:**             A true/false value indicating the end-of-file status for the file. A non-zero value indicates that it is at end-of-file.

**Conforms to:**                **feof**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                [clearerr](#), [eof](#), [ferror](#), [file\\_err](#), [perror](#)

---

**Example:**

```
#include <stdio.h>

void
main()
{
    FILE * fcb,
    int len;
    char buffer[100];

    if (fcb = fopen("my.file", "rw"))
    {
        while (!feof(fcb))
        {
            len = fread(buffer, 99, fcb);
            buffer[len] = 0;
            printf("%s", buffer);
        }
        fclose(fcb);
    }
}
```



## **fferror**

Test a file's error status.

```
#include <stdio.h>
int fferror ( FILE * file )
```

---

*file*                      »    pointer to file's fcb

**Operation:**        The file's error status indicator is returned.

**Returns:**           A non-zero return value indicates that the file is in an error condition; a zero return value indicates that no error condition is set.

**Notes:**            File errors are set by the various functions that access files. This error status is returned by this function. A file's error status is not reset or cleared until the file is closed, rewound, or until the [clearerr](#) function is used on the file.

**Conforms to:**                **fferror**    n ANSI       q DOS       n THEOS       q POSIX

**See also:**            [clearerr](#), [eof](#), [\\_errbot](#), [errmsg](#), [feof](#), [file\\_err](#), [fperror](#), [getch](#), [perror](#), [strerror](#), [syserr](#)

---

**Example:**            See example for [clearerr](#) function.

**fgetc, fgetl, fgets, fgetsn, fgetw, getc, getchar**

Read a byte, character, integer, word or string of characters from an open file.

```
#include <stdio.h>
int fgetc ( FILE * file )
int fgetl ( FILE * file )
char * fgets ( char * buffer, size_t len, FILE * file )
char * fgetsn ( void * buffer, size_t len, FILE * file )
unsigned fgetw ( FILE * file )
int getc ( FILE * file )
int getchar ( void )
```

---

<i>buffer</i>	»	pointer to storage location
<i>len</i>	»	number of bytes to read
<i>file</i>	»	pointer to file's fcb

**Operation:** All of these functions operate on open files using the current position pointer. After reading the byte, character, word or string of bytes, the position pointer is advanced by the number of bytes read.

**fgetc** Reads one character from *file*. If *file* is the console input device (or stdin mapped to the console input device), the [getc](#) function is used to read the character.

When no character is available from file and the file is not at EOF, processing is suspended until one becomes available or until an interrupt signal is received.

**fgetl** Reads a long integer from *file*.

No alignment of data is presumed. That is, from the current file position, exactly `sizeof long int` bytes of data are read in and returned.

Note that a return of EOF might not indicate end-of-file or an error because EOF is also a legitimate value for a binary integer. The [feof](#) function must be used to determine if an error has actually occurred.

**fgets** Reads a string of characters from *file*. If *file* is the console input device (or stdin mapped to the console input device), the [cgets](#) function is used to read the character. This means that the editing characters interpreted by [cgets](#) are acted upon in this situation.

The line or string of characters from *file* is read and stored in *buffer*. The string accepted consists of all of the characters up to and including the first new-line character, up to the end of the file, or until *len*-1 characters are read, whichever occurs first. The string terminating character '\0' is appended to the end of the characters accepted.

**fgetsn** Reads a string of characters from *file*. If *file* is the console input device (or stdin mapped to the console input device), the

[cgets](#) function is used to read the character. This means that the editing characters interpreted by [cgets](#) are acted upon in this situation.

The line or string of characters from *file* is read and stored in *buffer*. The string accepted consists of all of the characters up to the end of the file, or until *len*-1 characters are read, whichever occurs first. The string terminating character ‘\0’ is NOT appended to the end of the characters accepted. (Compare with [fgets](#) above.)

**fgetw** Reads a short or two-byte integer value from *file*.

No alignment of data is presumed. That is, from the current file position exactly two bytes of data are read and returned.

**getc** Synonym name to the [fgetc](#) function, described above.

**getchar** Using the [fgetc](#) function with the *file* argument set to `stdin`, gets a character from the standard input device.

**Returns:**

<b>fgetc</b>	Returns the character read from <i>file</i> . Return of EOF indicates end-of-file read.
<b>fgetl</b>	Returns the long integer read from <i>file</i> .
<b>fgets</b>	Returns a pointer to the character string read from <i>file</i> . Return of NULL indicates end-of-file read.
<b>fgetsn</b>	Returns a pointer to the data string read from <i>file</i> . Return of NULL indicates end-of-file read.
<b>fgetw</b>	Returns the two-byte value read from <i>file</i> .
<b>getc</b>	Returns the character read from <i>file</i> . Return of EOF indicates end-of-file read.
<b>getchar</b>	Returns the character read from <code>stdin</code> . Return of EOF indicates end-of-file read.

A return of -1 indicates that the read was unsuccessful because the location is locked by another user and the lock wait time elapsed (see below).

**Errors:** Use the [feof](#) function to determine whether the end-of-file was encountered during the read.

**Notes:** If *file* has been opened with “r+” access, these functions check to see if the requested location in the file is locked by another user with the [filelock](#) function. If it is locked, the function waits for the lock to be released before reading the data. These functions do not check for locks by other users placed with automatic record-locking or the [relock](#) function.

If the read is not immediately successful because the location is locked by another user, these functions may wait for awhile to see if the lock is

cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in `scr.lock_wait` can be examined with the [\\_peekscrub](#), [\\_peekscrd](#), [\\_peekscrw](#) functions or changed with the [\\_pokescrb](#), [\\_pokescrd](#), [\\_pokescrw](#) functions.

When a read attempt fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`. This item can be examined with the [\\_peekscrub](#), [\\_peekscrd](#), [\\_peekscrw](#) functions.

If *file* was not opened with “r+” access, these functions do not check for locks by other users.

<b>Conforms to:</b>	<b>fgetc</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>fgetl</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fgets</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>fgetsn</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fgetw</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>getc</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>getcar</b>	n ANSI	n DOS	n THEOS	n POSIX

**See also:** [cgets](#), [getch](#), [getw](#), [feof](#), [ferror](#), [fread](#), [fscanf](#), [read](#)

---

### Example:

This example read data from a file that was created in the example for the [fputc](#), [fputl](#), [fputs](#), [fgetsn](#), [fputsnl](#), [fputw](#), [putc](#), [putchar](#) function shown on page 236.

```
#include <stdio.h>
#include <conio.h>

void main()
{
    FILE *    datafile;
    short int  i;
    long int   l;
    char       string1[256],
               string2[256];
    int        rec = 0;
    char       type;

    datafile = fopen("data.file","r");

    while((type=fgetc(datafile))!=EOF)
    {
        ++rec;
        string1[0] = 0;
        string2[0] = 0;
```

```
        c = l = i = 0;
        switch (type)
        {
            case 'A':
                fgets(string1, 256, datafile);
                break;
            case 'D':
                fgets(string1, 256, datafile);
                l = fgetl(datafile);
                break;
            case 'S':
                fgets(string1, 256, datafile);
                i = fgetw(datafile);
                fgets(string2, 256, datafile);
                break;
        }
        cprintf("Record %i\n", rec);
        cprintf("\tType      \ %c\n", type);
        cprintf("\tString 1   = %s\n", string1);
        cprintf("\tString 2   = %s\n", string2);
        cprintf("\tShort Int  = %hi\n", i);
        cprintf("\tLong Int   = %li\n", l);
    }
    fclose(datafile);
}
```

---

**Output:**

```
Record 1
    Type      A
    String 1   = Sample record
    String 2   =
    Short Int  = 0
    Long Int   = 0
Record 2
    Type      D
    String 1   = Another sample
    String 2   =
    Short Int  = 0
    Long Int   = 123456
Record 3
    Type      S
    String 1   = Yet another sample
    String 2   = End
    Short Int  = 25
    Long Int   = 0
```

>

## fgetpos

Get a file's current position pointer.

```
#include <stdio.h>
int fgetpos ( FILE * file, fpos_t * position )
```

---

*file* » pointer to file's fcb  
*position* » storage location for position pointer

**Operation:** The current input/output position pointer for *file* is retrieved and copied to the location *position*.

**Returns:** When an error is detected the return value is set to -1 and the [errno](#) variable is set to indicate the specific error.

**Errors:** When the return value is not zero, [errno](#) may have one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EBADF	24	File not open
EINVAL	428	The value of <i>file</i> is not a pointer to a file's fcb or a valid file handle.

**Notes:** This function should be used on stream files only.

**Conforms to:** **fgetpos** n ANSI n DOS n THEOS q POSIX

**See also:** [fsetpos](#), [ftell](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE *    out;
    fpos_t    len;

    if (!(out = fopen("some.text", "wa")))
        exit(ferror());

    fgetpos(out, &len);
    printf("There are %ld characters in the file.\n", len);
    ...
}
```

---

### Output:

>test

There are 143 characters in the file.

>

## fgrow

Sets the automatic growth factor for a file that is used when the file becomes full.

```
#include <stdio.h>
```

```
int fgrow ( FILE * file, int growth )
```

---

```
growth          » growth percentage divided by 10
```

```
file            » pointer to open file control block
```

**Operation:** The growth factor setting for the open *file* is set to *growth*.

**Returns:** Success code: Zero means success, non-zero is failure.

**Errors:** A non-zero return code indicates that the growth factor was invalid or that *file* was not an open disk file.

**Notes:** Although the growth parameter is expressed as an integer, it is coded and interpreted as a single byte composed of two hexadecimal nibbles, or half-bytes, such as 0x10 with the '1' and the '0' being the two nibbles.

The first nibble specifies the amount of growth for the file in multiples of 100%; the second nibble specifies the amount of growth in multiples of 10%. Only one of the two nibbles can be non-zero, with the first taking precedence. Thus you can specify that a file either grows in multiples of its current size or grows by a fraction of its current size. You cannot specify that a file grows by 150%.

A growth factor of zero specifies that the file will grow as required but will be trimmed, or chopped to its actual size when it is closed.

**Defaults:** The default growth factor for direct, indexed and keyed files is 30%; for stream files the default is 0%.

**Restrictions:** The maximum growth allowed is 1,500% expressed as 0xF0.

The *file* must be an open disk file.

**Conforms to:** **fgrow** q ANSI q DOS n THEOS q POSIX

**See also:** [creat](#), [fcreate](#), [\\_filechange](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>           // needed for exit function
#include <string.h>           // needed for strcpy

void
main(int argc, char *argv[]) {
    FILE *update;
    char filename[256];       // file name from cmd line
    int growth,               // growth factor from cmd line
        grow;                 // growth factor for fgrow

    if (argc<3) {              // not enough cmd line args?
        printf("File name and growth factor required.\n");
        exit(EXIT_FAILURE);
    }

    growth = atoi(argv[2]);    // get growth factor
    if (!growth) {             // invalid number
        printf("Growth factor invalid\n");
        exit(EXIT_FAILURE);
    }

    strcpy(filename,argv[1]);  // get filename spec
    if (!(update=fopen(filename,"dr"))) { // open file
        perror();              // can't--display error msg
        exit(EXIT_FAILURE);
    }

    if (growth>=100)           // growth is in 100s of % ?
        grow = (growth/100)<<4; // change to high nibble
    else grow = (growth/10);

    if (fgrow(update, grow)) { // growth factor changed?
        printf("Cannot change growth factor for \"%s\"\n",
            filename);
        fclose(update);        // close file
        exit(EXIT_FAILURE);
    }

    fclose(update);            // close file
}
```



## fifo functions

This group of functions maintains and accesses a fifo or first-in, first-out buffer.

```
#include <fifo.h>
void fifo_alloc ( FIFO _far * fifo, unsigned len )
void fifo_free ( FIFO _far * fifo )
unsigned fifo_get ( FIFO _far * fifo )
void fifo_put ( FIFO _far * fifo, char c )
unsigned fifo_rdy ( FIFO _far * fifo )
```

---

<i>c</i>	»	character to add
<i>fifo</i>	»	pointer to fifo structure and storage buffer
<i>len</i>	»	size of storage buffer

**Operation:** A fifo buffer is a “first-in, first-out” character buffer. The fifo structure and buffer must exist before these functions can be used to maintain or access the buffer.

<b>fifo_alloc</b>	Initializes the fifo buffer to zeros.
<b>fifo_free</b>	Doesn't do anything. This function is provided for consistency with the type-ahead buffer maintenance functions used by device-drivers.
<b>fifo_get</b>	Gets the next character available from the buffer. If no character is available, this function returns a zero.
<b>fifo_put</b>	Adds the character <i>c</i> to the buffer.
<b>fifo_rdy</b>	Returns the number of characters available in the buffer.

<b>Returns:</b>	<b>fifo_get</b>	The next character available from the buffer or a zero if the buffer is empty.
	<b>fifo_rdy</b>	The number of characters available in the buffer.

<b>Notes:</b>	<b>fifo_get</b>	Because a zero might be a valid character in the buffer, the program should use the <b>fifo_rdy</b> function prior to calling this function to test whether or not a character is available.
---------------	-----------------	--

**Restrictions:** The allocation of the fifo structure and the buffer must be done before any of these functions can be used. The buffer must be contiguous with the fifo structure.

If the fifo structure is declared as global or static, the buffer must follow the structure:

```
FIFO  my_fifo;
char  my_fifo_buffer[1024];
```

If the structure is declared as automatic, the buffer must preceed the structure:

```
void sample_function(void)
{
    FIFO    *my_fifo_buffer;
    ...
    my_fifo = malloc(sizeof(FIFO)+1024);
    ...
    fifo_alloc(my_fifo, 1024);
    ...
}
```

Conforms to:	<b>fifo_alloc</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fifo_free</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fifo_get</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fifo_put</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fifo_rdy</b>	q ANSI	q DOS	n THEOS	q POSIX

See also: [typeahead buffer functions](#)

## **\_file\_alloc**

Determines the allocated size of a file, in bytes.

```
#include <sc.h>
unsigned long _file_alloc ( const char far * filename )
```

---

*filename*                »    pointer to string containing file description

**Operation:**        The directory entry for the file specified by *filename* is examined and the total of the disk space allocated to the file is determined.

**Returns:**           The number of bytes allocated to *filename* on disk.

**Errors:**            If *filename* cannot be found, a zero is returned.

**Notes:**            The specification of the file in *filename* is assumed to be in the current working directory unless *filename* specifies the path to the file.

**Restrictions:**    This function can only be used when the operating system is THEOS 32 Version 4 or later.

**Conforms to:**        **\_file\_alloc**    q ANSI    q DOS    n THEOS    q POSIX

**\_filechange**

Changes a file's directory entry, either the date, ownership, protection codes or size.

```
#include <sc.h>
unsigned short _filechange ( const char _far * name, int type, size_t value )
```

---

<i>name</i>	»	name of file to change
<i>type</i>	»	coded type of change
<i>value</i>	»	change value

**Operation:** The file indicated by *name* is located and its directory entry is changed. The specific change made is indicated by *type* with the new value provided by *value*.

The *type* field is a one or two-character integer specifying the type of change:

<i>type</i>	Field changed	<i>value</i>	Content
'd'	Date and time	char *	Three-character, coded date and time (old format)
'da'	Date and time	char *	Four-character, coded date and time (new format)
'o'	Ownership	short int *	New owning account #
'p'	Protection codes	short int *	New protection codes
's'	File size	long int *	New size of file
'g'	Growth factor	unsigned int	New growth factor
'm'	Mode of organization	short int *	New mode
'ma'	Rec/key length	char *	New rec/key length

As indicated, *value* is a pointer to the new value for the change. Many of the values for *value* are obvious, except:

**'d'** The new date and time value is coded in the old date format. This is a 24-bit field with the following structure:

```
struct {
    year   : 4;    // years since 1980 [0-15]
              //           [1980 - 1995]
    month  : 4;    // month [1-12]
    day    : 5;    // day [1-31]
    hour   : 5;    // hour [0-23]
    min    : 6;    // minute [0-59]
};
```

Do not use this format for writing new dates for file. Instead, use the 'da' code described next.

**‘da’** The new date and time value is coded as a 32-bit field with the following structure:

```
struct {
    year   : 6;    // years since 1986 [0-63]
                //           [1986 - 2049]
    month  : 4;    // month [1-12]
    day    : 5;    // day [1-31]
    hour   : 5;    // hour [0-23]
    min    : 6;    // minute [0-59]
    sec    : 6;    // second [0-59]
};
```

**‘p’** The protection field is a coded 8-bit field with one of the two following structures, depending upon the operating system version:

Version 4:

```
struct {
    n_hidden: 1; // not hidden protection
    modified: 1; // modified status
    shar_w : 1; // shared-write protection
    shar_r : 1; // shared-read protection
    owner_e: 1; // owner-erase protection
    owner_x: 1; // owner-execute protection
    owner_w: 1; // owner-write protection
    owner_r: 1; // owner-read protection
};
```

Version 3.2:

```
struct {
    shar_e : 1; // shared-erase protection
    shar_x : 1; // shared-execute protection
    shar_w : 1; // shared-write protection
    shar_r : 1; // shared-read protection
    owner_e: 1; // owner-erase protection
    owner_x: 1; // owner-execute protection
    owner_w: 1; // owner-write protection
    owner_r: 1; // owner-read protection
};
```

**‘g’** A growth factor of zero means that the file is trimmed back to the actual size in use when closed. Non-zero growth factors specify the amount that the file is to grow when an attempt is made to write beyond the current end-of-file. The growth factor is specified as a percentage divided by 10. For instance, to indicate that a file is to double in size (100%) when it is extended, specify a growth of 10.

**‘m’** Changes to the organization mode should only be made with great care as the organization mode affects the type of record access that can be made on the file and, if the file already contains data and the mode is changed, the information will be interpreted very differently.

The value for the organization is an 8-bit field coded as:

fdb Mode	Code
<code>_FDB_STAT_EMPTY</code>	0x00
<code>_FDB_STAT_ERASED</code>	0xff
<code>_FDB_STAT_LIBRARY</code>	0x80
<code>_FDB_STAT_DIRECTORY</code>	0x40
<code>_FDB_STAT_STREAM</code>	0x10
<code>_FDB_STAT_RELATIVE</code>	0x08
<code>_FDB_STAT_KEYD</code>	0x04
<code>_FDB_STAT_INDEXED</code>	0x02
<code>_FDB_STAT_RANDOM</code>	0x0e
<code>_FDB_STAT_PROGRAM</code>	0x01
<code>_FDB_STAT_16_BIT_PROGRAM</code>	0x21
<code>_FDB_STAT_32_BIT_PROGRAM</code>	0x41

**Returns:** A success code is returned: zero means success; non-zero indicates failure.

**Errors:** Other than the normal file-access errors or file-not-found errors, the only unique type of error that might occur with this function pertains to a change of ownership. It is possible that a file with the same name already exists under the new owning account.

**Notes:** The header file `FDB.H` contains structure definitions and field name definitions that can be used to make your program code more readable. Refer to the example.

Care should be taken when changing the ownership of a file. The new owning account id code is not validated and, if it is a code for an account that does not exist, it might be very difficult to access the file again.

When changing the protection codes in a program that might be executing on systems with different versions of the operating system, use the [\\_getosver](#) function.

**Conforms to:** `_filechange`    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [fgrow](#)

## file\_err

Test a file's error status and exit when error exists.

```
#include <stdio.h>
void file_err ( FILE * file )
```

---

*file*                      »    pointer to file's fcb

**Operation:**        The error status of *file* is tested. If it is non-zero, the [syserr](#) function is used to display the appropriate message and exit the program.

**Notes:**            The error status of a file may be cleared by subsequent accesses to the file. Always check the return codes from file-access functions or use this function before a new operation is performed on the file.

**Defaults:**        When no error exists for the indicated file, execution of your program continues.

**Conforms to:**        **file\_err**    q ANSI        q DOS        n THEOS        q POSIX

**See also:**           [clearerr](#), [eof](#), [errmsg](#), [feof](#), [ferror](#), [fperror](#), [perror](#), [putmsg](#), [strerror](#), [syserr](#)

---

### Example:

```
#include <stdio.h>

void main() {
    FILE *output;
    char record[80];

    if (!(out = fopen("new.file","w")))          // open the file
        perror();                               // report any error
    while (1) {                                  // repeat till no entry
        putchar('*');                            // display prompt
        gets(record);                            // get string
        if (record[0]==0)                        // empty?
            break;                               // then done
        fprintf(output, "%s\n", record);         // write to file
        file_err(output);                       // exit if error
    }
    fclose(output);                             // close the file
}
```

**filelock**

Locks a portion of a file so that other users cannot access it while this program has it locked. This function can also unlock a file or test to see if a portion of a file is already locked by another user.

```
#include <stdio.h>
unsigned filelock ( FILE * file, int code, unsigned long from,
                    unsigned long to )
```

---

<i>code</i>	»	type of lock or unlock to be performed
<i>file</i>	»	pointer to open file's FCB
<i>from</i>	»	starting location in file
<i>to</i>	»	ending location in file

**Operation:** The operation of this function depends upon the value of the *code* argument.

**FILELOCK\_TEST** Test the region of *file* from *from* through *to* to see if any user has a lock placed on that portion of the file.

**FILELOCK\_LOCK** Lock the region of *file* from *from* through *to*.

**FILELOCK\_UNLOCK** Unlock the region of *file* from *from* through *to*. Only locks placed by this user are cleared.

**FILELOCK\_UNLOCK\_ALL** Unlock all regions in *file* locked by this process. In this situation, the values of *from* and *to* are not used (but they are required). Only locks placed by this user are cleared.

**Returns:** The return of this function depends upon the value of the *code* argument.

**FILELOCK\_TEST** Returns a true/false value. A zero is returned if no user has any of the bytes in the specified region locked. If any user including the calling program has a lock on any byte of the region specified, a non-zero value is returned.

**FILELOCK\_LOCK** Returns a true/false value. A zero return indicates that the region was locked successfully. A non-zero return value indicates that the lock failed.

A return value of 3 indicates that the lock failed because the lock table (FLT) used to record these locks is full. A return value of 404 indicates that one or more bytes of the region are already locked. (See notes, below).

**FILELOCK\_UNLOCK** Always returns a zero value.

**FILELOCK\_UNLOCK\_ALL** Always returns a zero value.

**Notes:** Locking, unlocking or testing for locks on a file does not cause the file to be read or written.

The *from* and *to* values may be outside of the current limits of the file, that is, beyond the current end of the file.



When testing or locking a region of a file when part of the region is already locked by a user, the function may wait for awhile to see if the lock is cleared by another user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is lockable (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the lock request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`.

The `filelock` function uses the FLT (File Lock Table) to record its locks. This is **not** the same table that is used by the automatic record-locking mechanism used for direct, indexed and keyed files, nor is it the same table used by the `reclock` function. However, the FLT is tested by all file input and output functions prior to accessing the file. Therefore, any lock placed on a portion of a file with the `filelock` function will prevent all other users from writing to that portion of the file. It will also prevent all other users from reading that portion of the file if they have the file open with “r+” access (update).

Locks on a file are removed with this function (`code=2` or `code=3`), with the `unlock` function or by closing the file.

The `filelock` function is used by the `lockf` and `locking` functions.

**Restrictions:** *file* must be a pointer to a disk stream file’s FCB. The *to* value must be greater than or equal to the *from* value.

**Conforms to:** `filelock` q ANSI q DOS n THEOS q POSIX

**See also:** `fclose`, `fcloseall`, `lockf`, `locking`, `reclock`, `recunlock`, `unlock`

**Example:**

```
#include <stdio.h>
#include <scr.h>
#include <timer.h>

void main()
{
    FILE *    datafile;
    int      loc;

    datafile = fopen("data.file","r"); // open file

    loc = 1024;                        // define location

    while (filelock(datafile, FILELOCK_LOCK, loc, loc+255)) // lock
    {
        printf("File in use by partition %i\n", Scr->lock_pid-1);
        printf("    Will retry in 5 seconds.\n");
        sleep(5000);
    }

    fseek(datafile, loc, SEEK_SET);    // position to location
    fgets(string, 256, datafile);      // read data

    fclose(datafile);                  // close file
}
```

## fileno

Determine the file handle of a file opened with a pointer to an FCB..

```
#include <stdio.h>
int fileno ( FILE * file )

_____
file          »   pointer to an open file
```

**Operation:** The number of the file control block (fcb) for *file* is determined.

**Returns:** The file handle for *file*.

**Errors:** If *file* is not an open file, then [errno](#) is set to EBADF and -1 is returned.

**Restrictions:** When used on a system with a version of THEOS prior to Version 4, `fileno` only operates correctly when *file* was opened with the [fopen](#) function, which allows 50 files to be open at any one time. Use the `xfileno` function name (see extended file functions on page 749) if your program used the `xopen` function.

You should not mix usage of `fileno` and `xfileno` in the same program.

**Conforms to:** `fileno`    q ANSI    n DOS    n THEOS    n POSIX

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main() {
    FILE      *in;

    if (!(in=fopen("TEST.FILE","r"))) {
        perror();
        exit(EXIT_FAILURE);
    }
    printf("File handle of \"%s\" is %d.\n","TEST.FILE",fileno(in));
    printf("File handle of stdin is %d.\n",fileno(stdin));
    printf("File handle of stdout is %d.\n",fileno(stdout));
    fclose(in);                      // close the file
}
```

### Output:

```
>test
File handle of "TEST.FILE" is 3.
File handle of stdin is 0.
File handle of stdout is 1.
```

### **find\_acc**

Find and read the ACB (Account Control Block) information for a given account.

```
#include <acb.h>
ACB find_acc ( char * acc_name )
```

---

*acc\_name*           »   pointer to account name

**Operation:**       The SYSTEM.TEOS32.ACCOUNT file is searched for the account definition of *acc\_name*. When it is found, an ACB structure is created and returned as the value of this function call.

**Returns:**         The ACB structure for *acc\_name*.  
  
When *acc\_name* is not found, a blank structure is returned.

**Notes:**           The value of *acc\_name* is case insensitive.

**Conforms to:**       **find\_acc**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**         [get\\_acc\\_name](#), [read\\_acc](#), [sch\\_acc](#)

**Example:**

```
#include <stdio.h>
#include <acb.h>
#include <string.h>

void print_info(ACB);

void main()
{
    ACB          account_info;

    account_info = find_acc("system");
    print_info(account_info);
    account_info = find_acc("badname");
    print_info(account_info);
}

void print_info(ACB info)
{
    char          name[9];          // work space for name string

    strmake(name, info.name, 8); // change to c-style string
    printf("\nAccount name..... %s\n",name);
    printf("Account id..... %hi\n",info.id);
    printf("Location..... %li\n",info.loc);
    printf("Mail?..... %c\n",info.mail ? 'Y' : 'N');
}
```

---

**Output:**

>test

```
Account name..... SYSTEM
Account id..... 0
Location..... 3500
Mail?..... N
```

```
Account name.....
Account id..... 0
Location..... 0
Mail?..... N
```

>

### find\_first, find\_next

These functions provide directory search capabilities.

```
#include <diskfind.h>

int find_first ( char * fname_mask, DISKFIND * diskfind )
int find_next ( DISKFIND * diskfind )
```

---

*diskfind*           »   pointer to directory search structure  
*fname\_mask*        »   pointer to string containing file name specification

**Operation:**       **find\_first**   The *diskfind* structure is initialized and the first directory entry matching the *fname\_mask* is located. The file description for that file is copied to the *filename* member of the structure and the fdb for the file is copied to the *retdir* member of the structure.

**find\_next**   The next directory entry is located that matches the *fname\_mask* specified when this *diskfind* structure was initialized with the function. The file description and the fdb for the directory entry are copied to the appropriate members in the structure.

**Returns:**           A zero is returned when a directory entry is found. A non-zero is returned when no more files match.

**Notes:**            These two functions operate similarly to the functions [diropen](#), [dirread](#) and [dirclose](#). However, these functions are more flexible and can search multiple paths without closing prior searches.

**Conforms to:**       **find\_first**   q ANSI       q DOS       n THEOS       q POSIX  
                      **find\_next**   q ANSI       q DOS       n THEOS       q POSIX

**See also:**           [directory search functions](#)

---

#### Example:

```
#include <stdio.h>
#include <diskfind.h>
#include <string.h>

void search(int, char *);

main(int argc, char *argv[])
{
    putchar('\n');

    search(0, argv[1]);
}

void
search(int level, char *arg)
{
    DISKFIND     directory;
    char         type[20];
    int          rc;
```

---

```

for (rc=find_first(arg, &directory);
    rc==0;
    rc=find_next(&directory))
{
    switch (directory.retdir.filestat)
    {
        case _FDB_STAT_DIRECTORY:
        {
            char    search_dir[256];

            printf("%s%s (%s)\n",4*level, "",
                directory.filename, "directory");
            strcat(strcpy(search_dir, directory.filename),
                "/*.*");
            search(++level, search_dir);
            --level;
            continue;
        }
        case _FDB_STAT_LIBRARY:
            strcpy(type, "library");
            break;
        case _FDB_STAT_STREAM:
            strcpy(type, "stream");
            break;
        case _FDB_STAT_RELATIVE:
            strcpy(type, "relative");
            break;
        case _FDB_STAT_KEYED:
            strcpy(type, "keyed");
            break;
        case _FDB_STAT_INDEXED:
            strcpy(type, "indexed");
            break;
        case _FDB_STAT_RANDOM:
            strcpy(type, "random");
            break;
        case _FDB_STAT_16_BIT_PROGRAM:
            strcpy(type, "16-bit program");
            break;
        case _FDB_STAT_32_BIT_PROGRAM:
            strcpy(type, "32-bit program");
            break;
        case _FDB_STAT_PROGRAM:
            strcpy(type, "program");
            break;
        default:
            strcpy(type, "");
    }
    printf("%s%s (%s)\n",4*level, "", directory.filename,
        type);
}
}

```

## 212 *find\_first, find\_next*

---

### Output:

```
>testdir dos\*.*
```

```
DOS\DOS.EXEC:S (stream)
```

```
DOS\DOS.BACKUP:S (stream)
```

```
DOS\DOS.DOSCFG:S (stream)
```

```
DOS\DOS.:S (directory)
```

```
    DOS\DOS./AUTOEXEC.01_:S (stream)
```

```
    DOS\DOS./AUTOEXEC.02_:S (stream)
```

```
    DOS\DOS./AUTOEXEC.04_:S (stream)
```

```
    DOS\DOS./AUTOEXEC.06_:S (stream)
```

```
    DOS\DOS./AUTOEXEC.BAT:S (stream)
```

```
...
```



## floor

Compute the largest whole number that is less than or equal to a given value.

```
#include <math.h>
double floor( double x )

_____
x                » floating-point value
```

**Operation:** The floor of  $x$  is computed and returned.

For instance, the `floor(3.5) == 3.0`; the `floor(-5.6) == -6.0`.

**Returns:** The floor of  $x$ .

**Restrictions:** The operation of this function is dependant upon the `#pragma float bcd` or `#pragma float ieee` directive currently in effect.

**Conforms to:** `floor` n ANSI n DOS n THEOS n POSIX

**See also:** [ceil](#), [fmod](#)

**flush, flushall, fflush**

Write any data remaining in a file's output buffer to the actual file.

```
#include <stdio.h>
int fflush ( FILE * file )
int flushall ( void )
```

```
#include <handle.h>
int flush ( int fhandle )
```

---

*fhandle*               »   file handle or number

*file*                 »   pointer to file's fcb

**Operation:**

<b>fflush</b>	Writes any data remaining in a file's output buffer to <i>file</i> .
<b>flush</b>	Writes any data remaining in a file's output buffer to <i>fhandle</i> , just like the fflush function above. This function differs from fflush in that it uses a file handle to specify the file, not a pointer to a file control block.
<b>flushall</b>	Writes any data remaining in any of the open files' output buffers to the files. Uses fflush.

**Returns:**

<b>fflush</b>	A zero is returned when the buffer is successfully flushed. A return of EOF indicates that an error occurred.
<b>flush</b>	A zero is returned when the buffer is successfully flushed. A return of EOF indicates that an error occurred.
<b>flushall</b>	The number of files open is returned.

**Notes:** Most stream files have input and output buffers associated with them. When data is written to a file, it is placed in the file's output buffer first. When the output buffer becomes full or when data is written to a different location in the file, it is then written to the file. This is called flushing.

These functions force the early flushing of the output buffer even though the buffer is not full. If the buffer is empty or there is no output buffer for the file, no action is taken.

A file is automatically flushed when it is closed, when a read operation is performed on the file, or when the file's position pointer is changed.

**Conforms to:**

<b>fflush</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>flush</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>flushall</b>	q ANSI	n DOS	n THEOS	q POSIX

**See also:** [close](#), [fclose](#), [fcloseall](#), [fseek](#), [lseek](#), [seek](#), [\\_seek](#), [setbuf](#), [setvbuf](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE *    output;
    char      ans;

    load_yn();

    if (!(output = fopen("some.data", "wa")))
        exit(fperror());

    setvbuf(output, NULL, _IOFBF, 4096); // large buffer

    ... // write some data to file

    fflush(output);                // flush before prompt
    printf("Ready to proceed...");
    ans = yesno();

    ...
}
```

**far memory functions**

This group of functions is similar to the memory functions described on page 433 except that these functions operate on far memory objects. They copy one area to another, initialize an area to a specific value, find the location of a value in an area or compare two memory areas.

```
#include <string.h> or <memory.h>
void _far * _fmemccpy ( void _far * buffer1, void _far * buffer2, char c,
                        size_t len )
void _far * _fmemchr ( void _far * buffer, char c, size_t len )
int _fmemcmp ( void _far * buffer1, void _far * buffer2, size_t len )
void _far * _fmemcpy ( void _far * buffer1, void _far * buffer2, size_t len )
int _fmemdcmp ( void _far * buffer1, void _far * buffer2, size_t len )
int _fmemeq ( void _far * buffer1, void _far * buffer2, size_t len )
int _fmemicmp ( void _far * buffer1, void _far * buffer2, size_t len )
int _fmemieq ( void _far * buffer1, void _far * buffer2, size_t len )
void _far * _fmemmove ( void _far * buffer1, void _far * buffer2, size_t len )
void _fmemrcpy ( void _far * buffer1, void _far * buffer2, size_t len )
void _far * _fmemset ( void _far * buffer, char val, size_t len )
```

---

<i>buffer</i>	»	pointer to far memory buffer
<i>buffer<sub>1</sub></i>	»	pointer to first far memory buffer
<i>buffer<sub>2</sub></i>	»	pointer to second far memory buffer
<i>c</i>	»	byte value of terminating byte in buffer
<i>len</i>	»	length of buffer
<i>val</i>	»	byte value to initialize memory

- Operation:**
- \_fmemccpy** Copies bytes from *buffer<sub>2</sub>* to *buffer<sub>1</sub>*. Bytes are copied until either *len* number of bytes are copied or until a byte with the value *c* is copied.
  - \_fmemchr** Locates the byte with value *c* in *buffer*. Only the first *len* bytes of *buffer* are examined.
  - \_fmemcmp** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>*.
  - \_fmemcpy** Copy *len* bytes from *buffer<sub>2</sub>* to *buffer<sub>1</sub>*.
  - \_fmemdcmp** This function is identical in operation to the **\_fmemcmp** function unless there are ASCII decimal digits in the buffers.

When *buffer<sub>1</sub>* and *buffer<sub>2</sub>* contain ASCII decimal digits that start in the same relative location, the two buffers will compare as if the portions containing the decimal digits were the same length, with zeros added to pad the shorter string of dig-

its to the same length as the other buffer's string of digits. For example:

Original set of strings:	Sorted with _fmemcmp:	Sorted with _fmemdcmp:
a1xxx	a101xxx	a1xxx
a83xxx	a11xxx	a11xxx
a12xxx	a12xxx	a12xxx
a11xxx	a1xxx	a83xxx
b1xxx	a83xxx	a101xxx
a101xxx	b100xxx	b1xxx
b15xxx	b15xxx	b15xxx
b100xxx	b1xxx	b100xxx

**\_fmemeq** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>* for equality.

**\_fmemicmp** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>*. This is a “case-insensitive” comparison. That is, if a position in one buffer contains a lowercase letter and the other buffer contains the uppercase form of the same letter, the comparison is determined to be equal.

**\_fmemieq** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>* for equality. This is a “case-insensitive” comparison.

**\_fmemmove** Copies the contents of *buffer<sub>2</sub>* to *buffer<sub>1</sub>* for *len* bytes. This function ensures that the copy will be accurate even when the two buffers overlap.

**\_fmemrcpy** Copies *len* bytes from *buffer<sub>2</sub>* to *buffer<sub>1</sub>*, in the reverse direction. That is, the last byte of *buffer<sub>2</sub>* is copied to the last position of *buffer<sub>1</sub>*, the next-to-the-last byte of *buffer<sub>2</sub>* is copied to the next-to-the-last position of *buffer<sub>1</sub>*, etc.

**\_fmemset** Copies the value *c* into the first *len* bytes of *buffer*.

**Returns:**

**\_fmemccpy** If a byte of value *c* is copied, a pointer to the location following *c* in *buffer<sub>1</sub>* is returned. Otherwise a NULL is returned.

**\_fmemchr** A pointer to the location of the first occurrence of *c* in *buffer*. If *c* cannot be located in the first *len* bytes of *buffer*, a NULL is returned.

**\_fmemcmp** A signed integer value.

Comparison	Return value
$buffer_1 < buffer_2$	< 0
$buffer_1 = buffer_2$	0
$buffer_1 > buffer_2$	> 0

**\_fmemcpy** A pointer to *buffer<sub>1</sub>*.

**\_fmemdcmp** A signed integer value, similar to **\_fmemcmp**.

**\_fmemeq** A non-zero value when the two buffers are equal for their first *len* bytes. A zero return value indicates inequality.

**\_fmemicmp** A signed, integer value.

Comparison	Return value
$buffer_1 < buffer_2$	< 0
$buffer_1 = buffer_2$	0
$buffer_1 > buffer_2$	> 0

A position is equal when the two buffers are equal or if they have the same letter independent of its case. When a position is unequal, the lowercase form of each position is compared. Thus, “a” is less than “b” and “B” even though the value of “a” is actually greater than the value of “B.”

**\_fmemieq** A non-zero value when the two buffers are equal for their first *len* bytes. A zero return value indicates inequality.

**\_fmemmove** A pointer to *buffer<sub>1</sub>*.

**\_fmemrcpy** A pointer to *buffer<sub>1</sub>*.

**\_fmemset** A pointer to *buffer<sub>2</sub>*.

### Notes:

**\_fmemicmp** This function uses the [tolower](#) function when determining if a position in the two buffers is the same letter with different cases.

**\_fmemmove** The functions `_fmemcpy` or `_fmemrcpy` are used to ensure that the *buffer<sub>2</sub>* is moved to *buffer<sub>1</sub>* even when the two buffers overlap.

### Restrictions:

**\_fmemccpy** If *buffer<sub>2</sub>* and *buffer<sub>1</sub>* overlap, the resulting buffer may be erroneous. When *buffer<sub>2</sub>* is greater than *buffer<sub>1</sub>*, the copy is performed correctly; when *buffer<sub>1</sub>* is greater than *buffer<sub>2</sub>*, the process of copying the bytes in *buffer<sub>2</sub>* may destroy subsequent bytes in *buffer<sub>2</sub>*. For example:

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
    buffer1 buffer2          len = 10, c = 'm'

```

Result: ghijklmnopklmnopqrstuvwxyz

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
    buffer2 buffer1          len = 10, c = 'm'

```

Result: abcdefabcdefabcdqrstuvwxyz

**\_fmemcpy** Same as `memcpy`.

**\_fmemrcpy** Because the copy operation is performed in the reverse direction, the restrictions of a `_fmemrcpy` are the opposite of a `_fmemcpy`.

If *buffer<sub>2</sub>* and *buffer<sub>1</sub>* overlap, the resulting buffer may be erroneous. When *buffer<sub>1</sub>* is greater than *buffer<sub>2</sub>*, the copy is performed correctly; when *buffer<sub>2</sub>* is greater than *buffer<sub>1</sub>*, the process of copying the bytes in *buffer<sub>1</sub>* may destroy subsequent bytes in *buffer<sub>1</sub>*.

<b>Conforms to:</b>	<b>_fmemccpy</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fmemchr</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fmemcmp</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fmemcpy</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fmemdcmp</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fmemeq</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fmemicmp</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fmemieq</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fmemmove</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fmemrcpy</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fmemset</b>	q ANSI	n DOS	n THEOS	q POSIX

**See also:** [far string functions](#), [memory functions](#), [string functions](#)

**fmod**

Calculate the floating-point remainder.

```
#include <math.h>
double fmod ( double x, double y )
```

---

*x*                   »   dividend  
*y*                   »   divisor

**Operation:**       The modulo or remainder function of *x* divided by *y* is computed. This is equivalent to:

$$x - ip\left(\frac{x}{y}\right) \bullet y$$

**Returns:**        The floating-point modulo of *x* divided by *y*.

**Notes:**         This function uses BCD or IEEE arithmetic, depending upon the #pragma float bcd or #pragma float ieee directive.

**Conforms to:**               **fmod**    n ANSI    n DOS    n THEOS    n POSIX

---

**Example:**

```
#include <stdio.h>
#include <math.h>

void
main()
{
    double       twenty,
               ten,
               five,
               one,
               quarter,
               dime,
               nickel,
               penny,
               amount;

    if (scanf("%f",&amount)) {
        change_maker(amount, &twenty, &ten, &five, &one,
                      &quarter, &dime, &nickel, &penny);
        printf("\nProper change for amount %,10.2f is:\n",amount);
        if (twenty)
            printf("\n%6.0f x $20",twenty);
        if (ten)
            printf("\n%6.0f x $10",ten);
        if (five)
            printf("\n%6.0f x $5",five);
        if (one)
            printf("\n%6.0f x $1",one);
        if (quarter)
            printf("\n%6.0f x 25¢",quarter);
        if (dime)
```



```
        printf("\n%6.0f x 10¢",dime);
    if (nickel)
        printf("\n%6.0f x 5¢",nickel);
    if (penny)
        printf("\n%6.0f x 1¢",penny);
    }
    else
        printf("\nInvalid amount entered.");
}

void
change_maker(double a, double *twenty, double *ten, double *five,
double *one, double *quarter, double *dime, double *nickel,
double *penny)
{
    *twenty = ip(a/20);
    a = fmod(a, 20.0);
    *ten = ip(a/10);
    a = fmod(a, 10.0);
    *five = ip(a/5);
    a = fmod(a, 5.0);
    *one = ip(a/1);
    a = fmod(a, 1.0);
    *quarter = ip(a/.25);
    a = fmod(a, .25);
    *dime = ip(a/.10);
    a = fmod(a, .10);
    *nickel = ip(a/.05);
    a = fmod(a, .05);
    *penny = a * 100.0;
}
```

**fopen**

Opens a file for access.

```
#include <stdio.h>
FILE * fopen ( char * filename, char * mode )
```

---

*filename*               »   pointer to string containing file description  
*mode*                    »   access mode

**Operation:**       The *filename* and *mode* are used in a request to the operating system to open a file named *filename* with access type *mode*.

**Returns:**         A pointer to the opened file's file control block (fcb). If the file cannot be opened, a NULL pointer is returned and [errno](#) is set to the error code describing the reason for the failure.

**Errors:**         When the file cannot be opened and a NULL pointer is returned, [errno](#) is set to one of the following codes:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EACCES	18	Protected file
EBADF	24	Invalid <i>filename</i>
ENODEV	13	Disk not attached
ENOENT	19	File not found
ENOMEM	3	Insufficient memory
ENOSPC	42	Disk full
EMFILE	15	Too many open files

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [fpererror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:**         The specification of the file in *filename* is assumed to be in the current working directory unless *filename* specifies the path to the file.

The character string *mode* may contain one or more of the following codes:

<b>code</b>	<b>Meaning</b>
One of the following codes <u>must</u> be used to specify the access type.	
r	Open an existing file for read access.
w	Open a new file for write access. If the file already exists, it is first erased (stream files).
a	Open a file for write access. All new records written to the file will be written at the current end-of-file. The file may, or may not, already exist.
r+	Opens an existing file for both read and write access.
w+	Opens a new file for both read and write access. If the file already exists, it is first erased (stream files).

code	Meaning
a+	Open a file for both read and write access. All new records written to the file will be written at the current end-of-file. The file may, or may not, already exist.
One of the following codes should be used to specify the organization and interpretation of the data in the file.	
b	File is a binary stream. This is the default mode when no organization code is used.
t	File is a text stream. File must have stream organization.
p	File is a compiled program.
d	Direct file organization. File must have direct organization.
k	Keyed file organization. File must have keyed organization.
i	Indexed file organization. File must have indexed organization.
One of the following codes may be used to specify that the file is locked to other users trying to open it. If none of these codes are used, the file is opened but not locked.	
l	Locked file access. While this file is open, no other users may open it.
lA	Locked file access, deny all opens. Same as “l” specification. While this file is open, no other users may open it.
lR	Locked file access, deny read opens. While this file is open, no other users may open it with read access. They may open it with write-only or append access.
lW	Locked file access, deny write opens. While this file is open, no other users may open it with write access. They may open it with read-only access.
The following code may be used if the file access and organization is d, k or i and access is r+ (update).	
m	<p>Use multi-record locking. Records are locked when they are read and remain locked until an unlock or fclose operation is performed on the file.</p> <p>When this code is not used, then only single record-locking is performed. A record is automatically locked when it is read and unlocked when it is written, deleted, another record is read or when an unlock or fclose operation is performed on the file.</p> <p>No record-locking is performed if the file is opened with read-only or write-only access (r or w).</p>

### Stream Files

When a file is opened with the “a” or “a+” access type, all write operations occur at the end of the file. Although the file pointer can be positioned using [fseek](#), [rewind](#), *etc.*, the file pointer is always moved back to the current end-of-file prior to any write operation. Existing data is never overwritten in this mode.

### Direct, Keyed and Indexed Files

Files opened with “d,” “k” or “i” access are accessed using the functions [deletk](#), [lreadk](#), [lreadn](#), [readk](#), [readn](#), [readp](#) and [writek](#). Files opened with “b,” “t” or “p” access are accessed using any of the file-access functions except those just listed.

### Named Pipes

A named pipe may be opened with this function. A named pipe is a special type of file that is used for interprocess communication. Unlike the pipes that are used in a multitask environment via the [pipe](#) or [popen](#) functions, a named pipe can be used between two tasks in separate environments or time. That is, the named pipe can transfer data between two separate user tasks or between two tasks that are executed at different times.

To use a **named pipe**, the application must decide upon the name of the pipe. All named pipes are named as if they were a file in the special directory /PIPE. The two programs that use a named pipe each open the pipe as a file. For example, the pipe name is “WORK.DATA.” The program that will write to the pipe opens it with the `fopen` function call:

```
fopen ( " /PIPE/WORK.DATA" , "w" );
```

and the program that will read from the pipe opens it with:

```
fopen ( " /PIPE/WORK.DATA" , "r" );
```

It doesn’t matter if the receiving program opens the pipe first—the program will wait for the pipe to be created by the originator.

A program writing lots of data to the pipe might be delayed by the receiving program if the receiving program doesn’t empty the pipe. Pipes have a rather large, but fixed buffer size. When the output buffer becomes full, the writing program’s execution pauses until the buffer is read and space becomes available.

This is normally not a consideration because pipes are not used to transfer large databases.

### **Automatic Record-Lock- ing:**

When a direct, indexed or keyed organization file is opened with the appropriate “d,” “i” or “k” access mode and “r+” access is requested, automatic record-locking is enabled for the file. **Automatic record-locking** means that every time that a record is successfully read from this file by this user, it is locked so that other users cannot access the record until this user releases the record lock.

If the “m” access is not specified for the file, the record lock is released automatically when the record is written, deleted or a different record is read, or the lock is removed with the [unlock](#) function or the file is closed. When the “m” access is specified, the record remains locked until all locks on the file are removed with the [unlock](#) function or the file is closed.

All of the direct, indexed and keyed record access functions test for record-locked condition prior to reading, writing or deleting a record. This includes the functions [deletk](#), [ldeletk](#), [lreadk](#), [lreadn](#), [lwritek](#), [readk](#), [readn](#), [readp](#) and [writek](#).

Automatic record-locking does not prevent another user from accessing the record if the user opens the file with read-only access or if the user opens the file as a stream file. The stream file input/output functions do not test the location being accessed for automatic record-lock condition by another user. For this reason, direct, indexed and keyed organization files opened with stream access should be opened with the "l" or "lW" access modifiers to prevent other users from writing to the file while the user is accessing the file as a stream of bytes rather than logical records.

**Defaults:** If the file organization is not specified as "d," "k" or "i," then the file is opened with stream organization.

When fopen must create the file, it sets the following default permissions: shared-read and Write protected, owner-execute protected and not hidden. This default can be changed by defining an environment variable named CREATE whose value is the protection codes desired. Refer to the CREATE environment variable description in the *THEOS System Reference* manual.

**Restrictions:** When used on a system with a version of THEOS prior to Version 4, fopen only allows 50 files to be open at any one time. Use the x fopen function name (see extended file functions on page 749) if your program needs more than 50 files open. When fopen is used on a system with THEOS 32 Version 4 and above, it can have as many as 1000 files at one time.

Using the *mode* codes of "t," "d," "k" or "i" requires that the file matching *filename* have the corresponding organization.

Direct, keyed and indexed files cannot be created with this function and should not be opened with "a," "a+" or "w+" access. Opening these types of files with "w" access does not clear the current contents of the file. However, if an existing direct, keyed or indexed file is opened with "w" or "w+" access and the organization is not specified or is specified as "b," the existing file is erased and a new stream file is created with the same name.

#### Named Pipes

A named pipe is implemented as a "memory file." Its duration lasts until it is written and closed, and read and closed. The named pipe does not exist after the system is reset.

Pipe file always have a file name beginning with "/pipe/" with any file-name and file-type. For example:

```
p = fopen("/pipe/test.file", "w");
```

**Conforms to:**                    **fopen**    n ANSI    n DOS    n THEOS    n POSIX

Note: The access modes "t," "i," "k," "d," "l," "lA," "lR," "lW," "m" and "p" are extensions to the ANSI standard.

**See also:**                    [access](#), [fcreate](#), [fdopen](#), [file\\_err](#), [freopen](#), [open](#), [openhlp](#), [openmenu](#), [pipe](#), [popen](#), [topen](#)

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE *    datafile,
             prtfile;

    if (!(datafile=fopen("my.data", "r1"))) // open data file
        exit(fperror());
    if (!(prtfile=fopen("prt1", "w")) // open printer
        exit(fperror());
    ...
    fcloseall();    // close all files
}
```

## **\_force**

Force another user process to terminate and execute a specific command.

```
#include <sc.h>
unsigned _force ( int pid, const char _far * cmdstr )
```

---

<i>cmdstr</i>	»	pointer to string containing csi command
<i>pid</i>	»	process number

**Operation:** If user process *pid* is logged on it is forced to perform a **Break**, **Q** operation and, when that is successful, the user is forced to execute *cmdstr*.

**Returns:** A success/fail code: A zero indicates success; a non-zero indicates that the process could not be forced.

**Notes:** This function can operate in one of two modes: Wait until successful or try once and report result.

When *pid* is a value less than 256, the *\_force* function tries to force the user process and will wait until it is successful. In this situation there is only one possible return value: zero indicating success.

If *pid* is a value greater than 256, it is treated as a coded value in the form of:

*0xttpp*

The *pp* is the value of the process to force. The *tt* is the number of seconds to attempt the force. If *tt* is "02" the *\_force* function first enables the remote user's **Break**, **Q**, then it attempts to force the user process but only for a two-second period of time. If it is not successful in that time period, it returns with a non-zero value indicating failure. Other, non-zero values of *tt* cause *\_force* to attempt to force the user process for the specified number of seconds.

**Restrictions:** If the user process is not logged on or has its **Break**, **Q** disabled, it cannot be forced by this function (except as noted above). In this situation, the *\_force* function may not return until the user is successfully forced.

*cmdstr* must be a pointer to a string. Using NULL causes a fatal error.

**Conforms to:**            *\_force*    q ANSI    q DOS    n THEOS    q POSIX





## fork, forktask

Fork or spawn the current task as a subtask of itself.

```
#include <process.h>
short fork ( void )
short forktask ( pid )

_____
pid          »   process number
```

- Operation:** Another process is activated as a “child” task to this “parent” task. The child task process is activated with a copy of this parent task’s System Communication Region (SCR) and a copy of the data segment.
- The child task begins execution at the same location in this current program. That is, it is returning from the fork or forktask function call. (The child task does not receive a copy of the program as it is shared. The use count of the program is merely incremented to indicate that another task is using the code.)
- fork**        The spawned child task is activated in a process number selected by the operating system.
- forktask**    The spawned child task is activated as process number *pid*.
- Returns:** Both functions have the same type of return. The parent task returns with the value of the spawned child task’s process number. The child task returns with a zero value.
- Errors:** A return value of -1 indicates that the task could not be spawned. When this occurs it is normally due to insufficient memory for the copies of the SCR and data segment.
- Notes:** Because the child task receives a copy of the SCR and data segment of the parent task, it has access to all of the resources that the parent task had access to.
- ▶ The same files will be open, their input and output position pointers will be the same.
  - ▶ The same environment variables and values are the same.
  - ▶ The pool and status of semaphores are the same.
- When a parent task exits or uses the [system](#) functions, all of its child tasks are terminated. This include the child tasks spawned by the parent and the child tasks spawned by the children, grandchildren, greatgrandchildren, *etc.*
- Restrictions:** The child task does not inherit any resource, file or record locks.
- Conforms to:**
- |                 |        |       |         |         |
|-----------------|--------|-------|---------|---------|
| <b>fork</b>     | q ANSI | q DOS | n THEOS | n POSIX |
| <b>forktask</b> | q ANSI | q DOS | n THEOS | q POSIX |
- See also:** [getmpid](#), [getpid](#), [getppid](#), [killtask](#), [spawn functions](#), [system](#)

### Example:

```
#include <stdio.h>
#include <process.h>

void
main()
{
    int    childpid;

    if ((childpid=fork()) == 0)    // return zero?
        goto subtask_start;

    if (childpid==-1) {
        printf("Subtask could not be started.\n");
        exit(253);
    }

    // resume parent task code

    ...

    killtask(childpid);
    exit(0);

subtask_start:

    // subtask code
    ...

}
```

## formask, formclear, formincr, formparm

These functions interface to a disk driver's formatting routines.

```
#include <format.h>
void formask ( void )
void formclear ( DISK_UCB * ucb, char * label, int direct, int trackct, int headct,
                int sectct, int density )
void formincr ( int incr, int sectct, short * interlacetable )
void formparm ( short argc, char ** argv )
```

---

<i>argc</i>	»	number of arguments in <i>argv</i>
<i>argv</i>	»	pointer to array of pointers to strings
<i>density</i>	»	code for density
<i>direct</i>	»	number of directory entries to allocate
<i>headct</i>	»	number of heads
<i>incr</i>	»	increment count between logical sector numbers
<i>label</i>	»	pointer to string containing label for disk
<i>interlacetable</i>	»	pointer to list of sectors numbers used for interlacing
<i>sectct</i>	»	number of sectors per track
<i>trackct</i>	»	number of tracks
<i>ucb</i>	»	pointer to unit control block structure for disk

**Operation:** These functions are used in disk format or maintenance utility programs to interface with a disk-driver's low-level formatting capabilities. For instance, the utility programs DISK and SETUP use these functions when formatting or clearing a disk.

**formask** This function displays the geometry of the disk to be formatted using message text from the SYSTEM.TEOS $nnn$ .MESSAGE $n$  file.

```
Number of main directory entries = 112
Increment between adjacent sectors = 1
Number of cylinders = 80
Number of heads per cylinder = 2
Number of sectors per track = 36
Density = 7 (Removable, 512 bytes per sector)
Formatted capacity = 1,474,560 bytes.
```

Is the above information correct (Y/N)

The program is exited if the user responds with a N.

The values come from the following global variables:

```
int lub;           // ucb index of disk
int size;          // number of main directory entries
int cyls;          // number of cyls on disk
int heads;         // number of tracks per cylinder
int sects;         // number of THEOS sectors per track,
                  // 256 bytes per sector
int den;           // density code, described in VLB
int incr;          // sector increment, seldom used
char label[8];     // volume label, right space padded
```

```
DISK_UCB *ucb; // ucb pointer to disk
```

These values are encoded and copied to the disk's UCB if the user answers Y.

**formclear** This function writes an empty THEOS file system to the disk drive, writing the label sector, blank main directory sectors, and the first free space table onto the disk.

The parameters are encoded onto the VLB (Volume Label Block) before it is written to the new disk.

**formincr** Builds *interlacetable* to remap the sectors on a track to be formatted. The interleave table that results maps the physical contiguous sector on the track to the logical sector numbers in the sector header so that consecutive logical sectors are not necessarily consecutive physical sectors.

An increment of 1 means that adjacent logical sectors are in adjacent physical sectors. An increment of 2 means that there is at least 1 physical sector separating adjacent logical sectors.

The *incr* must be between 1 and *sectct*.

**formparm** Interprets the command-line arguments pointed to by *argv* as disk-formatting options and copies the values to the global variables listed above in the *formask* function description. Only the first *argc* arguments are processed.

The *SYSTEM.TEOSnnn.KEYWORDn* file is used to identify the individual option names. Unrecognized option names or values are ignored.

**Restrictions:** These functions should only be used in utility programs that will use the disk-formatting driver of a disk. For instance, the *DISK* and *SETUP* utilities use these functions.

<b>Conforms to:</b>	<b>formask</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>formclear</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>formincr</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>formparm</b>	q ANSI	q DOS	n THEOS	q POSIX

## fperror

Test error status and output file-related error message on stderr.

```
#include <stdio.h>
int fperror ( void )
```

**Operation:** Tests [\\_errno](#) and, if it is non-zero, writes the appropriate message from the `SYSTEM.TEOS $nnn$ .MESSAGE $n$`  file to `stderr`. Parameter substitution is performed on the message text using the reserved field `_errarg`.

**Returns:** The value of [\\_errno](#) used by this function.

**Notes:** The [\\_errno](#) field is cleared by this function.

Refer to [errno](#), [\\_errno](#), [\\_errarg variables](#) for a list of the functions that set the [\\_errno](#) variable.

**Conforms to:** [fperror](#) q ANSI q DOS n THEOS q POSIX

**See also:** [clearerr](#), [eof](#), [\\_errbot](#), [errmsg](#), [feof](#), [ferror](#), [file\\_err](#), [perror](#), [putmsg](#), [strerror](#), [syserr](#)

---

### Example:

```
#include <stdio.h>

void
main()
{
    FILE *input;

    if (!(infile=fopen("some.file","r"))
        fperror();
    ...
}
```

---

### Output:

```
>test
```

```
File "some.file" not found.
```

```
>
```

## fprintf

Write formatted characters, strings, literal text and numbers to an open stream file.

```
#include <stdio.h>
int fprintf ( FILE * file, char * format, [,argument-list] )
```

---

*file*                   »   pointer to an open, stream file  
*format*                »   formatting control mask  
*argument-list*        »   optional arguments to format and print

**Operation:**       Format and print the *arguments* using the *format* mask. Output is to the open stream file indicated by *file*.

**Returns:**         The number of characters written.

A return of -1 indicates that the write was unsuccessful because the location is locked by another user and the lock wait time elapsed (see below).

**Notes:**           This function operates identically to the [printf](#) function described on page 484 except that the resulting formatted string is written to a stream file rather than stdout.

If *file* has been opened with “r+” access, fprintf checks to see if the requested location in the file is locked by another user with the [filelock](#) function. If it is locked, the function waits for the lock to be released before reading the data. fprintf does not check for locks by other users placed with automatic record locking or the [relock](#) function.

If the write is not immediately successful because the location is locked by another user, this function may wait awhile to see if the lock is cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in `scr.lock_wait` can be examined with the [\\_peekscrbr](#), [\\_peekscrd](#), [\\_peekscrw](#) functions or changed with the [\\_pokescrb](#), [\\_pokescrd](#), [\\_pokescrw](#) functions.

When a read attempt fails with a return value of -1 (region already lock by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`. This item can be examined with the [\\_peekscrbr](#), [\\_peekscrd](#), [\\_peekscrw](#) functions.

If *file* was not opened with “r+” access, fprintf does not check for locks by other users.

**Conforms to:**       **fprintf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [cprintf](#), [printf](#), [sprintf](#), [vfprintf](#)

**Example:**        See also the [printf](#) examples.

```
#include <stdio.h>

main()
{
    FILE *      outfile;

    if (outfile = fopen("sample.output", "w")) {
        fprintf(outfile, "This is a only a test.\n");
        fclose(outfile);
    }
}
```

**fputc, fputl, fputs, fputsn, fputsnl, fputw, putc, putchar**

Write a byte, character, integer, string or word to an open stream file.

```
#include <stdio.h>
int fputc ( int c, FILE * file )
void fputl ( unsigned long word, FILE * file )
int fputs ( char * string, FILE * file )
char * fputsn ( char * string, size_t len, FILE * file )
int fputsnl ( char * string, FILE * file )
void fputw ( unsigned short word, FILE * file )
int putc ( char c, FILE * file )
int putchar ( int c )
```

---

<i>c</i>	»	character to write
<i>file</i>	»	pointer to file's fcb
<i>len</i>	»	number of bytes to output
<i>string</i>	»	pointer to characters
<i>word</i>	»	integer data value

**Operation:** Each of these functions writes data to an open file at the file's current position pointer. The position pointer is updated to reflect the amount of data written.

**fputc** Writes one byte to *file*. If *file* is the console output (or stdout mapped to the console output device), the [putc](#) function is used to display the character.

**fputl** Writes a long integer to *file*. No alignment of data is presumed. That is, exactly [sizeof\(long int\)](#) bytes of data are written at the current file position.

**fputs** Writes a string to *file*. If *file* is the console output device (or stdout mapped to the console output), the [cputs](#) function is used to display the string.

The string terminating character '\0' is not written.

**fputsn** Writes *len* character of *string* to *file*. If *file* is the console output (or stdout mapped to the console output), the [putc](#) function is used to write the individual characters of *string*.

**fputsnl** Write *string* to *file*, appending the new-line character. This function is implemented as a macro:

```
(fputs(string, file), fputc("\n", file))
```

**fputw** Write the short integer *w* to *file*. No alignment of data is presumed. That is, exactly two bytes of data are written at the current file position.

**putc** This function name is a synonym to the **fputc** function.



	<b>putc</b>	This function name is a synonym to the <code>fputc</code> function. When <code>putc</code> is used, the <i>file</i> specification for <code>fputc</code> is always set to <code>stdout</code> .
<b>Returns:</b>	<b>fputc</b>	The character <code>c</code> is returned. A return of EOF indicates an error, in which case <code>errno</code> is set to indicate the specific error encountered.
	<b>fputl</b>	No value is returned.
	<b>fputs</b>	An error status code is returned: Zero indicates no error; a non-zero code indicates the specific error encountered. Also, <code>errno</code> is set.
	<b>fputsn</b>	The value of <code>string</code> is returned. A return of a NULL pointer indicates an error, with the <code>errno</code> variable indicating the specific error.
	<b>fputsnl</b>	An error status code is returned: Zero indicates no error; a non-zero code indicates the specific error encountered. Also, <code>errno</code> is set.
	<b>fputw</b>	No value is returned. The <code>errno</code> variable is set if an error is encountered.
	<b>putc</b>	Same as <code>fputc</code> .
	<b>putchar</b>	Same as <code>fputc</code> .

A return of -1 indicates that the write was unsuccessful because the location is locked by another user and the lock wait time has elapsed (see below).

**Notes:** These functions check to see if the requested location in the file is locked by another user with the `filelock` function. If it is locked, the function waits for the lock to be released before writing the data. These functions do not check for locks by other users placed with automatic record-locking or the `relock` function.

If the write is not immediately successful because the location is locked by another user, these functions may wait awhile to see if the lock is cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

## 238 *fputc, fputl, fputs, fputsn, fputsnl, fputw, putc, putchar*

---

If the program is used on a THEOS 32 Version 4 or later system, and the input request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`.

Conforms to:	<b>fputc</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>fputl</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fputs</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>fputsn</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fputsnl</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>fputw</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>putc</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>putchar</b>	n ANSI	n DOS	n THEOS	n POSIX

**See also:** [feof](#), [ferror](#), [fwrite](#), [putw](#), [write](#)

---

### Example:

```
#include <stdio.h>

void main()
{
    FILE *    datafile;
    char      c = 'x';
    short int  i = 25;
    long int   l = 123456;

    datafile = fopen("data.file", "w");

    fputc('A', datafile);          // record type = A
    fputs("Sample record", datafile);
    fputc(0, datafile);           // mark end of string

    fputc('D', datafile);          // record type = D
    fputs("Another sample", datafile);
    fputc(0, datafile);           // mark end of string
    fputl(l, datafile);

    fputc('S', datafile);          // record type = S
    fputs("Yet another sample", datafile);
    fputc(0, datafile);           // mark end of string
    fputw(i, datafile);
    fputs("End", datafile);

    fclose(datafile);
}
```

**Example:**

This example uses the putchar function to implement a “linput using” function, similar to the LINPUT USING statement in the BASIC language.

```
// Special Library: LINPUTU
//
// linputu(char *dest, char *mask)
//
// Performs similar function to the LINPUT USING statement in
// BASIC. Displays mask on console at current location, positions
// to the beginning of the displayed mask and accepts operator
// modifications to that mask. Operator not allowed to position
// outside of the displayed mask length.
//
// Modification keys include:
//
// BEGLINE    position to beginning of field
// DELCHAR    delete character
// DEL        delete prior character
// ENDLINE    position to end of current field
// RIGHT      non-destructive advance
// LEFT       non-destructive backspace
// REPLACE    insert space - all characters pushed to right
//            last character lost
// CASE       change case of character and advance
// NEWLINE    exit from function
// QUIT       exit from function
// ESC        restore to original text, position beginning,
//            continue linputu
// ERASE      change all characters from current position
//            to end of line to spaces - cursor remains at
//            current position
// TRANSPOS   transpose the two characters starting with the
//            character under the current position
//
// all other characters replace current character and advance
//
// The resulting input string is in the dest field.
// Mask field is unchanged.
//
// Returns the address of the dest field.

#include <stdio.h>
#include <string.h>
#include <ascii.h>
#include <crt.h>
#include <func_key.h>
#include <ctype.h>

int linp;          // terminating character storage

// function to accept input similar to LINPUT USING statement of BASIC

static char * ptr;

static void
del_char()
{
    int j;

    if (j=strlen(ptr))
    {
        crt(KOFF);
        printf("%s",strcpy(ptr,ptr+1));
        putchar(' ');
        while (j-->0)
            crt(BS);
        crt(KON);
    }
}
```

```
}
char *
linputu(char *dest, char *mask)
{
    static      int    len, len2, l;

    strcpy(dest, mask);          // initialize return string

    if (!(len=strlen(mask)))
        return dest;            // null mask
    len2 = len-1;

    // display mask, position to beginning

    printf("%s",dest);
    l = len;

    crt(KOFF);

    while (l--)
        crt(BS);

    // accept modifications to field

    ptr = dest;                  // point to beginning of string
    l = 0;                       // character number
    conmask("n");

    crt(KON);

    while (((linp=getchar()) != NEWLINE)
        && (linp != QUIT)
        && (linp != TOP))
    {
        if (isprint(linp) || linp==' ')
        {
            if (l < len)
            {
                *ptr++ = linp;
                putchar(linp);
                ++l;
            }
            else crt(BEL);
        }

        else switch (linp)
        {
            case BEGLINE:        // position to beginning
                crt(KOFF);
                while (l)
                {
                    --l; --ptr;
                    crt(BS);
                }
                crt(KON);
                break;

            case DELCHAR:        // delete current character
                if (l < len)
                    del_char();
                else crt(BEL);
                break;

            case DEL:            // delete prior character
                if (l)
                {
                    --l; --ptr;
                    crt(BS);
                    del_char();
                }
            }
        }
    }
}
```

```
    }
    else
        crt(BEL);
    break;

case ENDLINE: // position to ending
    crt(KOFF);
    while (*ptr++)
    {
        ++l;
        crt(RIGHT);
    }
    --ptr;
    while (*(--ptr) == ' ')
    {
        --l;
        crt(BS);
    }
    ++ptr;
    crt(KON);
    break;

case RIGHTKEY: // right character
    if (l < len)
    {
        ++l;
        if (!*ptr) {*ptr = ' '; *(ptr+1) = 0;}
        ++ptr;
        crt(RIGHT);
    }
    else crt(BEL);
    break;

case LEFTKEY: // left character
    if (l)
    {
        --l;
        --ptr;
        crt(BS);
    }
    else crt(BEL);
    break;

case REPLACE: // Insert/replace
{
    static int i;
    static char *k;

    i = len - 1;
    k = dest+len; // point to end of str
    *(k-1) = 0;
    while (i--)
        *k = *(--k);
    *ptr = ' ';
    printf("%s", ptr);
    i = (int)strlen(ptr);
    while (i--)
        crt(BS);
    break;
}

case CASE: // lowercase
    if (l < len)
    {
        if (isupper(*ptr))
        {
            *ptr = tolower(*ptr);
            putchar(*ptr++);
            ++l;
        }
        else if (islower(*ptr))
```

```
        {
            *ptr = toupper(*ptr);
            putchar(*ptr++);
            ++l;
        }
    }
    else crt(BEL);
    break;

case ESC: // original text
    crt(KOFF);
    strcpy(dest,mask); // restore orig
    while (l)
    {
        --l;
        crt(BS);
    }
    printf("%s",dest); // display orig
    l = len;
    while (l--)
        crt(BS);
    l = 0; ptr = dest; // point beginning
    crt(KON);
    break;

case ERASE: // erase to end of string
    while (*ptr)
        del_char();
    break;

case TRANSPOS: // transpose characters
    lnp = *ptr;
    ptr[i] = ptr[0];
    ptr[0] = lnp;
    putchar(*ptr++);
    putchar(*ptr++);
    break;

default:
    crt(BEL);
    break;
}
}
conmask("e");
return dest;
}
```

## fread

Read multiple data items from a file.

```
#include <stdio.h>
```

```
size_t fread ( void * buffer, size_t size, size_t count, FILE * file )
```

---

<i>buffer</i>	»	pointer to storage area
<i>count</i>	»	number of data items to read
<i>file</i>	»	pointer to file's fcb
<i>size</i>	»	size of each data item

**Operation:** *count* number of data fields are read from *file* and placed in consecutive locations in *buffer*. Each data item is of length *size*.

The file's current position pointer is incremented by the number of bytes actually read.

**Returns:** The number of items actually read is returned as the value of the function.

**Errors:** Normal file access errors are detected and cause [errno](#) and [\\_errno](#) to be set, if appropriate.

**Conforms to:** **fread**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [fgetc](#), [fgetl](#), [fgets](#), [fgetsn](#), [fgetw](#), [getc](#), [getchar](#), [fscanf](#), [read](#)

**free**

Deallocate memory that was allocated with the [calloc](#), [malloc](#) or [realloc](#) functions.

```
#include <stdlib.h>
void free ( void * ptr )
```

---

*ptr*                      »    pointer to allocated buffer

**Operation:**        The memory pointed to by *ptr* is returned to the heap space available memory table. The number of bytes freed is the length of the block allocated or reallocated with the [calloc](#), [malloc](#) or [realloc](#) function used to allocate the memory.

**Notes:**            A *ptr* value of NULL is ignored as are *ptr* values that do not point to a region of memory allocated with the [calloc](#), [malloc](#) or [realloc](#) function.

Although it is not necessary to **free** memory prior to exiting your program, it is good programming practice.

To free memory allocated with the [\\_getmem](#) function, you must use the [\\_putmem](#) function.

**Defaults:**        Any and all memory allocated to your program from the heap is released when the program exits.

**Conforms to:**                **free**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [calloc](#), [malloc](#), [realloc](#)

---

**Example:**

```
#include <stdlib.h>

void
main()
{
    char *buffer;

    if (!(buffer=calloc(1024, 1)))
        syserr(16, 3, NULL);

    ...                               // use the memory

    free(buffer);

    ...
}
```



**free\_sel, \_free\_sel**

Release a memory segment selector.

```
#include <driver.h>
void free_sel ( unsigned selector )

#include <sc.h>
void _free_sel ( unsigned selector )
```

---

*selector*                    »    memory selector

**Operation:**        Release the memory segment *selector*.

**Notes:**            These functions should only be used to release a selector created with the [make\\_alias](#), [\\_make\\_alias](#), [make\\_sel](#), [\\_make\\_sel](#) functions.

These functions do not deallocate the memory associated with *selector* and can be very dangerous when used on a selector created with the [\\_getmem](#) function. Use the [\\_putmem](#) function to deallocate the memory and release the selector.

**Restrictions:**    This function is intended for use by device-driver authors.

Do not release a memory selector that you do not own!

**Conforms to:**        [free\\_sel](#)    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [make\\_alias](#), [\\_make\\_alias](#), [make\\_sel](#), [\\_make\\_sel](#), [\\_putmem](#)

**freopen**

Reassign a file pointer.

```
#include <stdio.h>
FILE * freopen ( char * filename, char * mode, FILE * file )
```

---

<i>file</i>	»	pointer to an already open file's fcb
<i>filename</i>	»	pointer to string containing file description
<i>mode</i>	»	access mode

**Operation:** Using the [fclose](#) function, any file currently associated with *file* is closed. An error on this close operation is ignored.

The [fopen](#) function is then used to open *filename* with access *mode*. The file pointer for this file uses the same location as *file*.

**Returns:** A pointer to the opened file's file control block (FCB). If the file cannot be opened, a NULL pointer is returned and [errno](#) is set to the error code describing the reason for the failure.

**Errors:** When the file cannot be opened and a NULL pointer is returned, [errno](#) is set to one of the following codes:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EACCES	18	Protected file
EBADF	24	Invalid <i>filename</i>
ENODEV	13	Disk not attached
ENOENT	19	File not found
ENOMEM	3	Insufficient memory
ENOSPC	42	Disk full
EMFILE	15	Too many open files

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [ferror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:** The `freopen` function is typically used to redirect the preopened files `stdin`, `stdout` and `stderr`.

The specification of the file in *filename* is assumed to be in the current working directory unless *filename* specifies the path to the file.

The character string *mode* is the same as described for the `fopen` function on page 222.

**Defaults:** If the file organization is not specified as “d,” “k” or “i,” then the file is opened with stream organization.

When `freopen` must create the file, it sets the following default permissions: shared-read and write protected, owner-execute protected and not hidden. This default can be changed by defining an environment variable named

CREATE whose value is the protection codes desired. Refer to the CREATE environment variable description in the *THEOS System Reference* manual.

**Restrictions:** Using the *mode* codes of “d,” “k” or “i” requires that the file matching *file-name* have the corresponding organization.

**Conforms to:**                **freopen**    n ANSI        n DOS        n THEOS    n POSIX

Note: The access modes “t,” “i,” “k,” “d,” “l,” “lA,” “lR,” “lW,” “m” and “p” are extensions to the ANSI standard.

**See also:**                [access](#), [fdopen](#), [fopen](#), [open](#), [openhlp](#), [openmenu](#), [pipe](#), [popen](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    if (!freopen("prt1", "w", stdout))
        exit(fperror());

    ...           // all subsequent output to stdout will be sent
                  // to the device attached as PRT1
}
```

**frexp**

Get the mantissa and exponent power of two for a floating-point value.

```
#include <math.h>
double frexp ( double x, int * exptr )
```

---

*exptr*                   »   pointer to exponent storage  
*x*                        »   value to analyze

**Operation:**       The value of *x* is recomputed as a mantissa and a power of 2 such that

$$x = \text{mantissa} \times 2^{\text{exp}}$$

and

$$0.5 \leq \text{mantissa} < 1.0$$

**Returns:**       The value of the mantissa. The value of the exponent is copied to the location pointed to by *exptr*. When *x* is zero, the mantissa and exponent are both set to zero.

**Notes:**        This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                **frexp**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**       [ldexp](#)

## fscanf

Read formatted data from a file stream.

```
#include <stdio.h>
```

```
int fscanf ( FILE * file, const char * mask, argument... )
```

---

<i>file</i>	»	pointer to file's fcb
-------------	---	-----------------------

<i>mask</i>	»	pointer to string containing input mask
-------------	---	---

<i>argument...</i>	»	pointers to storage locations for items scanned
--------------------	---	---

**Operation:** Identical in operation to the [scanf](#) function except that the input source is *file*, not stdin. Data is read from *file* and converted according to the format specifications in *mask*. The converted data is saved in the locations pointed to by the *argument* list.

**Returns:** The number of fields successfully converted and assigned. The return value does not include fields read but not assigned.

**Errors:** A return value of EOF means that an error or end-of-file was detected on *file* before the first conversion. A return value of zero means that no fields were assigned.

**Notes:** The *mask* field is fully described in the description of the [scanf](#) function on page 540.

**Conforms to:**                    **fscanf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                    [scanf](#), [sscanf](#)

**fseek**

Sets a file's input/output position pointer to a specified location.

```
#include <stdio.h>
int fseek ( FILE * file, long offset, int base )
```

---

*base*                   »   starting point within file  
*file*                    »   pointer to file's fcb  
*offset*                  »   relative position to seek to

**Operation:** The file-position pointer for *file* is changed according to *offset* and *base*. That is, the position pointer is set to *offset* bytes from *base*. *offset* is a signed value and *base* is coded to be one of three values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
SEEK_SET	0	<i>offset</i> is relative to the beginning of the file.
SEEK_CUR	1	<i>offset</i> is relative to the current value of the position pointer.
SEEK_END	2	<i>offset</i> is relative to the current end-of-file.

**Returns:** A success/fail indicator. A zero return value indicates success; a non-zero return value indicates failure and [errno](#) is set to the specific error. On devices incapable of seeking (terminals, printers, *etc.*) the return value is undefined.

**Errors:** When the return value is not zero, [errno](#) may have one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EBADF	24	File not open
EINVAL	428	The value of <i>offset</i> and/or <i>base</i> is invalid.

**Notes:** You may position to any location in the file or even beyond the current end-of-file. However, you may not position to a location before the beginning of the file.

The `fseek` operation clears the end-of-file indicator and discards any prior `ungetc` characters from *file*.

Because this function does not actually read or write any data to *file*, no record or location locking is tested.

**Defaults:** When a file is opened for append, the current file position is determined by the last input/output operation, not by the location of the next write operation. When no read, write or seek has been performed on a file stream, the file position is at the start of the file.

**Restrictions:** This function should be used on stream files only.

**Conforms to:** `fseek`   n ANSI    n DOS    n THEOS   n POSIX

**See also:** [fsetpos](#), [ftell](#), [lseek](#), [rewind](#), [seek](#), [\\_seek](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE *    in;
    char      string[256];

    if (!(in=fopen("some.data","r")))
        exit(fperror());

    fseek(in, -10L, SEEK_END); // position 10 bytes from eof
    fread(string, 10, 1, in); // read last 10 bytes of file

    ...
}
```

**fsetpos**

Sets a file’s input/output position pointer.

```
#include <stdio.h>
int fsetpos ( FILE * file, fpos_t * pos )
```

---

<i>file</i>	»	pointer to file’s fcb
<i>pos</i>	»	position to seek to

**Operation:** Sets the file-position pointer for *file* to the value of *pos*. This is equivalent to performing:

```
fseek(file, pos, SEEK_SET)
```

That is, the file-position pointer is set to *pos* bytes from the beginning of the file.

Before changing the position pointer, a flush operation is performed.

**Returns:** A success/fail indicator. A return of zero indicates success; a non-zero return value indicates failure and the [errno](#) variable is set to indicate the specific error.

**Errors:** When the return value is not zero, [errno](#) may have one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EBADF	24	File not open
EINVAL	428	The value of <i>file</i> is not a pointer to a file’s fcb or a valid file handle.

**Notes:** The end-of-file indicator is cleared by this function and the effects of the [ungetc](#) function are undone.

When used in an ANSI C conforming program, the *fsetpos* function should only use values of *pos* that have been previously set with the [fgetpos](#) function. In ANSI C, the value of *pos* is usable only by the [fgetpos](#) and *fsetpos* functions.

Because this function does not actually read or write any data to *file*, no record or location-locking is tested.

This function should be used on stream files only.

**Conforms to:** **fsetpos**    n ANSI    n DOS    n THEOS    q POSIX

**See also:** [fgetpos](#), [fseek](#), [ftell](#), [lseek](#), [rewind](#)



**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE *    out;
    fpos_t    pos;

    if (!(out = fopen("some.text", "r")))
        exit(fperror());

    ...
    fgetpos(out, &pos);           // perform some read operations
    ...                           // remember current position
    ...                           // perform more read operations
    fsetpos(out, pos);           // position back to rembered loc
    ...
}
```

**fsign**

Determine the mathematical sign of a floating-point value.

```
#include <math.h>
int fsign ( double x )
_____
x                » floating-point value
```

**Operation:** The sign of the value *x* is tested and returned.

**Returns:** The sign of *x* as a -1 or +1. If *x* is zero, then 0 is returned.

**Conforms to:** **fsign**    q ANSI    q DOS    n THEOS    q POSIX

## far string functions

This group of functions is similar to the [string functions](#) described on page 596 except these manipulate far memory, null-terminated strings. They can copy, append, create, duplicate, compare, initialize, *etc.* strings.

```
#include <string.h>
char _far * _fstrcat ( char _far * string1, const char _far * string2 )
char _far * _fstrchr ( const char _far * string, char c )
int _fstrcmp ( const char _far * string1, const char _far * string2 )
char _far * _fstrcpy ( char _far * string1, const char _far * string2 )
size_t _fstrcspn ( const char _far * string1, const char _far * string2 )
int _fstrdcmp ( char _far * string1, const char _far * string2 )
char _far * _fstrend ( const char _far * string )
int _fstreq ( const char _far * string1, const char _far * string2 )
int _fstricmp ( const char _far * string1, const char _far * string2 )
int _fstrieq ( const char _far * string1, const char _far * string2 )
char _far * _fstristr ( const char _far * string1, const char _far * string2 )
size_t _fstrlen ( const char _far * string )
char _far * _fstrpbrk ( const char _far * string1, const char _far * string2 )
char _far * _fstrrchr ( const char _far * string, char c )
char _far * _fstrset ( char _far * string, char c )
size_t _fstrspn ( const char _far * string1, const char _far * string2 )
char _far * _fstrstr ( const char _far * string1, const char _far * string2 )
char _far * _fstrtok ( const char _far * string1, const char _far * string2 )
```

---

<i>c</i>	»	single character
<i>string</i>	»	pointer to string of characters
<i>string<sub>1</sub></i>	»	pointer to first string of characters
<i>string<sub>2</sub></i>	»	pointer to second string of characters

<b>Operation:</b>	<b>_fstrcat</b>	The contents of <i>string<sub>2</sub></i> are copied to the end of <i>string<sub>1</sub></i> .
	<b>_fstrchr</b>	<i>string</i> is searched for the first occurrence of the character value <i>c</i> . <i>c</i> may be the null character, in which case the terminating character of <i>string</i> is located. The <i>_fstrend</i> function is a better choice for this purpose.
	<b>_fstrcmp</b>	The contents of <i>string<sub>1</sub></i> are compared with the contents of <i>string<sub>2</sub></i> . A greater than, equal to, or less than result is returned.
	<b>_fstrcpy</b>	The contents of <i>string<sub>2</sub></i> are copied to <i>string<sub>1</sub></i> , replacing the prior contents of <i>string<sub>1</sub></i> .
	<b>_fstrcspn</b>	Similar to the function <i>_fstrpbrk</i> , the contents of <i>string<sub>1</sub></i> are searched for any occurrence of a character that is also in <i>string<sub>2</sub></i> . The null-terminating character is not included in the search.

**\_fstricmp** This function is identical in operation to the `_fstrcmp` function unless there are ASCII decimal digits in the strings.

When *string*<sub>1</sub> and *string*<sub>2</sub> contain ASCII decimal digits starting in the same relative location, the strings will compare as if the portions containing the decimal digits were the same length, with zeros added to pad the shorter string of digits to the same length as the other string's string of digits. For instance:

Original set	with <code>_fstrcmp</code> :	with <code>_fstricmp</code> :
a1xxx	a101xxx	a1xxx
a83xxx	a11xxx	a11xxx
a11xxx	a1xxx	a83xxx
a101xxx	a83xxx	a101xxx
b15xxx	b100xxx	b15xxx
b100xxx	b15xxx	b100xxx

**\_fstrend** The null-terminating character of *string* is located.

**\_fstreq** The contents of *string*<sub>1</sub> are compared with the contents of *string*<sub>2</sub>. Only equality or inequality is determined.

**\_fstricmp** A “case-insensitive” comparison is performed. The lowercased contents of *string*<sub>1</sub> are compared with the lowercased contents of *string*<sub>2</sub>. A greater-than, equal-to, or less-than result is returned.

**\_fstricq** A “case-insensitive” comparison is performed between the contents of *string*<sub>1</sub> and the contents of *string*<sub>2</sub>. Only equality or inequality is determined.

**\_fstristr** A “case-insensitive” substring search is performed. The lowercased contents of *string*<sub>1</sub> are searched for the first occurrence of the lowercased contents of *string*<sub>2</sub>.

**\_fstrlen** Determines the current length of *string*, not including the null-terminating character.

**\_fstrpbrk** Similar to the function `_fstrcspn`, the contents of *string*<sub>1</sub> are searched for any occurrence of a character that is also in *string*<sub>2</sub>. The null-terminating character is not included in the search.

The difference between this function and `_fstrcspn` is that `_fstrpbrk` returns a pointer to the matching character and `_fstrcspn` returns the index of the matching character.

**\_fstrchr** *string* is searched for the last occurrence of the character value *c*. *c* may be the null character, in which case the terminating character of *string* is located.

**\_fstrset** The contents of *string* are set to the value *c*.

**\_fstrspn** This is the opposite of the function `_fstrcspn`. The contents of *string*<sub>1</sub> are searched for any occurrence of a character that is not in *string*<sub>2</sub>. The null-terminating character is not included in the search. In other words, this function determines the ini-

tial length of *string<sub>1</sub>* that consists solely of characters in *string<sub>2</sub>*.

**\_fstrstr** The contents of *string<sub>1</sub>* are searched for the first occurrence of the substring *string<sub>2</sub>*.

**\_fstrtok** Finds the next “token” in *string<sub>1</sub>*.

The contents of *string<sub>2</sub>* define the characters that delimit tokens. A token is defined as a series of one or more characters that are not delimiting characters. For instance, the words in a typical English sentence can be treated as tokens if the delimiting characters are defined as “ . , ”.

The *string<sub>1</sub>* argument may be a pointer to a string of characters or it may be a NULL pointer. When *string<sub>1</sub>* is a character pointer, **\_fstrtok** saves this pointer in an internal variable. When *string<sub>1</sub>* is a NULL pointer, **\_fstrtok** uses the previously saved pointer.

Each time that **\_fstrtok** is called it searches *string<sub>1</sub>*. First it skips any leading delimiters identified by *string<sub>2</sub>* and remembers this first non-delimiting character location. Then it searches for the next delimiter and changes it to a null terminator. The internal variable is then set to point to the character following this location.

Subsequent calls to **\_fstrtok** (using a NULL pointer for *string<sub>1</sub>*) cause **\_fstrtok** to use its saved pointer to the string. It repeats the same sequence as above: Skipping leading delimiters, remembering the location of the first non-delimiting character and changing the next delimiting character to a null-terminator.

**Returns:** **\_fstrcat** A pointer to *string<sub>1</sub>*.

**\_fstrchr** A pointer to the first occurrence of *c* in *string*. A return of a NULL pointer indicates that *c* cannot be found in *string*.

**\_fstrcmp** A signed, integer value.

Comparison	Return value
<i>string<sub>1</sub></i> < <i>string<sub>2</sub></i>	< 0
<i>string<sub>1</sub></i> = <i>string<sub>2</sub></i>	0
<i>string<sub>1</sub></i> > <i>string<sub>2</sub></i>	> 0

**\_fstrcpy** A pointer to *string<sub>1</sub>*.

**\_fstrcspn** The length of the portion of *string<sub>1</sub>* that contains no characters found in *string<sub>2</sub>*.

A return value of zero indicates that the first character of *string<sub>1</sub>* is one of the characters in *string<sub>2</sub>*.

A return value of the length of *string<sub>1</sub>* indicates that none of the characters in *string<sub>1</sub>* is also in *string<sub>2</sub>*.

**\_fstricmp** A signed integer value, similar to **\_fstrcmp**.

**\_fstrend** A pointer to the null-terminating character of *string*.

**\_fstreq** A true/false value. A non-zero value indicates that the two strings are equal; a zero value indicates that they are unequal. (Note that this is the opposite result of a **\_fstrcmp** comparison.)

**\_fstricmp** A signed integer value, similar to **\_fstrcmp**.

A position is equal when the two strings are equal or if they have the same letter independent of its case. When a position is unequal, the lowercase form of each position is compared. Thus, “a” is less than “b” and “B,” even though the value of “a” is actually greater than the value of “B.”

**\_fstricq** A true/false value. A non-zero value indicates that the two strings are equal, independent of their case; a zero value indicates that they are unequal. (Note that this is the opposite result of a **\_fstricmp** comparison.)

**\_fstristr** A pointer to the start of the first occurrence of the lowercase form of *string<sub>2</sub>* that was found in the lowercase form of *string<sub>1</sub>*. A NULL pointer is returned when *string<sub>2</sub>* cannot be found in *string<sub>1</sub>*.

**\_fstrlen** The current length of the contents of *string*.

**\_fstrpbrk** A pointer to the first character in *string<sub>1</sub>* that is also contained in *string<sub>2</sub>*. A NULL pointer indicates that no character in *string<sub>1</sub>* matched any of the characters in *string<sub>2</sub>*.

**\_fstrchr** A pointer to the last occurrence of *c* in *string*. A return of a NULL pointer indicates that *c* cannot be found in *string*.

**\_fstrset** A pointer to *string*.

**\_fstrspn** The length of the portion of *string<sub>1</sub>* that contains only characters found in *string<sub>2</sub>*.

A return value of zero indicates that the first character of *string<sub>1</sub>* is not one of the characters in *string<sub>2</sub>*.

A return value of the length of *string<sub>1</sub>* indicates that all of the characters in *string<sub>2</sub>* are also in *string<sub>2</sub>*.

**\_fstrstr** A pointer to the first occurrence of the substring *string<sub>2</sub>* in *string<sub>1</sub>*. A NULL pointer indicates that *string<sub>2</sub>* cannot be found in *string<sub>1</sub>*.

**\_fstrtok** A pointer to the next token found in *string<sub>1</sub>*. (If *string<sub>1</sub>* was a NULL pointer, it returns a pointer to the next token found in the last usage of **\_fstrtok** using a non-NULL pointer for *string<sub>1</sub>*.)

A return of a NULL pointer indicates that there are no more tokens in *string<sub>1</sub>*.

<b>Notes:</b>	<b>_fstrcat</b>	The contents of <i>string</i> are modified.
	<b>_fstrcpr</b>	The contents of <i>string</i> are modified.
	<b>_fstrset</b>	The contents of <i>string</i> are modified.
	<b>_fstrtok</b>	The content of <i>string<sub>1</sub></i> is modified by the operation of this function. (Some delimiting characters are changed to null-terminating characters.)

Multiple, contiguous delimiters are treated as one delimiter.

**Restrictions:** All of these functions operate on null-terminated strings. The string arguments to these functions should contain a null character marking the end of the string.

<b>_fstrcat</b>	The contents of <i>string<sub>2</sub></i> are added directly to the end of <i>string<sub>1</sub></i> . Make sure that the allocation for <i>string<sub>1</sub></i> is sufficient to contain both strings.
<b>_fstrcpy</b>	If <i>string<sub>2</sub></i> and <i>string<sub>1</sub></i> overlap, the resulting string may be erroneous. When <i>string<sub>2</sub></i> is greater than <i>string<sub>1</sub></i> , the copy is performed correctly; when <i>string<sub>1</sub></i> is greater than <i>string<sub>2</sub></i> , the process of copying the characters in <i>string<sub>2</sub></i> may replace subsequent characters in <i>string<sub>2</sub></i> .

For example:

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
string1 string2

```

Result: ghijklmnopklmnopqrstuvwxyz

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
string2 string1

```

Result: abcdefabcdefabcdqrstuvwxyz

Also, make sure that the allocation for *string<sub>1</sub>* is sufficient to contain the copy of *string<sub>2</sub>*.

<b>Conforms to:</b>	<b>_fstrcat</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fstrchr</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fstrcmp</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fstrcpy</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fstrcspn</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>_fstrdcmp</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fstrend</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fstreq</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_fstricmp</b>	q ANSI	n DOS	n THEOS	q POSIX

## 260 *far string functions*

---

<code>_fstireq</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_fstristr</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrlen</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrpbrk</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrchr</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_fstrset</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrspn</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrstr</code>	q ANSI	n DOS	n THEOS	q POSIX
<code>_fstrtok</code>	q ANSI	n DOS	n THEOS	q POSIX

**See also:** [far memory functions](#), [memory functions](#), [string functions](#)



## ftell

Get a file's current position pointer.

```
#include <stdio.h>
long ftell ( FILE * file )
```

---

*file*                      »    pointer to file's fcb

**Operation:**      The current input/output position of *file* is determined, relative to the beginning of the file.

**Returns:**          The current file position.

A return of a negative value indicates an error occurred, in which case [errno](#) is set to the specific cause of the error.

When *file* is on a device incapable of seeking, such as a terminal or printer, the return value is undefined.

**Errors:**            When the return value is negative, [errno](#) may have one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EBADF	24	File not open

**Conforms to:**                      **ftell**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                      [fgetpos](#), [fseek](#), [lseek](#), [tell](#), [\\_tell](#)

**Example:** This example opens a file and displays each record of the file on the console. At the beginning of each display line the percentage of the file read is displayed.

```
#include <stdio.h>
#include <stdlib.h>

void
main(void)
{
    char        textline[256];
    FILE *      file;
    long        end_pos, pos;

    if (!(file=fopen("sample.text", "r")))
        exit(fperror());
    fseek(file, 0, SEEK_END);    // position to end of file

    end_pos = ftell(file);      // get length of data in file

    rewind(file);               // position to start of file

    while (!feof(file)) {
        fgets(textline, 256, file); // read a record
        pos = ftell(file);      // get current position
        printf("%3.0f%% %s", ((double)pos/end_pos)*100, textline);
    }

    fclose(file);
}
```

---

**Output:**

```
>test

 8% #include <stdio.h>
16% #include <stdlib.h>
...
99%
100% }

>
```

**ftoa**

Convert a floating-point value into its ASCII representation.

```
#include <stdlib.h>
char * ftoa ( char * string, double f )
```

---

<i>f</i>	»	value to convert
<i>string</i>	»	pointer to buffer

**Operation:** The floating-point value *f* is converted to its ASCII representation, suitable for printing. The result is copied to *string* as a null-terminated string of characters.

**Returns:** A pointer to *string*.

**Notes:** This function is similar in effect to the [sprintf](#) function:

```
sprintf(string, "%g", f)
```

except that all significant digits of *f* are retained without zero padding short values.

This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:** **ftoa** q ANSI q DOS n THEOS q POSIX

**See also:** [itoa](#), [lltoa](#), [ltoa](#), [ltoc3](#), [ltoi2](#), [ltol3](#), [sprintf](#), [utoa](#)

## **fwrite**

Write multiple data items to a file.

```
#include <stdio.h>
size_t fwrite ( void * buffer, size_t size, size_t count, FILE * file )
```

---

<i>buffer</i>	»	pointer to data being written
<i>count</i>	»	number of items to write
<i>file</i>	»	pointer to file's fcb
<i>size</i>	»	size of each item to write

**Operation:** *count* number of data items from *buffer* are written to *file* at its current position pointer. Each data item is of length *size*.

The file's current position pointer is incremented by the number of bytes actually written.

**Returns:** The number of items actually written is returned. Normally this is the same as *count* unless an error was encountered during the write.

**Errors:** Normal file access errors are detected and cause [errno](#) and [\\_errnum](#) to be set, if appropriate.

**Conforms to:** **fwrite**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [fread](#), [write](#)

## gcd

Compute the greatest common denominator between two values.

```
#include <math.h>
double gcd ( double x, double y )
```

---

*x*                   » first value  
*y*                   » second value

**Operation:**       Computes the largest or greatest value which, when divided into both values *x* and *y*, gives a whole number result.

**Returns:**         The greatest common denominator for the two values. The sign of the returned value is the same as the sign of the *y* parameter.

**Conforms to:**               **gcd**    q ANSI    q DOS    n THEOS    q POSIX

**gcv**

Convert a floating-point value to a string.

```
#include <stdlib.h>
char * gcv ( double value, int sig_digits, char * buffer )
```

---

<i>buffer</i>	»	pointer to storage area
<i>sig_digits</i>	»	number of significant digits
<i>value</i>	»	number to convert

**Operation:** Convert *value* storing the result in *buffer*. An attempt is made to convert at least *sig\_digits* number of significant digits. When there are more significant digits in *value* than *sig\_digits*, exponential format is used.

**Returns:** A pointer to the converted string.

**Errors:** No error is detected.

**Notes:** The [sprintf](#) function can perform the same operations as this function.

**Restrictions:** *buffer* must be allocated large enough to hold the largest converted string, with its trailing null terminator.

**Conforms to:** **gcv** q ANSI n DOS n THEOS n POSIX

**See also:** [ecvt](#), [fcvt](#), [ftoa](#), [sprintf](#)

**Example:**

```
#include <stdio.h>           // needed for printf function
#include <stdlib.h>          // needed for gcvf function

void
main() {
    double   fvalue = 1234.5678;
    char     buffer[20];

    printf("\nThe original value is %g.\n",fvalue);

    gcvf(fvalue, 2, buffer);
    printf("\nThe gcvf string with 2 digits is %s\n",buffer);
    gcvf(fvalue, 4, buffer);
    printf("\nThe gcvf string with 4 digits is %s\n",buffer);
    gcvf(fvalue, 6, buffer);
    printf("\nThe gcvf string with 6 digits is %s\n",buffer);
    gcvf(fvalue, 8, buffer);
    printf("\nThe gcvf string with 8 digits is %s\n",buffer);
    gcvf(fvalue, 10, buffer);
    printf("\nThe gcvf string with 10 digits is %s\n",buffer);
}
```

---

**Output:**

>test

The original value is 1234.5678

The gcvf string with 2 digits is 1.2E+03

The gcvf string with 4 digits is 1235

The gcvf string with 6 digits is 1234.57

The gcvf string with 8 digits is 1234.5678

The gcvf string with 10 digits is 1234.567800

***\_get\* builtin functions***

This group of “functions” generates in-line code to get frequently accessed registers or memory selectors.

```
#include <builtin.h>
unsigned _getar ( unsigned sel )
unsigned _getcsc ( void )
unsigned _getcscsa ( void )
unsigned _getdsc ( void )
unsigned long _geteax ( void )
unsigned long * _getebp ( void )
unsigned long _getflag ( void )
unsigned _getldt ( void )
unsigned long _getlimit ( unsigned sel )
unsigned long _getmsw ( void )
int _getpid ( void )
unsigned _getss ( void )
unsigned _gettib ( void )
```

---

*sel*                      »    memory segment selector

**Operation:**        ***\_getar***        Gets the access rights of memory segment *sel*.

This is not truly a “built-in” function in that it does not produce “in-line” code. It is included here because of its similarity with the other functions.

***\_getcsc***        Gets the code segment selector.

***\_getcscsa***    Gets the code segment alias selector. Only applies when used in a device-driver.

***\_getdsc***        Gets the data segment selector.

***\_geteax***        Gets the value in the EAX register.

***\_getebp***        Gets the value in the EBP register.

***\_getflag***      Gets the cpu flags.

***\_getldt***        Gets the local descriptor table segment alias selector.

***\_getlimit***    Gets the size of a memory segment.

***\_getmsw***        Gets the machine status word (MSW).

***\_getpid***        Gets your process number, base 0.

***\_getss***        Gets the stack segment selector.

***\_gettib***        Gets the task information block segment alias selector.



**Returns:** These “functions” all place their return value in the EAX register, just like all normal functions do. However, if one of these functions is used in an assignment statement, the code is optimized so that the value is placed directly in the destination variable, bypassing the EAX register.

**`_getar`** The access rights of the memory segment. A return of zero means that *sel* is an invalid memory selector.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions or requiring the programmer to drop into assembly language with the `#asm` and `#endasm` directives.

**Restrictions:** These functions are intended for device-driver authors.

<b>Conforms to:</b>	<b><code>_getar</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getcs</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getcsa</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getds</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_geteax</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getebp</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getflag</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getldt</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getlimit</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getmsw</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getpid</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_getss</code></b>	q ANSI	q DOS	n THEOS	q POSIX
	<b><code>_gettib</code></b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** `CREGS.H`, `GDT.H`

### **get\_acc\_name**

Find an account name given an account number.

```
#include <acb.h>
char * get_acc_name ( int acc_num )
_____
acc_num          »   number of desired account
```

**Operation:** The SYSTEM.TEOS32.ACCOUNT file is searched for the first account definition of account number *acc\_num*. When a definition is found, the account name is copied to local storage.

**Returns:** A pointer to the account name that was found.

A pointer to a NULL string is returned if no matching account definition can be found.

**Notes:** When there are multiple account names defined with *acc\_num* (synonym accounts), this function only finds the first account definition. This account name need not be the primary account name for the id.

**Restrictions:** The value is read into a static buffer internal to the *get\_acc\_name* function. This buffer will be overwritten by the next call to *get\_acc\_name*.

**Conforms to:**    *get\_acc\_name*    q ANSI    q DOS    n THEOS    q POSIX

**See also:**    [find\\_acc](#), [read\\_acc](#), [sch\\_acc](#)

**Example:**

```
#include <stdio.h>
#include <acb.h>
#include <string.h>

void print_info(ACB);

void main()
{
    ACB account_info;

    account_info = find_acc(get_acc_name(8));
    print_info(account_info);
}

void print_info(ACB info)
{
    char    name[9];           // work space for name string

    strmake(name, info.name, 8); // change to c-style string
    printf("\nAccount name..... %s\n",name);
    printf("Account id..... %hi\n",info.id);
    printf("Location..... %li\n",info.loc);
    printf("Mail?..... %c\n",info.mail ? 'Y' : 'N');
}
```

---

**Output:**

```
>test

Account name..... PACKAGE
Account id..... 8
Location..... 1449
Mail?..... N

>
```

**getbaud**

Returns the attached baud rate for a specified device.

```
#include <stdio.h>
long getbaud ( int lub )
```

---

*lub*                   »   logical unit number of serial device

**Operation:**       The device-driver for the device specified by *lub* is interrogated and the associated baud rate is returned.

**Returns:**         The attached baud rate.

If the device is not attached, is not a serial device, or the device-driver does not support programmable baud rates, a zero is returned.

**Notes:**           Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of lub numbers.

**Conforms to:**       **getbaud**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [\\_baud2cd](#), [\\_cd2baud](#)

---

**Example:**

```
#include <stdio.h>

void main()
{
    int     lub;
    long    baud;

    printf("\nEnter lub number: ");
    scanf("%i", &lub);

    baud = getbaud(lub);

    if (baud)
        printf("\nThe baud rate for %s is %li.\n",
               _lubname(lub), baud);
    else
        printf("\nDevice %i has no baud rate.\n", lub);
}
```

---

**Output:**

>test

Enter lub number: 32

The baud rate for COM1 is 38400.

>

## getch

Accepts one character from the console input device.

```
#include <conio.h>
unsigned short getch ( void )
```

**Operation:** Accepts one character from the console input device.

If a character is ready in the exec stack or the console device-driver's input buffer, that character is read. When no character is ready, this function always turns the cursor on and then waits for a character to be typed. After the character is read, the cursor on/off status is restored to its prior state.

**Returns:** The character read is returned.

**Errors:** No error is detected.

**Notes:** This function is used directly or indirectly by all console input functions.

This function uses the console input device, not stdin. Redirection of stdin does not affect this function.

**Conforms to:** **getch** q ANSI q DOS n THEOS q POSIX

**See also:** [cgets](#), [conrdy](#), [\\_conrdy](#), [fgetc](#), [fgetl](#), [fgets](#), [fgetsn](#), [fgetw](#), [getc](#), [getchar](#), [fread](#), [getchar](#), [\\_getline](#), [gets](#), [getw](#), [read](#)

---

### Example:

```
#include <stdio.h>
#include <conio.h>
#include <crt.h>           // screen control char defs

void
main(void) {
    char    fi, mi, li;

    conmask("eu");
    putch(CLEAR);         // clear the screen

    at(5,5); cputs("What are your initials: ");
    fi = getch();          // get first initial
    mi = getch();          // get middle initial
    li = getch();          // get last initial
}
```

---

### Output:

```
>test
```

```
    What are your initials: ABC
```

```
>
```

### **getclass**

This function returns the class code number for the console or one of the attached printers.

```
#include <stdlib.h>
int getclass ( int lub )
```

---

*lub*                   »   logical unit number of console or printer

**Operation:**       The class code number for the specified console or printer is returned.

**Returns:**         The class code number. A zero is returned if the device is not attached or is not the console or one of the printers.

**Notes:**          The lub number given to this function should be for the console or one of the printers. Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of these lub numbers.

                  The include file LUB.H defines names for each of the lubs supported.

**Conforms to:**       **getclass**    q ANSI     q DOS     n THEOS    q POSIX

---

#### **Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <lub.h>

void main()
{
    int       class;

    printf("\nThe console is using class code number %i.\n",
           getclass(CONSOLE));
}
```

---

#### **Output:**

>test

The console is using class code number 90.

getcwd

Gets the current working directory name.

```
#include <direct.h> or <stdlib.h>
char * getcwd ( char * dirname, int size )
```

---

*dirname*               »   pointer to string buffer for directory name  
*size*                   »   length of *dirname* storage

**Operation:**       The full path name of the current working directory is stored in *dirname*. *size* specified the maximum size of the *dirname* buffer. An error occurs if *size* is smaller than required to store the current working directory path name.

**Returns:**         A pointer to *dirname*. A return value of NULL indicates an error and [errno](#) is set.

**Errors:**          When the return is NULL, [errno](#) is set to one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EINVAL	428	<i>size</i> argument must be positive
ENOMEM	3	Insufficient memory to allocate <i>dirname</i>
ERANGE	427	<i>size</i> is smaller than required for path name

**Notes:**          The *dirname* argument may be NULL. When it is, [getcwd](#) will allocate a buffer using the [malloc](#) function and the value of *size*. This buffer can later be deallocated with the [free](#) function.

**Conforms to:**        **getcwd**    q ANSI       n DOS       n THEOS       n POSIX

**See also:**          [chdir](#), [mkdir](#), [rmdir](#)

## 276 *getcwd*

---

### Example:

```
#include <stdio.h>
#include <direct.h>

void main()
{
    char    buffer[FILENAME_MAX];    // provide for maximum size

    if (getcwd(buffer, FILENAME_MAX)==0)
        perror("Error getting cwd.");
    else
        printf("The current directory is:\n  %s\n",buffer);
}
```

---

### Output:

```
>test
The current directory is:
  /C32:S

>
```



## getdate

Get the current system date.

```
#include <timer.h> or <sc.h>
char * getdate ( char * buffer )
```

---

*buffer*                    »    pointer to string storage area

**Operation:**        The current system date is retrieved and formatted as a string. The resulting string is stored in the location pointed to by *buffer*.

**Returns:**            A pointer to the formatted date string.

**Notes:**             The format of the date string depends upon the current value of the DATEFORM system environment variable.

<i>DATEFORM</i>	String Format
1	mm/dd/yy
2	dd-mm-yy
3	yy.mm.dd

The different formats use different delimiters between the month, day and year values.

**Restrictions:**     The *buffer* should be an area allocated for at least nine bytes. Whether it is or not, nine bytes of data will be copied to that location.

The year number is always formatted as a two-digit year, excluding the century number. Since this is the current date, the century number can be assumed.

**Conforms to:**                **getdate**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                [gettime](#), [time](#)

## 278 *getdate*

---

### Example:

```
#include <stdio.h>
#include <time.h>

main()
{
    char    datestr[9],
           timestr[9]
           mtimestr[13];

    printf("\nThe current date is %s.", getdate(datestr));
    printf("\nThe current time is %s.", gettime(timestr));
    printf("\nThe current time is %s.", getmsec(mtimestr));
}
```

---

### Output:

```
>test
```

```
The current date is 07/04/96.
The current time is 14:24:15.
The current time is 14:24:15.064
```

```
>
```

## getdev

Determine the logical unit number for the device used by an open file.

```
#include <stdio.h>
int getdev ( FILE * file )
```

---

*file*                    »    pointer to an open file's FCB

**Operation:**        Determines the lub number for the device used by *file*.

**Returns:**           The lub number. An EOF is returned when an error is detected.

**Errors:**            The only error possible occurs when *file* is not open.

**Notes:**            Refer to Appendix D: "Logical Unit Block Numbers" on page 769 for a list of lub numbers and their standard names.

**Conforms to:**        **getdev**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**           [\\_lubname](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    FILE *  infile;

    if (!(infile=fopen(argv[1], "r")))
        exit(fperror());

    printf("\nThe file \"%s\" is on the device \"%s\"\n",
        argv[1], _lubname(getdev(infile)));

    ...

    fcloseall();
}
```

---

### Output:

```
>test data.file

The file "DATA.FILE" is on the device "S"

>test com1

The file "COM1" is on the device "COM1"

>
```

**getenv**

Get the system-defined or user-defined value for an environment parameter.

```
#include <stdlib.h>
char * getenv ( char * name )
```

---

*name*                   »   pointer to environment name string

**Operation:**       The list of environment variables is searched for a variable name matching *name*. When a match is found, a pointer to the defined value of that name is returned.

**Returns:**         A pointer to a string containing the current value for the environment *name*.

When *name* is not defined, a NULL pointer is returned.

**Notes:**           Environment names are not case-sensitive.

Besides any user-defined environment names, the following system-defined names may be used.

ABBREV	DATEFORM	MAIL	RDYMSG
BREAK	DECIMAL	MSG	SEARCH
CASE	HISTORY	NO_QUIT	STDSYN
CMDLIB	LANG	OBJLIB	SUBDIR
COPYLIB	LIBRARY	PATH	SYNONYM
CWD	LINKLIB	PRIV	WORK

Table 3: System-defined Environment Names

The SUBDIR name is a synonym to the CWD name.

**Restrictions:**    The value is read into a static buffer internal to the `getenv` function. This buffer will be overwritten by the next call to `getenv`.

**Conforms to:**       **getenv**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**         [putenv](#), [\\_putenv](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char    name[20];

    while (1)                // loop until break
    {
        printf("\nEnvironment name: ");
        gets(name);          // get name
        if (name[0]==0)      // exit?
            break;
        printf("The value of %s is \"%s\".\n",name,getenv(name));
    }
}
```

---

**Output:**

>test

Environment name: dateform  
The value of dateform is "A".

Environment name: path  
The value of path is "(null)".

Environment name: cmdlib  
The value of cmdlib is "C32.CMD32".

Environment name:

>

**getexvar, putexvar**

Access the calling EXEC language program variables.

```
#include <execvar.h>
char * getexvar ( void * name )
void putexvar ( void * name, void * value )
```

---

<i>name</i>	»	pointer to variable name string
<i>value</i>	»	pointer to new value string

**Operation:**      **getexvar**      Search the calling EXEC language program environment for a variable whose name matches the string *name*.

**putexvar**      Search the calling EXEC language program environment for a variable whose name matches the string pointed to by *name*. If it is found, the value of that variable name is set to the string pointed to by *value* (the *value* string is copied to the EXEC environment). If the name is not found, a new variable is created in the EXEC environment with a value of *value*.

**Returns:**          **getexvar**      A pointer to a character string containing the value for the referenced variable name. A NULL pointer is returned if *name* is not found in the EXEC environment.

**Errors:**           **getexvar**      If the program was not invoked by an EXEC language program, the **getexvar** function returns a NULL pointer. It also returns a NULL pointer if *name* is not found in the EXEC program.

**putexvar**      No errors are detected by this function. If there is no calling EXEC language program, the function returns immediately.

**Notes:**            Do not use the ampersand ( & ) character in the name.

**Restrictions:**    The *name* string is case-sensitive. The maximum length allowed for *name* and *value* strings is 128 bytes.

The string pointed to by the return value is local to the **getexvar** function and will be reused the next time that **getexvar** is called.

**Conforms to:**           **getexvar**      q ANSI      q DOS      n THEOS      q POSIX  
                         **putexvar**      q ANSI      q DOS      n THEOS      q POSIX

**Example:**

EXAMPLE1.EXEC

&state\_name = California

&type EXEC: State name is &state\_name

example

&type EXEC: State name is now &state\_name

EXAMPLE.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <execvar.h>

main()
{
    char    name[32];
    char    value[32];
    char    * valptr;

    valptr = getexvar("0");          // see if EXEC exists
    if (valptr==NULL) {
        printf("C: No EXEC environment available\n");
        exit(254);
    }

    printf("C: EXEC var name to get: ");
    gets(name);
    valptr = getexvar(name);

    strcpy(value, valptr ? valptr : "");
    printf("\nC: %s variable contains %s\n",name, value);

    printf("C: New value for %s\n", name);
    gets(value);
    putexvar(name, value);

    printf("\nC: Value of %s after putexvar is %s\n", name,
        getexvar(name));
}
```

## 284 *getexvar, putexvar*

---

### Output:

```
>example1
EXEC: State name is California

>example
C: EXEC var name to get: state_name

C: state_name variable contains California
C: New value for state_name
West Virginia

C: Value of state_name after putexvar is West Virginia
EXEC: State name is now West Virginia

>example
C: No EXEC environment available

>
```



## getdate, getfiledate

Get the last change date of a file.

```
#include <stdlib.h>
```

```
int getdate ( const char * filename, time_t * date )
```

```
time_t getfiledate ( FDB * dirptr )
```

---

```
date » pointer to storage for file's date
```

```
dirptr » pointer to file's directory entry
```

```
filename » pointer to string containing file description
```

**Operation:**      **getdate**      The file directory entry for *filename* is examined and the date field is extracted and copied to *date* as a `time_t` value.

**getfiledate**      The file directory pointed to by *dirptr* is examined and the date field is extracted and returned as a `time_t` value.

**Returns:**            **getdate**      A success/fail indicator. A zero return value indicates success; a non-zero return value indicates failure and `errno` is set to a value indicating the specific error.

**getfiledate**      The file's last change date.

**Notes:**            The `getdate` function is used when all the program knows about a file is its path; `getfiledate` is used when the file's directory entry has already been retrieved with `_dirfdb`, `_get_sect` or `locate`.

The date/time retrieved by these functions is a `time_t` value which is the date/time expressed as the number of seconds since the beginning of 1970. Use one of the various time-related functions to convert the `time_t` value to another form, such as `ctime`, `gmtime` or `localtime`.

**Conforms to:**            **getdate**      q ANSI      q DOS      n THEOS      q POSIX  
                         **getfiledate**      q ANSI      q DOS      n THEOS      q POSIX

**See also:**            [putdate](#)

## getflat

Get the “flat file” portion of a file description.

```
#include <stdlib.h>
```

```
char * getflat ( char * path )
```

---

*path*                   »   pointer to string containing file’s complete path description

**Operation:**       The file description specified in *path* is analyzed and the first character that is not part of the path specification is determined.

**Returns:**         A pointer to the start of the flat file description in *path*.

**Notes:**           A *flat file* description is a file name specification that does not include any account name or directory specification. Only file-name, file-type, member-name and disk-letter are used in a flat file description.

*path* does not have to specify a file that exists. It does not even have to specify a valid file name.

**Conforms to:**       **getflat**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**         [getfn](#), [longfname](#), [\\_norm\\_fn](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void
main(int argc, char * argv[])
{
    char    buffer[256];
    char    * ptr;

    ptr = _norm_fn(argv[1], buffer);

    printf("\nFile name is \"%s\"",argv[1]);
    printf("\n_norm_fn is  \"%s\"",ptr);

    printf("\n\ngetflat is   \"%s\"",getflat(buffer));
}
```

**Output:**

```
>test sample.file
```

```
File name is "SAMPLE.FILE"
```

```
_norm_fn is  "/C32/SAMPLE.FILE:S"
```

```
getflat is   "SAMPLE.FILE:S"
```

```
>test account\subdir1\subdir2\my.library.member:d
```

```
File name is "ACCOUNT\SUBDIR1\SUBDIR2\MY.LIBRARY.MEMBER:D"
```

```
_norm_fn is  "ACCOUNT\SUBDIR1\SUBDIR2\MY.LIBRARY.MEMBER:D"
```

```
getflat is   "MY.LIBRARY.MEMBER:D"
```

```
>
```

**getfn**

Determine the full file name for a file.

```
#include <stdio.h>
char * getfn ( char _far * name )
```

---

*name*                    »   pointer to string containing file name

**Operation:**        The file indicated by *name* is located using the current environment values (current working directory, default library, disk search sequence, *etc.*). The full path and file name are generated, stored in the location *name* and returned as the value of the function call.

**Returns:**           A pointer to the resulting string. A return of a NULL pointer indicates that *name* cannot be found.

**Restrictions:**    The *name* argument should point to a buffer of at least FILENAME\_MAX bytes.

Do not use a *name* value of argv[n] as the transformation will be a longer string and the result destroys the contents of the program's stack.

**Conforms to:**                **getfn**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [getflat](#), [locate](#), [longfname](#), [\\_norm\\_fn](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    char fn[FILENAME_MAX];
    strcpy(fn, argv[1]);
    printf("\nThe path for \"%s\" ", fn);
    printf("is \"%s\".\n", getfn(fn));
    printf("Filename is now \"%s\"\n", fn);
}
```

---

**Output:**            Assuming that this program is run while the current working directory is C32 and the default library is SAMPLES.C32:

>test test

The path for "TEST" is "/C32/SAMPLES.C32.TEST:S".  
Filename is now "/C32/SAMPLES.C32.TEST:S"

## **`_get_help_filename, _get_menu_filename`**

Generate the complete file name for the help file or menu file associated with the current program.

```
#include <stdlib.h>
char * _get_help_filename ( void )
char * _get_menu_filename ( void )
```

**Operation:** Get the full path name of the file accessed with the last call to [openhlp](#) or [openmenu](#).

**`_get_help_filename`** Gets the full path name used by the last call to the [openhlp](#) function.

**`_get_menu_filename`** Gets the full path name used by the last call to the [openmenu](#) function.

**Returns:** A pointer to the path name string.

**Errors:** A NULL pointer is returned when there has been no prior call to [openhlp](#) or [openmenu](#).

**Restrictions:** These functions are declared in the [openhlp](#) and [openmenu](#) function modules. A program must reference the [openhlp](#) function or else the `_get_help_filename` function will be an “Unresolved symbol” during the program’s link phase. Similarly, a program must reference the [openmenu](#) function or the `_get_menu_filename` function will be unresolved.

These functions can only return a valid string after the respective [openhlp](#) or [openmenu](#) function is called.

**Conforms to:**

<code>_get_help_filename</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_get_menu_filename</code>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [openhlp](#), [openmenu](#)

---

**Example:** See [openhlp](#), [openmenu](#) on page 462.

**gethostbyaddr, gethostbyname**

Get host information corresponding to an address or a host-name.

```
#include <socket.h>
HOSTENT * gethostbyaddr ( const void * addr, int len, int type )
HOSTENT * gethostbyname ( const char * name )
```

---

<i>addr</i>	»	pointer to an address in network byte order
<i>len</i>	»	length of address (must be 4)
<i>name</i>	»	pointer to name of the host
<i>type</i>	»	type of address (must be PF_INET)

**Operation:** Both of these functions return a pointer to a HOSTENT structure corresponding to the host-address or the host-name.

**Returns:** Returns a pointer to a HOSTENT structure. If an error occurs, a NULL pointer is returned and the specific error number may be retrieved by using [TSAGetLastError](#).

**Errors:** When an error occurs the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAHOST_NOT_FOUND	Specified host was not found.
TSANO_DATA	Valid name, no data record of requested type.
TSANO_RECOVERY	Non-recoverable errors, FORMERR, REFUSED, NOTIMP.
TSATRY_AGAIN	Host not found or SERVERFAIL.
TSAEWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.

**Restrictions:** The pointer returned points to a structure used by the `gethostbyaddr`, `gethostbyname` and [gethostname](#) functions. The application must never attempt to modify this structure or to free any of its components.

**Conforms to:**

<b>gethostbyaddr</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>gethostbyname</b>	q ANSI	q DOS	n THEOS	q POSIX

**Example:** See the example for the function [socket](#).

## gethostname

Get the standard host-name for the local machine.

```
#include <socket.h>
int gethostname ( char * name, int namelen )
```

---

*name*                   »   pointer to buffer to receive host name  
*namelen*                »   length of name buffer

**Operation:** This routine returns the name of the local host into the buffer specified by the *name* parameter. The host-name is returned as a null-terminated string.

The form of the host-name is dependent on the THEOS Sockets implementation—it may be a simple host-name, or it may be a fully qualified domain name. However, it is guaranteed that the name returned will be successfully parsed by `gethostbyname`.

**Returns:** A zero. When an error is detected, `SOCKET_ERROR` is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is `SOCKET_ERROR` the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAEFAULT	The <i>namelen</i> argument is too small.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.

**Conforms to:**        **gethostname**    q ANSI        q DOS        n THEOS        q POSIX

**See also:**         [gethostbyaddr](#), [gethostbyname](#)

### **getkey**

Gets a keyword token from the SYSTEM.TEOS32.KEYWORD $n$  file.

```
#include <stdlib.h>
```

```
int getkey ( int keynbr, char * buffer )
```

---

<i>buffer</i>	»	pointer to storage location for keyword string
---------------	---	--

<i>keynbr</i>	»	number of keyword token to read
---------------	---	---------------------------------

**Operation:** Open the SYSTEM.TEOS32.KEYWORD $n$  file, if necessary. The keyword number *keynbr* is read into *buffer*.

**Returns:** The minimum abbreviation length for the keyword.

**Errors:** When the keyword cannot be found, a zero is returned.

**Notes:** The keywords file is left open by this function. Subsequent calls to `getkey` will not try to reopen it. When the program is finished reading keywords, it should use the [keyclose](#) function to close the keywords file.

**Restrictions:** *buffer* should be allocated for at least nine bytes.

**Conforms to:** `getkey` q ANSI q DOS n THEOS q POSIX

**See also:** [keyclose](#), [load\\_yn](#), [matcharg](#), [testarg](#)



**Example:**

```
#include <stdlib.h>

void
main(int argc, char * argv[])
{
    char    typekey[9];

    if (argc<2)
        syserr(1, 133, NULL);    // missing keyword
    if (*argv[1] != '(')
        syserr(1, 134, NULL);    // missing parenthesis
    if (!matcharg(argv[2], typekey, getkey(2, typekey)))
        syserr(1, 27, argv[2]);  // invalid option
    keyclose();

    ...
}
```

---

**Output:**

```
>test
Keyword missing or misspelled.

>test type
Missing parenthesis.

>test (type
Invalid option: "TIPE".

>test (tipe
Invalid option: "TIPE".
```

**getlang**

Get the currently defined language code.

```
#include <stdlib.h>
int getlang ( void )
```

**Operation:** The current language code from the System Communication Region (SCR) is returned.

**Returns:** The current language code.

**Notes:** The language code is normally a zero, even when your computer uses a language other than English.

The language code is used to form the complete file names for members of SYSTEM.HELP32*n* and SYSTEM.MENU32*n* and for the files SYSTEM.TEOS32.MESSAGE*n* and SYSTEM.TEOS32.KEYWORD*n*.

**Conforms to:**                    **getlang**    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    printf("\nThe current language code is %i.\n",getlang());
}
```

---

**Output:**

>test

The current language code is 0.

>test

---

## getlib

Get the current default library name.

```
#include <stdlib.h>
char * getlib ( void )
```

**Operation:** A copy of the current default library name is made and a pointer to this copy is returned.

**Returns:** A pointer to a copy of the current default library name.

**Notes:** The current library name may also be retrieved with the function `getenv( "LIBRARY" )`.

The `getlib` function may be useful when forming the complete file name of a member of the current library, but only if the default library name does not include the drive code specification.

**Restrictions:** The value is read into a static buffer internal to the `getlib` function. This buffer will be overwritten by the next call to `getlib`.

**Conforms to:** `getlib` q ANSI q DOS n THEOS q POSIX

**See also:** [getenv](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE * infile;

    printf("\nOpening file \"%s.%s\".", getlib(), "sample");
    infile = fopen("sample");
    ...
}
```

---

### Output:

>test

Opening file "SAMPLE.LIBRARY.sample".

**\_getline**

Accept one line of text from the console, with editing capabilities.

```
#include <stdio.h>
char * _getline ( char * buffer )
```

---

*buffer*                      »    pointer to input storage area

**Operation:**        The `_getline` function provides text line input and editing capabilities. Text input may come from the console keyboard or the EXEC stack. The standard input device is not used.

This function accepts one line of text from the operator, saves it in *buffer* and returns a pointer to *buffer*. It also saves a copy of the text entered in a local buffer. This copy can be used by the next call to `_getline` as a “line recall” string. (See **F3** in the following table.)

The `_getline` function has its own insert and replace modes with insert mode being the initial mode. This means that, initially, any characters that you type are inserted into the line at the cursor position. A replace mode is also available that allows you to type characters that replace the existing characters in the line.

While using the `_getline` function, there are several special editing keys that can be used. The following table describes all of the editing keys available with the `_getline` function.
















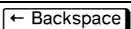




Key	Function
	Terminate entry and exit function.
 	When entered with no characters in the line, recalls last line of text entered with the <code>_getline</code> function.
 	Move left one character position.
 	Move right one character position.
  	Move to the beginning of the text line.
  	Move to the end of the text line.
 	Delete the character under the cursor.
	Delete the character to the left of the cursor.
 	Erase all characters from the present cursor position to the end of the line.
 	Erase entire line. Cursor positioned to beginning of line.

Table 4: `_getline` Function Editing Keys

Key	Function
<div><div>F2</div>,c</div> <div>Ctrl+W,c</div>	Invoke a search operation. The next character typed (c) is the character searched for, starting at the current cursor location. If the character exists in the remainder of the line, then the cursor is positioned to it. If there is no matching character, then the cursor is positioned to the end of the line.
<div>Insert</div> <div>Ctrl+R</div>	Switch between insert mode and replace mode or between replace mode and insert mode.

Table 4: `_getline` Function Editing Keys

- Returns:** A pointer to buffer which contains the string of characters input from the console or the EXEC stack.
- Notes:** When the line of text comes from the EXEC stack, it is not saved as a local copy and therefore cannot be recalled when the `_getline` function is called again from this program.
- Defaults:** The first time that `_getline` is called there is not previous string and the recall key will not function.
- Initially, `_getline` is in insert mode each time that it is called.
- Restrictions:** The local buffer used for prior line recall is 256 bytes in length. Only the first 255 characters of the previous input line are saved in this buffer.
- Conforms to:** `_getline`    q ANSI    q DOS    n THEOS    q POSIX
- See also:** [cgets](#), [gets](#), [wEdit](#)

## getll, getpl

Get the attached line length or page length for the console or one of the printers.

```
#include <stdio.h>
int getll ( int lub )
int getpl ( int lub )
```

---

*lub*                      »    logical unit block number for console or printer

**Operation:**        The *lub* is tested to determine if it is the console or the printer. If it is neither, then a zero is returned. Otherwise, the attached line length or page depth is retrieved.

**Returns:**            **getll**            The attached line length for the console or one of the printers.

**getpl**            The attached page depth for the console or one of the printers.

If *lub* is not the lub for the console or one of the printers, a zero is returned. If the device is not attached, a zero is returned.

**Notes:**            When getting the line or page length of the console, these functions actually return the size of the currently selected window. If a window other than window 0 is selected, the line and page lengths will be for that active window, not the console's physical screen.

The line or page length is base 1.

Refer to Appendix D: "Logical Unit Block Numbers" on page 769 for a list of lub numbers and standard names for those numbers.

**Conforms to:**            **getll**        q ANSI        q DOS        n THEOS        q POSIX

**getpl**        q ANSI        q DOS        n THEOS        q POSIX

**See also:**            [GetScreenSize](#), [SetScreenSize](#)

---

### Example:

```
#include <stdio.h>
#include <lub.h>

void main()
{
    printf("\nConsole attached with line length of %i",
        getll(CONSOLE));
    printf("\nConsole attached with page depth of %i",
        getpl(CONSOLE));
}
```

---

### Output:

>test

Console attached with line length of 80  
Console attached with page depth of 24

## getlogin

Retrieves the account name that you are currently logged on to.

```
#include <stdlib.h>
char * getlogin ( void )
```

**Operation:** A local copy is made of the currently logged on account name. This copy has all trailing spaces removed.

**Returns:** A pointer to the currently logged on account name.

When the program is operating as a background user, a pointer to a NULL string is returned.

**Notes:** This function is similar to the [cuserid](#) function, except that it does not have the capability of copying the login name for you.

**Restrictions:** The copy of the trimmed account name is local to the `getlogin` function and will be reset each time that `getlogin` is called.

**Conforms to:** `getlogin` q ANSI q DOS n THEOS q POSIX

**See also:** [cuserid](#), [getuid](#), [logname](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
    FILE * logfile;
    char account[9];

    logfile = fopen("log.file","wa");
    strcpy(account, getlogin());
    if (account[0]==0)
        strcpy(account,"Phantom");
    fprintf(logfile, "User name = %s (%i)\n",account,getuid());
    fclose(logfile);
}
```

---

### Output:

```
>test
```

```
>start test
```

```
>list log.file
```

```
User name = SAMPLES (2)
```

```
User name = Phantom (2)
```

```
>
```

### getlub

Get the unit control block (ucb) number associated with a logical unit block (lub) number.

```
#include <stdlib.h>
int getlub ( int lub )
```

---

*lub*                    »    logical unit block number for device

**Operation:**        The number of the ucb attached to the device *lub* is returned.

**Returns:**           The number of the ucb attached to *lub*.

When there is no device attached to *lub*, a 255 is returned.

**Notes:**            Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of lub numbers.

The include file LUB.H defines names for each of the lubs supported.

**Conforms to:**                **getlub**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [getucb](#), [GetUCB](#)

---

#### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <lub.h>

void main()
{
    if (getlub(COM1)==255)
        syserr(20, 20, NULL);        // device not attached message
    ...
}
```



***\_getmem, \_putmem***

Allocate memory from the system memory pool.

```
#include <getmem.h>
unsigned short _getmem ( size_t len, int type )
void _putmem ( unsigned segment )
```

---

<i>len</i>	»	size of requested memory
<i>segment</i>	»	memory segment selector
<i>type</i>	»	type of memory requested

**Operation:**     **\_getmem**   Allocate a contiguous region of memory, *len* bytes in length, from the system memory pool. The privilege level, access rights, expansion direction, *etc.* are specified by the bit-mapped *type* argument.

**\_putmem**   Deallocates the memory specified by *segment*.

**Returns:**       **\_getmem**   The segment address of the memory allocated.

**Errors:**       **\_getmem**   When there is insufficient memory available to satisfy the request, a zero value is returned.

**Notes:**        **\_getmem**   Allocated memory is cleared to zeros.

**\_putmem**   When a program is exited, all local-access memory allocated by the program is automatically deallocated. Global-access memory (memory allocated by **\_getmem**) is not deallocated until an explicit **\_putmem** call is used or until the system is restarted.

The *type* argument is bit-mapped with the following meanings for the bits:

<b><i>bits</i></b>	<b><i>Meaning</i></b>
1-0	Requested privilege level
2	Access (global or local)
3	Expansion direction
4	4K granuals
5	Set big bit in selector
6	Align on 4096 byte boundary
7	Lock the memory after allocate
8	Align on 64K byte boundary
9	Align on 128K byte boundary
10	Align on 256 byte boundary
11	Align on 512 byte boundary

For convenience and self-documentation, the GETMEM.H file defines names for the various values that can be used in this *type* field. Merely OR the desired names together, along with the requested privilege level.

<i>Name</i>	<i>Meaning</i>
MEM_GDT	Global memory
MEM_LDT	Local memory
MEM_EXPAND_DOWN	Expand down
MEM_BIG	Set big bit in selector
MEM_LOCK	Lock memory after allocation
MEM_GRANULARITY	4K granuals
MEM_ALIGN_256	Align on 256 byte boundary
MEM_ALIGN_512	Align on 512 byte boundary
MEM_ALIGN_4k	Align on 4K byte boundary
MEM_ALIGN_64k	Align on 64K byte boundary
MEM_ALIGN_128k	Align on 128K byte boundary
SELECTOR_ONLY	Allocate a selector only

The actual privilege level assigned to the memory is always the greater of the requested privilege level and the current privilege level of the program.

The access type specifies whether other users will be able to read and write to this memory area (assuming that they know its location). An access bit of 0 means that it is globally accessible; an access bit of 1 means that only this program may access it.

The expansion direction bit controls whether the memory is accessed from the bottom up or the top down. Normal memory expands up (value 0). A value of 1 for this bit means that the memory direction is down. Downward expanding memory is accessed with negative offsets as the base address of the memory refers to its end.

**Defaults:** The standard *type* value to use should be MEM\_LDT. This specifies local memory expanding up with the same privilege as your program.

**Restrictions:** The SELECTOR\_ONLY *type* value is reserved for device-driver and system level programmers.

Do not use \_putmem with a segment that your program did not allocate with \_getmem.

**Conforms to:**        *\_getmem*    q ANSI    q DOS    n THEOS    q POSIX  
                      *\_putmem*    q ANSI    q DOS    n THEOS    q POSIX

**See also:**        [\\_alloca](#), [calloc](#), [malloc](#), [mem\\_avail](#), [shared](#)

**Example:**

```
#include <string.h>
#include <getmem.h>

void
main()
{
    short extra_data;
    char stuff[200];

    ...
    extra_data = _getmem(0x1000, MEM_LDT);
    _fmemcpy(_MK_FP(extra_data, 0), stuff, sizeof(stuff));
    ...

    _putmem(extra_data);
}
```

**getmpid, getpid, getppid**

Get the process number of your program, your program’s main task or your program’s parent task.

```
#include <stdlib.h>
int getmpid ( int pid )
int getpid ( void )
int getppid ( void )
```

---

*pid*                    »    process number

**Operation:**        **getmpid**    Determine the main task’s process number for the task in process *pid*.

**getpid**     Determine your program’s process number.

**getppid**    Determine the program’s parent process number.

**Returns:**        **getmpid**    The main task’s process number. If the program in *pid* is not a subtask to another task, the program’s own process number is returned.

**getpid**     Your program’s process number.

**getppid**    The program’s parent task process number. If the current program is not a subtask to another task, the program’s own process number is returned.

<b>Conforms to:</b>	<b>getmpid</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>getpid</b>	q ANSI	q DOS	n THEOS	n POSIX
	<b>getppid</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**        [fork](#), [forktask](#)

**Example:**

```
#include <stdlib.h>
#include <stdio.h>
#include <process.h>

void
main()
{
    int childpid;

    if ((childpid=fork()) == 0)        // return zero?
        goto subtask_start;

    if (childpid==-1) {
        printf("Subtask could not be started.\n");
        exit(253);
    }

    // resume parent task code

    ...

    killtask(childpid);
    exit(0);

subtask_start:

    // subtask code
    printf("\nThis task is using pid %i",getpid());
    printf("\nThe parent task is pid %i",getppid());
    printf("\nThe main task is pid  %i",getmpid(getpid()));
    ...

}
```

---

**Output:**

>example

```
This task is using pid 20
The parent task is pid 5
The main task is pid  5
```

**getmsec**

Get the current system time to the nearest millisecond.

```
#include <timer.h> or <sc.h>
char * getmsec ( char * buffer )
```

---

*buffer*                    »   pointer to string storage area

**Operation:**        The current system time is retrieved and formatted as a string. The resulting string is stored in the location pointed to by *buffer*.

**Returns:**           A pointer to the formatted time string.

**Notes:**            The format of the time string is

                  hh:mm:ss.mmm

The hours are always converted using the 24-hour notation.

**Restrictions:**    The *buffer* should be an area allocated for at least 13 bytes. Whether it is or not, 13 bytes of data will be copied to that location.

The time returned by this function may not be the same time as returned by the [gettime](#) or [time](#) functions. Other functions use the time as supported by any hardware time-of-day components. The time reported by the `getmsec` function uses a software timer that is initialized when the computer is booted. Over long periods of time, this timer may show a difference from the hardware time-of-day timer.

**Conforms to:**            **getmsec**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [getdate](#), [gettime](#), [time](#)

---

**Example:**            Refer to [getdate](#) example on page 278.



**`_getosver`**

Get the complete operating system version number.

```
#include <sc.h>
long _getosver ( void )
```

**Operation:** The operating system version number is computed as a long integer and returned.

**Returns:** The operating system version number.

**Notes:** The version number returned is coded as a six-digit long integer. The first two high-order digits represent the version number, the middle two digits represent the subversion number, and the last two digits represent the build number, if appropriate.

The version number reported by this function is the true version number of the operating system. It is not a mere copy of the plain text version number maintained in memory.

**Restrictions:** This function only operates properly when the operating system is THEOS 32 Version 4 or higher. When the version is less than 4, a number is returned that is less than 40000 but it will not necessarily be the version number.

**Conforms to:** `_getosver` q ANSI q DOS n THEOS q POSIX

**See also:** [getver](#) function, [\\_osmajor](#), [\\_osminor](#), [\\_osversion](#) variables

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>

void main()
{
    if (_getosver() < 40000)
    {
        printf("\nVersion 4 or higher required!");
        exit(255);
    }
    ...
}
```



## getpeername

Get the address of the peer to which a socket is connected.

```
#include <socket.h>
int getpeername ( SOCKET s, SOCKADDR * name, int * namelen )
```

---

<i>name</i>	»	Pointer to buffer for socket address
<i>namelen</i>	»	Pointer to length of socket address buffer
<i>s</i>	»	Socket number

**Operation:** getpeername retrieves the name of the peer connected to the socket *s* and stores it in the *name*. It is used on a connected datagram or stream socket.

**Returns:** A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is SOCKET\_ERROR, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEADDRINUSE	The specified address is already in use.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAEFAULT	The <i>namelen</i> argument is incorrect.
TSAENOTCONN	The socket is not connected.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.

**Notes:** On return, the *namelen* argument contains the actual size of the *name* returned, in bytes.

**Conforms to:**      **getpeername**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**      [bind](#), [getsockname](#), [socket](#)

### getpriv

Get the user's current privilege level.

```
#include <stdlib.h>
int getpriv ( void )
```

**Operation:** The privilege level of the current user is returned.

**Returns:** The privilege level.

**Conforms to:** **getpriv** q ANSI q DOS n THEOS q POSIX

---

#### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    printf("\nYou are logged onto account \"%s\"", getlogin());
    printf(" with a privilege level of \"%i\".\n", getpriv());
}
```

---

#### Output:

```
>test
```

```
You are logged onto account "SAMPLES" with a privilege level of "5."
```

```
>
```

## getprotobyname, getprotobynumber

Get protocol information corresponding to a protocol name or number.

```
#include <socket.h>
PROTOENT * getprotobyname ( const char * name )
PROTOENT * getprotobynumber ( int proto )
```

---

*name*                   »   pointer to a protocol name  
*proto*                   »   protocol number

**Operation:** Both of these functions return a pointer to a PROTOENT structure matching the given *name* or *proto*.

**Returns:** Returns a pointer to a PROTOENT structure. If an error occurs, a NULL pointer is returned and the specific error number may be retrieved by using [TSAGetLastError](#).

**Errors:** When an error occurs the possible codes returned by [TSAGetLastError](#) include:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSANO_DATA	Valid name, no data record of requested type.
TSANO_RECOVERY	Non-recoverable errors, FORMERR, REFUSED, NOTIMP.

**Restrictions:** The pointer returned points to a structure that is allocated by the THEOS Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per process, and so the application should copy any information that it needs before issuing any other THEOS Sockets API calls.

**Conforms to:**

<b>getprotobyname</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>getprotobynumber</b>	q ANSI	q DOS	n THEOS	q POSIX

**gets**

Gets a line from the stdin device.

```
#include <stdio.h>
char * gets ( char * buffer )
```

---

*buffer*                    »    pointer to storage

**Operation:**        Using the [fgets](#) function (which uses the [fgetc](#) function), a line or string of characters is read from the standard input device `stdin` and saved in the location *buffer*. The string of characters consists of all of the characters up to and including the first new-line character. This new-line character is replaced with the string terminating character `'\0.'`

**Returns:**            A pointer to *buffer*. A NULL pointer is returned if an end-of-file or error condition is detected. Use the `ferror` or `feof` to determine which one occurred.

**Notes:**             Assuming the same input source, the difference between `gets`, [fgets](#) and [fgetsn](#) is that `gets` terminates on a new-line character only and replaces the new line with a null terminator; [fgets](#) terminates on new line or *n*-1 characters; [fgetsn](#) terminates on *n* characters only.

**Defaults:**          Unless redirected, `stdin` is the console keyboard.

**Restrictions:**      *buffer* must point to an area large enough to hold the longest string that might be accepted.

**Conforms to:**                **gets**      n ANSI      n DOS      n THEOS      n POSIX

**See also:**            [fgets](#), [fgetl](#), [fgets](#), [fgetsn](#), [fgetw](#), [getc](#), [getchar](#)

## GetScreenSize, SetScreenSize

Get the attached screen size of the console or set it.

```
#include <wmapi.h>
int GetScreenSize ( int * width, int * height )
int SetScreenSize ( int width, int height )
```

---

*height*                   » height of display, in lines  
*width*                    » width of display, in characters

**Operation:**       **GetScreenSize** Get the currently attached console screen size. The console width and height are stored in the locations pointed to by *width* and *height* respectively. Also tests the console device-driver to see if it supports multiple screen sizes.

**SetScreenSize** The attached width and height of the console screen is changed to *width* and *height*. If the console device and its device-driver supports multiple screen sizes, then the device-driver is instructed to change the size of the display, using different character size if necessary.

**Returns:** Both functions return a zero to indicate that the console device and its device-driver support multiple screen sizes. A non-zero return indicates that the device and/or its device-driver does not support multiple screen sizes.

**Notes:**       **SetScreenSize** When the console or the device-driver does not support multiple screen sizes, the attached width and height are always set to *width* and *height*. When the console and device-driver do support multiple screen sizes, the attached width and height are set to the size that the device-driver used.

When the device-driver supports multiple screen sizes, this function only passes the requested *width* and *height* to the device-driver. The driver will try to set the screen to the requested size using the following priorities:

1. Use the requested *width* and *height* if that combination is supported.
2. Using the requested *width* vary the requested *height* by +1, -1, +2 and -2 until a supported combination is found.
3. Use the default screen size of 80 × 24.

For instance, if the console supports screen sizes 40 × 24, 80 × 24 and 80 × 42:

### 314 *GetScreenSize, SetScreenSize*

---

<i>Request</i>	<i>Size used</i>
80 × 42	80 × 42
80 × 43	80 × 42
80 × 45	80 × 24
50 × 24	80 × 24
40 × 22	40 × 24

**Defaults:**           The default screen size is 80 × 24.

**Conforms to:**       **GetScreenSize**   q ANSI     q DOS     n THEOS   q POSIX  
                  **SetScreenSize**   q ANSI     q DOS     n THEOS   q POSIX

**See also:**           [getll, getpl](#)

---

**Example:**

```
#include <wmap.h>

void
main()
{
    int      width,
            height;
    int      rc;

    rc = GetScreenSize(&width, &height);

    if (rc==0 && width<40)
        rc = SetScreenSize(80,59);

    ...
}
```

## **\_get\_sect**

Locate and read a file's directory entry.

```
#include <sc.h>
```

```
FDB * _get_sect ( char _far * filename, char buffer[256], long * sect )
```

---

<i>buffer</i>	»	pointer to storage buffer
---------------	---	---------------------------

<i>filename</i>	»	pointer to string containing file description
-----------------	---	---

<i>sect</i>	»	pointer to sector number storage
-------------	---	----------------------------------

**Operation:** Using the file name specified in *filename*, the attached disks are searched in the default disk search sequence.

If *filename* specifies a drive code, then only that one disk is searched.

If *filename* is a simple name (i.e., does not contain a period character), then the current default library is searched for that member name.

*filename* may include explicit path specifications, either relative to the current directory or from the root.

When the directory for *filename* is found, the sector containing the directory entry is read into *buffer*, *sect* is set to the sector number containing the directory entry, and the location of the directory entry in *buffer* is returned.

**Returns:** A pointer to the directory entry in *buffer*. The sector number is also returned by copying it to the variable *sect*.

When the file is not found, a NULL pointer is returned.

**Notes:** This function is used by the DISK *filename* utility command.

**Restrictions:** This function is only available when used on a THEOS 32 Version 4 system or later.

To locate a file owned by another account, you must specify the account name in the path specifications of *filename*.

*buffer* must point to a buffer of at least 256 bytes.

**Conforms to:**        \_get\_sect   q ANSI    q DOS    n THEOS   q POSIX

**See also:**        [locate](#)

### Example:

```
#include <stdio.h>
#include <sc.h>

void
main()
{
    char    buffer[256];
    long    sector;
    FDB     * myfile;

    if (myfile = _get_sect("my.data", buffer, &sector)) {
        printf("\nThe file \"my.data\" directory entry at sector
               %,li.",sector);
        printf("\nCurrent file size is %,li.", myfile->filesize);
    }
    else
        printf("\nFile not found.");
}
```

---

### Output:

>example

The file "my.data" directory entry at sector 94,522.  
Current file size is 48.

>



## getservbyname, getservbyport

Get service information corresponding to a service name and protocol or a port and protocol.

```
#include <socket.h>
SERVENT * getservbyname ( const char * name, const char * proto )
SERVENT * getservbyport ( int port, const char * proto )
```

---

<i>name</i>	»	pointer to service name
<i>port</i>	»	port number in network byte order
<i>proto</i>	»	pointer to a protocol name

**Operation:** **getservbyname** Returns a pointer to a SERVENT structure which contains the name(s) and service number that correspond to the given service *name* and *proto*.

**getservbyport** Returns a pointer to a SERVENT structure which contains the name(s) and service number that correspond to the given service *port* and *proto*.

**Returns:** Returns a pointer to a SERVENT structure. If an error occurs, a NULL pointer is returned and the specific error number may be retrieved by using [TSAGetLastError](#).

**Errors:** When an error occurs the possible codes returned by [TSAGetLastError](#) include:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSANO_DATA	Valid name, no data record of requested type.
TSANO_RECOVERY	Non-recoverable errors, FORMERR, REFUSED, NOTIMP.

**Notes:** If *proto* is NULL, getservbyname or getservbyport returns the first service entry for which the *name* matches the *s\_name* or one of the *s\_aliases* or the first service entry for which the *port* matches the *s\_port*. Otherwise, getservbyname matches both the *name* and the *proto* and getservbyport matches both the *port* and *proto*.

**Restrictions:** The pointer that is returned points to a structure which is allocated by the THEOS Sockets library. The application must never attempt to modify this structure or to free any of its components. Furthermore only one copy of this structure is allocated per process, and so the application should copy any information that it needs before issuing any other THEOS Sockets API calls.

**Conforms to:**

<b>getservbyname</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>getservbyport</b>	q ANSI	q DOS	n THEOS	q POSIX

**getsockname**

Gets the local name for a socket.

```
#include <socket.h>
int getsockname ( SOCKET s, SOCKADDR * name, int * namelen )
```

---

<i>name</i>	»	Pointer to buffer for socket address
<i>namelen</i>	»	Pointer to length of socket address buffer
<i>s</i>	»	Socket number

**Operation:** getsockname retrieves the current name for the specified socket descriptor in *name*. It is used on a bound or connected socket specified by *s*. The local association is returned. This call is especially useful when a [connect](#) call has been made without doing a [bind](#) first; this call provides the only means by which you can determine the local association that has been set by the system.

**Returns:** A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is SOCKET\_ERROR, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEFAULT	The <i>namelen</i> argument is incorrect.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.
TSAEINVAL	The socket has not been bound to an address with bind.

**Notes:** On return, the *namelen* argument contains the actual size of the *name* returned in bytes.

If a socket was bound to INADDR\_ANY, indicating that any of the host's IP addresses should be used for the socket, getsockname will not necessarily return information about the host IP address, unless the socket has been connected with [connect](#) or [accept](#). A THEOS Sockets application must not assume that the IP address will be changed from INADDR\_ANY unless the socket is connected. This is because for a multi-homed host the IP address that will be used for the socket is unknown unless the socket is connected.

**Conforms to:**      **getsockname**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**        [bind](#), [getpeername](#), [socket](#)

## getsockopt

Gets a socket option for a specific socket.

```
#include <socket.h>
int getsockopt ( SOCKET s, int level, int optname, void * optval, int * optlen )
```

---

<i>level</i>	»	level for option
<i>optlen</i>	»	Pointer to the size of the optval buffer
<i>optname</i>	»	Socket option
<i>optval</i>	»	Pointer to buffer for returned option value
<i>s</i>	»	Socket number

**Operation:** The current value for a socket option associated with a socket is retrieved and stored in *optval*.

**Returns:** A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is SOCKET\_ERROR the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAEFAULT	The <i>namelen</i> argument is incorrect.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSANOPROTOOPT	The option is unknown or unsupported. In particular, SO_BROADCAST is not supported on sockets of type SOCK_STREAM, while SO_ACCEPTCONN, SO_CONTLINGER, SO_KEEPAIVE, SO_LINKER and SO_OOINLINE are not supported on sockets of type SOCK_DGRAM.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.

**Notes:** Options may exist at multiple protocol levels, but they are always present at the uppermost “socket” level. Options affect socket operations, such as whether an operation blocks or not, the routing of packets, out-of-band data transfer, *etc.* The value associated with the selected option is returned in the buffer *optval*. The integer pointed to by *optlen* should originally contain the size of this buffer; on return, it will be set to the size of the value returned. For SO\_LINGER, this will be the size of a struct linger; for all other options it will be the size of an integer.

If the option was never set with [setsockopt](#), then getsockopt returns the default value for the option.

The only supported *level* values are SOL\_SOCKET and IPPROTO\_TCP.

The following options are supported for `getsockopt`. The Type identifies the type of data addressed by *optval*.

<i>Value</i>	<i>Type</i>	<i>Meaning</i>
SO_ACCEPTCONN	BOOL	Socket is listening.
SO_BROADCAST	BOOL	Socket is configured for the transmissions of broadcast messages.
SO_DEBUG	BOOL	Debugging is enabled.
SO_DONTLINGER	BOOL	If true, the SO_LINGER option is disabled.
SO_DONTROUTE	BOOL	Routing is disabled.
SO_ERROR	int	Retrieve error status and clear.
SO_KEEPALIVE	BOOL	Keep-alives are being sent.
SO_LINGER	struct LINGER far *	Returns the current linger options.
SO_OOBINLIN	BOOL	Out-of-band data is being received in the normal data stream.
SO_RCVBUF	int	Buffer size for receives.
SO_REUSEADDR	BOOL	The socket may be bound to an address which is already in use.
SO_SNDBUF	int	Buffer size for sends.
SO_TYPE	int	The type of the socket (e.g. SOCK_STREAM).

BSD options not supported for `getsockopt` include:

<i>Value</i>	<i>Type</i>	<i>Meaning</i>
SO_RCVLOWAT	int	Receive low water mark.
SO_RCVTIMEO	int	Receive timeout.
SO_SNDLOWAT	int	Send low water mark.
SO_SNDTIMEO	int	Send timeout.
IP_OPTIONS		Get options in IP header.
TCP_MAXSEG	int	Get TCP maximum segment size.

**Conforms to:**            **getsockopt**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [setsockopt](#), [socket](#)

## gettime

Get the current system time.

```
#include <timer.h> or <sc.h>
char * gettime ( char * buffer )
```

---

*buffer*                    »    pointer to string storage area

**Operation:**        The current system time is retrieved and formatted as a string. The resulting string is stored in the location pointed to by *buffer*.

**Returns:**            A pointer to the formatted time string.

**Notes:**             The format of the time string is

hh:mm:ss

The hours are always converted using the 24-hour notation.

**Restrictions:**     The *buffer* should be an area allocated for at least nine bytes. Whether it is or not, nine bytes of data will be copied to that location.

**Conforms to:**                **gettime**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [getdate](#), [getmsec](#), [time](#)

---

**Example:**            Refer to the [getdate](#) example on page 278.

**getucb, GetUCB**

Get a pointer to a device’s unit control block (ucb).

```
#include <stdlib.h>
UCB * getucb ( int lub )
UCB far * GetUCB ( int lub )
```

---

*lub*                      »    logical unit block number of device

**Operation:**            Both of these functions return a pointer to the unit control block attached to the device specified by *lub*.

- getucb**

Compute and return a pointer to the ucb relative to the start of the nucleus memory segment. This value can be used with the [\\_peeknucb](#), [\\_peeknucd](#), [\\_peeknucw](#) and [\\_pokenucb](#), [\\_pokenucd](#), [\\_pokenucw](#) functions.
- GetUCB**

Compute and return a far pointer to the ucb. This value can be used with the far memory functions.

**Returns:**            **getucb**            A pointer to the ucb that is the offset within the nucleus memory segment.

**GetUCB**            A far pointer to the ucb.

A NULL pointer is returned if the device is not attached or if *lub* is invalid.

**Notes:**             Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of lub numbers.

When the GetUCB function is called by an application program, the far pointer returned may only be used to read ucb elements. When called by a device-driver program, the far pointer may be used to read or write ucb elements. This difference in usability is due to the fact that a device-driver operates as an extension to the operating system.

<b>Conforms to:</b>	<b>getucb</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>GetUCB</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**            [getlub](#)

**Example:**

```
#include <stdlib.h>
#include <ucb.h>
#include <peek.h>
#include <conio.h>

void main()
{
    UCB * ucb;

    display("\nThis is a test !");

    ucb = getucb(27);
    at(peeknuc(&ucb->curcol)-1, peeknuc(&ucb->curline)+1);
    putchar('^');
}
```

---

**Output:**

>test

This is a test !  
                  ^

>

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <lub.h>

void
main() {
    UCB _far *ucb;

    ucb = GetUCB(COM1);

    if (!ucb) {
        puts("COM not attached");
        exit(1);
    }

    switch (ucb->option & UCB_FLOW_MASK) {
        case UCB_FLOW_NONE:
            printf("COM1 has no flow control\n");
            break;
        case UCB_FLOW_XON:
            printf("COM1 uses XON/XOFF flow control\n");
            break;
        case UCB_FLOW_XPC:
            ...
        ...
    }
}
```

### getuid

Get the account number of the account that the user is logged onto.

```
#include <stdlib.h>
int getuid ( void )
```

**Operation:** The account number of the account that is currently logged onto is returned.

**Returns:** The user account id.

**Notes:** The account number is returned even when the program is operating as a background task, unlike the [cuserid](#) and [getlogin](#) functions.

**Conforms to:** **getuid** q ANSI q DOS n THEOS q POSIX

**See also:** [cuserid](#), [getlogin](#), [logname](#)

---

#### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE * logfile;

    logfile = fopen("log.file","a");
    fprintf(logfile, "User name = %s (%i)\n",getlogin(),getuid());
    fclose(logfile);
}
```



## getver

Get the operating system name, version number or serial number, or get the program version number or version date.

```
#include <stdlib.h>
```

```
void getver ( int type, char * string )
```

---

*string*                   »   pointer to storage location

*type*                    »   value specifying type of request

**Operation:**       This function can return one of five different values, depending upon the value of the *type* argument.

**type=0**       Gets the operating system version number. The version number is copied to *string* as a string of characters.

**type=1**       Gets the current program version number. The program version number is copied to *string* as a string of characters.

**type=2**       ]Gets the operating system name. The name of the operating system is copied to *string*. For instance, "THEOS 32."

**type=3**       Gets the operating system serial number. The serial number of the operating system is concatenated onto the end of the *string*.

**type=4**       Get the current program version date. The program version date is copied to *string*.

**Returns:**       This function does not have a return value.

**Notes:**       For all *types* except 3, the value is copied to *string*. When *type*=3, the value is concatenated onto the end of the current value in *string*.

**Restrictions:**   For *type* values of 1 and 4, the program must have a `#pragma prog` defined specifying the version number.

**Conforms to:**       **getver**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**       [\\_getosver](#) function, [\\_osmajor](#), [\\_osminor](#), [\\_osversion](#) variables

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>
#include <string.h>

#pragma prog 12.34x

void main()
{
    char buffer[101];

    printf("\nThe OS version is %li",_getosver());
    printf("\nosmajor = %i, osminor = %i, osversion=%i",
        _osmajor, _osminor, _osversion);

    getver(0,buffer);
    printf("\nThe OS version is %s",buffer);
    getver(2,buffer);
    printf("\nThe OS name is %s",buffer);
    strcat(buffer, ", ");
    getver(3,buffer);
    printf("\nThe OS serial is %s",buffer);

    getver(1,buffer);
    printf("\nThe program version is %s",buffer);
    getver(4,buffer);
    printf("\nThe program date is %s",buffer);
}
```

---

### Output:

```
>test

The OS version is 40000
osmajor = 4, osminor = 0, osversion=1024
The OS version is 4.0
The OS name is THEOS 32
The OS serial is THEOS 32, 102-12345
The program version is 12.34x
The program date is 04JUL96

>
```

## getvol

Gets the disk or tape volume label for an attached device.

```
#include <stdlib.h>
char * getvol ( int lub )
```

---

*lub*                    »    logical unit block number of device

**Operation:**        The disk volume name or the tape volume name, whichever applies for *lub*, is determined. A volume name is the name of the disk/tape assigned to the unit when it was last formatted. This is a string of up to nine characters, including the null-terminating character.

**Returns:**            A pointer to the volume name. A NULL pointer is returned if *lub* is not a disk or tape device.

**Notes:**             If there is no disk or tape mounted in the device, the operator is asked to mount a disk or tape.

See “Logical Unit Block Numbers” on page 769 for a listing of lub numbers for disks and tapes.

**Restrictions:**      The volume name pointed to on return is local to the `getvol` function. It is overwritten each time that `getvol` is called by your program.

**Conforms to:**            `getvol`    q ANSI    q DOS    n THEOS    q POSIX

## getw

Read an integer from an open file.

```
#include <stdio.h>
unsigned getw ( FILE * file )
```

---

*file*                      »    pointer to open file's FCB

**Operation:**        Reads a short or two-byte unsigned integer value from *file*.

No alignment of data is presumed. That is, from the current file position, exactly two bytes of data are read and returned.

After the integer is read, the current position pointer for file points to the byte immediately following the integer.

**Returns:**            The integer read. A return of EOF might indicate an error (see notes).

**Notes:**             A return of EOF might not indicate end-of-file because EOF is also a legitimate value for a binary integer. The [feof](#) functions must be used to determine if an error has actually occurred.

**Restrictions:**      Usage of this function is not portable because the size of an unsigned integer might be different on other implementations of C or on other computers. You should restrict usage of this function to reading from work files created earlier in the same program.

**Conforms to:**                **getw**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**                [fgetc](#), [fgetl](#), [fgets](#), [fgetsn](#), [fgetw](#), [getc](#), [getchar](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE * infile;
    unsigned value;

    if (!(infile=fopen("data.file","r")))
        exit(ferror());

    value = getw(infile);

    if (feof(infile))
        printf("End of file\n");
    else
        printf("Value read was %u.\n",value);

    fclose(infile);
}
```

---

## gmtime

Converts a UTC calendar time value into a time structure.

```
#include <time.h>
struct tm * gmtime ( const time_t * timer )
```

---

*timer*                   »   pointer to calendar time

**Operation:**       The gmtime function converts the calendar time pointed to by *timer* into a structure of type `tm`, expressed as Universal Time Coordinate (UTC), formally known as Greenwich Mean Time. The [tzset](#) function is called. The converted `tm` structure is in static storage, local to the gmtime function and is re-used each time that gmtime is called.

**Returns:**         A pointer to the converted `tm` structure.

**Notes:**           The calendar time value *timer* is assumed to be the current UTC time.

When *timer* is NULL, the current UTC time is used.

**Restrictions:**   Similar to gmtime and [mktime](#), the `tm` structure used by this function is local to the function and is re-used each time that the function is called.

**Conforms to:**        **gmtime**    n ANSI       n DOS       n THEOS    n POSIX

**See also:**           [asctime](#), [clock](#), [ctime](#), [difftime](#), [localtime](#), [mktime](#), [strftime](#), [time](#), [tzset](#)

---

**Example:**         Refer to the [strftime](#) example on page 612.

**has, hascolor**

Determine if console is capable of performing a specific screen-editing operation or a video attribute, or if it displays color.

```
#include <stdio.h>
int has ( int attribute )
int hascolor ( void )
```

---

*attribute* » screen editing or video attribute code

- Operation:**

The class code for the current console is analyzed.

**has** The class code definition for *attribute* is examined.

**hascolor** The class code definition for color changes is examined.
- Returns:**

**has**

A return value of 1 indicates that the console can perform the action in the normal manner.

A return value of 2 indicates that the console can perform the action but that it takes a display position.

A return value of 0 indicates that the console cannot perform the action.

**hascolor** A non-zero value indicates that the console supports colors; a zero value indicates that colors are not defined for the console.
- Notes:**

*attribute* names are defined in the CRT.H header file.
- Conforms to:**

<b>has</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>hascolor</b>	q ANSI	q DOS	n THEOS	q POSIX
- See also:**

[crt](#), [crtcolor](#)

**Example:**

```
#include <stdio.h>
#include <crt.h>
#include <colors.h>                                // define color names

void main(void) {
    char          highlight=0, normal=0;

    if (hascolor())
        crtcolor(WHITE, BLACK, -1, -1);

    if (has(RVON)==1)
    {
        highlight = RVON;
        normal = RVOFF;
    }
    else if (has(PON)==1)
    {
        highlight = PON;
        normal = POFF;
    }
    else if (has(ULON)==1)
    {
        highlight = ULON;
        normal = ULOFF;
    }
    else if (has(BON)==1)
    {
        highlight = BON;
        normal = BOFF;
    }

    printf("\nThe word %c%s%c is unknown to me.",
        highlight,"sysygy",normal);
}
```

---

**Output:**

>test

The word **sysygy** is unknown to me.

>

### HasMouse

Determine if console has a mouse attached.

```
#include <wmapi.h>
int HasMouse ( void )
```

**Operation:** Tests the console setup for a mouse configuration.

**Returns:** A true/false value. A true or non-zero value indicates that a mouse is configured for the console.

**Notes:** This function does not test whether a mouse is present or not, only whether one has been configured for this console.

**Conforms to:** **HasMouse** q ANSI q DOS n THEOS q POSIX

**See also:** [wMouse functions](#)

---

#### Example:

```
#include <stdio.h>
#include <wmapi.h>

void main()
{
    int mouse;

    if (mouse=HasMouse())
    {
        wMouseEnable(...)
    }
}
```



## hasprt

Determine if an attached printer is capable of performing a specific print attribute.

```
#include <stdio.h>
```

```
int hasprt( int attribute, int lub )
```

---

```
attribute          »   printing attribute code
```

```
lub                »   logical unit number of attached printer
```

**Operation:** The class code definition associated with the printer attached as *lub* is examined.

**Returns:** A true/false value. A true or non-zero value indicates that the printer is capable of performing *attribute*. A false value indicates that the class code has no definition for *attribute*.

**Notes:** A false value is always returned when *lub* is not one of the printers (PRT1—PRT16), it is not currently attached or there is no class code associated with the printer.

Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of lub numbers.

The include file LUB.H defines names for each of the lubs supported.

**Conforms to:** **hasprt**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [getclass](#), [getlub](#), [getucb](#), [GetUCB](#), [has](#), [hascolor](#),

---

### Example:

```
#include <stdio.h>
```

```
void main(void) {
    char    highlight=0,
           normal=0;
    FILE    * prt;

    if (prt = fopen("prt1","w"))
    {
        if (hasprt(RVON, 28))
        {
            highlight = RVON;
            normal = RVOFF;
        }
        else if (hasprt(ULON, 28))
        {
            highlight = ULON;
            normal = ULOFF;
        }

        fprintf(prt, "\nThe word %c%s%c is unknown to me.",
                highlight,"sysygy",normal);
    }
    else
        perror();
}
```

// printer not attached

htonl, htons

Convert numbers from host-byte order to network-byte order.

```
#include <socket.h>
u_long  htonl ( u_long  hostlong )
u_short htons ( u_short hostshort )
```

---

*hostlong*           »   32-bit number in host-byte order  
*hostshort*        »   16-bit number in host-byte order

**Operation:**        **htonl**        Converts a 32-bit number from host-byte order and returns a 32-bit number in network-byte order.

**htons**        Converts a 16-bit number from host-byte order and returns a 16-bit number in network-byte order.

**Returns:**           The converted value in network-byte order.

**Notes:**            A THEOS network defines host-byte order as “little-endian” which is the Intel-standard sequence for multiple-byte numeric values. Network-byte order is “big-endian,” which sequences the bytes in reverse, or least significant byte first. For instance:

Value: 305,419,896 (0x12345678)

Host-byte order: 78 56 34 12

Network-byte order: 12 34 56 78

**Conforms to:**               **htonl**    q ANSI    q DOS    n THEOS   q POSIX

**htons**    q ANSI    q DOS    n THEOS   q POSIX

**See also:**            [ntohl](#), [ntohs](#)

## hypot

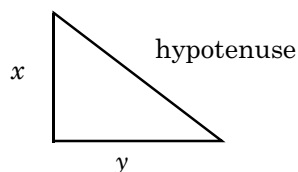
Compute the length of the hypotenuse of a right triangle.

```
#include <math.h>
double hypot ( double x, double y )
```

---

*x*                   » length of side 1  
*y*                   » length of side 2

**Operation:** The length of the hypotenuse of a right triangle having two sides of length *x* and *y* is computed using the Pythagorean theorem.



$$\text{hypotenuse} = \sqrt{x^2 + y^2}$$

**Returns:** The value of the hypotenuse.

**Errors:** If an overflow occurs during the computation, HUGE\_VAL is returned and [errno](#) is set to ERANGE.

**Restrictions:** This function uses BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:** **hypot**   q ANSI   n DOS   n THEOS   n POSIX

**See also:** [sqrt](#)

**Example:**

```
#include <stdio.h>
#include <math.h>

void main()
{
    double  x,
            y,
            h;

    printf("\nEnter the length of the two sides: ");
    scanf("%lf %lf", &x, &y);

    h = hypot(x, y);

    printf("\nThe hypotenuse of a right triangle with");
    printf("\ntwo sides of &lg and &lg is &lg.\n", x, y, h);
}
```

---

**Output:**

>test

Enter the length of the two sides: 6 8

The hypotenuse of a right triangle with  
two sides of 6 and 8 is 10.

>

## i2tol

Convert a value from a short integer array to a long integer.

```
#include <stdlib.h>
long i2tol ( short i[2] )
```

---

*i*                      »    two-element integer array value

**Operation:**        The value stored in the two elements of the array *i* is converted to a long integer.

**Returns:**            The converted long integer value.

**Notes:**             The short integer array values used by this function are created with the `ltoi2` function.

**Conforms to:**                **i2tol**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [lltoa](#), [ltoa](#), [ltoc3](#), [ltoi2](#), [ltol3](#)

### increment memory assembly functions

This group of “functions” generates in-line code to increment a byte, short integer or long integer in a specified memory area, your program’s code segment or in the nucleus segment.

```
#include <builtin.h>

void _incb ( void _far * far_offset )
void _incd ( void _far * far_offset )
void _incw ( void _far * far_offset )

void _inccsb ( void * offset )
void _inccsd ( void * offset )
void _inccsw ( void * offset )

void _incnucb ( void * offset )
void _incnucd ( void * offset )
void _incnucw ( void * offset )
```

---

*far\_offset*           »   pointer to remote data object  
*offset*                »   pointer to data item in code segment

**Operation:**        To increment means to add one to a value.

- \_incb**            Increments the character or byte value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_incd**            Increments the long integer value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_incw**            Increments the short integer value in the location *far\_offset*. The result is placed back in that location of the remote memory area.
- \_inccsb, \_inccsd, \_inccsw** These functions perform the same operation as **\_incb**, **\_incd** and **\_incw** except that the memory segment is predefined to be your program’s code segment alias. These three functions can only be used in device-drivers because other programs do not have an alias to their code segment and they will generate a general protection error.
- \_incnucb, \_incnucd, \_incnucw** These functions perform the same operation as **\_incb**, **\_incd** and **\_incw** except that the memory segment is predefined to be the nucleus memory segment.

**Returns:**        No value is returned by these functions. The memory locations are incremented in-place.

**Notes:**         These “built-in” functions generate in-line code, thus avoiding the expensive usage of the call and ret instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can increment whatever type of object desired in any location.

For instance, a `_incw` function can increment a word value (double-byte value) in a location that is declared as a character string. Of course, the program would have to be coded such that it could handle this situation.

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

**Conforms to:**

<code>_incb</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_incd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_incnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_incnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_incnucd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_inccsb</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_inccsd</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_inccsw</code>	q ANSI	q DOS	n THEOS	q POSIX
<code>_incw</code>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**

[add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

**input port assembly functions**

This group of “functions” generates in-line code to get a byte, word or double-word from an input port.

```
#include <builtin.h>
unsigned _inb ( unsigned port )
long _ind ( unsigned port )
unsigned _inw ( unsigned port )

void _insb ( unsigned port, void _far * offset, size_t count )
void _insd ( unsigned port, void _far * offset, size_t count )
void _insw ( unsigned port, void _far * offset, size_t count )
```

---

<i>count</i>	»	number of items to read
<i>offset</i>	»	pointer to storage location
<i>port</i>	»	input port number to read from

**Operation:**

<b>_inb</b>	Reads a single byte from the input port number <i>port</i> .
<b>_ind</b>	Reads a double-word from the input port number <i>port</i> .
<b>_inw</b>	Reads a single word from the input port number <i>port</i> .
<b>_insb</b>	Reads <i>count</i> number of bytes from the input port number <i>port</i> . The data read is stored in consecutive locations starting at <i>offset</i> .
<b>_insd</b>	Reads <i>count</i> number of double-words from the input port number <i>port</i> . The data read is stored in consecutive locations starting at <i>offset</i> .
<b>_insw</b>	Reads <i>count</i> number of single words from the input port number <i>port</i> . The data read is stored in consecutive locations starting at <i>offset</i> .

**Returns:**

<b>_inb</b>	The single byte read.
<b>_ind</b>	The double-word read.
<b>_inw</b>	The single word read.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions.

**Restrictions:** These functions are intended for device-driver authors and cannot be executed by programs operating with greater than ring 2 access.

**Conforms to:**

<b>_inb</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>_ind</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>_insb</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>_insd</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>_insw</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>_inw</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [output port assembly functions](#)



## ***\_incrmem***

Increase the size of the program's data segment.

```
#include <sc.h>
char * _incrmem ( size_t len )
```

---

*len*                   »   additional size requested, in bytes

**Operation:**       The size of the memory segment currently pointed to by the DS register (data segment selector) is increased by *len* bytes.

**Returns:**         A pointer to the first available byte of the additional memory added to the data segment.

**Notes:**           It is normally not necessary to increase the size of your data segment with this function because the memory allocation routines will increase the size automatically when more memory is requested than is currently allocated to the segment.

**Conforms to:**        *\_incrmem*   q ANSI    q DOS    n THEOS   q POSIX

**See also:**         [\\_mem\\_grow](#)

**inet\_addr, inet\_ntoa**

Converts internet address between the Internet standard “dotted” form and the struct IN\_ADDR form.

```
#include <socket.h>
```

```
u_long inet_addr ( const char * string )
```

```
char * inet_ntoa ( IN_ADDR addr )
```

---

*addr* » structure representing the Internet host address

*string* » pointer to string containing Internet standard “dotted” address

**Operation:**     **inet\_addr**     The value in *string* is interpreted as an Internet standard “dotted” address and converted to a network address in network-byte order.

**inet\_ntoa**     The value in the structure *addr* is interpreted as a network address in network-byte order and converted to an Internet standard “dotted” address.

**Returns:**        **inet\_addr**     If no error occurs, the internet address is returned in network-byte order. If *string* does not contain a legitimate Internet address then INADDR\_NONE is returned.

**inet\_ntoa**     A pointer to a character string containing the text address in standard “dotted” notation.

**Errors:**         **inet\_addr**     If *string* does not contain a legitimate Internet address, then INADDR\_NONE is returned.

**inet\_ntoa**     If *addr* cannot be converted, a NULL is returned.

**Notes:**          The standard “dotted” form of the address is a character string containing one to four numbers separated by periods. Each number has a potential value in the range of 0–255. For instance:

```
a.b.c.d
a.b.c
a.b
a
```

Each number has a potential value in the range of 0–255.

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on the Intel architecture, the bytes referred to above appear as “d.c.b.a”. That is, the bytes on an Intel processor are ordered from right to left.

Note: The following notations are only used by Berkeley, and nowhere else on the Internet. In the interests of compatibility with their software, they are supported as specified.

When a three-part address is specified, the last part is interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as “128.net.host”.

When a two-part address is specified, the last part is interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as “net.host”.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

**Restrictions:**     **inet\_ntoa**     The pointer returned points to a static buffer. This buffer may be overwritten each time that a windows socket function is called. Copy the contents of this buffer to your program before another windows socket function call is performed.

<b>Conforms to:</b>	<b>inet_addr</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>inet_ntoa</b>	q ANSI	q DOS	n THEOS	q POSIX

**Example:**     See the example for the function [socket](#).

## InitFile functions

These functions open, close, read or write records in an initialization or configuration file. For a description of an initialization file see the section on “Notes.”

```
#include <initfile.h>

int InitFileClose ( INITFILE _far * initfile )
int InitFileOpen ( INITFILE _far * initfile,
    char _far * filename, char _far * mode )
int InitFileRead ( INITFILE _far * initfile, char _far * section, char _far * entry,
    char _far * default, char _far * buffer, int length )
int InitFileReadInt ( INITFILE _far * initfile,
    char _far * section, char _far * entry, int base, int default )
int InitFileReadNext ( INITFILE _far * initfile,
    char _far * section, char _far * entry, char _far * buffer, int length )
int InitFileWrite ( INITFILE _far * initfile,
    char _far * section, char _far * entry, char _far * value )
int InitFileWriteInt ( INITFILE _far * initfile,
    char _far * section, char _far * entry, int base, int value )
```

---

<i>base</i>	»	10 or 16 for decimal or hexadecimal base
<i>buffer</i>	»	Buffer to read entry value into
<i>default</i>	»	Default value to use if section and or entry not found
<i>entry</i>	»	Record entry label
<i>filename</i>	»	File name of initialization file
<i>initfile</i>	»	Pointer to an INITFILE structure
<i>length</i>	»	Length of buffer
<i>mode</i>	»	File access mode, one of “r” “w” or “rw”
<i>section</i>	»	Name of section in file
<i>value</i>	»	Value to write for entry

**Operation:**     **InitFileClose** Closes the initialization file *initfile*. If the file was opened with *mode* “w” or “rw” and entries were written to the file, the disk file is updated at this time.

**InitFileOpen** Opens initialization file *filename* with access *mode*. The *mode* may be “r,” “w” or “rw.” As with each of these functions, the *initfile* must be a pointer to an INITFILE structure.

**InitFileRead** Locates *section* and *entry* within *section* of the *initfile*. Reads the value following the equal sign of *entry* from *initfile*, placing it in *buffer*. *Length* is the maximum length of *buffer* and may cause the value to be truncated.

*Section* may be a null string or a pointer to a null string, in which case *entry* is located in the unnamed *section* at the beginning of the initialization file.

If *entry* cannot be found in *section*, then *buffer* is filled in with the default value *dflt*.

If *entry* is a null string, then all records (entry names, equal sign and values) in *section* are read and stored in *buffer*.

**InitFileReadInt** Similar to `InitFileRead` above, except that no success/failure return is set. Instead, the value is interpreted as an integer of number *base* and is returned as the value of the function. *Base* must be 10 or 16, indicating decimal or hexadecimal base.

If the value in the file is “On,” “Yes” or “True,” it is interpreted as the value 1. If the value is “Off,” “No” or “False,” it is interpreted as the value 0. These interpreted strings are case-insensitive.

**InitFileReadNext** Locates *section* and *entry* within *section* of the *initfile*. When this record is found, the record following is read and, if this record is not a section record, then the *entry* field value is copied to *entry* and the string following the equal sign is copied to *buffer*. *Length* is the maximum length of *buffer* and may cause the value to be truncated.

*entry* may be NULL or a pointer to a null string, in which case `InitFileReadNext` locates the first record following the *section* record, copies the entry label to *entry* and reads the entry value to *buffer*.

Similar to `InitFileRead`, *section* may be a NULL string or a pointer to a null string, in which case *entry* is located in the unnamed *section* at the beginning of the initialization file.

If *entry* is specified but cannot be found in *section* then *buffer* is set to a null string.

**InitFileWrite** Updates *entry* in *initfile* to have *value*. If *section* does not exist, it is added to the file with a blank line before it. If *entry* does not exist, it is added to the file. The upper or lower case mode used in *section* and *entry* is used only if the line must be added to *initfile*, not if it is only updating an existing entry.

**InitFileWriteInt** Similar to `InitFileWrite` above, except *value* is specified as a numeric integer. This number is converted (using *base*) to its string equivalent and used to update *entry*.

#### Returns:

**InitFileClose** Success code: 0 = success, non-zero is failure reason code.

**InitFileOpen** Success code: 0 = success, non-zero is failure reason code.

**InitFileRead** Success code: 0 = success, non-zero is failure reason code.

**InitFileReadInt** The value from the entry or, if the entry or section is not found, the default value (*default*).

**InitFileReadNext** Success code: 0 = success, non-zero is failure reason code.

**InitFileWrite** Success code: 0 = success, non-zero is failure reason code.

**InitFileWriteInt** Success code: 0 = success, non-zero is failure reason code.

**Errors:** The errors that might be detected and reported by these functions are:

Number	Description	Function
1	<i>Entry</i> not found.	InitFileRead InitFileReadInt InitFileReadNext InitFileWrite InitFileWriteInt
2	<i>Section</i> not found.	InitFileRead InitFileReadInt InitFileReadNext InitFileWrite InitFileWriteInt
3	Insufficient memory.	InitFileRead InitFileReadInt InitFileReadNext InitFileWrite InitFileWriteInt
19	<i>Filename</i> not found.	InitFileOpen
24	<i>Initfile</i> not open.	InitFileRead InitFileReadInt InitFileReadNext InitFileWrite InitFileWriteInt
47	<i>Initfile</i> not open for write.	InitFileWrite InitFileWriteInt

Additional errors might be detected by the InitFileOpen function. Refer to the [fopen](#) function for a list of the possible error codes that apply to opening files.

Although neither the [errno](#) nor the [\\_errnum](#) variables are set by these functions, you may assign the return value to [\\_errnum](#) and the file name to [\\_errarg](#) to perform standard error processing. For instance:

```
if (rc=InitFileOpen(&config_file, config_name, "r")) {  
    _errnum = rc;  
    _errarg = config_name;  
    exit(fperror());  
}
```

**Notes:** These functions access and maintain an *initialization file*. An initialization file is a stream text file with zero or more unique *sections* and zero or more *entries* that are unique within the section. Section names and entry names are case-insensitive on reads and writes of existing sections or entries. The case mode specified is only used when a new line is added to the file.

Section names are always enclosed within square brackets in the file. Section names may have embedded spaces. Entry names are always followed by an equal sign and may have embedded spaces and the equal sign may have optional white space before and after it.

The general form of an initialization file is:

```
[Section1]
Entry1=Entry1 value
Entry2=Entry2 value

[Section2]
Entry1=Entry1 value
Entry2=Entry2 value
```

Although the disk file is only updated when the InitFileClose function is used, an InitFileRead or InitFileReadInt function following an InitFileWrite will retrieve the updated version of the record.

**Defaults:** When reading an *entry* with InitFileRead or InitFileReadInt, and the *entry* line is not found, the value returned is the value indicated by *default*.

<b>Conforms to:</b>	<b>InitFileClose</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>InitFileOpen</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>InitFileRead</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>InitFileReadInt</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>InitFileWrite</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>InitFileWriteInt</b>	q ANSI	q DOS	n THEOS	q POSIX

**Example:**

```
#include <stdio.h>
#include <initfile.h>

void get_config(void);
void dsp_config(void);

struct config {
    char    coname[40];
    char    coaddr[40];
    char    cocity[16];
    char    costate[3];
    char    cozip[11];
    char    cophone[13];
    char    cofax[13];
    char    path[256];
    char    appath[256];
    char    arpath[256];
} config;

INITFILE config_file;           // fcb for config file

void
main()
{
    get_config();                // get current config info
    dsp_config();                // display current config
}

void get_config(void)
{
    int          rc;

    if (rc=InitFileOpen(&config_file, "sample.config", "r"))
    {
        printf("Error opening file = %i",rc);
        exit(rc);
    }

    InitFileRead(&config_file, "General", "CoName",
        "", config.coname, 40); // get company name
    InitFileRead(&config_file, "General", "CoAddr",
        "", config.coaddr, 40); // get company address
    InitFileRead(&config_file, "General", "CoCity",
        "", config.cocity, 16); // get company city
    InitFileRead(&config_file, "General", "CoState",
        "", config.costate, 3); // get company state
    InitFileRead(&config_file, "General", "CoZip",
        "", config.cozip, 11);  // get company zip code
    InitFileRead(&config_file, "General", "CoPhone",
        "", config.cophone, 13); // get company telephone #
    InitFileRead(&config_file, "General", "CoFax",
        "", config.cofax, 13);  // get company fax #
    InitFileRead(&config_file, "General", "Path",
        "", config.path, 256);  // get default path for files
```



---

```

    InitFileRead(&config_file, "AccountsPayable", "APpath",
        "", config.appath, 256); // default path for AP files
    InitFileRead(&config_file, "AccountsReceivable", "ARpath",
        "", config.arpah, 256); // default path for AR files

    InitFileClose(&config_file);    // close config file
}

void dsp_config(void)
{
    printf("\n Company name:   %s\n", config.coname);
    printf("      Address:      %s\n", config.coaddr);
    printf("City-state-zip:      %s, %s %s\n", config.cocity,
        config.costate, config.cozip);
    printf("      Telephone:    %s\n", config.cophone);
    printf("      Fax:         %s\n", config.cofax);
    printf("\n");
    printf("Path:              %s\n", config.path);
    printf("AR Path:          %s%s\n", config.path, config.arpah);
    printf("AP Path:          %s%s\n", config.path, config.appath);
}

```

sample.config file:

```

[General]
CoName=ABC Software Corporation
CoAddr=1234 North Main Street
CoCity=Anytown
CoState=CA
CoZip=12345-6789
CoPhone=510 555-1234
CoFax=510 555-4321
Path=/company.files/

[AccountsPayable]
APpath=accounts.payable.

[AccountsReceivable]
ARpath=accounts.receive.

```

---

### Output:

>example

```

      Company name:   ABC Software Corporation
      Address:      1234 North Main Street
City-state-zip:    Anytown, CA 12345-6789
      Telephone:    510 555-1234
      Fax:         510 555-4321

Path:              /company.files/
AR Path:          /company.files/accounts.receive.
AP Path:          /company.files/accounts.payable.

```

>

**intoff, inton**

These “functions” produce in-line code that enables or disables interrupts.

```
#include <driver.h>
void intoff ( void )
void inton ( void )

#include <builtin.h>
void _cli ( void )
void _sti ( void )
void _disable ( void )
void _enable ( void )
```

**Operation:**      **intoff**      Disable or turn off processor interrupts.

**inton**      Enable or turn on processor interrupts.

The **\_disable** and **\_enable** names are synonym names to **intoff** and **inton**, respectively. Similarly, the **\_cli** and **\_sti** names are synonym names to **intoff** and **inton**, respectively.

**Notes:**            These “built-in” functions generate in-line code, thus avoiding the expensive usage of the **call** and **ret** instructions.

**Restrictions:**    These functions are intended for device-driver authors.

Interrupts should not be turned off for any significant length of time. When interrupts are disabled, other devices and processes such as disk, tapes, multiport serial cards, *etc.*, may not perform properly. When necessary, interrupts should only be disabled for the length of time required to execute a few instructions and then interrupts should be immediately enabled.

<b>Conforms to:</b>	<b>_cli</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_disable</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_enable</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>intoff</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>inton</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_sti</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**          [\\_saveints](#), [\\_restoreints](#)

## **\_ioctl, \_ioctl\_ucb**

These functions perform basic input/output control and status report operations for devices.

```
#include <ioctl.h> or <sc.h>
unsigned short _ioctl ( FILE * file, void _far * ctl )
```

```
#include <sc.h>
long _ioctl_ucb ( UCB * ucb, void _far * ctl )
```

---

<i>ctl</i>	»	pointer to i/o control structure
<i>file</i>	»	pointer to open file's fcb
<i>ucb</i>	»	pointer to UCB structure

**Operation:** The control operation specified in the *ctl* structure is performed on the device associated with *file* or *ucb*. The types of operations performed depend upon the type of device associated with *file* or *ucb* (disk, tape or byte i/o device).

**\_ioctl** This function provides control and status information about the device associated with an open *file*.

**\_ioctl\_ucb** This function provides control and status information about the device associated with a *ucb*. (Note: This function is only available when used on a THEOS 32 Version 4 system or later.)

**Returns:** The status of the device or the requested information about the device. The meaning of this return value is different for each type of device and is described below for each of the device types.

**Notes:** The control structures for disk, tape and byte devices are defined in the IOCTL.H file. This file also defines names for the command codes used in these structures.

**■ Disk Devices**

A disk device is accessed with a DISK\_CTL structure:

```
struct diskcntl {
    short c_cmd;           // command code
    long c_sect;           // sector number
    char *c_buf;           // buffer offset
    unsigned short c_seg;  // buffer segment
    short c_count;         // sector count
    short c_cyl;           // cylinder number
    short c_head;          // head number
    short c_den;           // density code
} DISK_CTL;
```

Although there are many commands that can be used for disk devices, only two are appropriate for user-written applications and utilities.

**DISK\_IOCTL\_DISK\_TYPE**

Report the type of disk drive and the cache configuration for the device.

Only the c\_cmd field of DISK\_CTL need be specified.

A zero return value indicates that the drive type is unknown or not supported. A non-zero return value specifies the type of disk drive and the current disk cache options enabled for the drive. The possible values for disk types and cache options are defined in IOCTL.H in the enumeration list DISK\_TYPES. The disk type is ANDed with the cache option.

**DISK\_IOCTL\_REPORT\_NAME**

Report disk volume name.

The c\_cmd, c\_seg and c\_buf fields of DISK\_CTL must be specified. The c\_seg and c\_buf define the location of the buffer used to return the name of the disk volume.

A zero return value indicates that disk names are not supported by the device-driver or the disk drive. Otherwise REPORT\_NAME\_SUPPORTED is returned and the buffer is filled in with a null-terminated string of 64 or fewer bytes.

## ■ Tape Devices

A tape device is accessed with a TAPE\_CTL structure:

```
struct tapectl {
    short c_cmd;           // command code
    unsigned short c_tseg; // buffer segment
    char *c_tbuf;          // buffer offset
    unsigned short c_tlen; // buffer length
} TAPE_CTL;
```

Although there are many commands that can be used for tape devices, only a few are appropriate for user-written applications and utilities.

### **TAPE\_IOCTL\_STATUS**

Returns the current tape drive status.

Only the `c_cmd` field of TAPE\_CTL need be specified.

The return value is bit-mapped:

- Bit 0 indicates the tape is at BOT (Begin-of-Tape).
- Bit 1 indicates the tape is at EOT (End-of-Tape).
- Bit 2 indicates that the tape is write protected.
- Bit 3 indicates that the tape drive is ready.

If supported by the device, early warning of end-of-tape is indicated by both bits 0 and 1 being set (value is 3).

### **TAPE\_IOCTL\_OPEN\_STATUS**

Returns the current tape device-driver status.

Only the `c_cmd` field of TAPE\_CTL need be specified.

A zero return value indicates that the device-driver does not support this command. A return value of OPEN\_STATUS\_SUPPORTED indicates that the command was performed.

When the command is performed, the field `c_tseg` in TAPE\_CTL is set to one of the following values:

- 0      Tape driver attached properly.
- 1      Insufficient memory for tape buffer.
- 2      Fatal error or configuration error.

This command is used by BOOTER3 and the ATTACH command to determine if the requested device was attached properly and should always return as attached properly when called by an application program. Improperly attached devices are detached by BOOTER3 and ATTACH.

### **TAPE\_IOCTL\_REWIND**

Rewind to the beginning of the tape volume.

Only the `c_cmd` field of TAPE\_CTL need be specified.

Prior to rewinding the tape, any data still in the device-driver's output buffer is flushed to the tape volume and any data in the read buffer is dis-

carded. The tape is then rewound to the start of the tape with the following return codes:

- |   |  |
|---|--|
| 4 | Error detected during rewind operation.            |
| 9 | Tape rewound successfully and is at start-of-tape. |

#### **TAPE\_IOCTL\_REWIND\_UNLOAD**

Rewind to the beginning of the tape volume and unload the tape.

Operation is identical to the [TAPE\\_IOCTL\\_REWIND](#) command except that, if the tape drive is capable of it, the tape volume is ejected.

#### **TAPE\_IOCTL\_WRITE\_TAPE\_MARK**

Write a tape mark at the current location of the tape.

Only the `c_cmd` field of `TAPE_CTL` need be specified.

Prior to writing the tape mark, any data still in the device-driver's output buffer is flushed to the tape volume and any data in the read buffer is discarded.

A zero return value indicates that the tape mark was written successfully; a non-zero return value indicates failure.

#### **TAPE\_IOCTL\_FWD\_SPACE\_FILE**

Advance tape to the next tape mark.

Only the `c_cmd` field of `TAPE_CTL` need be specified.

Prior to advancing the tape, any data still in the device-driver's output buffer is flushed to the tape volume and any data in the read buffer is discarded. The tape is then advanced to the end of the next tape mark and the function returns with one of the following codes:

- |   |  |
|---|--|
| 0 | Successful                               |
| 2 | End-of-tape encountered                  |
| 3 | Early warning of end-of-tape encountered |
| 4 | Error occurred                           |

This command should not be performed after a write operation because write-data is always written at the end of the tape and there is no tape mark following the write-data.

#### **TAPE\_IOCTL\_ERASE**

Erase the tape.

Only the `c_cmd` field of `TAPE_CTL` need be specified.

The exact operation of this command depends upon the tape drive. Some drives completely erase the data on the tape while others might only set a mark indicating physical end-of-tape at the beginning of the tape. In either case, the tape is positioned to the beginning of the tape.

The return values for this command include:

- 4 Error detected during erase operation
- 9 Tape erased and rewound successfully and is at start-of-tape

This command should only be used following a [TAPE\\_IOCTL\\_REWIND](#) command.

#### **TAPE\_IOCTL\_RETENSION**

Retension the tape volume by advancing to the end of the tape and then performing a high-speed rewind to the beginning of the tape volume.

Only the `c_cmd` field of `TAPE_CTL` need be specified.

The return values for this command include:

- 4 Error detected during operation
- 9 Tape retensioned successfully and is at start-of-tape

This command should only be used following a [TAPE\\_IOCTL\\_REWIND](#) command.

#### **TAPE\_IOCTL\_READ**

Read a standard variable-length tape record from the tape volume.

All parameters must be set in the `TYPE_CTL` structure:

```
c_cmd    = TAPE_IOCTL_READ
c_tseg   = read buffer selector
c_tbuf   = read buffer offset
c_tlen   = buffer length in bytes (1–32768)
```

The function returns with one of the following codes:

- 0 Successful
- 2 Tape mark encountered
- 3 End-of-tape encountered
- 4 Unrecoverable error occurred

The actual number of bytes read can be determined by using the [TAPE\\_IOCTL\\_READ\\_LENGTH](#) command described below.

This command is used to read in THEOS-formatted, variable-length tape blocks written with [TAPE\\_IOCTL\\_WRITE](#) or the ARCHIVE, BACKUP or COPY-FILE utilities.

#### **TAPE\_IOCTL\_READ\_RAW**

Read a fixed-length tape record from the tape volume.

All parameters must be set in the `TYPE_CTL` structure:

```
c_cmd    = TAPE_IOCTL_READ_RAW
c_tseg   = read buffer selector
c_tbuf   = read buffer offset
c_tlen   = buffer length in 512-byte blocks (1–32768)
```

The function returns with one of the following codes:

0	Successful
2	Tape mark encountered
3	End-of-tape encountered
4	Unrecoverable error occurred

The actual number of bytes read can be determined by using the [TAPE\\_IOCTL\\_READ\\_LENGTH](#) command described below.

This command is used to read in 512-byte, fixed-length tape blocks written with the [TAPE\\_IOCTL\\_WRITE\\_RAW](#) command.

#### **TAPE\_IOCTL\_READ\_LENGTH**

Return the length of the last read operation.

Only the `c_cmd` field of `TAPE_CTL` need be specified.

If the last tape read operation was a [TAPE\\_IOCTL\\_READ](#) command or a `read` or `_read` function, the number of bytes in the last record read is returned. If the last tape read operation was a [TAPE\\_IOCTL\\_READ\\_RAW](#) command, the number of bytes transferred to the read buffer is returned.

No operation is performed on the tape itself.

#### **TAPE\_IOCTL\_WRITE**

Write a standard variable-length tape record to the tape volume.

All parameters must be set in the `TYPE_CTL` structure:

```
c_cmd    = TAPE_IOCTL_WRITE
c_tseg   = write buffer selector
c_tbuf   = write buffer offset
c_tlen   = buffer length in bytes (1–32768)
```

The function returns with one of the following codes:

0	Successful
1	Physical end-of-tape encountered
4	Unrecoverable error occurred

This command is used to write out THEOS-formatted, variable-length tape blocks that can be read with [TAPE\\_IOCTL\\_READ](#) or the `BACKUP`, `COPY-FILE` or `RESTORE` utilities.

The last data sent to the tape driver with this command may not be physically written to the tape volume until a [TAPE\\_IOCTL\\_REWIND](#), [TAPE\\_IOCTL\\_REWIND\\_UNLOAD](#), [TAPE\\_IOCTL\\_WRITE\\_TAPE\\_MARK](#), [TAPE\\_IOCTL\\_FWD\\_SPACE\\_FILE](#) or [TAPE\\_IOCTL\\_ERASE](#) is performed.

After this command executes successfully the [TAPE\\_IOCTL\\_STATUS](#) command should be used to check for early end-of-tape warning. On many tapes there will be at least 1MB of space between the early warning indicator and the physical end-of-tape. However, there can be as much as 256K of data in the write buffer when this occurs. When early warning occurs the application should write at least one tape mark to ensure that all of the write data is flushed to the tape.



If the physical end-of-tape encountered error is reported your application should abort writing to the tape because it has ignored the early warning for the end-of-tape and there is no space left to write a tape mark.

### **TAPE\_IOCTL\_WRITE\_RAW**

Write a fixed-length tape record to the tape volume.

All parameters must be set in the `TYPE_CTL` structure:

```
c_cmd    = TAPE_IOCTL_WRITE_RAW
c_tseg   = write buffer selector
c_tbuf   = write buffer offset
c_tlen   = buffer length in 512-byte blocks (1-32768)
```

The function returns with one of the following codes:

0	Successful
1	Physical end-of-tape encountered
4	Unrecoverable error occurred

This command is used to write out fixed-length tape blocks that can be read with [TAPE\\_IOCTL\\_READ\\_RAW](#) command.

The last data sent to the tape driver with this command may not be physically written to the tape volume until a [TAPE\\_IOCTL\\_REWIND](#), [TAPE\\_IOCTL\\_REWIND\\_UNLOAD](#), [TAPE\\_IOCTL\\_WRITE\\_TAPE\\_MARK](#), [TAPE\\_IOCTL\\_FWD\\_SPACE\\_FILE](#) or [TAPE\\_IOCTL\\_ERASE](#) is performed.

After this command executes successfully, the [TAPE\\_IOCTL\\_STATUS](#) command should be used to check for early end-of-tape warning. When early warning occurs, the application should write at least one tape mark to ensure that all of the write data is flushed to the tape.

If the physical end-of-tape encountered error is reported your application should abort writing to the tape because it has ignored the early warning for the end-of-tape and there is no space left to write a tape mark.

### ■ **Byte Devices**

A byte i/o device such as `CON`, `COMnn` or `PRTnn` is accessed with a `BYTE_CTL` structure:

```
struct bytectl {
    short c_cmd;           // command code
    short c_word1;         //
    short c_word2;         //
    char *c_buf;           // buffer offset
    unsigned short c_seg;  // buffer segment
} BYTE_CTL;
```

Although there are many commands that can be used for byte i/o devices, only a few are appropriate for user-written applications and utilities. It is more appropriate to use the library function provided that invokes that particular command. For instance, instead of using the `CLEAR_BYPASS` command of the `_ioctl` function, use the [ClrBypass](#) function.

The commands that are appropriate for application programs include:

**BUF\_STATUS**

Return buffer status in a bit-mapped value.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

The return value is bit-mapped:

- Bit 0 is transmit buffer status (1=not full)
- Bit 1 is receive buffer status (1=not empty)

**MODEM\_STATUS**

Return modem status in a bit-mapped value.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

A zero return value indicates that modem signals are not supported by the device-driver. A non-zero return value is bit-mapped:

- Bit 0 is DSR status
- Bit 1 is CTS status
- Bit 2 is RI status
- Bit 3 is DCD status

The high-order 12 bits of this value are 0xF0F. It indicates that the driver conforms to the current standards.

**SEND\_BREAK**

Sends the break signal.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

A zero return value indicates that the device-driver does not support this command. A non-zero return value indicates that the command was performed.

**USER\_BUFFER**

Defines a user-supplied input buffer.

A user-supplied input buffer is defined with the `USER_BUFFER` command. When this command is used, the `c_seg` and `c_buf` fields are set to point to the buffer. This buffer must be a structure with the following form:

```
struct {
    unsigned short buf_len;
    unsigned short buf_count;
    unsigned short next_store;
    unsigned short next_fetch;
    char buf[BUF_LEN];
};
```

When a user-supplied input buffer is defined for a device, the program must disable this buffer before exiting the program. If it doesn't, the buffer will probably be deallocated when the program exits and the device-driver will cause a memory-access error the next time it tries to use the input buffer. To disable the buffer, use the `USER_BUFFER` command with `NULL` specified for the `c_buf` and `c_seg` fields.

**RAISE\_DTR, DROP\_DTR, RAISE\_RTS, DROP\_RTS**

These four commands raise or drop the Data Terminal Ready or the Request To Send signal.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

A zero return value indicates that the device-driver does not support this command. A non-zero return value indicates that the command was performed.

Some multi-port or intelligent boards may have a significant delay between the time this command is executed and when the signal actually changes state. This time lag may be several milliseconds.

**CLEAR\_TYPEAHEAD**

Clear the default type-ahead buffer.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

Only the default type-ahead buffer is cleared with this command. A user-specified buffer is not cleared.

A zero return value indicates that the device-driver does not support this command. A non-zero return value indicates that the command was performed.

**REPORT\_MAX\_BAUD**

Report the maximum baud rate supported by the device.

Only the `c_cmd` field of `BYTE_CTL` need be specified.

A zero return value indicates that the device-driver does not support this command. A return value of `MAX_BAUD_SUPPORTED` indicates that the command was performed.

When the command is performed, the fields `c_word1` and `c_word2` in `BYTE_CTL` are filled in with the 32-bit value for the maximum baud rate supported. `c_word1` has the lower 16-bits of data.

**REPORT\_NAME**

Report the serial adapter name.

The `c_cmd`, `c_seg` and `c_buf` fields of `BYTE_CTL` must be specified. The `c_seg` and `c_buf` define the location of the buffer used to return the name of the adapter.

A zero return value indicates that adapter names are not supported by the device-driver or the device. Otherwise `REPORT_NAME_SUPPORTED` is returned and the buffer is filled in with a null-terminated string of 64 or fewer bytes.

<b>Conforms to:</b>	<b>_ioctl</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_ioctl_uch</b>	q ANSI	q DOS	n THEOS	q POSIX

**ioctlsocket**

Control the mode of a socket.

```
#include <socket.h>
```

```
int ioctlsocket ( SOCKET s, long cmd, u_long * argp )
```

---

*argp*                   »   pointer to a parameter for *cmd*

*cmd*                    »   command to perform on socket

*s*                      »   socket number

**Operation:**       This routine may be used on any socket in any state. It is used to set or clear blocking mode.

**Returns:**         A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:**         When the return is SOCKET\_ERROR, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAEINVAL	<i>cmd</i> is not a valid command or <i>argp</i> is not an acceptable parameter for <i>cmd</i> or the command is not applicable to the type of socket supplied.
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.

**Notes:**         The following *cmd* is supported:

<i>Code</i>	<i>Meaning</i>
FIONBIO	<p>Enable or disable non-blocking mode on the socket <i>s</i>. <i>argp</i> points at an unsigned long, which is non-zero if non-blocking mode is to be enabled and zero if it is to be disabled.</p> <p>When a socket is created, it operates in blocking mode (i.e. non-blocking mode is disabled). This is consistent with BSD sockets.</p>

**Conforms to:**       *ioctlsocket*   q ANSI    q DOS    n THEOS   q POSIX

**See also:**         [getsockopt](#), [setsockopt](#), [socket](#)

## **ip**

Get the integer portion of a floating-point value.

```
#include <math.h>
double ip ( double x )
```

---

*x*                      »   floating-point value

- Operation:**        The integer or whole number portion of the value *x* is determined and returned.
- Returns:**            The integer portion of *x*. The sign of the integer portion is the same as the sign of *x*.
- Notes:**             This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.
- Conforms to:**        `ip`    q ANSI    q DOS    n THEOS    q POSIX
- See also:**            [modf](#)

**is character functions**

These functions test a character and determine its type or category of character.

```
#include <ctype.h>
int isalnum ( int c )
int isalpha ( int c )
int isascii ( int c )
int iscntrl ( int c )
int iscsym ( int c )
int iscsymnum ( int c )
int isdigit ( int c )
int isdisp ( int c )
int isfnsym ( int c )
int isgraph ( int c )
int ishex ( int c )
int islower ( int c )
int isoctal ( int c )
int isprint ( int c )
int ispunct ( int c )
int isspace ( int c )
int issymbol ( int c )
int issymnum ( int c )
int isupper ( int c )
int isxdigit ( int c )
```

---

*c*                                »    character value to test

**Operation:**        The character *c* is tested for specific character types, depending upon the function reference:

- |                  |  |
|------------------|--|
| <b>isalnum</b>   | Alphanumeric character type ('A'—'Z', 'a'—'z' or '0'—'9').   |
| <b>isalpha</b>   | Alphabetic character type ('A'—'Z' or 'a'—'z').  |
| <b>isascii</b>   | ASCII character type (0x00—0x7F).  |
| <b>iscntrl</b>   | ASCII control character type (0x00—0x1F or 0x7F).  |
| <b>iscsym</b>    | C language, valid first character symbol name type (isalpha type or '_').  |
| <b>iscsymnum</b> | C language, valid character symbol name type (isalnum type or '_').  |
| <b>isdigit</b>   | ASCII digit character type ('0'—'9').  |
| <b>isdisp</b>    | Visible character type on most terminals (isprint type plus 0xA0—0xB9 and 0xC0—0xED).  |
| <b>isfnsym</b>   | Valid file name character type (isalnum type, or one of the set {\$, _, Æ, Ä, Å, Ç, É, Ñ, Ö, Ü, æ, ä, å, ç, é, ñ, ö, ü or ß}). |

<b>isgraph</b>	Visible character type (0x21—0x7E).
<b>ishex</b>	Hexadecimal digit ('A'—'F', 'a'—'f' or '0'—'9'). Non-zero return values are the normalized value of the hexadecimal digit. That is, the returned value is the value of the digit in the range 0—15.
<b>islower</b>	Lowercase alphabetic character type ('a'—'z').
<b>isocal</b>	Octal digit ('0'—'7').
<b>isprint</b>	Visible character type plus space (0x20—0x7E).
<b>ispunct</b>	Punctuation character type (isprint type minus isalnum type).
<b>isspace</b>	White space character (TAB, LF, VT, FF, CR, US and SPACE).
<b>issymbol</b>	Valid first character for file names (isalpha type or '\$').
<b>issymnum</b>	Valid character file names (isalnum type or '\$').
<b>isupper</b>	Uppercase alphabetic character type ('A'—'Z').
<b>isxdigit</b>	Hexadecimal digit ('A'—'F', 'a'—'f' or '0'—'9').

**Returns:** A zero return indicates that the character *c* is not the specified character type; a non-zero return value indicates that it is.

The value returned by *ishex* is the normalized value of the hexadecimal digit, in the range 0—15.

**Errors:** No errors are detectable by these functions.

**Notes:** This set of functions is normally implemented as in-line macros. Except for the functions *iscsym*, *iscsymnum*, *isfnsym*, *issymbol* and *issymnum*, it is possible to use true functions instead of the in-line macros by defining the label `_CTYPE_NO_MACRO` prior to including the `CTYPE.H` header file.

For instance:

```
#include <stdio.h>
#define _CTYPE_NO_MACRO
#include <ctype.h>
```

When this is done, references to these functions will refer to external function modules, not in-line macro expansions. The functions do not use the current locale; the default macros do.

Refer to Table 18, “Character Types,” on page 757 for a listing of all characters and their character types.

<b>Conforms to:</b>	<b>isalnum</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>isalpha</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>isascii</b>	q ANSI	n DOS	n THEOS	n POSIX
	<b>iscntrl</b>	n ANSI	n DOS	n THEOS	n POSIX
	<b>iscsym</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>iscsymnum</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>isdigit</b>	n ANSI	n DOS	n THEOS	n POSIX

## 364 *is* character functions

---

<b>isdisp</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>isfnsym</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>isgraph</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>ishex</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>islower</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>isoctal</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>isprint</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>ispunct</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>isspace</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>issymbol</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>issymnum</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>isupper</b>	n ANSI	n DOS	n THEOS	n POSIX
<b>isxdigit</b>	n ANSI	n DOS	n THEOS	n POSIX

**See also:** [setlocale](#), [toascii](#), [tolower](#), [\\_tolower](#), [toupper](#), [\\_toupper](#)

---

### Example:

```
#include <stdio.h>
#include <ctype.h>

void
main()
{
    int    c;

    for (c=0; c<256; ++c)
    {
        printf("\n0x%.2x %c ", c, isdisp(c) ? c : ' ');
        printf("%3s ", isalnum(c) ? "al#" : "");
        printf("%5s ", isalpha(c) ? "alpha" : "");
        printf("%5s ", isascii(c) ? "ascii" : "");
        printf("%1s ", iscntrl(c) ? "^" : "");
        printf("%4s ", iscsym(c) ? "csym" : "");
        printf("%5s ", iscsymnum(c) ? "csym#" : "");
        printf("%1s ", isdigit(c) ? "#" : "");
        printf("%2s ", isfnsym(c) ? "fn" : "");
        printf("%1s ", isgraph(c) ? "g" : "");
        printf("%3s ", ishex(c) ? "hex" : "");
        printf("%3s ", islower(c) ? "low" : "");
        printf("%3s ", isoctal(c) ? "oct" : "");
        printf("%3s ", isprint(c) ? "prt" : "");
        printf("%3s ", ispunct(c) ? "pun" : "");
        printf("%2s ", isspace(c) ? "sp" : "");
        printf("%3s ", issymbol(c) ? "sym" : "");
        printf("%4s ", issymnum(c) ? "sym#" : "");
        printf("%2s ", isupper(c) ? "up" : "");
        printf("%1s", isxdigit(c) ? "x" : "");
    }
    printf("\n");
}
```



**Output:**

```
>test
```

```

...
0x27 '      ascii      g      prt pun
0x28 (      ascii      g      prt pun
0x29 )      ascii      g      prt pun
0x2a *      ascii      g      prt pun
0x2b +      ascii      g      prt pun
0x2c ,      ascii      g      prt pun
0x2d -      ascii      g      prt pun
0x2e .      ascii      g      prt pun
0x2f /      ascii      g      prt pun
0x30 0 al#   ascii      csym# # fn g hex oct prt      sym# x
0x31 1 al#   ascii      csym# # fn g hex oct prt      sym# x
0x32 2 al#   ascii      csym# # fn g hex oct prt      sym# x
0x33 3 al#   ascii      csym# # fn g hex oct prt      sym# x
0x34 4 al#   ascii      csym# # fn g hex oct prt      sym# x
0x35 5 al#   ascii      csym# # fn g hex oct prt      sym# x
0x36 6 al#   ascii      csym# # fn g hex oct prt      sym# x
0x37 7 al#   ascii      csym# # fn g hex oct prt      sym# x
0x38 8 al#   ascii      csym# # fn g hex      prt      sym# x
0x39 9 al#   ascii      csym# # fn g hex      prt      sym# x
0x3a :      ascii      g      prt pun
0x3b ;      ascii      g      prt pun
0x3c <      ascii      g      prt pun
...
>

```

**IsActive**

Test if program's console display is active.

```
#include <wmapi.h>
int IsActive ( void )
```

**Operation:** Test the Session Manager to see if this program's console display is the active session.

**Returns:** A true/false value. A non-zero return value means that it is active; a zero return value means that the current program is not the active session.

**Notes:** When the display is not active, it means that the operator cannot see the text displayed by the program and may not be aware of any prompts for information or error messages that might be displayed by the program. The display can be forced to be the active session by using the [wSwitchTo](#) function.

**Conforms to:** **IsActive** q ANSI q DOS n THEOS q POSIX

**See also:** [IsSession](#), [wGetActive](#), [wGetSess](#), [wSwitch](#), [wSwitchTo](#)

---

**Example:**

```
#include <stdio.h>
#include <wmapi.h>

void
main()
{
    ...
    if (IsActive()==0)           // This session active?
        wSwitchTo(wGetSess());  // No, then activate it
    ...
}
```

---

## isatty

Test an open file to see if it is a serial communications device.

```
#include <stdio.h> or <io.h>
int isatty ( FILE * file )
```

---

*file*                   »   pointer to open file's FCB

**Operation:**       The file associated with *file* is tested to see if its device is a serial communications device.

**Returns:**         A true/false value. Non-zero return values indicate that *file* is on a serial communications device; a zero value indicates that it is not.

**Conforms to:**       **isatty**    q ANSI    n DOS    n THEOS    n POSIX

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    FILE * infile;

    if (!(infile = fopen(argv[1], "r")))
        exit(fperror());

    printf("\nThe file \"%s\" is on the device \"%s\"\n",
        argv[1], _lubname(getdev(infile)));
    printf("That file %s on a serial device.\n",
        isatty(infile) ? "is" : "is not");
}
```

---

### Output:

>test con

The file "CON" is on the device "CONIN"  
That file is on a serial device.

## IsBypass

Test the current Session Manager bypass status.

```
#include <wmapi.h>
int IsBypass ( void )
```

**Operation:** Test the Session Manager for the window bypass status in use by this program's display.

**Returns:** The current Session Manager bypass status: zero indicates it is not in bypass mode; non-zero indicates that it is.

**Notes:** Session Manager bypass refers to the ability to bypass the Session Manager control of multiple sessions on one console and multiple windows on one session. When in normal or clear bypass mode, characters are only displayed on the console if the session is the active session and the selected window is in update-on mode.

When Session Manager bypass mode is set, the Session Manager control of windows and sessions is bypassed. Characters are displayed on the console just as though the console was a single-session, single-window display. Characters are not saved by the Session Manager in this mode and, when bypass mode is cleared, cannot be "refreshed."

**Defaults:** When a console is first started, it is not in bypass mode. The bypass mode when the current program begins execution is dependent upon the state that the last program left it in.

**Conforms to:** **IsBypass**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [SetBypass](#), [ClrBypass](#)

---

**iscon**

Test to see if a file is really the console.

```
#include <stdio.h>
int iscon ( FILE * file )
```

---

*file*                   »   pointer to open file's FCB

- Operation:**       The file associated with *file* is tested to see if it is the console.
- Returns:**         A true/false value. A non-zero return value indicates that *file* is the console; a zero return value indicates that it is not.
- Errors:**          No error is reported. If *file* is not open, it is not the console and a false value is returned.
- Notes:**          The normal use of this function is with the standard file `stdin` or `stdout` to see if the file's input or output has been redirected or not.

**Conforms to:**               **iscon**    q ANSI      q DOS      n THEOS      q POSIX

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    FILE * infile;

    cprintf("stdout %s been redirected.\n\n",
           iscon(stdout) ? "has not" : " ");
    if (!(infile = fopen(argv[1], "r")))
        exit(fperror());

    printf("\nThe file \"%s\" is on the device \"%s\"\n",
           argv[1], _lubname(getdev(infile)));

    printf("That file %s on a serial device.\n",
           isatty(infile)? "is" : "is not");
    if (isatty(infile))
        printf("%s the console.\n",
              iscon(infile)? "And it is" : "But it is not");
    fclose(infile);
}
```

### Output:

```
>test con
stdout has not been redirected.

The file "CON" is on the device "CONIN"
That file is on a serial device.
And it is the console.

>test com > test.out
stdout has been redirected.

>list test.out
The file "COM" is on the device "COM1"
That file is on a serial device.
But it is not the console.

>
```

## ispcterm

Test the attached console to see if it is PC Term-compatible.

```
#include <stdlib.h>
int ispcterm ( void )
```

**Operation:** The current console attachment is examined to see if it is PC Term-compatible.

**Returns:** True/false indicator. A return of zero indicates that the console is not PC Term-compatible; a non-zero return indicates that it is.

**Notes:** PC Term-compatible consoles use scancodes for input keys and display the IBM PC character set.

To qualify as a PC Term-compatible console, the console must be attached with one of the following class code numbers: 53, 90–99, 170–199 and 210–219.

**Conforms to:**            **ispcterm**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [isatty](#), [iscon](#), [isvga](#)

### isprtnum

Determines the logical unit block (lub) number for a printer name.

```
#include <stdlib.h>
```

```
int isprtnum ( char * name )
```

---

*name*                    »    pointer to printer name to look up

**Operation:**        The *name* is examined and, if it is a valid name for one of the printers, the corresponding lub is returned.

**Returns:**            The lub number for the printer.

                      If *name* is an invalid printer name, a zero is returned.

**Notes:**            The name may contain any of the three valid forms for printer names. For instance, PRINTER3 may be specified with “PRT3,” “PRINT3” or “PRINTER3.”

                      The printer number may be omitted, in which case PRINTER1 is assumed. For instance, “PRT” and “PRT1” both refer to PRINTER1.

                      The indicated printer does not have to be attached to return the proper lub number.

**Conforms to:**            **isprtnum**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**              [\\_lubname](#)

---

#### Example:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```
{
```

```
    printf("\nPrinter 5 is using class code number %i.\n",  
           getclass(isprtnum("PRT5")));
```

```
}
```



## IsSession

Test if console is session-capable.

```
#include <wmapi.h>
int IsSession ( void )
```

**Operation:** Test the Session Manager to see if this program's console is capable of session-switching.

**Returns:** A true/false value. A non-zero return value means that it is capable of session-switching; a zero return value means that it is not.

**Notes:** Session-switching refers to the ability to share one console display and keyboard between two or more separate user partitions and programs. When a display is capable of session-switching, then it is possible that the information displayed by this program is not seen by the operator because another session is active on the console.

The [IsActive](#) function can test if the program's console is the active session. The [wSwitch](#) function can enable/disable the session-switching operation.

**Conforms to:** [IsSession](#) q ANSI q DOS n THEOS q POSIX

**See also:** [IsActive](#), [wGetActive](#), [wGetSess](#), [wSwitch](#), [wSwitchTo](#)

**isvga**

Test the attached console to see if it is VGA-compatible.

```
#include <stdlib.h>
int isvga ( void )
```

**Operation:** The current console attachment is examined to see if it is VGA-compatible.

**Returns:** True/false indicator. A return of zero indicates that the console is not VGA-compatible; a non-zero return indicates that it is.

**Notes:** A VGA console means that it can display the copyright ( © ) and registered ( ® ) symbols, and that it is VGA/EGA, meaning that it can do text or graphics.

**Conforms to:**                    **isvga**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [iscon](#), [ispcterm](#)

## itoa

Convert an integer to an ASCII string.

```
#include <stdlib.h>
char * itoa ( char * string, int i )
```

---

*i*                      »   integer value  
*string*                »   pointer to string storage

**Operation:**        Converts the short integer *i* to a character string and saves it in *string*.

**Returns:**           A pointer to *string*.

**Conforms to:**                **itoa**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [atof](#), [atoi](#), [atol](#), [atoll](#), [lltoa](#), [ltoa](#), [ltoc3](#), [ltoi2](#), [ltoi3](#)

**keyclose**

Close the system keywords file used by the [getkey](#) function.

```
#include <stdlib.h>
void keyclose ( void )
```

**Operation:** The SYSTEM.TEOS32.KEYWORD $n$  file used by the [getkey](#) function is closed.

**Returns:** No value is returned.

**Errors:** No errors are detected.

**Notes:** The file is also closed by the [fcloseall](#) function.

**Conforms to:** **keyclose** q ANSI q DOS n THEOS q POSIX

**See also:** [exit](#), [fclose](#), [fcloseall](#), [getkey](#)

---

**Example:**

```
#include <stdlib.h>

void
main(int argc, char * argv[])
{
    char    typekey[9];

    if (argc<2)
        syserr(1, 133, NULL);           // missing keyword
    if (*argv[1] != '(')
        syserr(1, 134, NULL);           // missing parenthesis
    if (!matcharg(argv[2], typekey, getkey(2, typekey)))
        syserr(1, 27, argv[2]);         // invalid option
    keyclose();

    ...
}
```

## killtask

Stop a child subtask of the current task.

```
#include <process.h>
void killtask ( int pid )



---


pid           »   process number
```

**Operation:** The child subtask executing in *pid* is stopped by forcing it to perform an [exit](#) function call.

**Notes:** All child tasks of *pid* are also terminated.

**Restrictions:** *pid* must be the process number of a task that is subordinate to the current task.

**Conforms to:** **killtask** q ANSI q DOS n THEOS q POSIX

**See also:** [exit](#), [fork](#), [forktask](#), [spawn](#) functions, [system](#)

---

### Example:

```
#include <process.h>
#include <semaphore.h>

void
main()
{
    int    child_pid;
    int    done;

    done = semaphore("done");

    if ((child_pid=fork())==0) {
        ...        // child task code
    }
    ...            // parent task code
    semawait(done);                // wait for child to say it's done

    killtask(child_pid);           // finished with child, stop it
    ...
}
```

## **l3tol**

Converts an array of three-byte integers to an array of long integers.

```
#include <stdlib.h>
```

```
void l3tol ( long * lptr, char * cptr, int n )
```

---

<i>cptr</i>	»	pointer to array of three-byte integers
-------------	---	---

<i>lptr</i>	»	pointer to storage for long integers
-------------	---	--------------------------------------

<i>n</i>	»	number of values to convert
----------	---	-----------------------------

**Operation:** The first *n* three-byte values stored in *cptr* are converted to long integers and stored in the array pointed to by *lptr*.

**Notes:** There are no standard functions that can manipulate or use the three-byte integer value. It must be converted back to a long integer value for actual usage. The sole purpose of the three-byte integer value is more compact external storage of long integer values that can fit in three bytes.

Three-byte integer values are created with the [ltoc3](#) or [ltoi3](#) functions.

Three-byte integers are used in the FDB (File Directory Block). Refer to the header files DISKADDR.H, FDB.H and ULB.H.

**Conforms to:** **l3tol** q ANSI q DOS n THEOS q POSIX

**See also:** [lltoa](#), [ltoa](#), [ltoc3](#), [ltoi2](#), [ltoi3](#)

## l64a

Convert a long integer value to a base-64 ASCII string.

```
#include <stdlib.h>
char * l64a ( long l )

_____
l           » long integer value
```

**Operation:** The base-10 value *l* is converted to a base-64 value.

**Returns:** A pointer to the converted, base-64 string.

**Notes:** In a base-64 ASCII number, the characters used to represent the base-10 digits are:

<i>base-10 value</i>	<i>base-64 character</i>
0	.
1	/
2–11	“0”–“9”
12–37	“A”–“Z”
38–63	“a”–“z”

**Restrictions:** The value long integer value *l* should not be negative.

The converted base-64 string is local to this function and will be overwritten the next time that it is called. You should copy this string immediately after using this function.

**Conforms to:** l64a q ANSI q DOS n THEOS q POSIX

**See also:** [a64l](#)

**labs**

Compute the absolute or unsigned value of a long integer value.

```
#include <stdlib.h>
long labs ( long l )

_____
l          »   long integer value
```

**Operation:**        The absolute or unsigned value of the long integer argument *l* is computed and returned.

**Returns:**         The long integer absolute value of *l* is returned.

**Conforms to:**        labs    n ANSI    n DOS    n THEOS    q POSIX

**See also:**         [abs](#), [cabs](#), [fabs](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main(void) {
    long l;

    // Test abs
    printf("\nEnter a large integer: "); // get value from operator
    scanf("%ld",&l);                     // convert to numeric
    printf("The absolute value of %li is %li\n",l,labs(l));
}
```

---

**Output:**

>example

```
Enter a large integer: -87654321
The absolute value of -87654321 is 87654321
```

>



## **ldeletek**

Deletes a record from a direct-access organization file.

```
#include <stdio.h>
unsigned short ldeletek ( FILE * file, long _far * key )
```

---

*file*                   »   pointer to file's fcb  
*key*                    »   pointer to record number

- Operation:**       The record whose relative record number is the value in *key* is deleted by writing binary zeros over the entire allocated length of the record.
- Returns:**         A zero. No errors are detected or reported unless the record is locked by another user and you have requested that locked records are detected and reported (see below).
- Notes:**          There is no difference between a record that has never been written, one that has been deleted or one whose record contents is all binary zeros.

If *file* has been opened with access “r+” and without specifying the access modifier “m,” then deleting a record releases any other record locks set by this user for this file.

If *file* is in use by another user and that user has the desired record locked, the *ldeletek* function will wait for that lock to be released before deleting the record. If the delete is not immediately successful because the record is locked by another user, the function may wait awhile to see if the lock is cleared by another user. The *scr.lock\_wait* item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in *scr.lock\_wait* can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the delete request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in *scr.lock\_pid*.

See also “Automatic Record-Locking:” on page 224.

## 382 *ldeletek*

---

**Restrictions:** This function may only be used with a file opened with direct organization, and write or update access.

**Conforms to:** **ldeletek**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [deletek](#), [lreadk](#), [lreadn](#), [lwritek](#)

## ldexp

Compute a value given the mantissa and the base-2 exponent.

```
#include <math.h>
double ldexp ( double x, int exp )
```

---

*exp*                   » exponent  
*x*                    » floating-point value

**Operation:**        Compute the value of  $x \times 2^{\textit{exp}}$ .

**Returns:**         The computed value.

**Errors:**           If an overflow occurs, the return value is  $\pm\text{HUGE\_VAL}$  and [errno](#) is set to ERANGE.

**Restrictions:**    This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**        **ldexp**    n ANSI      n DOS      n THEOS    n POSIX

**See also:**          [frexp](#), [modf](#)

**`_leapyear`**

Determines if a given year is a leap year.

```
#include <time.h>
int _leapyear ( int year )
_____
year           »   year number, including century
```

**Operation:** The *year* is tested to see if it is a leap year in the Gregorian calendar.

**Returns:** A true/false value indicating whether or not *year* is a leap year. A false value is zero, a true value is non-zero.

**Notes:** A leap year is evenly divisible by 4 but not by 100, unless it is also evenly divisible by 400.

**Conforms to:** `_leapyear`    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>
#include <time.h>

main()
{
    int      year, ly;

    while (1) {
        printf("Enter year number: ");
        scanf("%d",&year);                // get year number
        if (year==0)
            break;                        // end program
        ly = _leapyear(year);

        printf("The year %d %s a leap year.\n", year,
            ly ? "is" : "is not");
    }
}
```

---

**Output:**

```
>test
Enter year number: 1902
The year 1902 is not a leap year.
Enter year number: 1700
The year 1700 is a leap year.
Enter year number: 1800
The year 1800 is not a leap year.
Enter year number: 0

>
```

## listen

Set a socket to listen for incoming connection.

```
#include <socket.h>
int listen ( SOCKET s, int backlog )
```

---

*backlog*               »   value for maximum queue of pending connections  
*s*                        »   socket number

**Operation:**       The socket *s* is put into “passive” mode where incoming connections are acknowledged and queued pending acceptance by the process.

**Returns:**           A zero. When an error is detected, `SOCKET_ERROR` is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:**            When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

<i>Code</i>	<i>Meaning</i>
TSAEADDRINUSE	An attempt has been made to listen to a socket address in use.
TSAEINPROGRESS	A blocking THEOS Sockets call is in progress.
TSAEINVAL	The socket has not been bound with bind or is already connected.
TSAEISCONN	The socket is already connected.
TSAEMFILE	No more file descriptors are available.
TSAENETDOWN	The THEOS Sockets implementation has detected that the network subsystem has failed.
TSAENOBUFS	No buffer space is available.
TSAENOTSOCK	The descriptor <i>s</i> is not a socket.
TSAEOPNOTSUPP	The referenced socket is not of a type that supports the listen operation.

**Notes:**            To accept connections, a socket is first created with [socket](#), a backlog for incoming connections is specified with `listen`, and then the connections are accepted with [accept](#). `listen` applies only to sockets that support connections, i.e. those of type `SOCK_STREAM`.

This function is typically used by servers that could have more than one connection request at a time: If a connection request arrives with the queue full, the client will receive an error with an indication of `TSAECONNREFUSED`.

`listen` attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call to `listen` or [accept](#) will re-fill the queue to the current or most recent “backlog,” if possible, and resume listening for incoming connections.

## 386 *listen*

---

**Conforms to:** `listen` q ANSI q DOS n THEOS q POSIX

**See also:** [accept](#), [connect](#), [socket](#)

**Example:** See the example for the function [socket](#).

## load, unload

Load a program or file into memory.

```
#include <process.h>
unsigned short load ( const char _far * filename, int type )
void unload ( unsigned segment )
```

---

<i>filename</i>	»	pointer to string containing program name
<i>segment</i>	»	memory segment number
<i>type</i>	»	type of program to load

<b>Operation:</b>	<b>load</b>	The file specified by <i>filename</i> is loaded into memory. The memory segment used is returned.
	<b>unload</b>	The file loaded into memory in memory <i>segment</i> is unloaded.
<b>Returns:</b>	<b>load</b>	The memory segment address used to load the file into memory.
<b>Errors:</b>	<b>load</b>	A return of zero indicates that the file could not be loaded.
<b>Notes:</b>	<b>load</b>	The specification of the file in <i>filename</i> is assumed to be in the current working directory unless <i>filename</i> specifies the path to the file.

The *type* field tells the operating system the type and usage of the file. Valid *type* codes include:

<i>type</i>	<i>Meaning</i>
0	<i>filename</i> is a data file and will be loaded with read-only access.
1	<i>filename</i> is a device-driver program.
2	<i>filename</i> is a command program.
3	<i>filename</i> is an overlay program with code and possibly data.
4	Reload the code segment only of the overlay program <i>filename</i> .

Requesting a load of a file that is already in memory does not load an additional copy of the file. Instead, the usage count of the file is incremented and the memory segment address of the already loaded file is returned.

**unload** Unloading a file that is still in use by another task does not remove the file from memory. Instead, the usage count of the file is decremented. Only when the usage count decrements to zero will the file be removed from memory.

The usage counter of a loaded file is also decremented when your program uses the `exit`, `execl`, `execvp`, `execv`, `system`, or `csi` functions.

## 388 *load, unload*

---

<b>Restrictions:</b>	<b>load</b>	The loaded file is available for read or execution but it cannot be changed (no write access).			
	<b>unload</b>	You may not unload a file or program that you did not load. Such requests will be ignored.			
<b>Conforms to:</b>	<b>load</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>unload</b>	q ANSI	q DOS	n THEOS	q POSIX

---

### **Example:**

```
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>

void
main()
{
    unsigned data_seg;
    char      s[100];

    data_seg = load("critical.data", 0);

    farcpy(getds(), s, data_seg, NULL, 99);
    ...
    unload(data_seg);
}
```



## load\_yn

Initialize the strings used by the yesno functions.

```
#include <stdio.h>
void load_yn ( void )
```

**Operation:** The keywords numbered 0, 1, 203 and 264 are read from the SYSTEM.TEOS32.KEYWORD $n$  file (“ $n$ ” represents the current language code). The first letters of each of these keywords is then saved for usage by the [yesno](#) and [\\_yesno](#) functions.

**Returns:** No value is returned.

**Notes:** The yesno functions accept a “yes,” “no,” “all” and “go” responses from the operator. By using this load\_yn function, the [yesno](#) functions are initialized to the current language being used on this computer system.

**Conforms to:** [load\\_yn](#) q ANSI q DOS n THEOS q POSIX

**See also:** [getkey](#), [yesno](#), [\\_yesno](#), [yesnoall](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    int    reply;

    load_yn();                // initialize yesno function

    printf("\nAre you ready to proceed? ");
    reply = yesno();          // get yes or no response

    if (!reply)
        exit(0);              // they are not ready

    ...
}
```

**localeconv**

Get information about the current locale settings for numeric data formatting.

```
#include <locale.h>
struct lconv * localeconv ( void )
```

**Operation:** Get detailed information about the locale settings for formatting of numeric and monetary quantities.

**Returns:** A pointer to an lconv type structure.

**Notes:** The lconv structure is defined in the LOCALE.H file. The meanings or usages of the members of this structure are:

<i>member</i>	<i>Meaning</i>
*currency_symbol	Currency symbol.
*decimal_point	Decimal point character used in non-monetary formats.
*grouping	Size of each group of digits to the left of the decimal point in non-monetary formats.
*int_curr_symbol	International currency symbol.
*mon_decimal_point	Decimal point character for monetary quantities.
*mon_grouping	Size of each group of digits to the left of the decimal point in monetary quantities.
*mon_thousands_sep	Character used to separate digit groups to the left of the decimal point in monetary formats.
*negative_sign	String denoting the sign for negative monetary values.
*positive_sign	String denoting the sign for non-negative monetary values.
*thousands_sep	Character used to separate digit groups to the left of the decimal point in non-monetary formats.
frac_digits	Number of digits to the right of the decimal point in formatted monetary quantities.
int_frac_digits	Number of digits to the right of the decimal point in internationally-formatted monetary quantities.
n_cs_precedes	Set to one if the currency symbol precedes the value for a negative-formatted monetary quantity. Set to zero if the symbol follows the value.
n_sep_by_space	Set to one if the currency symbol is separated by a space from the value for a negative-formatted monetary quantity. Set to zero if there is no space separation.

<i>member</i>	<i>Meaning</i>
n_sign_posn	Position of positive sign in negative-formatted monetary quantities.
p_cs_precedes	Set to one if the currency symbol precedes the value for a non-negative-formatted monetary quantity. Set to zero if the symbol follows the value.
p_sep_by_space	Set to one if the currency symbol is separated by a space from the value for a non-negative formatted monetary quantity. Set to zero if there is no space separation.
p_sign_posn	Position of positive sign in non-negative formatted monetary quantities.

**Restrictions:** The structure pointed to by the return value is local to the localeconv function and is overwritten by each call to localeconv.

**Conforms to:** **localeconv** n ANSI n DOS n THEOS q POSIX

**See also:** [setlocale](#), [strcoll](#), [strftime](#), [strxfrm](#)

---

**Example:**

```
#include <stdio.h>
#include <locale.h>

main()
{
    struct lconv *lc;

    lc = localeconv();

    printf("currency_symbol = %s\n",lc->currency_symbol);
    printf("decimal_point = %s\n",lc->decimal_point);
    printf("grouping = %s\n",lc->grouping);
    printf("int_curr_symbol = %s\n",lc->int_curr_symbol);
    printf("mon_decimal_point = %s\n",lc->mon_decimal_point);
    printf("mon_grouping = %s\n",lc->mon_grouping);
    printf("mon_thousands_sep = %s\n",lc->mon_thousands_sep);
    printf("negative_sign = %s\n",lc->negative_sign);
    printf("positive_sign = %s\n",lc->positive_sign);
    printf("thousands_sep = %s\n",lc->thousands_sep);
    printf("frac_digits = %i\n",lc->frac_digits);
    printf("int_frac_digits = %i\n",lc->int_frac_digits);
    printf("n_cs_precedes = %i\n",lc->n_cs_precedes);
    printf("n_sep_by_space = %i\n",lc->n_sep_by_space);
    printf("n_sign_posn = %i\n",lc->n_sign_posn);
    printf("p_cs_precedes = %i\n",lc->p_cs_precedes);
    printf("p_sep_by_space = %i\n",lc->p_sep_by_space);
    printf("p_sign_posn = %i\n",lc->p_sign_posn);
}
```

**Output:**

```
>example
currency_symbol = $
decimal_point = .
grouping =
int_curr_symbol = USD
mon_decimal_point = .
mon_grouping =
mon_thousands_sep = ,
negative_sign = -
positive_sign = +
thousands_sep = ,
frac_digits = 2
int_frac_digits = 2
n-cs_precedes = 1
n_sep_by_space = 0
n_sign_posn = 4
p-cs_precedes = 1
p_sep_by_space = 0
p_sign_posn = 4

>
```

## localtime

Converts a UTC calendar time value into a local-time time structure.

```
#include <time.h>
struct tm * localtime ( const time_t * timer )
```

---

*timer*                    »    pointer to calendar time

**Operation:**        The `localtime` function converts the calendar time pointed to by *timer* into a structure of type `tm`, expressed as local time. The [tzset](#) function is called. The converted `tm` structure is in static storage, local to the `localtime` function and is re-used each time that `localtime` is called.

**Returns:**            A pointer to the converted `tm` structure.

**Notes:**             The calendar time value *timer* is assumed to be the current UTC time.

When *timer* is `NULL`, the current local time is used.

**Restrictions:**      Similar to [gmtime](#) and [mktime](#), the `tm` structure used by this function is local to the function and is re-used each time that the function is called.

**Conforms to:**                `localtime`    n ANSI        n DOS        n THEOS    n POSIX

**See also:**            [asctime](#), [clock](#), [ctime](#), [difftime](#), [gmtime](#), [mktime](#), [strftime](#), [time](#), [tzset](#)

### Example:

```
#include <stdio.h>
#include <time.h>

void
main(void)
{
    struct tm curtime;
    time_t timer;
    char buffer[80];

    time(&timer);                // get current time
    curtime = * localtime(&timer); // convert to struct
    strftime(buffer, 80, "%A, %B %d, %Y, at %H:%M %p %Z",&curtime);
    printf("It is now %s\n",buffer);

    // adjust date for new deadline

    curtime.tm_mday += 3;        // add 3 days
    curtime.tm_hour = 8;        // set to 8 am
    curtime.tm_sec = curtime.tm_min = 0; // no minutes, seconds
    mktime(&curtime);           // normalize
    if (curtime.tm_wday==0)      // Sunday?
        curtime.tm_mday += 1;    // change to Monday
    else if (curtime.tm_wday==6) // Saturday?
        curtime.tm_mday += 2;    // change to Monday
    mktime(&curtime);           // normalize again
    strftime(buffer, 80, "%A, %B %d, %Y, at %H:%M %p %Z", &curtime);
    printf("\nYour deadline has been moved to:\n%s\n", buffer);
}
```

Also, refer to the `strftime` example on page 612.

---

### Output:

It is now Thursday, July 04, 1997, at 12:58 pm PDT

Your deadline has been moved to:  
Monday, July 08, 1997, at 08:00 am PDT

## locate

Find a file on disk and get a copy of the file's fdb.

```
#include <sc.h>
```

```
FDB * locate ( const char _far * filename, char buffer[256], short * lub )
```

---

*buffer* » pointer to working storage

*filename* » pointer to string containing file description

*lub* » pointer to logical unit number storage

**Operation:** Using the file name specified in *filename*, the attached disks are searched in the default disk search sequence.

If *filename* specifies a drive code, then only that one disk is searched.

If *filename* is a simple name (i.e., does not contain a period character), then the current default library is searched for that member name.

*filename* may include explicit path specifications, either relative to the current directory or from the root.

**Returns:** A pointer to a copy of the file's fdb. The lub number of the drive where the file was found is assigned to the location pointed to by *lub*.

If the file cannot be found, a NULL pointer is returned.

**Notes:** Only files that are accessible from the current account are searched. That is, if *filename* does not specify an account name, then only files owned by the current account or the public system account are searched.

When used on a THEOS 32 Version 4 system, the *lub* argument may be specified as NULL. In that situation, the lub number is not returned.

**Restrictions:** *buffer* must point to a buffer of at least 256 bytes.

**Conforms to:** **locate** q ANSI q DOS n THEOS q POSIX

**See also:** [getfn](#), [\\_get\\_sect](#)

**lockf, locking**

Locks a portion of a file so that other users cannot access it while this program has it locked. These functions can also unlock a file or test to see if a portion of a file is already locked by another user.

```
#include <unistd.h>
int lockf ( int fhandle, int mode, unsigned long size )

#include <locking.h>
int locking ( int fhandle, int mode, unsigned long size )
```

---

<i>fhandle</i>	»	file handle or number
<i>mode</i>	»	code for lock or unlock operation
<i>size</i>	»	length of region to lock or unlock

**Operation:** These two functions are almost identical in operation and perform functions that are similar to the [filelock](#) function described on page 204. (Internally, these functions use the [filelock](#) function.)

Although these two functions perform similar operations, and their operation is dependent upon the *mode* argument, the values used for *mode* differ between the two functions.

The main differences between these two functions and the [filelock](#) function are that [filelock](#) uses a pointer to an open file's FCB to identify the file and that region of the file is identified by its starting location and its ending location. The `lockf` and `locking` functions use the handle of an open file to identify it and the region of the file is defined by the current position pointer for a specified length from that position pointer.

**lockf** The values of *mode* may be any value as defined in the UNISTD.H file:

mode	Value	Meaning
F_ULOCK	0	Unlock the region
F_LOCK	1	Lock the region
F_TLOCK	2	Lock the region
F_TEST	3	Test region for locks

**locking** The values of *mode* may be any value as defined in the LOCKING.H file:

mode	Value	Meaning
LK_LOCK LK_RLCK	1	Lock the region. If locked by another user, retries once per second for a period of five seconds.
LK_NBLCK LK_NBRLOCK	2	Lock the region. If Locked by another user, does not try again.
LK_UNLCK	3	Unlock the region.



**Returns:** Both of these functions return a zero if the operation was successful. A non-zero value is returned when the attempt was unsuccessful, in which case the [errno](#) variable is set to indicate the specific cause of the error.

**Errors:** When an error occurs, the two functions set [errno](#) to indicate the cause of the error.

lockf	locking	Value	Meaning
EACCES	EACCES	18	Region of file is locked by another user.
EBADF	EBADF	24	Invalid file handle.
EDEADLOCK	EDEADLOCK	404	Region of file is locked by another user, even after five seconds.
	EINVAL	428	Invalid argument.

**Notes:** Locking, unlocking or testing for locks on a file does not cause the file to be read or written.

The specified region of the file may be outside of the current limits of the file, that is, beyond the current end of the file.

More than one region of a file may be locked by a user.

When unlocking a region of a file, be sure to use the same length from the same starting location. Adjacent, locked regions are maintained separately and they must be unlocked separately.

Closing a file removes all locks for that file.

**Restrictions:** *handle* must be a handle to an open disk stream file.

<b>lockf</b>	q ANSI	q DOS	n THEOS	n POSIX
<b>locking</b>	q ANSI	n DOS	n THEOS	n POSIX

**See also:** [fclose](#), [fcloseall](#), [filelock](#), [relock](#), [recunlock](#), [unlock](#)

***\_lockres, \_lockset, \_lockshare, \_locktest, \_lockwait***

Maintain a lock on a memory semaphore.

```
#include <sc.h>
void _lockres ( void * offset, unsigned seg )
void _lockset ( void * offset, unsigned seg )
void _lockshare ( void * offset, unsigned seg )
unsigned short _locktest ( void * offset, unsigned seg )
void _lockwait ( void * offset, unsigned seg, int type )
```

---

<i>offset</i>	»	offset within memory segment
<i>seg</i>	»	memory segment selector
<i>type</i>	»	coded type of lock requested

**Operation:** All of these functions use the arguments *offset* and *seg*. Typically, these refer to an actual memory location. However, there is no actual correlation between the value of these arguments and memory. The value is used as a semaphore number and can be any value that two tasks agree upon.

- \_lockres*** Release any lock (exclusive or shared) that your task has placed on the semaphore specified by *offset* and *seg*.
- \_lockset*** Place an exclusive lock on the semaphore specified by *offset* and *seg*. If the semaphore is already locked by another task (either exclusive or shared), wait for the lock to be released. If the semaphore is currently locked as a shared lock by your task and you are the only one with a lock on the semaphore, convert the lock to an exclusive lock.
- \_lockshare*** Place a shared lock on the semaphore specified by *offset* and *seg*. If the semaphore is already exclusively locked by another task, then the function waits for that lock to be released. If the semaphore is currently exclusively locked by your task, then it is converted to a shared lock.
- \_locktest*** Test the semaphore specified by *offset* and *seg*.
- \_lockwait*** This is the general lock interface used by the *\_lockset* and *\_lockshare* functions. A *type* code of zero is a request for an exclusive lock (*\_lockset*); a *type* code of one is a request for a shared lock (*\_lockshare*).

In addition to this basic locking operation, the *\_lockwait* function can attempt to perform the requested lock once or it can wait until it is successful. The *type* field is actually bit-mapped:

<i>bit</i>	<i>value</i>	<i>Meaning</i>
0	0	Exclusive lock request.
	1	Shared lock request.
1	0	Wait for lock to be set.
	1	Try once. If can't be done, then return.

If the try-once option is used, then use the `_locktest` function to determine if the request was successful or not.

**Returns:**        `_locktest`    The status of the lock(s) placed on the semaphore. A zero return means that it is not locked by any task; a positive value return is the process number of the locking task; a negative value return is the negative number of tasks that have placed a shared lock on the semaphore.

**Errors:**        No errors are detected.

**Notes:**        The semaphore referred to here has no relationship to the semaphores used by the [semaphore functions](#) described on page 550.

An ***exclusive lock*** is just that: No other processes can place a lock on the semaphore while your task has an exclusive lock on it. This type of lock is used when you want to be sure that your task is the only one performing some action, for instance, updating some system resource.

A ***shared lock*** means that other processes can place a shared lock on the semaphore but they cannot place an exclusive lock on it. This type of lock is used when it is okay if others are also using the resource but no other task needs exclusive usage of it. For instance, a shared lock would be used when you are merely examining a system resource.

The specific *offset* and *seg* values used must be some value that is agreed upon by all tasks that need to refer to the resource. For instance, the user accounting structure is a system resource that is accessed by several system calls. These system calls use a specific memory location to restrict access to the resource so that no task tries to read information while another task is changing it.

**Restrictions:**    These functions do not actually place any kind of hardware or low-level lock on a location. If a program does not use these functions and has access rights to the resource, it can use it.

The functions `_lockshare`, `_locktest` and `_lockwait` can only be used on THEOS 32 Version 4 or above.

<b>Conforms to:</b>	<code>_lockres</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>_lockset</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>_lockshare</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>_locktest</code>	q ANSI	q DOS	n THEOS	q POSIX
	<code>_lockwait</code>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**        [acc\\_access](#), [acc\\_maint](#), [filelock](#), [lockf](#), [locking](#), [reclock](#), [recunlock](#), [unlock](#)

**log, log2, log10**

These functions compute the natural, base 2 or common logarithm of a value.

```
#include <math.h>
double log ( double x )
double log2 ( double x )
double log10 ( double x )
```

---

*x*                      »    value to compute log of

**Operation:**      **log**              The natural logarithm of *x* is computed. A natural logarithm is defined as:

$$x = e^y$$

$$y = \log_e x$$

Where  $e \approx 2.7182818$

**log2**              The base-2 logarithm of *x* is computed. A base-2 logarithm is defined as:

$$x = 2^y$$

$$y = \log_2 x$$

**log10**            The common logarithm of *x* is computed. A common logarithm is defined as:

$$x = 10^y$$

$$y = \log_{10} x$$

**Returns:**              The requested logarithm of *x*.

**Errors:**              When *x* is less than zero, [errno](#) is set to EDOM and -HUGE\_VAL is returned.

When *x* is zero, [errno](#) is set to ERANGE and a zero is returned.

**Notes:**              Natural logarithms are also called Napierian or hyperbolic logarithms.

Common logarithms are also called Briggsian logarithms.

This function uses BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

Logarithms using bases other than 2, *e* or 10 can be computed using the formula for change of base:

$$\log_{\text{base1}} x = \log_{\text{base2}} x / \log_{\text{base2}} \text{base1}$$

For instance, to compute the  $\log_3 x$  you could compute  $\log_e x / \log_e 3$ . Thus,  $\log_3 x$  is computed with  $\log(x) / \log(3)$ .

<b>Conforms to:</b>	<b>log</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>log2</b>	q ANSI	q DOS	n THEOS	q POSIX

**log10**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**    [exp](#)

---

**Example:**

```
#include <math.h>
#include <errno.h>

void
main()
{
    double x;

    x = log(0);
    printf("\nlog(0) produces %g with errno set to %i",x,errno);
    errno = 0;           // clear any error code set
    x = log(2);
    printf("\nlog(2) produces %g with errno set to %i",x,errno);
    errno = 0;           // clear any error code set
    x = log10(2);
    printf("\nlog10(2) produces %g with errno set to %i",x,errno);
}
```

---

**Output:**

>test

```
log(0) produces 0 with errno set to 427
log(2) produces 0.693147180559945286 with errno set to 0
log10(2) produces 0.301029995663981198 with errno set to 0
```

### logname

Retrieves the account name that you are currently logged on to.

```
#include <stdlib.h>
char * logname ( void )
```

**Operation:** A local copy is made of the currently logged on account name. This copy has all trailing spaces removed.

**Returns:** A pointer to the currently logged on account name.

When the program is operating as a background user, a NULL pointer is returned.

**Notes:** This function is identical to the [getlogin](#) function, except that it might return a NULL pointer rather than a pointer to a null string.

**Restrictions:** The copy of the trimmed account name is local to the `logname` function and will be reset each time that `logname` is called.

**Conforms to:** `logname` q ANSI q DOS n THEOS n POSIX

**See also:** [cuserid](#), [getlogin](#), [getuid](#)

---

#### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
    FILE * logfile;
    char account[9];

    logfile = fopen("log.file", "wa");
    if (logname()==NULL)
        strcpy(account, "Phantom");
    else
        strcpy(account, logname());
    fprintf(logfile, "User name = %s (%i)\n", account, getuid());
    fclose(logfile);
}
```

---

#### Output:

```
>test

>start test

>list log.file
```

```
User name = SAMPLES (2)
User name = Phantom (2)
```

## \_logon

Log onto another account.

```
#include <stdlib.h>
int _logon ( char * name, char * password, int change_name_flag )
```

---

*change\_name\_flag* » code to change/not change logname in SCR  
*name* » pointer to account name string  
*password* » pointer to password string

**Operation:** The user is logged onto the account with the account *name* and *password*.

When *change\_name\_flag* is non-zero, the logonname field in the SCR is set to the new account. When it is zero the logonname field is not changed. The useumum field in the SCR is always set to the number of the account that you are currently logged on to.

**Returns:** A success/fail indicator: A zero return indicates success; a non-zero return indicates failure.

**Errors:** When \_logon fails, the return value indicates the specific reason: a return value of one indicates that *name* is not a valid account name; a return value of two indicates that *name* is valid but that *password* is not.

**Notes:** The *password* field is specified with plain text. System security can be easily violated by users that know a program uses this function to log onto an account protected with a password that is normally encrypted.

**Restrictions:** *name* cannot be NULL or a null string and it cannot have any leading, trailing or embedded space characters.

**Conforms to:**                    \_logon    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [logname](#)

**Example:** This example program logs onto another account temporarily and then logs back onto the prior account. It assumes that the current account is not password-protected because there is no way to determine the password if there was one.

```
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <scr.h>

void
main()
{
    char  curacc[9];

    // get current account name

    memset(curacc, 0, 9);           // set to string terminators
    _fmemcpy(curacc, Scr->logonname, 8);
    strtrim(curacc);                // remove trailing spaces

    // log onto account but leave user name field the same

    if (_logon("worklogs", "syzzgy", 0)==0) {
        ...
        _logon(curacc, "", 0);
    }
}
```



## longfname

This function attempts to create the longest complete description of a file specification given a limiting text length.

```
#include <stdlib.h>
```

```
char * longfname ( const char * filename, char buffer[256], int size )
```

---

<i>buffer</i>	»	pointer to buffer for normalized file name
<i>filename</i>	»	pointer to string containing file description
<i>size</i>	»	maximum length of generated file name

**Operation:** Given the limitations of *size*, an attempt is made to create and return a string that contains as much of *filename* as possible with at least the flat file portion of *filename* retained.

The contents of *filename* are analyzed as a complete entity and as the result of [getflat](#).

- ▶ If the length of the contents of *filename* is less than or equal to *size*, then *filename* is copied to *buffer* and a pointer to *buffer* is returned.
- ▶ If the length of the string returned by [getflat](#) is greater than *size*, a NULL string is returned by **longfname**.
- ▶ If *size* is greater than the length of the string returned by [getflat](#), the flat file portion is copied to *buffer* and then as much of the path specification as will fit is copied to the beginning of *buffer*. First the right-most portion of the path is included and then the left-most portion. This process is repeated until no more portions of the path specification will fit. The omitted portions are indicated by using an ellipsis (...).

**Returns:** A pointer to the generated string in *buffer* or NULL if the flat file portion of *filename* is longer than *size*.

**Notes:** The string created by this function might not be usable as a file description except for display purposes. There may be critical information missing.

**Conforms to:**        **longfname**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**        [getflat](#), [getfn](#), [\\_norm\\_fn](#)

## 406 longfname

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main(int argc, char * argv[])
{
    char    buffer[256],
            work[256];
    char *  ptr;

    ptr = _norm_fn(argv[1], buffer);

    printf("\nFile name is \"%s\"",argv[1]);
    printf("\n_norm_fn is  \"%s\"",ptr);

    printf("\ngetflat is   \"%s\"",getflat(buffer));

    printf("\n\nlongfname is \"%s\"",longfname(buffer, work, 40));
    printf("\nlongfname org  \"%s\"",longfname(argv[1], work, 50));
}
```

---

### Output:

```
>test sample.file:s
```

```
File name is "SAMPLE.FILE:S"
_norm_fn is  "/C32/SAMPLE.FILE:S"
getflat is   "SAMPLE.FILE:S"
```

```
longfname is "/C32/SAMPLE.FILE:S"
longfname org "SAMPLE.FILE:S"
```

```
>test "account\subdir1\subdir2\subdir3/example.library.member"
```

```
File name is "account\subdir1\subdir2\subdir3/example.library.member"
_norm_fn is  "account\subdir1\subdir2\subdir3/example.library.member"
getflat is   "example.library.member"
```

```
longfname is "...subdir3/example.library.member"
longfname org "account\...subdir2\subdir3/example.library.member"
```

## longjmp

Restore the stack environment and instruction pointer saved by a [setjmp](#) function call.

```
#include <setjmp.h>
```

```
void longjmp ( jmp_buf * env, int value )
```

---

*env*                   »   buffer variable holding saved environment

*value*                »   return value to use for [setjmp](#) return

**Operation:**       The stack environment is restored using the saved information in the *env* structure. The instruction pointer is set to the saved instruction pointer in that structure, causing control to transfer to a location internal to the [setjmp](#) function, which saved the environment information.

**Returns:**          There is no return from this function. Control is transferred to a location internal to the [setjmp](#) function, which performs a return using the *value* argument as its return value. If *value* is zero, the return value of the [setjmp](#) is set to one.

**Errors:**          It is erroneous and will produce unpredictable results if the `longjmp` function is used after the routine that used the [setjmp](#) function returns.

**Notes:**           A call to [setjmp](#) saves the stack environment in *env*. A subsequent call to `longjmp` restores this saved environment. Thus, when the `longjmp` executes, the values of variables are unchanged (except register variables which are restored to the values they had when the [setjmp](#) call was originally performed).

The values of register variables are restored by this function call. However, this is not required by the ANSI standard and relying on this may not be portable to other C language environments.

**Restrictions:**   Do not use `longjmp` to transfer control from one overlay to another. Both the `longjmp` and the [setjmp](#) must be in the same overlay.

Do not use the `longjmp` to transfer control out of an interrupt service routine. `longjmp` may be used to transfer control out of a [signal](#) interrupt handling routine.

**Conforms to:**       **longjmp**    n ANSI    n DOS    n THEOS   n POSIX

**See also:**          [setjmp](#)

### Example:

```
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <sc.h>

void timeout(void);
jmp_buf env;

void
main()
{
    char string[80];

    printf("\nEnter text: ");
    gettimed(string, 10);           // accept string with 10 sec limit

    printf("\nYour text is: %s\n", string);
}

char *
gettimed(char *s, int timelimit)
{
    char *sptr;

    signal(SIGALRM, timeout);      // prepare for timed event
    alarm(timelimit);              // program timer

    if (!(setjmp(env))) {          // save environment and location
        sptr = gets(s);            // get text from operator
        alarm(0);                  // disable timer, text accepted
        signal(SIGALRM, SIG_DFL);
    }
    else {                          // executed if timeout occurred
        *s = 0;
        return NULL;              // NULL return = timeout
    }
    return sptr;                  // non-NULL return = okay
}

void
timeout(void)
{
    longjmp(env, 1);              // return to gettimed routine
}
```

## lreadk, lreadn

Reads records from direct-access organization files.

```
#include <stdio.h>
```

```
unsigned short lreadk ( FILE * file, long _far * key, void * record )
```

```
unsigned short lreadn ( FILE * file, long _far * key, void * record )
```

---

<i>file</i>	»	pointer to open file's fcb
<i>key</i>	»	pointer to record number
<i>record</i>	»	pointer to record storage buffer

**Operation:** Each of these functions reads a single record from a file opened as a direct-access organization file.

**lreadk** The record, whose relative record number is the value in *key*, is read and stored in *record*. The input position pointer is set to the end of this record's location in the file, whether it is successfully read or not.

**lreadn** Using the current input position pointer for *file*, the next valid record in the file is read. The key for that record is stored in *key* and the record contents are stored in *record*. The input position pointer is set to the end of the record's location in the file.

**Returns:** The file's allocated record length.

A zero return value indicates that the record was not found in the file.

A return value of -1 indicates that the read was unsuccessful because the record is locked by another user (see below).

**Errors:** A zero return indicates that the record was not read.

**lreadk** The record with *key* does not exist in the file. The buffer pointed to by *record* is set to binary zeros.

**lreadn** There are no records in the file following the current input position pointer. The end-of-file flag is set and can be tested with the `feof` function. The buffer pointed to by *record* is set to binary zeros but the value of *key* is not changed.

**Notes:** There is no difference between a record that has never been written, one that has been deleted or one whose record contents are all binary zeros.

If *file* has been opened with access "r+", then the record read by these functions is locked. If it was opened without specifying the access modifier "m," then attempting to read a record releases any other record locks set by this user for this file.

If *file* has been opened with access "r+", it is in use by another user and that user has the desired record locked, the `lreadk` and `lreadn` functions will wait for that lock to be released before reading the record.

If the read is not immediately successful because the record is locked by another user these functions may wait for awhile to see if the lock is cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked (default)
1	Wait for 50 milliseconds
<i>n</i>	Wait for <i>n</i> seconds

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the read request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`.

See also “Automatic Record-Locking:” on page 224.

**Defaults:** When a direct file is first opened, the input position pointer is set to the beginning of the file.

**Restrictions:** These functions may only be used with files opened with the direct access mode and read access.

**Conforms to:**

<b><code>lreadk</code></b>	q ANSI	q DOS	n THEOS	q POSIX
<b><code>lreadn</code></b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [eof](#), [feof](#), [fseek](#), [fsetpos](#), [ldeletek](#), [lseek](#), [lwritek](#), [rewind](#), [seek](#), [\\_seek](#), [tell](#), [\\_tell](#)

## lsearch, lfind

Perform a linear search of an array for a key.

```
#include <search.h>

void * lfind ( const void * key, const void * array_base, size_t nbrmemb,
               size_t size, int (*compar)() )
void * lsearch ( const void * key, const void * array_base, size_t nbrmemb,
                 size_t size, int (*compar)() )
```

---

<i>array_base</i>	»	pointer to array to search
<i>compare</i>	»	address/name of comparison function
<i>key</i>	»	pointer to key to find
<i>nbrmemb</i>	»	number of members in array
<i>size</i>	»	size of each item in array

**Operation:** Both of these functions perform a linear search on the array of items pointed to by *array\_base*. The array has *nbrmemb* items in it (starting with *array\_base*) and each item is *size* bytes wide. The item searched for is identified by the object pointed to by *key*. The function *compar* is used to test each item against *key* for a match.

**lfind** Search the *array\_base* and report if it finds the requested *key*.

**lsearch** Searches the *array\_base* and either finds the existing item that matches *key* or, if it is not found, adds a new item to the array.

**Returns:** **lfind** A pointer to the item that matches *key*. If *key* is not found in *array\_base*, a NULL pointer is returned.

**lsearch** If *key* was found in *array\_base*, the return value is the pointer to the item. If *key* was not found the return value is the pointer to the location where the new item was added.

**Notes:** These functions make no presumption regarding the contents or organization of each item in the array; the *compar* function must know that.

*compar* is a function that is given two pointer arguments, similar to [strcmp](#). The first argument to *compar* will be a pointer to a member of *array\_base*; the second argument will be a pointer to *key*. The *compar* function must return an integer of zero if the *key* matches the item or a non-zero if *key* does not match.

**Restrictions:** **lsearch** Since this function might add an item to *array\_base*, there must be sufficient space available in the array for a new item.

**Conforms to:**

<b>lfind</b>	q ANSI	q DOS	n THEOS	q POSIX
<b>lsearch</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [bsearch](#), [qsort](#)

## 412 lsearch, lfind

---

### Example:

```
#include <stdio.h>
#include <string.h>
#include <search.h>

void
main(int argc, char * argv[])
{
    char    ** magic_found,
            * key = "MAGIC";

    magic_found = lfind(&key, argv, argc, sizeof(char *), strcmp);

    if (magic_found)
        printf("\nFound");
    else
        printf("\nNot found");
}
```

---

### Output:

>example one two three magic five six

Found



## lseek

Sets a file's input/output position pointer to a specified location.

```
#include <io.h>
unsigned long lseek ( int file_handle, long offset, int base )
```

```
#include <sc.h>
unsigned long _lseek ( FILE * file, long offset, int base )
```

---

<i>base</i>	»	starting point within file
<i>file</i>	»	pointer to open file's fcb
<i>file_handle</i>	»	pointer to file's fcb or number of open file
<i>offset</i>	»	relative position to seek to

**Operation:** These functions are identical in operation to the [fseek](#) function except for the return values.

**lseek** The *file\_handle* is converted to a file pointer and the `_lseek` function is called.

**\_lseek** The file-position pointer for *file* is set according to *offset* and *base*. That is, the position pointer is set to *offset* bytes from *base*. *offset* is a signed value and *base* is coded.

**Returns:** Both functions return the new file position relative to the beginning of the file. A return value of -1 indicates an error occurred. On devices incapable of seeking (terminals, printers, *etc.*), the return value is undefined.

**Errors:** When the return value is -1, [errno](#) may have one of the following values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
EBADF	24	File not open.
EINVAL	428	The value of <i>offset</i> and/or <i>base</i> is invalid.

**Notes:** The *base* argument is coded to be one of three values:

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
SEEK_SET	0	<i>offset</i> is relative to the beginning of the file.
SEEK_CUR	1	<i>offset</i> is relative to the current value of the position pointer.
SEEK_END	2	<i>offset</i> is relative to the current end-of-file.

When *base* is SEEK\_SET, *offset* is treated as an unsigned long, rather than a signed long. This allows *offset* specifications greater than 2GB.

You may position to any location in the file or even beyond the current end-of-file. However, you may not position to a location before the beginning of the file. An attempt to position prior to the start of the file is interpreted as a request to position to the beginning of the file.

Because these functions do not actually read or write any data to *file*, no record or location locking is tested.

**Restrictions:** This function should be used on stream and relative-access files only.

No flush operation is performed on *file* prior to changing the position pointer. If the write buffer for *file* is dirty, you should perform a [fflush](#) operation or use the [fsetpos](#) function.

<b>Conforms to:</b>	<b>lseek</b>	q ANSI	n DOS	n THEOS	n POSIX
	<b>_lseek</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [fseek](#), [fsetpos](#), [ftell](#), [rewind](#), [seek](#), [\\_seek](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

void
main() {
    int    in;
    char    string[256];
    long    pos;

    if (!(in=open("some.data",O_RDONLY)))
        exit(f perror());
    ...
    pos = tell(in);                // get current position
    read(in, string, 30);          // read a record
    lseek(in, pos, SEEK_SET);       // return to prior position
    ...
}
```

## lltoa, ltoa, ltoc3, ltoi2, ltol3

These functions convert long integer values to ASCII strings, three-byte integers or to two integers.

```
#include <stdlib.h>
char * lltoa ( char * string, long long ll )
char * ltoa ( char * string, long l )
char * ltoc3 ( char c[2], long l )
int * ltoi2 ( short i[2], long l )
void ltol3 ( char * string, long * values, int n )
```

---

<i>c</i>	» pointer to 3-byte array
<i>i</i>	» array of two integers
<i>l</i>	» long integer to convert
<i>ll</i>	» long long integer to convert
<i>n</i>	» number of items in <i>values</i> array
<i>string</i>	» pointer to series of 3-byte arrays
<i>values</i>	» pointer to array of long integers to convert

<b>Operation:</b>	<b>lltoa</b>	Converts a long long integer (64-bit) to its ASCII representation. For instance:
		<pre>char    value[20];  printf("Value = %s\n",       lltoa(value, 1234567890123456));</pre>
	<b>ltoa</b>	Converts the long integer <i>l</i> to a character string and saves it in <i>string</i> .
	<b>ltoc3</b>	Converts the long integer <i>l</i> to a three-byte integer value and saves it in <i>c</i> .
	<b>ltoi2</b>	Converts the long integer <i>l</i> to a two-element short integer array <i>i</i> .
<b>Returns:</b>	<b>ltol3</b>	Converts an array of long integers to multiple three-byte integer values. The parameter <i>n</i> specifies the number of long integers in <i>values</i> to be converted. The converted values are stored in consecutive three-byte locations in <i>string</i> .
	<b>lltoa</b>	A pointer to <i>string</i> .
	<b>ltoa</b>	A pointer to <i>string</i> .
	<b>ltoc3</b>	A pointer to <i>c</i> .
<b>Notes:</b>	<b>ltoi2</b>	A pointer to <i>i</i> .
	<b>ltoc3</b>	No checking is performed to ensure that the value of <i>l</i> will fit in three bytes. The range of values that may fit in a three-byte integer is 0–16,777,215.

There are no standard functions that can manipulate or use the three-byte integer value. It must be converted back ([c3tol](#)) to a long integer for actual usage. The sole purpose of the three-byte integer value is more compact external storage of long integer values that can fit in three bytes.

**ltol3** See note for ltoc3, above. To convert the values back into long integers, use the [l3tol](#) function.

<b>Conforms to:</b>	<b>lltoa</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>ltoa</b>	q ANSI	n DOS	n THEOS	q POSIX
	<b>ltoc3</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>ltoi2</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>ltol3</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [atof](#), [atoi](#), [atol](#), [atoll](#), [c3tol](#), [i2tol](#), [ltoa](#), [l3tol](#), [sprintf](#)

---

**\_lubname**

Find the standard device name for a logical unit block (lub) number.

```
#include <stdlib.h>
char * _lubname ( int lub )
```

---

*lub*                    »    logical unit block number for device

**Operation:**        The standard name for the device indicated by *lub* is found.

**Returns:**           A pointer to the standard name for the device.

A NULL string is returned if *lub* is invalid.

**Notes:**            Refer to Appendix D: “Logical Unit Block Numbers” on page 769 for a list of lub numbers and their standard names.

**Conforms to:**        **\_lubname**    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**           Refer to [getbaud](#) example on page 272.

**lwritek**

Writes a record to a direct-access organization file.

```
#include <stdio.h>
unsigned short lwritek ( FILE * file, long _far * key, void * record )
```

---

<i>file</i>	»	pointer to file's fcb
<i>key</i>	»	pointer to record number
<i>record</i>	»	pointer to record storage buffer

**Operation:** Writes the data pointed to by *record* to *file* at record position number *key*.

**Returns:** The file's allocated record length.

A zero return value indicates that the record could not be written.

A return value of -1 indicates that the write was unsuccessful because the record is locked by another user (see below).

**Errors:** When the return value is zero, the record was not written for some reason. Possible reasons include: *file* not opened as direct access file with write access or *key* is zero or negative.

**Notes:** You may write a record to a record number that is beyond the current end-of-file. The file is merely extended to include the new record.

If *file* has been opened with access "r+" and without specifying the access modifier "m," then writing a record releases any other record locks set by this user for this file.

If *file* is in use by another user and that user has the desired record locked, the *lwritek* function will wait for that lock to be released before writing the record.

If the write is not immediately successful because the record is locked by another user, this function may wait to see if the lock is cleared by the other user. The *scr.lock\_wait* item controls the length of time of this wait. This item may have one of three values:

Value	Meaning
0	Wait until region is not locked.
1	Wait for 50 milliseconds (default).
<i>n</i>	Wait for <i>n</i> seconds.

The value in *scr.lock\_wait* can be examined with the [\\_peekscr](#), [\\_peekscrd](#), [\\_peekscrw](#) functions or changed with the [\\_pokescr](#), [\\_pokescrd](#), [\\_pokescrw](#) functions.

When a write attempt fails with a return value of -1 (region already lock by a user), the process number (base 1) of the user locking the region is set in *scr.lock\_pid*. This item can be examined with the [\\_peekscr](#), [\\_peekscrd](#), [\\_peekscrw](#) functions.

See also “Automatic Record-Locking:” on page 224.

**Restrictions:** This function may only be used with a file opened with direct organization, and write or update access.

**Conforms to:** **lwritek** q ANSI q DOS n THEOS q POSIX

**See also:** [ldeletek](#), [lreadk](#), [lreadn](#), [writek](#)

### **make\_alias, \_make\_alias**

Make an alias to a memory segment selector.

```
#include <driver.h>
int make_alias ( int sel )

#include <sc.h>
unsigned short _make_alias ( unsigned sel )
```

---

*sel*                      »    memory segment selector

**Operation:**        An exact copy of *sel* is created as an alias in the GDT (Global Descriptor Table).

**Returns:**            The selector for the alias entry.

**Notes:**             Use the `_make_alias` system call in a program that will only be used with THEOS 32 Version 4 or later.

The `make_alias` function operates correctly with current or prior versions of the operating system. It tests to see if the operating system is Version 4 or later. If so, it executes the `_make_alias` system call, if not it performs the operation itself.

This function is normally used by a device-driver routine to create an alias of a selector in the GDT that is currently defined in the LDT (Local Descriptor Table).

**Restrictions:**      These functions are intended for device-driver authors.

<b>Conforms to:</b>	<b>make_alias</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_make_alias</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:**            [free\\_sel](#), [\\_free\\_sel](#), [make\\_sel](#), [\\_make\\_sel](#)



**makelib, \_makelib, \_mklib**

Create a new library.

```
#include <sc.h>
unsigned short makelib ( const char far * filename, long size )
unsigned short _makelib ( const char far * filename, short size )
unsigned short _mklib ( const char far * filename, long size )
```

---

*filename*               »   pointer to string containing name of library  
*size*                    »   size of new library

**Operation:**       If the string pointed to by *filename* is not the name of an existing file, and if it is a name that is valid for a library, then a new library is created with a size of *size*.

**Returns:**         A zero or non-zero value is returned. A zero return value indicates that the library was created successfully. A non-zero return value indicates the specific reason for the failure to create the library.

**Errors:**         Since these functions are system calls, they cannot use the [errno](#) or [\\_errnum](#) variables to report errors. However, the value returned by these functions conforms to the same error-numbering standard used by those variables.

<i>Name</i>	<i>Value</i>	<i>Meaning</i>
ENODEV	13	Disk not attached.
	21	Disk label specified, disk not mounted.
	25	File name missing.
ENOSPC	42	Disk full.
	43	Directory full.
EEXIST	44	File already exists.
	54	Account name specified, account not found.
	412	Library nesting.
	462	File type missing.

## 422 *makelib*, *\_makelib*, *\_mklib*

---

**Notes:** The *makelib* and *\_mklib* are synonyms of each other.

The difference between *makelib* and *\_makelib* is that *makelib* uses a long integer for the *size* argument and *\_makelib* uses a short integer for the *size* argument. The *makelib* function can be used on THEOS 32 Version 4 and above operating system to create libraries with a size larger than 32,767 members. The *\_makelib* function is limited to libraries with a size of 32,767 or fewer members.

**Restrictions:** Attempting to use the *makelib* function with a value for *size* exceeding 32,767 on an operating system prior to Version 4 will fail.

<b>Conforms to:</b>	<b>makeilib</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_makelib</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_mklib</b>	q ANSI	q DOS	n THEOS	q POSIX

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>

void main(int argc, char *argv[])
{
    int    rc;
    int    size;

    sscanf(argv[2], "%i", &size);           // convert size argument

    if (rc = makelib(argv[1], size))
        syserr(rc, rc, argv[1]);
}
```

---

### Output:

```
>test old.library 200
"OLD.LIBRARY" already exists.

>test old.library.library 20
Cannot create a library as a member of a library.

>test new.library 200

>
```

## make\_sel, \_make\_sel

Create a memory selector.

```
#include <driver.h>
unsigned make_sel( unsigned sel, size_t limit, unsigned long base, unsigned type )
```

```
#include <sc.h>
void _make_sel ( unsigned sel, size_t limit, unsigned long base, unsigned type )
```

---

<i>base</i>	»	selector base address
<i>limit</i>	»	selector size limit
<i>sel</i>	»	memory selector number to use
<i>type</i>	»	access rights to selector

**Operation:** Create a new selector to memory starting at *base* through *limit* with access rights of *type*. If *sel* is non-zero, then that selector number is used; otherwise, a selector number is assigned in the GDT (Global Descriptor Table).

**Returns:** The selector number used.

**Notes:** The *type* value is the 12-bits of information specifying the privilege level, read/write/execute, granularity, *etc.* This is the same information returned by the [\\_getar](#) function and is described in the CPU hardware manual. Some useful names are defined in the GDT.h file.

Use the `_make_sel` system call in a program that will only be used with THEOS 32 Version 4 or later.

The `make_sel` function operates correctly with current or prior versions of the operating system. It tests to see if the operating system is Version 4 or later. If so, it executes the `_make_sel` system call. If not, it performs the operation itself.

This function is normally used by a device-driver routine to create a selector for memory-mapped i/o, for instance, memory-mapped video or dual-ported memory for intelligent devices.

**Restrictions:** These functions are intended for device-driver authors.

<b>Conforms to:</b>	<b>make_sel</b>	q ANSI	q DOS	n THEOS	q POSIX
	<b>_make_sel</b>	q ANSI	q DOS	n THEOS	q POSIX

**See also:** [free\\_sel](#), [\\_free\\_sel](#), [make\\_alias](#), [\\_make\\_alias](#), [uncache](#), [\\_uncache](#)

**malloc**

Allocate or reserve memory.

```
#include <stdlibmalloc.h> or <malloc.h>
void * malloc ( size_t size )
```

---

*size*                      »    size of memory requested

**Operation:**        Allocates a region of memory of at least *size* bytes in length. The block may be larger due to space required for alignment and maintenance.

**Returns:**            A pointer to allocated memory. If the memory cannot be allocated, a NULL pointer is returned.

**Notes:**             Always test the return value from the allocation routines to make sure that memory was actually allocated. If the return is not tested and a NULL pointer is returned, using that NULL pointer will cause an access error.

Use the [free](#) function to release memory allocated with this function or use the [calloc](#) or the [realloc](#) function to reallocate the memory.

**Comment:**          There are many functions that use the malloc function to reserve space for their own requirements. The following is a partial list of these functions using malloc:

<a href="#">_asctime</a>	<a href="#">fopen</a>	<a href="#">_gets</a>	<a href="#">qsort</a>	<a href="#">tempnam</a>
<a href="#">calloc</a>	<a href="#">_fprint</a>	<a href="#">_getw</a>	<a href="#">read_acc</a>	<a href="#">topen</a>
<a href="#">_ctime</a>	<a href="#">fputc</a>	<a href="#">_gmtime</a>	<a href="#">sbrk</a>	<a href="#">tmpfile</a>
<a href="#">diropen</a>	<a href="#">_fputs</a>	<a href="#">_localtime</a>	<a href="#">setvbuf</a>	<a href="#">_tmpnam</a>
<a href="#">execv</a>	<a href="#">_fread</a>	<a href="#">_mktime</a>	<a href="#">spawnv</a>	<a href="#">_ungetc</a>
<a href="#">execvp</a>	<a href="#">_fscanf</a>	<a href="#">_popen</a>	<a href="#">spawnvp</a>	<a href="#">_vfprintf</a>
<a href="#">execl</a>	<a href="#">_fseek</a>	<a href="#">_printf</a>	<a href="#">spawnl</a>	<a href="#">_vprintf</a>
<a href="#">execlp</a>	<a href="#">_fsetpos</a>	<a href="#">_putc</a>	<a href="#">spawnlp</a>	<a href="#">wChoice</a>
<a href="#">fbuf</a>	<a href="#">_fwrite</a>	<a href="#">_putchar</a>	<a href="#">_sterror</a>	<a href="#">wEdit</a>
<a href="#">fgetc</a>	<a href="#">_getchar</a>	<a href="#">_putenv</a>	<a href="#">strdup</a>	<a href="#">wMenuBar</a>
<a href="#">_fgetchar</a>	<a href="#">getcwd</a>	<a href="#">_puts</a>	<a href="#">_system</a>	
<a href="#">_fgets</a>	<a href="#">getenv</a>	<a href="#">_putw</a>	<a href="#">_scanf</a>	

**Conforms to:**                **malloc**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                [\\_alloca](#), [calloc](#), [free](#), [\\_getmem](#), [\\_putmem](#), [max\\_alloc](#), [mem\\_avail](#), [realloc](#), [tot\\_alloc](#)

**Example:**

```
#include <stdlib.h>

void
main()
{
    char * buffer;
    char * bptr;

    if ((buffer=malloc(1024))==NULL) // allocate memory
        exit(256);                // could not allocate

    bptr = buffer;                 // point to start of buffer

    ...

}
```

**matcharg**

Compare one string to another with minimum spelling test.

```
#include <stdlib.h>
int matcharg ( const char * string, const char * lit, int minimum)
```

---

<i>string</i>	»	string to match
<i>minimum</i>	»	minimum abbreviation count
<i>lit</i>	»	text to match against

**Operation:** The *string* is compared to *lit*. In order for the two strings to match, *string* must be at least *minimum* characters in length and *string* must match *lit* for *string*'s length.

**Returns:** A true/false value: A non-zero return indicates that the two strings do match; a zero return value indicates that the strings do not match.

**Notes:** The function is case-insensitive in its operation.

This function is normally used to compare command-line arguments to tokens from the keywords file, although the [testarg](#) function is better suited to that task.

**Conforms to:** **matcharg**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [getkey](#), [string functions](#), [testarg](#)

---

**Example:**

```
#include <stdlib.h>

void
main(int argc, char * argv[])
{
    if (argc<2)
        syserr(1, 133, NULL);                // missing keyword
    if (*argv[1] != '(')
        syserr(1, 134, NULL);                // missing parenthesis
    if (!matcharg(argv[2], "type", 1))
        syserr(1, 27, argv[2]);              // invalid option
    ...
}
```

---

**Output:**

```
>test
Keyword missing or misspelled.
```

```
>test type
Missing parenthesis.
```

```
>test (tipe
Invalid option: "TIPE".
```

```
>test (type
```

## max, min

These two functions compute the larger or smaller value of two values.

```
#include <stdlib.h>
type max ( type a, type b )
type min ( type a, type b )
```

---

*a*                   » first value  
*b*                    » second value  
*type*                » the type of the values (int, long, double, *etc.*)

- Operation:**
- |            |   |
|------------|---|
| <b>max</b> | The larger value of <i>a</i> and <i>b</i> is determined.  |
| <b>min</b> | The smaller value of <i>a</i> and <i>b</i> is determined. |
- Returns:**
- |            |  |
|------------|--|
| <b>max</b> | The larger value of <i>a</i> and <i>b</i> .  |
| <b>min</b> | The smaller value of <i>a</i> and <i>b</i> . |
- Notes:** These functions are implemented as macros and, as such, may have side effects.
- ```
#define max(a,b) (((a)>(b)) ? (a) : (b))
#define min(a,b) (((a)<(b)) ? (a) : (b))
```
- Restrictions:** The arguments *a* and *b* may be of any numeric type, signed or unsigned. However, they must be the same type and the return value will be the same type.
- Conforms to:**
- |            |        |       |         |         |
|------------|--------|-------|---------|---------|
| <b>max</b> | q ANSI | n DOS | n THEOS | q POSIX |
| <b>min</b> | q ANSI | n DOS | n THEOS | q POSIX |
- See also:** [max & min assembly functions](#)

**max\_alloc**

Determines the maximum size of memory available in your program's data segment that can be allocated with the [calloc](#) or [malloc](#) functions.

```
#include <malloc.h>
size_t max_alloc ( void )
```

**Operation:** The heap space is analyzed and the entry for the largest amount of contiguous, available memory is determined.

**Returns:** The size of the largest amount of contiguous memory that is currently available in your program's data segment.

**Notes:** Because the size of a program's data segment can grow dynamically, it is possible to allocate more memory than `max_alloc` indicates. `max_alloc` determines the largest amount of memory that is currently available, without any growth in the size of the data segment.

This function is similar to the DOS function `_memmax`.

**Conforms to:** `max_alloc` q ANSI q DOS n THEOS q POSIX

**See also:** [calloc](#), [malloc](#)

---

**Example:**

```
#include <stdlib.h>
#include <malloc.h>

void
main()
{
    char    * huge;           // pointer to large amount of memory
    size_t huge_size;         // size of large amount of memory

    huge = malloc(huge_size=max_alloc()); // get large memory chunk

    ...
}
```



## max & min assembly functions

These functions provide fast and efficient methods of determining the smallest or largest value from a set of values.

```
#include <builtin.h>
long _maxd ( long arg1, long arg2, ... )
short _maxw ( short arg1, short arg2, ... )
long _mind ( long arg1, long arg2, ... )
short _minw ( short arg1, short arg2 )
```

---

*arg*<sub>1</sub>               »   first value  
*arg*<sub>2</sub>               »   second value

|                      |                                                                                                                                                         |                                                                                                               |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <b>Operation:</b>    | <b>_maxd</b>                                                                                                                                            | The series of arguments are compared as 32-bit long integers. The largest one is determined.                  |
|                      | <b>_maxw</b>                                                                                                                                            | The series of arguments are compared as 16-bit short integers. The largest one is determined.                 |
|                      | <b>_mind</b>                                                                                                                                            | The series of arguments are compared as 32-bit long integers. The smallest one is determined.                 |
|                      | <b>_minw</b>                                                                                                                                            | The series of arguments are compared as 16-bit short integers. The smallest one is determined.                |
| <b>Returns:</b>      | <b>_maxd</b>                                                                                                                                            | The largest value of <i>arg</i> <sub>1</sub> , <i>arg</i> <sub>2</sub> , ... is returned as a long integer.   |
|                      | <b>_maxw</b>                                                                                                                                            | The largest value of <i>arg</i> <sub>1</sub> , <i>arg</i> <sub>2</sub> , ... is returned as a short integer.  |
|                      | <b>_mind</b>                                                                                                                                            | The smallest value of <i>arg</i> <sub>1</sub> , <i>arg</i> <sub>2</sub> , ... is returned as a long integer.  |
|                      | <b>_minw</b>                                                                                                                                            | The smallest value of <i>arg</i> <sub>1</sub> , <i>arg</i> <sub>2</sub> , ... is returned as a short integer. |
| <b>Errors:</b>       | No errors are detected.                                                                                                                                 |                                                                                                               |
| <b>Notes:</b>        | These “built-in” functions generate in-line code, thus avoiding the expensive usage of the <code>call</code> and <code>ret</code> instructions.         |                                                                                                               |
|                      | You should provide at least two arguments for each of these functions.                                                                                  |                                                                                                               |
| <b>Restrictions:</b> | These functions are intended for device-driver authors. Because these are “built-in” functions, there is no type-checking or syntax analysis performed. |                                                                                                               |
| <b>Conforms to:</b>  | <b>_maxd</b>                                                                                                                                            | q ANSI    q DOS    n THEOS   q POSIX                                                                          |
|                      | <b>_maxw</b>                                                                                                                                            | q ANSI    q DOS    n THEOS   q POSIX                                                                          |
|                      | <b>_mind</b>                                                                                                                                            | q ANSI    q DOS    n THEOS   q POSIX                                                                          |
|                      | <b>_minw</b>                                                                                                                                            | q ANSI    q DOS    n THEOS   q POSIX                                                                          |
| <b>See also:</b>     | <a href="#">max</a> , <a href="#">min</a>                                                                                                               |                                                                                                               |

**mblen, mbstowcs, mbtowc**

These functions manipulate “multi-byte” strings.

```
#include <stdlib.h>

int mblen ( const char * mbstring, size_t count )
size_t mbstowcs ( wchar_t * wcs, const char * mbstring, size_t count )
int mbtowc ( wchar_t * wc, const char * mbstring, size_t count )
```

---

|                 |   |                                           |
|-----------------|---|-------------------------------------------|
| <i>count</i>    | » | number of bytes in a multi-byte character |
| <i>mbstring</i> | » | pointer to multi-byte character           |
| <i>wc</i>       | » | pointer to wide character                 |
| <i>wcs</i>      | » | pointer to wide character array           |

**Operation:**      **mblen**      When *mbstring* is a NULL pointer, this function initializes the multi-byte functions to the current LC\_CTYPE locale category. Subsequent calls with a NULL pointer argument merely test to see if multi-byte translations are supported.

When *mbstring* is not a NULL pointer, this function determines the number of bytes that comprise the multi-byte character pointer to by *mbstring*. At most, *count* bytes of *mbstring* are examined.

**mbstowcs**      Converts the sequence of multi-byte characters pointed to by *mbstring* into an array of wide characters and stores these wide characters in the location pointed to by *wcs*. At most, *count* multi-byte characters are translated. Translation may stop earlier if a null character is encountered in the multi-byte string.

**mbtowc**      When *mbstring* is a NULL pointer, this function merely uses the *mblen* function to initialize the multi-byte functions to the current LC\_CTYPE locale category.

Using *mblen*, the number of bytes comprising the multi-byte character pointed to by *mbstring* is determined and then the multi-byte character is converted to a wide character and stored in the location pointed to by *wc*. At most, *count* bytes of *mbstring* are examined.

**Returns:**      **mblen**      A true/false value when *mbstring* is a NULL pointer. A non-zero return indicates that multi-byte encodings are supported in the current LC\_CTYPE locale; a zero return indicates that they are not.

When *mbstring* is not a NULL pointer, the length of the multi-byte character is returned.

**mbstowcs**      The number of bytes used in *wcs*.

**mbtowc**      A true/false value when *mbstring* is a NULL pointer. A non-zero return indicates that multi-byte encodings are supported in the current LC\_CTYPE locale; a zero return indicates that they are not.

When *mbstring* is not a NULL pointer, the length of the multi-byte character is returned.

**Errors:** These functions return a -1 if an invalid multi-byte character is encountered.

**Notes:** The only LC\_TYPE locale supported is the default C type, which does not support multi-byte characters. These functions are provided for ANSI conformance.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>mblen</b>    | n ANSI | n DOS | n THEOS | q POSIX |
|                     | <b>mbstowcs</b> | n ANSI | n DOS | n THEOS | q POSIX |
|                     | <b>mbtowc</b>   | n ANSI | n DOS | n THEOS | q POSIX |

**See also:** [wcstombs](#), [wcstrlen](#), [wctomb](#)

**mem\_avail**

Determine the amount of system memory available at this time.

```
#include <getmem> or <malloc.h>
size_t mem_avail ( void )
```

**Operation:** The total amount of unused system memory is computed.

**Returns:** The total amount of unused system memory.

**Notes:** The amount of unused system memory returned by this function is available for usage by the [\\_getmem](#), [\\_incrmem](#), [\\_mem\\_grow](#) and [shared](#) functions and for dynamic extension of program data segments. However, this memory is only available at the time the function is executed. Because many programs and tasks use this available system memory its size grows and shrinks constantly.

**Conforms to:** **mem\_avail** q ANSI q DOS n THEOS q POSIX

**See also:** [\\_getmem](#), [\\_incrmem](#), [\\_mem\\_grow](#), [\\_putmem](#), [shared](#)

---

**Example:**

```
#include <stdlib.h>
#include <getmem.h>

void
main()
{
    short    extra_data;

    if (mem_avail()>0x80000)    // 500K available?
        extra_data = _getmem(0x80000, MEM_LDT);
    else
        syserr(3, 3, NULL);    // insufficient memory

    ...
    _putmem(extra_data);
}
```

## memory functions

This group of functions manipulate memory areas such as buffers. They copy one area to another, initialize an area to a specific value, find the location of a value in an area or compare two memory areas.

```
#include <string.h> or <memory.h>

void * memccpy ( char * buffer1, char * buffer2, char c, size_t len )
void * memchr ( char * buffer, char c, size_t len )
int memcmp ( char * buffer1, char * buffer2, size_t len )
void * memcpy ( char * buffer1, char * buffer2, size_t len )
void memdcmp ( char * buffer1, char * buffer2, size_t len )
int memeq ( char * buffer1, char * buffer2, size_t len )
int memicmp ( char * buffer1, char * buffer2, size_t len )
int memieq ( char * buffer1, char * buffer2, size_t len )
void * memmove ( char * buffer1, char * buffer2, size_t len )
void * memrcpy ( char * buffer1, char * buffer2, size_t len )
void * memset ( void * buffer, char c, size_t len )
```

```
#include <builtin.h>

void _memcpy ( char * buffer1, char * buffer2, size_t len )
void _memset ( void * buffer, char c, size_t len )
```

---

|                           |   |                               |
|---------------------------|---|-------------------------------|
| <i>buffer</i>             | » | pointer to memory area        |
| <i>buffer<sub>1</sub></i> | » | pointer to first memory area  |
| <i>buffer<sub>2</sub></i> | » | pointer to second memory area |
| <i>c</i>                  | » | single-byte value             |
| <i>len</i>                | » | number of bytes to manipulate |

|                   |                |                                                                                                                                                                                                                 |
|-------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Operation:</b> | <b>memccpy</b> | Copies bytes from <i>buffer<sub>2</sub></i> to <i>buffer<sub>1</sub></i> . Bytes are copied until either <i>len</i> number of bytes are copied or until a byte with the value <i>c</i> is copied.               |
|                   | <b>memchr</b>  | Locates the byte with value <i>c</i> in <i>buffer</i> . Only the first <i>len</i> bytes of <i>buffer</i> are examined.                                                                                          |
|                   | <b>memcmp</b>  | Compares the first <i>len</i> bytes of <i>buffer<sub>1</sub></i> and <i>buffer<sub>2</sub></i> .                                                                                                                |
|                   | <b>memcpy</b>  | Copy <i>len</i> bytes from <i>buffer<sub>2</sub></i> to <i>buffer<sub>1</sub></i> .                                                                                                                             |
|                   | <b>_memcpy</b> | Similar to <i>memcpy</i> except that <i>_memcpy</i> is a built-in function and thus generates in-line code without using <i>call</i> and <i>ret</i> instructions. This function also does not return any value. |
|                   | <b>memdcmp</b> | This function is identical in operation to the <i>memcmp</i> function unless there are ASCII decimal digits in the buffers.                                                                                     |

When the two buffers contain ASCII decimal digits that start in the same relative location, the two buffers will compare as if the portions containing the decimal digits were the same length, with zeros added to pad the shorter string of digits to

the same length as the other buffer's string of digits. For example:

| Original set<br>of strings: | Sorted with<br>memcmp: | Sorted with<br>memdcmp: |
|-----------------------------|------------------------|-------------------------|
| a1xxx                       | a101xxx                | a1xxx                   |
| a83xxx                      | a11xxx                 | a11xxx                  |
| a12xxx                      | a12xxx                 | a12xxx                  |
| a11xxx                      | a1xxx                  | a83xxx                  |
| b1xxx                       | a83xxx                 | a101xxx                 |
| a101xxx                     | b100xxx                | b1xxx                   |
| b15xxx                      | b15xxx                 | b15xxx                  |
| b100xxx                     | b1xxx                  | b100xxx                 |

**memeq** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>* for equality.

**memicmp** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>*. This is a “case-insensitive” comparison. That is, if a position in one buffer contains a lowercase letter and the other buffer contains the uppercase form of the same letter, the comparison is determined to be equal.

**memieq** Compares the first *len* bytes of *buffer<sub>1</sub>* and *buffer<sub>2</sub>* for equality. This is a “case-insensitive” comparison.

**memmove** Copies the contents of *buffer<sub>2</sub>* to *buffer<sub>1</sub>* for *len* bytes. This function ensures that the copy will be accurate even if the two buffers overlap.

**memrcpy** Copies *len* bytes from *buffer<sub>2</sub>* to *buffer<sub>1</sub>*, in the reverse direction. That is, the last byte of *buffer<sub>2</sub>* is copied to the last position of *buffer<sub>1</sub>*, the next to the last byte of *buffer<sub>2</sub>* is copied to the next to the last position of *buffer<sub>1</sub>*, etc.

**memset** Copies the value *c* into the first *len* bytes of *buffer*.

**\_memset** Similar to **memset** except that **\_memset** is a built-in function and thus generates in-line code without using **call** and **ret** instructions. This function also does not return any value.

**Returns:** **memccpy** If a byte of value *c* is copied, a pointer to the location following *c* in *buffer<sub>1</sub>* is returned. Otherwise a NULL is returned.

**memchr** A pointer to the location of the first occurrence of *c* in *buffer*. If *c* cannot be located in the first *len* bytes of *buffer*, a NULL is returned.

**memcmp** A signed, integer value.

| Comparison            | Return value |
|-----------------------|--------------|
| $buffer_1 < buffer_2$ | < 0          |
| $buffer_1 = buffer_2$ | 0            |
| $buffer_1 > buffer_2$ | > 0          |

**memcpy** A pointer to *buffer<sub>1</sub>*.

|                      |                |                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      | <b>_memcpy</b> | No value is returned.                                                                                                                                                                                                                                                                                                                                                 |
|                      | <b>memdcmp</b> | A signed, integer value, similar to memcmp.                                                                                                                                                                                                                                                                                                                           |
|                      | <b>memeq</b>   | A non-zero value when the two buffers are equal for their first <i>len</i> bytes. A zero return value indicates inequality.                                                                                                                                                                                                                                           |
|                      | <b>memicmp</b> | A signed, integer value, similar to the memcmp function.<br><br>A position is equal when the two buffers are equal, or if they have the same letter independent of its case. When a position is unequal, the lowercase form of each position is compared. Thus, “a” is less than “b” and “B,” even though the value of “a” is actually greater than the value of “B.” |
|                      | <b>memieq</b>  | A non-zero value when the two buffers are equal for their first <i>len</i> bytes. A zero return value indicates inequality. The case-mode of the characters is ignored in this comparison.                                                                                                                                                                            |
|                      | <b>memmove</b> | A pointer to <i>buffer<sub>1</sub></i> .                                                                                                                                                                                                                                                                                                                              |
|                      | <b>memrcpy</b> | A pointer to <i>buffer<sub>1</sub></i> .                                                                                                                                                                                                                                                                                                                              |
|                      | <b>memset</b>  | A pointer to <i>buffer</i> .                                                                                                                                                                                                                                                                                                                                          |
|                      | <b>_memset</b> | No value is returned.                                                                                                                                                                                                                                                                                                                                                 |
| <b>Notes:</b>        | <b>_memcpy</b> | This “built-in” function generates in-line code, thus avoiding the expensive usage of the call and ret instructions.                                                                                                                                                                                                                                                  |
|                      | <b>memicmp</b> | This function uses the <a href="#">tolower</a> function when determining if a position in the two buffers is the same letter with different cases.                                                                                                                                                                                                                    |
|                      | <b>memieq</b>  | The <a href="#">tolower</a> function is used to determine if characters are equal except for their case.                                                                                                                                                                                                                                                              |
|                      | <b>memmove</b> | The function memcpy or memrcpy is used internally to ensure that the <i>buffer<sub>2</sub></i> is moved to <i>buffer<sub>1</sub></i> , even when the two buffers overlap.                                                                                                                                                                                             |
|                      | <b>_memset</b> | This “built-in” function generates in-line code, thus avoiding the expensive usage of the call and ret instructions.                                                                                                                                                                                                                                                  |
| <b>Restrictions:</b> | <b>memccpy</b> | If <i>buffer<sub>2</sub></i> and <i>buffer<sub>1</sub></i> overlap, the resulting buffer may be erroneous. When <i>buffer<sub>2</sub></i> is greater than <i>buffer<sub>1</sub></i> , the copy is performed correctly; when <i>buffer<sub>1</sub></i> is greater than <i>buffer<sub>2</sub></i> , the                                                                 |

process of copying the bytes in *buffer<sub>2</sub>* may destroy subsequent bytes in *buffer<sub>2</sub>*. For example:

          abcdefghijklmnopqrstuvwxyz  
          ↑          ↑  
*buffer<sub>1</sub>* *buffer<sub>2</sub>*           len = 10, c = 'm'

Result:  ghijklmnopklmnopqrstuvwxyz

          abcdefghijklmnopqrstuvwxyz  
          ↑          ↑  
*buffer<sub>2</sub>* *buffer<sub>1</sub>*           len = 10, c = 'm'

Result:  abcdefabcdefabcdqrstuvwxyz

- memcpy**   Same as memccpy.
- \_memcpy**   This function is intended for device-driver authors. Because it is a “built-in” function, there is no type-checking or syntax analysis performed.
- memrcpy**   Because the copy operation is performed in the reverse direction, the restrictions of a memrcpy are the opposite of a memcpy. Also, *buffer<sub>1</sub>* and *buffer<sub>2</sub>* point to the last byte rather than the first byte of the memory array.
- If *buffer<sub>2</sub>* and *buffer<sub>1</sub>* overlap, the resulting buffer may be erroneous. When *buffer<sub>1</sub>* is greater than *buffer<sub>2</sub>*, the copy is performed correctly; when *buffer<sub>2</sub>* is greater than *buffer<sub>1</sub>*, the process of copying the bytes in *buffer<sub>1</sub>* may destroy subsequent bytes in *buffer<sub>1</sub>*.
- \_memset**   This function is intended for device-driver authors. Because it is a “built-in” function, there is no type-checking or syntax analysis performed.

|              |                |        |       |         |         |
|--------------|----------------|--------|-------|---------|---------|
| Conforms to: | <b>memccpy</b> | q ANSI | n DOS | n THEOS | q POSIX |
|              | <b>memchr</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|              | <b>memcmp</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|              | <b>memcpy</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|              | <b>_memcpy</b> | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>memdcmp</b> | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>memeq</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>memicmp</b> | q ANSI | n DOS | n THEOS | n POSIX |
|              | <b>memieq</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>memmove</b> | n ANSI | n DOS | n THEOS | q POSIX |
|              | <b>memrcmp</b> | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>memset</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|              | <b>_memset</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**   [far memory functions](#), [far string functions](#), [string functions](#)



## ***\_mem\_grow***

Increase the size of a memory segment.

```
#include <sc.h>
unsigned short _mem_grow ( int sel, size_t adjust )
```

---

|               |   |                                    |
|---------------|---|------------------------------------|
| <i>adjust</i> | » | additional size in bytes requested |
| <i>sel</i>    | » | memory segment selector            |

**Operation:** The memory segment specified by *sel* is increased in size by *adjust* bytes.

**Returns:** A success/fail indicator. A non-zero return indicates that the segment was increased in size successfully; a zero indicates an error and that the segment size was not changed.

**Notes:** The return value of this function uses inverse logic compared to most other functions.

**Restrictions:** You cannot use this function to increase the size of the data or stack segments. Instead, use the [\\_incrmem](#) function.

**Conforms to:** ***\_mem\_grow***    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [\\_incrmem](#)

**Example:**

```
#include <stdlib.h>
#include <string.h>
#include <sc.h>

char      shared_name[] = "COMMON_AREA";
unsigned  shared_seg;      // share memory segment
struct {
    char use_count;
    char data[256];
} far *shared_mem;

void
main()
{
    int      wakeup;
    char * mystring = "some of my data";
    size_t cur_size;

    shared_seg = shared(shared_name, 0x1000); // find common memory
    shared_mem = _MK_FP(shared_seg, 0);      // create pointer to memory

    _lockset(0, shared_seg);                // exclusive use

    ++shared_mem->use_count;                 // count me as user of mem

    cur_size = _getlimit(shared_seg); // get its current size
    if (cur_size < 0x1000)              // big enough?
        _mem_grow(shared_seg, 0x1000-cur_size);

    _fmemcpy(4, shared_mem->data, mystring, getds());

    _lockres(0, shared_seg);                // release lock

    ...

    _lockset(0, shared_seg);                // lock it
    --shared_mem->use_count;                 // uncount me as user
    _lockres(0, shared_seg);                // release lock
}
```

## menu, menu2

Displays a selection menu on the screen and waits for operator to select one item.

```
#include <stdlib.h>
int menu ( char * items[], int count )
int menu2 ( char * items[], int count, int start, int nodisp )
```

---











|               |   |                                              |
|---------------|---|----------------------------------------------|
| <i>count</i>  | » | number of items in array items               |
| <i>items</i>  | » | pointer to array of pointers to text strings |
| <i>nodisp</i> | » | true/false flag suppressing redisplay        |
| <i>start</i>  | » | initial index of item to position to         |

- Operation:**
- |              |                                                                                                                                                                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>menu</b>  | Displays the first <i>count</i> items from the list of menu items pointed to by <i>items</i> . The first item is then highlighted in reverse video and the function waits for the operator to select one of the items before exiting the function.                                        |
| <b>menu2</b> | Identical in operation to the menu function if <i>start</i> is 1 and <i>nodisp</i> is 0. When <i>nodisp</i> is non-zero, the menu is not displayed. This argument would be used when a menu that was previously displayed with the menu or menu2 function is still visible on the screen. |
- The *start* argument specifies which menu item should be highlighted initially.
- Returns:** The item number (base 1) selected by the operator. A return value of -1 indicates that the operator used the **Esc** or **Quit** key to exit the menu without selecting any item.
- Errors:** If the row or column number specified is invalid (less than zero or greater than the number of columns or rows available on the console or active window), the item is displayed using a value of 1 for the invalid row or column number.
- Every item must have a column and row specified and some item-text defined, or the result is unpredictable.

**Notes:** Each menu item must start with the column and row number (base 0) for display of that item. These two values are separated with a single, nonnumeric character and they are separated from the item text by a nonnumeric character. (The first non-numeric character following the row number value is always treated as a separator and is not part of the item text.) For instance:

```
1,1 This is item one
1,2 This is item two
40,1 This is item three (2nd column, item 1)
40,2 This is item four (2nd column, item 2)
```

There are several keys that may be used to move the highlight bar and to select an item:

| Key                                                                                                                                                                             | Meaning                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
|                                                                                                | Select item that is currently highlighted.                                                               |
|                                                                                                | Position down to the next item in <i>items</i> .                                                         |
|                                                                                                | Position to previous item in <i>items</i> .                                                              |
|                                                                                                | Position left to previous item displayed on same line.                                                   |
|                                                                                                | Position right to next item displayed on same line.                                                      |
|                                                                                                | Position to first item in list.                                                                          |
|                                                                                                | Position to last item in list.                                                                           |
| <br>or<br> | Exit function with a return value of -1.                                                                 |
|                                                                                              | Advance to next item in <i>items</i> . If currently positioned to last item, move to first item in list. |

In addition to the above keys, you may also enter a character. If that character matches the first character displayed for any of the items, that item is highlighted and selected. When there is more than one item that starts with a particular character, only the first item may be selected in this manner.

**Conforms to:**

|              |        |       |         |         |
|--------------|--------|-------|---------|---------|
| <b>menu</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>menu2</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [wChoice](#), [wChoiceFuncKeys](#), [wChoiceSelect](#), [wChoiceTimeout](#), [wMenuBar](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <crt.h>
#include <wmapi.h>
#include <colors.h>

void main()
{
    char    * menuitems[] = {
        "01,01 1 Payroll System",
        "01,02 2 Accounts Receivable System",
        "01,03 3 Accounts Payable System",
        "01,04 4 System Manager Functions",
        "01,05 5 Print Reports",
        "01,07 Exit"
    };

    int     menu_ret;
    int     menu_win = 1,

    wOpen(menu_win, 5, 5, 31, 9);          // open menu window
    wColor(menu_win, WHITE, BLUE, WHITE, RED);
    wFrame(menu_win, WFRAME_DOUBLE, WSHADOW_RIGHT, 0);

    crt(CLEAR);                            // clear background

    wSelect(menu_win, wUPDATE_ON);
    menu_ret = menu(menuitems, 7);

    while (menu_ret>0)
    {
        dispatch(menu_ret);                // process selection
        wSelect(menu_win, wUPDATE_ON);
        menu_ret = menu2(menuitems, 7, menu_ret-1, 1);
    }
    wFinish();
}
```

**mkdir**

Creates a new directory.

```
#include <stdio.h> or <direct.h>
```

```
int mkdir ( char * dirname )
```

---

*dirname*                    »   pointer to directory name

**Operation:**            A new, empty directory is created with a name of *dirname*.

**Returns:**              A zero when the new directory is successfully created. A non-zero return value indicates an error and [errno](#) is set.

**Errors:**                When the return is NULL, [errno](#) is set to one of the following values:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                         |
|-------------|--------------|----------------------------------------|
| EACCES      | 18           | The <i>dirname</i> already exists.     |
| ENOENT      | 19           | Path name of <i>dirname</i> not found. |

**Notes:**                *dirname* may be a simple name, in which case *dirname* is appended to the current working name to form the complete path specification.

**Conforms to:**                **mkdir**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**                [chdir](#), [creat](#), [fcreate](#), [fopen](#), [getcwd](#), [open](#), [rmdir](#)

---

**Example:**

```
#include <stdio.h>
```

```
main() {  
    if (mkdir("example"))  
        perror("Failure creating directory.");  
    if (mkdir("../newdir"))  
        perror("Failure creating directory.");  
    if (mkdir("/private/source/personal.files"))  
        perror("Failure creating directory.");  
}
```

## ***\_FP\_OFF, \_FP\_SEG, \_MK\_FP operators***

Convert a segment and offset into a far pointer or convert a far pointer into segment and offset.

```
void * _FP_OFF ( void _far * farptr )
unsigned _FP_SEG ( void _far * farptr )
void _far * _MK_FP ( unsigned seg, void * offset )
```

```
#include <types.h>
void * FP_OFF ( void _far * farptr )
unsigned FP_SEG ( void _far * farptr )
void _far * MK_FP ( unsigned seg, void * offset )
```

---

|               |   |                                              |
|---------------|---|----------------------------------------------|
| <i>farptr</i> | » | far memory pointer                           |
| <i>offset</i> | » | pointer to object relative to memory segment |
| <i>seg</i>    | » | memory segment selector                      |

**Operation:** These “functions” manipulate references to remote objects by either extracting out the two components of a far pointer or joining the two components together to form a far pointer.

***\_FP\_OFF*** Extract the relative offset of the object pointed to by *farptr*.

***\_FP\_SEG*** Extract the memory segment selector of the object pointed to by *farptr*.

***\_MK\_FP*** Combine the *offset* and *seg* into a far pointer.

**Returns:** None of these “functions” actually returns anything. However:

***\_FP\_OFF*** Returns the offset to the object pointed to by *farptr*.

***\_FP\_SEG*** Returns the memory segment of the object pointed to by *farptr*.

***\_MK\_FP*** Returns the far pointer specified by *offset* and *seg*.

**Notes:** These keywords are defined in this function library reference, not because they are functions, but because they look like functions. They are intrinsic operators built into the C language compiler.

|                     |                       |        |       |         |         |
|---------------------|-----------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b><i>FP_OFF</i></b>  | q ANSI | n DOS | n THEOS | q POSIX |
|                     | <b><i>_FP_OFF</i></b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b><i>FP_SEG</i></b>  | q ANSI | n DOS | n THEOS | q POSIX |
|                     | <b><i>_FP_SEG</i></b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b><i>_MK_FP</i></b>  | q ANSI | q DOS | n THEOS | q POSIX |

---

**Example:** See example for [\*\\_mem\\_grow\*](#) on page 437.

## mktemp

Create a unique file name.

```
#include <stdio.h> or <io.h>
char * mktemp ( char * mask )
```

---

*mask*                   »   pointer to pattern for new file name

**Operation:**       A new, unique file name is created by modifying the contents of *mask*.  
*mask* is a template in the form:

*base*XXXXXX

where *base* is the part of the new file name supplied by you and XXXXXX is a placeholder for the part generated by *mktemp*.

Six consecutive X characters, either uppercase or lowercase, found at the end of the *mask* field are replaced with your three-digit process number, followed by a three-digit sequence number, base 36 (digits and letters). The first time that *mktemp* is used in a program, the sequence number is 0. Subsequent calls to *mktemp* increment the sequence number by one.

When six consecutive X characters cannot be found in *mask*, a search is made for ##@ or ###@. If either of these are found in *mask*, they are replaced with your two- or three-digit process number, followed by a single letter. The first time that *mktemp* is used with a *mask* of this type, the letter A is used. Subsequent calls to *mktemp* use letters B, C, *etc.*

**Returns:**       A pointer to the new name. When the *mask* is invalid, a NULL pointer is returned. A NULL pointer is also returned if no more unique names can be generated with the given *mask*.

**Notes:**        This function does not create the new file, only the new name that may be used by the program to create a file.

**Restrictions:** The *mask* field is used by *mktemp* as buffer for the resulting file name. Thus, if *mktemp* is called twice with the same *mask* field, the second call will be erroneous because the *mask* will no longer contain a valid template. For instance, do not use this type of code:

```
strcpy(fname1,mktemp("abc.xxxxxx"));
strcpy(fname2,mktemp("abc.xxxxxx"));
```

The contents of *fname2* will be an empty string because the literal *mask* field was modified by the first call to *mktemp*. This literal field now contains "abc.001000" or something similar. There are no template characters remaining in the field.

Instead, it should be coded as:

```
strcpy(fname1,"abc.xxxxxx");
strcpy(fname2,"abc.xxxxxx");
mktemp(fname1);
mktemp(fname2);
```



A maximum of 46,656 names can be generated when using the XXXXXX pattern. A maximum of 26 names can be generated using the ##@ or ###@ pattern.

**Conforms to:**            **mktemp**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**            [tempnam](#), [tmpfile](#), [tmpnam](#), [\\_tmpnam\\_drive](#)

**mktime**

Converts the date and time from a time structure into a calendar time value.

```
#include <time.h>
time_t mktime ( struct tm * timeptr )
```

---

*timeptr*                   »   pointer to time structure

**Operation:**       mktime converts the local date and time specified in the time structure pointed to by *timeptr* into UTC calendar time value. This is the reverse operation of the [gmtime](#) function call.

See notes for additional operations performed by this function.

**Returns:**       The UTC calendar time value represented by the time structure.

**Notes:**       A significant side effect of this function is that it normalizes the date and time in the *timeptr* structure. The values in the member fields `tm_sec`, `tm_min`, `tm_hour`, `tm_mday` and `tm_mon` are not restricted to the ranges described in the declaration for `struct tm`. When the values in these member fields exceed their proper ranges, the member field and higher level fields are adjusted so that the result is in the proper range.

For instance, if the value in `tm_sec` is 70, then the value of `tm_min` is incremented by one and the value `tm_sec` is set to 10. As each field is analyzed, the next higher level field is adjusted as necessary. The sequence for this field analysis and normalization is:

```
tm_sec
tm_min
tm_hour
tm_mday
tm_mon
tm_year
```

After the time and date fields are normalized, the `tm_wday` and `tm_yday` are computed.

If `tm_isdst` is negative, it is also computed and set. If the value in `tm_isdst` is zero, it is treated as daylight savings time is not in effect locally. Positive values are treated as daylight savings time is in effect locally.

The [tzset](#) function is also called by this function.

**Restrictions:**   Since calendar time values are relative to 1970, any negative `tm_year` value in the time structure causes an illegal time value to be returned.

**Conforms to:**       **mktime**    n ANSI    n DOS    n THEOS    q POSIX

**See also:**       [asctime](#), [clock](#), [ctime](#), [difftime](#), [gmtime](#), [localtime](#), [strftime](#), [time](#), [tzset](#)

**Example:**

```
#include <stdio.h>
#include <time.h>

main(void)
{
    struct tm curtime;
    time_t timer;
    char buffer[80];

    time(&timer);                // get current time
    curtime = * localtime(&timer); // convert to struct
    strftime(buffer, 80, "%A, %B %d, %Y, at %H:%M %p %Z",&curtime);
    printf("It is now %s\n",buffer);

    // adjust date for new deadline

    curtime.tm_mday += 3;        // add 3 days
    curtime.tm_hour = 8;        // set to 8 am
    curtime.tm_sec = curtime.tm_min = 0; // no minutes, seconds
    mktime(&curtime);            // normalize
    if (curtime.tm_wday==0)      // Sunday?
        curtime.tm_mday += 1;    // change to Monday
    if (curtime.tm_wday==6)      // Saturday?
        curtime.tm_mday += 2;    // change to Monday
    mktime(&curtime);            // normalize again
    strftime(buffer, 80, "%A, %B %d, %Y, at %H:%M %p %Z", &curtime);
    printf("\nYour deadline has been moved to:\n%s\n", buffer);
}
```

---

**Output:**

It is now Thursday, July 04, 1996, at 12:58 pm PDT

Your deadline has been moved to:  
Monday, July 08, 1996, at 08:00 am PDT

**modf**

Get the integer portion and the fractional portion of a floating-point value.

```
#include <math.h>
double modf ( double x, double * iptr )
```

---

*iptr*                   »   pointer to integer portion storage  
*x*                       »   value to evaluate

**Operation:**       The value *x* is examined and the integer or whole number portion is extracted and copied to the location pointed to by *iptr*. The fractional portion is returned as the value of this function.

Both the integer and fractional portions of *x* will have the same sign as *x*.

**Returns:**           The fractional portion of the floating-point value *x*.

**Notes:**            This function uses BCD or IEEE arithmetic, depending upon the `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                **modf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [ip](#)

## **\_mount\_fs**

Mounts a disk file system.

```
#include <sc.h> or <diskfind.h>
void _mount_fs ( DISK_UCB * ucb )
```

---

*ucb*                    »    pointer to disk unit control block

**Operation:**        The *ucb* is marked as never accessed.

**Notes:**            No actual “mounting” is performed nor is the disk volume-label read. The information for the drive is set to indicate that the status of the drive is unknown so that the next access to the drive will read the volume label.

                     All disk cache information about the drive is discarded.

                     This function is used by the MOUNT command after a [\\_dir\\_mount](#) operation is performed.

**Conforms to:**        [\\_mount\\_fs](#)    q ANSI        q DOS        n THEOS        q POSIX

**See also:**            [\\_dir\\_mount](#)

---

### **Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <sc.h>

void
main()
{
    DISK_UCB *drive_A;

    drive_A = getucb(0);                    // get drive A ucb
    _dir_mount(drive_A);

    printf("\nPlace the diskette in drive A.");
    printf("\nPress RETURN when ready...");
    getchar();

    _mount_fs(drive_A);
}
```

### move memory assembly functions

This group of “functions” generates in-line code to move a series of bytes, words or double-words from one memory location to another.

```
#include <builtin.h>

void _movsb ( void _far * to_offset, void _far * from_offset, long count )
void _movsd ( void _far * to_offset, void _far * from_offset, long count )
void _movsw ( void _far * to_offset, void _far * from_offset, long count )
```

---

|                    |   |                                        |
|--------------------|---|----------------------------------------|
| <i>count</i>       | » | number of objects to move              |
| <i>from_offset</i> | » | pointer to remote source location      |
| <i>to_offset</i>   | » | pointer to remote destination location |

**Operation:**      **\_movsb**      Copies *count* number of bytes starting at location *from\_offset* to the memory location *to\_offset*.

**\_movsd**      Copies *count* number of double-words (long integers) starting at location *from\_offset* to the memory location *to\_offset*.

**\_movsw**      Copies *count* number of words (short integers) starting at location *from\_offset* to the memory location *to\_offset*.

**Returns:**              No value is returned by these functions.

**Notes:**              These “built-in” functions generate in-line code, thus avoiding the expensive usage of the call and ret instructions.

The arguments *from\_offset* and *to\_offset* are void pointers. This means that the functions can move whatever type of object desired to any location. For instance, a **\_movsw** function can move a word value (double-byte value) to a location that is declared as a character string.

**Restrictions:**      These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program’s code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_movsb</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_movsd</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_movsw</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**              [far memory functions](#), [memory functions](#)

## msalarm

Programs a timer to generate a signal interrupt after specified number of milliseconds have elapsed.

```
#include <timer.h> or <signal.h>
long msalarm ( long mseconds )
```

---

*mseconds*            »    number of milliseconds to elapse

- Operation:** A timer is programmed to raise SIGALRM after *mseconds* amount of time has elapsed.
- Returns:** If there was an [alarm](#) or msalarm timer already programmed, the time remaining for that timer is returned. Otherwise, zero is returned.
- Notes:** The default action for SIGALRM is SIG\_IGN. If you do not use the [signal](#) function to define your own trap for this event, your program will not detect when the *mseconds* have elapsed.
- Both [alarm](#) and msalarm use the same SIGALRM mechanism and they both share the same timer. The difference between the two functions is in the unit of measure for the time period specified.
- An alarm timer can be cleared by using a *mseconds* value of zero. When this is done, any previous alarm timer is reset.
- Defaults:** The default action for SIGALRM is SIG\_IGN.
- Restrictions:** Only one alarm timer is allowed at any one time. Multiple calls to [alarm](#) or msalarm reset the prior timer to the new value.
- Conforms to:**            **msalarm**    q ANSI    q DOS    n THEOS    q POSIX
- See also:**            [alarm](#), [signal](#), [timer](#)

```
#include <stdlib.h>
void msgclose ( void )
```

**See also:** [exit](#), [fclose](#), [fcloseall](#), [getmsg](#)

```
void main() {  
    char        tofmsg[80],  
               eofmsg[80];  
  
    strcpy(tofmsg, getmsg(123));           // get top-of-file msg  
    strcpy(eofmsg, getmsg(122));          // get end-of-file msg  
  
    msgclose();                           // close the message file  
    ...  
}
```



## **\_norm\_fn**

Creates a “normalized” file name by adding the path and drive code, if appropriate.

```
#include <stdlib.h>
```

```
char * _norm_fn ( char * filename, char * buffer )
```

---

*buffer*                   »   pointer to buffer for normalized file name

*filename*                »   pointer to string containing file description

**Operation:**       The contents of *filename* are analyzed and copied to *buffer* with additions. If *filename* does not specify a path or the path is relative to the current working directory, and there is no drive code specified or the drive code is the same as the drive code for the current working directory, the current working directory is added to the beginning of the file description.

If the contents of *filename* do not specify a drive code, the current working directory has a drive code specification, that drive code is added to the end of the file description.

**Returns:**       A pointer to *buffer*.

**Errors:**       No errors are detected. However, if *filename* contains a path specification and that path is invalid, *buffer* will contain a root specification only.

**Notes:**       The default library is not checked or used in normalizing the file name. Only the path is added if appropriate.

If the current working directory is not used to form the new name, the drive code is not added.

*buffer* should be allocated with a size sufficient to hold the longest possible path and file description (FILENAME\_MAX).

**Conforms to:**        **\_norm\_fn**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**       [getflat](#), [getfn](#), [longfname](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main(int argc, char * argv[])
{
    char buffer[FILENAME_MAX];
    char * ptr;

    ptr = _norm_fn(argv[1], buffer);

    printf("\nFile name is \"%s\"",argv[1]);
    printf("\n_norm_fn is  \"%s\"",ptr);
}
```

---

**Output:**

```
>show cwd
SUBDIR    = /C32:S

>test example.file

File name is "EXAMPLE.FILE"
_norm_fn is "/C32/EXAMPLE.FILE:S"

>show cwd
SUBDIR    = /SAMPLES:S

>test sample1.c

File name is "SAMPLE1.C"
_norm_fn is  "/SAMPLES/SAMPLE1.C:S"

>test /myprog.c

File name is "/MYPROG.C"
_norm_fn is  "/MYPROG.C"

>
```

## ntohl, ntohs

Convert numbers from network-byte order to host-byte order.

```
#include <socket.h>
u_long ntohl ( u_long netlong )
u_short ntohs ( u_short netshort )
```

---

*netlong*               »   32-bit number in network-byte order  
*netshort*             »   16-bit number in network-byte order

**Operation:**       **ntohl**       Converts a 32-bit number from network-byte order and returns a 32-bit number in host-byte order.

**ntohs**       Converts a 16-bit number from network-byte order and returns a 16-bit number in host-byte order.

**Returns:**           The converted value in host-byte order.

**Notes:**            A THEOS network defines host-byte order as “little-endian,” which is the Intel-standard sequence for multiple-byte numeric values. Network-byte order is “big-endian,” which sequences the bytes in reverse order, or most significant byte first. For instance:

Value: 305,419,896 (0x12345678)

Network-byte order: 12 34 56 78

Host-byte order: 78 56 34 12

**Conforms to:**           **ntohl**   q ANSI    q DOS    n THEOS   q POSIX

**ntohs**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**            [htonl](#), [htons](#)

**offsetof macro, sizeof operator**

Compute the offset of a member of a structure or the size of an object.

```
#include <stddef.h>
```

```
void * offsetof ( struct * structname, void memname )
```

```
size_t sizeof ( object )
```

---

*memname*           »   name of a member of a structure

*object*            »   storage type name or name of a specific data item

*structname*       »   pointer to structure

**Operation:**       **offsetof**   This macro computes the relative offset of the start of *memname* item in *structname*. For instance, given the following structure declaration:

```
struct flt {
    short fsa_num;
    short lockpid;
    unsigned long start;
    unsigned long end;
} FLT;
```

The `offsetof(FLT, start)` would be 4, which is the offset of the member `start` in the structure `FLT`.

**sizeof**           Compute the size of *object* in bytes. When *object* is the name of an array or structure, `sizeof` computes the size of the total array or structure. For instance, using the above structure, `sizeof(FLT)` is 12, `sizeof(FLT.start)` is 4.

**Returns:**       Neither “function” actually returns anything. However, the value of the `offsetof` macro is the computed offset of *memname* within *structname*; the value of the `sizeof` operator is the size of *object* in bytes.

**Notes:**        These two keywords are defined in this function library reference, not because they are functions, but because they look like functions. The `offsetof` is a macro defined in the `STDDEF.H` file; the `sizeof` is one of the intrinsic operators built into the C language compiler.

**Conforms to:**       **offsetof**   n ANSI    n DOS    n THEOS   n POSIX  
                      **sizeof**    n ANSI    n DOS    n THEOS   n POSIX

## oneshot functions

Create or release a “one-shot” timer.

```
#include <driver.h>
```

```
void bell_oneshot ( void (*isr)() )
void crt_oneshot ( void (*isr)() )
void floppy_oneshot ( void (*isr)() )
void free_oneshot ( unsigned ohandle )
unsigned get_oneshot ( void )
void mvga_oneshot ( int pid, void (*isr)() )
void oneshot ( unsigned ohandle, long time, void (*isr)(void) )
void scsi_oneshot ( int pid, void (*isr)() )
```

---

|                |   |                                                               |
|----------------|---|---------------------------------------------------------------|
| <i>isr</i>     | » | name of interrupt service routine                             |
| <i>ohandle</i> | » | one-shot handle assigned by <code>get_oneshot</code> function |
| <i>pid</i>     | » | process number                                                |
| <i>time</i>    | » | elapsed trigger-time, in milliseconds                         |

A one-shot is a timer that can be programmed or scheduled and goes off only once.

- Operation:**
- bell\_oneshot** A reserved one-shot timer used by the main console driver (DEV3).
  - crt\_oneshot** A reserved one-shot timer used by the main console driver (DEV3).
  - floppy\_oneshot** A reserved one-shot timer used by the floppy disk driver (DEV1).
  - free\_oneshot** The one-shot timer number *ohandle* is terminated and the *ohandle* number is freed.
  - get\_oneshot** Allocates a handle for a new one-shot timer.
  - mvga\_oneshot** A reserved one-shot timer used by Maxpeed workstation device drivers.
  - oneshot** The one-shot timer number *ohandle* is programmed to “go off” after *time* milliseconds. When it does, the interrupt service routine named *isr* is invoked.
  - scsi\_oneshot** A reserved one-shot timer used by the SCSI driver.
- Returns:**
- get\_oneshot** The next one-shot handle number is returned.
- Errors:**
- get\_oneshot** If all one-shots are in use, the return value is zero. Normally, the one-shot handle number returned is a negative value.
- Notes:**
- There are only 32 one-shots allowed for the entire system so these special timers should be used sparingly.

The *isr* function must be a function defined as:

**458 oneshot functions**

---

```
_interrupt void fctname (void)
```

The prototype for the *isr* function name can be declared with a simple void.

**Restrictions:** These functions are intended to be used by device drivers only.

|                     |                       |        |       |         |         |
|---------------------|-----------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>bell_oneshot</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>crt_oneshot</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>floppy_oneshot</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>free_oneshot</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>get_oneshot</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>mvga_oneshot</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>oneshot</b>        | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>scsi_oneshot</b>   | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [schedint](#)

## open

Open a file.

```
#include <io.h> or <handle.h>
```

```
int open ( char * filename, int flag, [[int pmode]] )
```

---

*filename* » pointer to string containing file description

*flag* » file-access mode

*pmode* » file-permission codes used when creating new file (ignored)

**Operation:** The *filename* and *flag* are used in a request to the operating system to open a file named *filename* with access type *flag*. If the argument *pmode* is specified (it is optional), it is always ignored.

The possible values for the *flag* parameter are defined in the FCNTL.H file and include:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                                                                                              |
|-------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| O_APPEND    | 8            | All new records written to the file will be written at the current end-of-file.                                                             |
| O_CREAT     | 256          | Creates (if necessary) and opens a new file for writing. If the file exists, it is not erased.                                              |
| O_EXCL      | 1024         | Used with O_CREAT flag. If the file exists already, an open fails.                                                                          |
| O_RDONLY    | 0            | Opens the file for read access only. Neither O_RDWR nor O_WRONLY flags may be used with this flag.                                          |
| O_RDWR      | 2            | Opens the file for both read and write access. Neither O_RDONLY nor O_WRONLY flags may be used with this flag.                              |
| O_TRUNC     | 512          | Opens the file and destroys the existing contents. The file must have write permission and you cannot use the O_RDONLY flag with this flag. |
| O_WRONLY    | 1            | Opens the file for writing only. Neither O_RDONLY nor O_RDWR flags may be used with this flag.                                              |

The file is opened as an unstructured stream file. That is, if *filename* refers to an existing direct, keyed, indexed or a program file, it is opened for stream or byte access.

**Returns:** The opened file's file handle. A file handle is a small, positive integer. A return value of -1 indicates that an error occurred and that the file could not be opened.

**Errors:** When a -1 is returned, [errno](#) is set to one of the following codes:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                                       |
|-------------|--------------|--------------------------------------------------------------------------------------|
| EACCES      | 18           | Protected file.                                                                      |
| EBADF       | 24           | Invalid <i>filename</i> .                                                            |
| EEXIST      | 44           | File already exists. O_CREAT and O_EXCL flags specified and the file already exists. |
| EMFILE      | 15           | Too many open files.                                                                 |
| ENODEV      | 13           | Disk not attached.                                                                   |
| ENOENT      | 19           | File not found.                                                                      |
| ENOMEM      | 3            | Insufficient memory.                                                                 |
| ENOSPC      | 42           | Disk full.                                                                           |

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the `ferror`, `strerror`, *etc.* functions can be used to report the error.

**Notes:** The specification of the file in *filename* is assumed to be in the current working directory unless *filename* specifies the path to the file.

There are other flag names defined in the file `STAT.H` but they are unused by the `open` function.

**Defaults:** When `open` must create the file, it sets the following default permissions: shared-read and write-protected, owner execute-protected and not hidden. This default can be changed by defining an environment variable named `CREATE` whose value is the protection codes desired. Refer to the `CREATE` environment variable description in the *THEOS System Reference* manual.

**Restrictions:** The *flag* argument must specify one of the modes `O_RDONLY`, `O_RDWR` or `O_WRONLY`.

**Conforms to:** `open`    q ANSI    n DOS    n THEOS    n POSIX

**See also:** [access](#), [fdopen](#), [file\\_err](#), [fopen](#), [freopen](#), [openhelp](#), [openmenu](#), [pipe](#), [popen](#)

**Example:** This example opens a file named `MY.FILE`, in the current working directory, for update access. If the file does not exist, it creates it. If it does exist, it erases the existing contents of the file.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void
main()
{
    char    fn[30];
    int     fhandle;

    strcpy(fn, "my.file");

    if ((fhandle=open(fn, O_RDWR | O_CREAT | O_TRUNC))== -1)
        exit(ferror());
    ...
}
```



## openhlp, openmenu

Open the help file or the menu file associated with a program.

```
#include <stdlib.h>
```

```
FILE * openhlp ( const char * prgname )
```

```
FILE * openmenu ( const char * prgname )
```

---

*prgname*                   »   pointer to name of help or menu file to open

**Operation:**       These two functions open the help or menu file from the appropriate library.

**openhlp**   The member file *prgname* in the library SYSTEM.HELPVVVVn:S is opened.

**openmenu**   The member file *prgname* in the library SYSTEM.MENUVVVVn:S is opened.

The specific library used by these functions is determined by the operating system version (386 or 32) and the current language code in effect.

**Returns:**       A pointer to the opened file's fcb.

**Errors:**       If the file cannot be opened, a NULL pointer is returned and the global variables [\\_errno](#) and [\\_errarg](#) are set to indicate the specific error encountered.

**Notes:**        These functions use the [fopen](#) function to open the appropriate file in "rt" access mode.

The *prgname* value does not have to be the same as the current program name. For instance, if the current program is named TEST, *prgname* could be test, test1, myhelp or anything else. The value of *prgname* is also case-insensitive.

Use the functions [\\_get\\_help\\_filename](#) or [\\_get\\_menu\\_filename](#) after using these functions to retrieve the full name of the help or menu file opened or attempted to be opened.

**Restrictions:**   *prgname* should be eight characters or less. Characters in a string longer than eight characters are ignored.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>openhlp</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>openmenu</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**       [devnames functions](#), [fclose](#), [fcloseall](#), [file\\_err](#), [fopen](#), [fperror](#), [\\_get\\_help\\_filename](#), [\\_get\\_menu\\_filename](#)

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
main(int argc, char *argv[])
{
    char    helpname[40],
            menuname[40];
    FILE    * helpfile,
            * menufile;

    if (!(helpfile = openhelp("TEST")))
        perror();
    if (!(menufile = openmenu("TEST1")))
        perror();

    strcpy(helpname, _get_help_filename());
    strcpy(menuname, _get_menu_filename());

    printf("\nHelp file name is: %s", helpname);
    printf("\nMenu file name is: %s", menuname);

    fclose(menufile);
    fclose(helpfile);
}
```

---

### Output:

```
>test
Member "/SYSTEM.HELP32.TEST:S" not found.

Help file name is: /SYSTEM.HELP32.TEST:S
Menu file name is: /SYSTEM.MENU32.TEST1:S

>
```

## or memory assembly functions

This group of “functions” generates in-line code to perform an OR operation on bytes, short integers or long integers in another memory area or in your program’s code segment.

```
#include <builtin.h>

void _orb ( void _far * far_offset, char mask )
void _ord ( void _far * far_offset, long lmask )
void _orw ( void _far * far_offset, int imask )

void _orcsb ( void * offset, char mask )
void _orcsl ( void * offset, long lmask )
void _orcsw ( void * offset, int imask )
```

---

|                   |   |                                      |
|-------------------|---|--------------------------------------|
| <i>far_offset</i> | » | pointer to remote data object        |
| <i>imask</i>      | » | 16-bit mask value                    |
| <i>lmask</i>      | » | 32-bit mask value                    |
| <i>mask</i>       | » | 8-bit mask value                     |
| <i>offset</i>     | » | pointer to data item in code segment |

**Operation:** These functions perform a bitwise, Boolean OR operation. That is, a bit position in the result is set to “on” if the same bit position in either of the objects is “on.”

**\_orb** ORs the value of *bval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_ord** ORs the value of *lval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_orw** ORs the value of *wval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_orcsb, \_orcsl, \_orcsw** These functions perform the same operation as **\_orb**, **\_ord** and **\_orw** except that the memory segment is pre-defined to be your program’s code segment alias. These three functions can only be used in device drivers because other programs do not have an alias to their code segment and they will generate a general protection error.

**Returns:** No value is returned by these functions. The values are ORed directly to the values in the locations specified.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can OR whatever type of object desired to any location. For instance, an **\_orw** function can OR a word value (double-byte value) to a location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:** These functions are intended for device-driver authors.

The nucleus and scr memory regions are not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_orb</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_ord</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_orcsb</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_orcsd</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_orcsw</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_orw</b>   | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

---

**\_osmajor, \_osminor, \_osversion**

Variables that represent the operating system version number.

```
#include <stdlib.h>
_osmajor
_osminor
_osversion
```

**Operation:** These variables are defined as macros that are expanded into references to the operating system version number that is maintained in the nucleus memory segment.

**Returns:**

**\_osmajor** The integer portion of the operating system version number. For instance, if the version is 4.1, then \_osmajor is 4.

**\_osminor** The fractional portion of the operating system version number. For instance, if the version is 4.1, then \_osminor is 1.

**\_osversion** The \_osmajor and \_osminor values are combined into one number by multiplying \_osmajor by 256 and adding \_osminor. For instance, if the version is 4.1, then \_osversion is 1025 (4×256+1).

**Conforms to:**

|                   |        |       |         |         |
|-------------------|--------|-------|---------|---------|
| <b>_osmajor</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_osminor</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_osversion</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [\\_getosver](#), [getver](#)

---

**Example:** Refer to [getver](#) example on page 325.

**output port assembly functions**

This group of “functions” generates in-line code to output a byte, word or double-word to an output port.

```
#include <builtin.h>
void _outb ( unsigned port, char bvalue )
void _outd ( unsigned port, long lvalue )
void _outw ( unsigned port, short ivalue )

void _outsb ( unsigned port, void _far * buffer, long count )
void _outsd ( unsigned port, void _far * buffer, long count )
void _outsw ( unsigned port, void _far * buffer, long count )
```

---

|               |   |                               |
|---------------|---|-------------------------------|
| <i>buffer</i> | » | pointer to data to be output  |
| <i>bvalue</i> | » | byte value to output          |
| <i>count</i>  | » | number of objects to output   |
| <i>ivalue</i> | » | short integer value to output |
| <i>lvalue</i> | » | long integer value to output  |
| <i>port</i>   | » | port number of output port    |

|                   |               |                                                                                                                                                            |
|-------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Operation:</b> | <b>_outb</b>  | Writes a single byte to the output port number <i>port</i> .                                                                                               |
|                   | <b>_outd</b>  | Writes a double-word to the output port number <i>port</i> .                                                                                               |
|                   | <b>_outw</b>  | Writes a single word to the output port number <i>port</i> .                                                                                               |
|                   | <b>_outsb</b> | Writes <i>count</i> number of bytes to the output port number <i>port</i> . The data is read from consecutive locations starting at <i>offset</i> .        |
|                   | <b>_outsd</b> | Writes <i>count</i> number of double-words to the output port number <i>port</i> . The data is read from consecutive locations starting at <i>offset</i> . |
|                   | <b>_outsw</b> | Writes <i>count</i> number of single words to the output port number <i>port</i> . The data is read from consecutive locations starting at <i>offset</i> . |

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive use of the call and ret instructions.

**Restrictions:** These functions are intended for device-driver authors and cannot be executed by programs operating with greater than ring 2 access.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_outb</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_outd</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_outsb</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_outsd</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_outsw</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_outw</b>  | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [input port assembly functions](#)

## pagewait

Wait for operator to “release” the console screen display.

```
#include <stdio.h>
int pagewait ( void )
```

**Operation:** Using the [getch](#) and [putch](#) console i/o functions, display the “page-wait” character in the lower-left corner of the screen and wait for the operator to enter a control character or the space character.

**Returns:** The character entered by the operator is returned.

If page-waits are suppressed, a zero is returned.

**Notes:** The “page-wait” character is the up-caret ( ^ ).

All characters entered by the operator, except for the space character or a control character, are ignored and discarded.

The page-wait operation can be suppressed by the operator by toggling the screen page-wait ( [Break](#), [W](#) )

The page-wait operation is always suppressed when there is no console, i.e., a background task.

The control character value returned can be used by your program to indicate an operator request. For instance, the LIST utility uses the `pagewait` function to see if the operator wanted to page back, start at the beginning, go to the last page, *etc.*

The header file `FUNC_KEY.H` defines names for console control keys that might be useful when interpreting the return value of this function.

**Defaults:** If page-waits are suppressed, a zero is returned.

**Conforms to:** `pagewait` q ANSI q DOS n THEOS q POSIX

**See also:** [getch](#), [putch](#), [testhead](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <crt.h>
#include <func_key.h>
#include <lub.h>

void main()
{
    FILE    * in;
    int      line_nbr = 0;
    int      exit = 0;
    char     buffer[128];
    char     c;

    if (!(in = fopen("system.theos32.devnames","r")))
        exit(fperror());           // display and exit

    while (!feof(in))
    {
        fgets(buffer, 127, in);
        if (++line_nbr >= getpl(CONSOLE))
        {
            c = pagewait();
            line_nbr = 0;
            switch (c)
            {
                case TOP:           // start at beginning
                    rewind(in);
                    fgets(buffer,127, in);
                    line_nbr = 1;
                    break;
                case QUIT:          // terminate now
                    exit = 13;
                    break;
            }
            if (exit)               // terminate?
                break;
            crt(CLEAR);
        }
        fputs(buffer, stdout);
    }
    fclose(in);
}
```



## pause

The program execution is delayed for a precise amount of time.

```
#include <driver.h>
void pause ( unsigned count_15_us )

_____
count_15_us      »   time to pause, in 15 microsecond units
```

**Operation:** Interrupts are disabled and the program's execution is delayed for *count\_15\_us* units of 15-microseconds of time. When the time interval has elapsed, interrupts are restored and program execution resumes.

**Notes:** If *count\_15\_us* is zero, the program execution is delayed for a period of time less than 15 microseconds.

**Restrictions:** This function is intended for device-driver authors.

Disabling interrupts with this function, or any function, is a dangerous procedure and should be done as rarely as possible. Other, safer methods of pausing for a period of time are provided with the functions listed below.

**Conforms to:** **pause**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [delay](#), [sleep](#), [sleep\\_msec](#), [sleep\\_sec](#), [sleep\\_until](#), [yield](#), [\\_pre\\_empt](#), [\\_snu](#), [\\_yield](#)

### peek memory assembly functions

This group of “functions” generates in-line code to access bytes, short integers, long integers or pointers from another memory area or from the specific memory areas: nucleus, your csi or your program’s code segment.

```
#include <builtin.h> or <peek.h>
unsigned _peekb ( void _far * far_offset )
unsigned long _peekd ( void _far * far_offset )
void * _peekp ( void _far * far_offset )
unsigned _peekw ( void _far * far_offset )

unsigned _peekcsb ( void * offset )
unsigned long _peekcsd ( void * offset )
void * _peekcsp ( void * offset )
unsigned _peekcsw ( void * offset )

unsigned _peeknucb ( void * offset )
unsigned long _peeknucd ( void * offset )
void * _peeknucp ( void * offset )
unsigned _peeknucw ( void * offset )

unsigned _peekscrb ( void * offset )
unsigned long _peekscrd ( void * offset )
void * _peekscrip ( void * offset )
unsigned _peekscrw ( void * offset )
```

---

*far\_offset*           »   pointer to remote data object  
*offset*                »   pointer to data item in nuc, scr or code segment

**Operation:**    **\_peekb**       Gets a byte value (unsigned char) from location *far\_offset*.

**\_peekd**       Gets a double-byte value (unsigned long integer) from location *far\_offset*.

**\_peekp**       Gets a pointer value from location *far\_offset*.

**\_peekw**       Gets a word value (unsigned short integer) from location *far\_offset*.

**\_peekcsb, \_peekcsd, \_peekcsw** These functions operate identically to *\_peekb*, *\_peekd* and *\_peekw* except that the memory selector is predefined to be your program’s code segment.

**\_peeknucb, \_peeknucd, \_peeknucw** These functions operate identically to *\_peekb*, *\_peekd* and *\_peekw* except that the memory selector is predefined to be the nucleus memory segment.

**\_peekscrb, \_peekscrd, \_peekscrw** These functions operate identically to *\_peekb*, *\_peekd* and *\_peekw* except that the memory selector is predefined to be the scr memory segment.

**Returns:**

- `_peekb`     A byte value (unsigned char).
- `_peekd`     A double-word value (unsigned long).
- `_peekp`     A pointer value (void \*).
- `_peekw`     A word value (unsigned short).

The other functions return similar values, as defined by the last character of their names: `b` returns a byte, `d` returns a double-word, `p` returns a pointer and `w` returns a word.

**Notes:**     These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can return whatever type of object desired from any location. For instance, a `_peekw` function can return a word (double-byte value) from a location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:**     These functions are intended for device-driver authors.

**Conforms to:**

|                         |        |       |         |         |
|-------------------------|--------|-------|---------|---------|
| <code>_peekb</code>     | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekcsb</code>   | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekcsd</code>   | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekcsp</code>   | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekcsw</code>   | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekd</code>     | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peeknucb</code>  | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peeknucd</code>  | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peeknucp</code>  | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peeknucw</code>  | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekp</code>     | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekscrib</code> | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekscrd</code>  | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekscrip</code> | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekscriw</code> | q ANSI | q DOS | n THEOS | q POSIX |
| <code>_peekw</code>     | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**     [add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

**perror**

Write a message on `stderr` followed by standard message associated with [errno](#).

```
#include <stdio.h>
void perror ( char * string )
```

---

*string*                   »   pointer to program-defined message string

**Operation:**       The text from *string* is prepended to the standard message text from the `SYSTEM.TEOSnnn.MESSAGES` file and is written to `stderr`. The text of *string* is separated from the standard error message text by a colon, space.

If *string* is `NULL` or a pointer to a null string, only the system-defined message is output to `stderr`.

**Notes:**           The system-defined message is defined in the `SYSTEM.TEOSnnn.MESSAGES` file and is indexed by the value of [errno](#) variable. It is displayed using parameter substitution. The substitution text comes from the reserved field `_errarg`, which normally contains the name of the file associated with the last file error detected.

Because the value of [errno](#) might be changed by subsequent function calls, it is always best to call the `perror` function immediately after a library routine returns with an error.

**Conforms to:**               **perror**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [clearerr](#), [eof](#), [\\_errbot](#), [errmsg](#), [feof](#), [ferror](#), [file\\_err](#), [fperror](#), [putmsg](#), [strerror](#), [syserr](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    FILE     * f;

    if (!(f = fopen("my.file", "r")))
    {
        perror("Unexpected error");
        exit(1);
    }
    ...
}
```

---

**Output:**

```
>test

Unexpected error: File "my.file" not found.

>
```

---

***\_PhyAddr, \_phy\_addr***

Compute the physical or absolute address of a memory location and reserve that memory so that it does not move during memory compaction.

```
#include <getmem.h>
unsigned long _PhyAddr( void _far * ptr, size_t len)
unsigned long _phy_addr( unsigned selector, void * offset, size_t len)
```

---

|                 |   |                              |
|-----------------|---|------------------------------|
| <i>len</i>      | » | length of memory to reserve  |
| <i>offset</i>   | » | offset within memory segment |
| <i>ptr</i>      | » | pointer to memory location   |
| <i>selector</i> | » | memory selector              |

**Operation:** Both of these functions are the same except for the manner of specifying the memory location.

These routines perform two functions: Compute the physical address for a memory location and reserve that memory so that it does not participate in memory compaction. The *len* field specifies the length of the memory that will be reserved (see [\\_rsvmem](#) on page 538). If *len* is zero, the memory is released.

**Returns:** The physical address of the memory location.

**Errors:** A return of zero means that the memory reference is invalid or the memory location + *len* is invalid. The memory region must be in a data memory segment (read and write access).

**Notes:** A ***physical address*** is a 32-bit address reference. It refers to the absolute address of a memory location. A far pointer is a 48-bit reference to a memory location. It is a combination of a memory segment selector and a 16-bit offset within that memory segment. The Intel 386, 486 and Pentium processors always reference memory with an offset and a memory selector.

Some devices require physical address references because they bypass the CPU for data transfers (DMA). This function computes the physical address for a memory location.

These routines are used by device-drivers when they need to temporarily reserve memory. The [\\_rsvmem](#) function is used when the memory is “permanently” reserved, such as a memory-mapped video device.

It is very important to release the memory reserved by the [\\_PhyAddr](#) and [\\_phy\\_addr](#) functions when the driver is finished using the memory. If it is not released, the memory cannot be compacted and memory will become very fragmented. Programs may not be able to execute due to lack of contiguous memory.

|                     |                           |        |       |         |         |
|---------------------|---------------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <a href="#">_PhyAddr</a>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <a href="#">_phy_addr</a> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [\\_rsvmem](#)

**Example:**

```
char _far * xfer_buf;

_driver int device_read(UCB *ucb, char _far *buf, int len)
{
    unsigned long addr;

    ...
    // ready to do data transfer

    xfer_buf = buf;                // save virtual address
    addr = _PhyAddr(buf, len);

    // program DMA device with address and transfer direction
    // start transfer

    ...
    return len;
}

_interrupt void irq_routine ( void )
{
    ...
    // release the memory address
    _PhyAddr(xfer_buf, 0);
}
```

## pipe

Opens an input/output channel that can be used between two tasks.

```
#include <stdio.h> or <io.h>
```

```
int pipe ( int filenum[2] )
```

---

*filenum*                    »    two-element array for opened pipe file numbers

**Operation:**      A “pipe” is opened using the [popen](#) function. The [fileno](#) function is then used to determine the file handles for the pipe.

After successfully opening the pipe, the array *filenum* is filled in with *filenum*[0] being the file handle of the input channel of the pipe and *filenum*[1] being the file handle of the output channel of the pipe.

**Returns:**          A true/false value indicating the success of the pipe open operation. A zero indicates that the pipe was opened successfully. Otherwise, a non-zero value is returned.

**Errors:**            When the pipe cannot be opened, [errno](#) is set to one of the following codes:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>          |
|-------------|--------------|-------------------------|
| EACCES      | 18           | Protected file          |
| EBADF       | 24           | Invalid <i>filename</i> |
| ENODEV      | 13           | Disk not attached       |
| ENOENT      | 19           | File not found          |
| ENOMEM      | 3            | Insufficient memory     |
| ENOSPC      | 42           | Disk full               |
| EMFILE      | 15           | Too many open files     |

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [ferror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:**            This function should be used prior to starting a subtask. When the subtask is started, it will inherit the handles to the pipe and then both of the tasks can use it.

**Restrictions:**    Generally, only one task will write to the pipe and the other task will read from the pipe. Two separate pipes should be used if bidirectional communications are needed. If a single task tries to both read and write to a pipe, a “deadly embrace” may result.

**Conforms to:**                    **pipe**      q ANSI      q DOS      n THEOS      q POSIX

**See also:**            [fileno](#), [popen](#)

### poke memory assembly functions

This group of “functions” generates in-line code to access bytes, short integers, long integers or pointers from another memory area or from the specific memory areas: nucleus, your csi or your program’s code segment.

```
#include <builtin.h> or <peek.h>
void _pokeb ( char c, void _far * far_offset )
void _poked ( long lvalue, void _far * far_offset )
void _pokep ( void * ptr, void _far * far_offset )
void _pokew ( int ivalue, void _far * far_offset )
```

```
void _pokecsb ( char c, void * offset )
void _pokecsd ( long lvalue, void * offset )
void _pokecsp ( void * ptr, void * offset )
void _pokecsw ( int ivalue, void * offset )
```

```
void _pokenucb ( char c, void * offset )
void _pokenucd ( long lvalue, void * offset )
void _pokenucp ( void * ptr, void * offset )
void _pokenucw ( int ivalue, void * offset )
```

```
void _pokescrib ( char c, void * offset )
void _pokescrib ( long lvalue, void * offset )
void _pokescrip ( void * ptr, void * offset )
void _pokescriw ( int ivalue, void * offset )
```

---

|                   |   |                                                  |
|-------------------|---|--------------------------------------------------|
| <i>c</i>          | » | byte value                                       |
| <i>ivalue</i>     | » | short integer value                              |
| <i>far_offset</i> | » | pointer to remote data object                    |
| <i>lvalue</i>     | » | long integer value                               |
| <i>offset</i>     | » | pointer to data item in nuc, scr or code segment |
| <i>ptr</i>        | » | pointer value                                    |

|                   |                                     |                                                                                                                                                                                                                                                                                                                                       |
|-------------------|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Operation:</b> | <b>_pokeb</b>                       | Puts a byte value (unsigned char) in the location <i>far_offset</i> .                                                                                                                                                                                                                                                                 |
|                   | <b>_poked</b>                       | Puts a double-word value (unsigned long integer) in the location <i>far_offset</i> .                                                                                                                                                                                                                                                  |
|                   | <b>_pokep</b>                       | Puts a pointer value in the location <i>far_offset</i> .                                                                                                                                                                                                                                                                              |
|                   | <b>_pokew</b>                       | Puts a word value (unsigned short integer) in the location <i>far_offset</i> .                                                                                                                                                                                                                                                        |
|                   | <b>_pokecsb, _pokecsd, _pokecsw</b> | These functions operate identically to _pokeb, _poked and _pokew except that the memory selector is predefined to be your program’s code segment alias. These three functions can only be used in device drivers because other programs do not have an alias to their code segment and they will generate a general protection error. |



**\_pokenucb, \_pokenucd, \_pokenucw** These functions operate identically to `_pokeb`, `_poked` and `_pokew` except that the memory selector is predefined to be the nucleus memory segment alias.

**\_pokescrib, \_pokescrd, \_pokescrw** These functions operate identically to `_pokeb`, `_poked` and `_pokew` except that the memory selector is predefined to be the scr memory segment alias.

**Returns:** No values are returned by these functions.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can poke whatever type of object desired to any location. For instance, a `_pokew` function can put a word value (double-byte value) to a location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

|                     |                   |        |       |         |         |
|---------------------|-------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_pokeb</b>     | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokecsb</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokecsd</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokecsp</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokecsw</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_poked</b>     | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokenucb</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokenucd</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokenucp</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokenucw</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokep</b>     | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokescrib</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokescrd</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokescrp</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokescrw</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pokew</b>     | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

**popen**

Opens a pipe file.

```
#include <stdio.h>
```

```
int popen ( FILE ** rd_pipe, FILE ** wr_pipe )
```

---

```
rd_pipe          »  pointer to a pointer to the read file control block
```

```
wr_pipe          »  pointer to a pointer to the write file control block
```

**Operation:** A pipe is opened for both input and output. The pointer to the input file's fcb is copied to the location pointed to by *rd\_pipe* and the pointer to the output file's fcb is copied to the location pointed to by *wr\_pipe*.

**Returns:** A success/fail indicator. When the pipe is opened successfully, a zero is returned. When the open is unsuccessful, a non-zero is returned and [errno](#) is set to the error code describing the reason for the failure.

**Errors:** When the pipe cannot be opened, [errno](#) is set to one of the following codes:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>            |
|-------------|--------------|---------------------------|
| EACCES      | 18           | Protected file.           |
| EBADF       | 24           | Invalid <i>filename</i> . |
| ENODEV      | 13           | Disk not attached.        |
| ENOENT      | 19           | File not found.           |
| ENOMEM      | 3            | Insufficient memory.      |
| ENOSPC      | 42           | Disk full.                |
| EMFILE      | 15           | Too many open files.      |

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [ferror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:** A pipe is a special type of file that may be thought of as a “memory file,” that is, a file that resides in memory.

Pipes should only be used in a multi-task application and any one task should only use one end of the pipe. This means that one task writes data to a pipe while a different task reads the data.

Similar to a normal disk file, the task writing data to a pipe is not normally delayed—the data is immediately taken by the system and placed in the pipe in a first-in, first-out manner. The task reading data out of the pipe might be delayed if there is no data available in the pipe.

Although it is possible for a single program to use a pipe as temporary storage (writing data to the pipe, closing the pipe, then reading data from the pipe at a later time), it is not advised. This type of usage may result in a deadly embrace.

Once a pipe is created, your program can use the normal stream file functions to access the pipe file.

The end-of-file status is not set until the writing task closes the pipe and the reading task reads all of the data written to the pipe. If the reading task reads all of the data written to the pipe but the pipe is still open for writing, no end-of-file occurs and the reading task is suspended until data becomes available in the pipe.

**Restrictions:** Although multiple pipes may be opened and in use by a task, a pipe is treated like any other file and there is a limited number of files that may be open at any one time. Since a task should only use one end of a pipe, you can close the unused end of a pipe.

**Conforms to:** **popen** q ANSI q DOS n THEOS q POSIX

**See also:** [fopen](#), [open](#), [pipe](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <sem.h>

void
main()
{
    FILE    * pin,
            * pout;
    char    buffer[256],
            data[256];
    int     task_done;

    if (popen(&pin, &pout))           // open pipe for i/o
        exit(fperror());

    task_done = semaphore("finished"); // intertask flag

    semares(task_done);                // clear it

    if (fork()) {                     // fork a subtask
        fclose(pin);                  // parent task code
        while (1) {                   // get and process data
            ...
            fputs(data, pout);        // write to pipe
        }
        fclose(pout);                // close output end
        semawait(task_done);          // wait for subtask
    }
    else {                             // subtask code
        fclose(pout);                // close unused end
        while (fgets(buffer, 256, pin)) { // get pipe data
            ...                       // process
        }
        fclose(pin);                // close input end
        semaset(task_done);          // signal parent
        exit();                      // stop this subtask
    }
}
```

**\_popf, \_pushf assembly functions**

These two “functions” generate in-line code to pop or push the flags register of the CPU.

```
#include <builtin.h> or <driver.h>
void _popf ( void )
void _pushf ( void )
```

**Operation:**     **\_popf**       Pop the flags register from the stack.

**\_pushf**     Push the flags register onto the stack.

**Notes:**         These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions or the inconvenience of using the `#asm` and `#endasm` directives.

**Restrictions:**   These functions are intended for device-driver authors.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_popf</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_pushf</b> | q ANSI | q DOS | n THEOS | q POSIX |

**pow**

Compute the result of one number being raised to the power of another number.

```
#include <math.h>
double pow ( double x, double y )
```

---

|          |   |                         |
|----------|---|-------------------------|
| <i>x</i> | » | floating-point value    |
| <i>y</i> | » | floating-point exponent |

**Operation:** The value of *x* is raised to the power of *y*:

$$x^y$$

**Returns:** The result of  $x^y$ .

If  $x \neq 0$  and  $y = 0$ , the return value is 1.

If  $x = 0$  and  $y < 0$ , the return value is 0 and [errno](#) is set to EDOM.

If  $x = 0$  and  $y = 0$  or  $x < 0$  and  $y$  is not an integer value, the return value is 0, a DOMAIN error message is printed to `stderr` and [errno](#) is set to EDOM.

If an overflow value results, [errno](#) is set to ERANGE and  $\pm\text{HUGE\_VAL}$  is returned.

**Errors:** When an error occurs, [errno](#) is set to EDOM or ERANGE as appropriate.

**Notes:** This function uses BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:** **pow**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [exp](#), [log](#), [log2](#), [log10](#), [sqrt](#)

**prime, primel**

Compute a prime number.

```
#include <stdlib.h>
unsigned prime ( unsigned n )
long primel ( long ln )
```

---

*ln*                   » seed number  
*n*                    » seed number

**Operation:**       **prime**       Computes the smallest 16-bit prime number that is greater than or equal to *n*.

**primel**      Computes the smallest 16-bit prime number that is greater than or equal to *ln*.

**Returns:**         The computed prime number.

**Conforms to:**       **prime**    q ANSI    q DOS    n THEOS   q POSIX

**primel**   q ANSI    q DOS    n THEOS   q POSIX

**printf**

Write formatted characters, strings, literal text and numbers to the `stdout` device.

```
#include <stdio.h>
```

```
int printf ( char * format [,argument-list] )
```

---

*format*                   »   formatting control mask

*argument-list*       »   optional arguments to format and print

**Operation:**       The *argument-list* is formatted according to the specifications in *format* and the result is written to `stdout`.

**Returns:**         The number of characters displayed.

**Errors:**         Invalid formatting specifications or specifications that do not match the *argument-list* are ignored and no error is reported.

**Notes:**         The *format* string contains zero or more ordinary characters, escape sequence characters (i.e., “\n”) and format specifications. The ordinary characters and escape sequence characters are written to `stdout` in the same sequence as specified. For instance:

```
printf("Sample output\nSecond line\t\t\ttabbed text");
```

produces:

```
Sample output
Second line      tabbed text
```

When one or more arguments follow *format*, the *format* string should contain ***format specifications*** for each of the arguments in *argument-list*.

### ■ Format Specifications

Format specifications always begin with a percent sign ( % ) character and are used in the sequence that they are specified. The first specification is used for the first argument in *argument-list*, the second specification is used for the second argument, and so on.

When there are more arguments than format specifications, the extra arguments are ignored. When there are more format specifications than there are arguments, the extra specifications may cause an address error.

A format specification has the following form, with the components enclosed in square brackets optional:

```
% [flags] [width] [precision] [modifier] type
```

Although the *flags* may contain a space character, spaces are not used between the components.

Each component of the format specification is a single letter or whole number value. The simplest form of a format specification has a percent sign followed by a single-letter *type* specification. For instance: %c.



**Type** is a required component specifying the type of argument to format (character, string or number). **Flags** is an optional component specifying justification, signing, decimal points and octal and hexadecimal prefixes. **Width** is an optional component indicating the minimum width to display. **Precision** is an optional component indicating the maximum width or the minimum number of digits for integers. **Modifier** is an optional component indicating the size of the argument (short, long, *etc.*).

### ■ Type

The *type* component is required and specifies the type of argument to format.

| Type   | Field Type  | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| b      | int         | An unsigned binary integer.                                                                                                                                                                                                                                                                                                                                                                                                           |
| c      | char<br>int | Single character.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| d      | int         | Signed, decimal integer. (Synonym to ‘i’ type.)                                                                                                                                                                                                                                                                                                                                                                                       |
| e or E | double      | <p>Floating-point value formatted in scientific notation: -d.ddddde±dd. (An optional leading negative sign, one non-zero digit to the left of the decimal point, <i>precision</i> number of digits to the right of the decimal point, a lowercase e followed by a signed two- or three-digit exponent.</p> <p>The uppercase E type is identical except an uppercase E is used to introduce the exponent instead of a lowercase e.</p> |
| f      | double      | <p>Floating-point value formatted in non-scientific form: -ddd.dddd. (An optional leading negative sign followed by the value of the number.) The number of digits to the left of the decimal point depends upon the magnitude of the value. The number of digits to the right of the decimal point is controlled by <i>precision</i> with a default <i>precision</i> of six.</p>                                                     |
| g or G | double      | <p>Floating-point value formatted in either “e” or “f” format. The “e” format is used only when the exponent of the value is less than -4 or is greater than or equal to the <i>precision</i> argument. Trailing zeros are omitted and the decimal point is used only if there are one or more digits following it.</p> <p>The G specification is identical except that it uses either “E” or “F” format.</p>                         |
| i      | int         | Signed, decimal integer.                                                                                                                                                                                                                                                                                                                                                                                                              |
| n      | int *       | No argument is converted. The number of characters written so far to stdout is stored in the location pointed to by the argument.                                                                                                                                                                                                                                                                                                     |
| o      | int         | Unsigned octal integer.                                                                                                                                                                                                                                                                                                                                                                                                               |
| p      | void *      | The value of the pointer (not what is pointed to) is written to stdout using %x format.                                                                                                                                                                                                                                                                                                                                               |
| s      | char *      | Characters are written for the entire string length or until <i>precision</i> number of characters.                                                                                                                                                                                                                                                                                                                                   |
| t      | N/A         | Format current date and time as “yyyy/mm/dd hh:mm:ss” similar to the <a href="#">strftime</a> format of “%Y/%m/%d %T”.                                                                                                                                                                                                                                                                                                                |

Table 5: printf Type Codes

| <i>Type</i> | <i>Field Type</i> | <i>Meaning</i>                                                                                                                                                                           |
|-------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| u           | int               | Unsigned decimal integer.                                                                                                                                                                |
| x or X      | int               | An unsigned hexadecimal integer. The lowercase x specification uses lowercase hexadecimal digits (“abcdef”); the uppercase X specification uses uppercase hexadecimal digits (“ABCDEF”). |
| %           | N/A               | If a percent sign character is followed by a percent sign character, a single percent sign is written to stdout. For instance, a specification of %% outputs a single %.                 |

Table 5: printf Type Codes

### ■ Flags

The *flags* component is optional and controls justification, positive value signing, zero padding, *etc.* A format specification may have more than one *flag* code used.

| <i>flag</i>            | Meaning                                                                                                                                                                                                                                                                                                                                                                  | Default                                                                                                                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| –                      | Left justify output within <i>width</i> .                                                                                                                                                                                                                                                                                                                                | Right justify.                                                                                                                                   |
| +                      | Prefix the output with its sign (either + or –). Applies only to numeric, signed <i>types</i> .                                                                                                                                                                                                                                                                          | Sign character is output only when negative.                                                                                                     |
| 0                      | Left-pad the number with zeros for minimum <i>width</i> . If 0 and – specified, the 0 is ignored. Applies only to numeric <i>types</i> .                                                                                                                                                                                                                                 | No zero padding.                                                                                                                                 |
| <i>space character</i> | Use leading space character to indicate positive value. If ‘ ’ and + specified, the ‘ ’ is ignored. Applies only to numeric, signed <i>types</i> .                                                                                                                                                                                                                       | No space used for positive values.                                                                                                               |
| #                      | When used with octal or hexadecimal <i>types</i> (o, x or X), use leading zero for octal and leading 0x or 0X for hexadecimal.<br><br>When used with e, E or f <i>types</i> , force the output to contain a decimal point.<br><br>When used with g or G <i>types</i> , force the output to contain a decimal point and zero pad on the right for <i>precision</i> width. | No prefix.<br><br>Decimal point used only when digits follow it.<br><br>Decimal point used only when digits follow it; no zero padding on right. |
| ,                      | When used with numeric <i>types</i> (b, d, e, E, f, g, G, i, o, u, x and X) commas are added between every third character, counting from the right. (For base-10 value displays, this is the thousand’s separator).<br><br>The separating character used (“,” or “.”) depends upon the “Decimal is comma” environment variable value.                                   | No commas are used.                                                                                                                              |

Table 6: printf Flag Codes

Note: The comma flag specification is not compatible with ANSI or DOS.

### ■ Width

The *width* is an optional component that controls the minimum width of the formatted field. When the *argument* is formatted and its formatted length is less than the specified *width*, the output is space-padded on the left or right (depending upon the – flag) until the minimum *width* is reached. A negative *width* field is treated as a – flag followed by a positive *width* value. When the formatted length is greater than *width*, no spaces are added nor is the output truncated.

The *width* may be specified with an *\** instead of a numeric value. When the *\** is used, it means that the actual *width* value is supplied by the next argument in the *argument-list*. This argument must be of type *int* and precedes the argument being formatted.

For instance:

```
printf("%5s", "abc");           Writes "   abc" to stdout.
printf("%-6i", 22);            Writes "22    " to stdout.
printf("%07x", 12);            Writes "000000c" to stdout.
printf("%-*s", 9, "abc");       Writes "abc      " to stdout.
```

## ■ Precision

*Precision* is an optional component specified with a leading period character and a non-negative value. It controls the minimum number of digits to write (integers), the number of digits to the right of the decimal (floats) or the maximum number of characters to write (strings).

*Precision* can cause character truncation or rounding. If *precision* is specified as a zero and the numeric value is also zero, no characters are written.

Similar to *width*, the *precision* may be specified with an *\** instead of a numeric value. When the *\** is used, it means that the actual *precision* value is supplied by the next argument in the *argument-list*. This argument must be of type *int* and precedes the argument being formatted.

| <i>type</i>                     | <i>precision</i> Meaning                                                                                                                                              | Default                                                                                                                                |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| b<br>i<br>d<br>o<br>u<br>x or X | Specifies the minimum number of digits to print. When the number of digits in the <i>argument</i> is less than <i>precision</i> the value is zero-padded on the left. | When <i>precision</i> is omitted, specified as 0 or merely a period with no number following, the <i>precision</i> is set to 1.        |
| e or E                          | Specifies the number of digits to write after the decimal point. The last digit is rounded.                                                                           | The default <i>precision</i> is 6. When <i>precision</i> is 0 or merely a period with no number following, no decimal point is output. |
| f                               | Specifies the number of digits to write after the decimal point. The last digit is rounded. If a decimal point appears, at least one digit appears before it.         | The default <i>precision</i> is 6. When <i>precision</i> is 0 or merely a period with no number following, no decimal point is output. |
| g or G                          | Specifies the maximum number of significant digits to write.                                                                                                          | Six significant digits are printed with any trailing zeros truncated.                                                                  |
| c                               | <i>Precision</i> is ignored.                                                                                                                                          | The character is written.                                                                                                              |
| s                               | Specifies the maximum number of characters to write. Excess characters are not written.                                                                               | The entire string is written.                                                                                                          |

Table 7: printf Precision & Type Effects

### ■ Modifier

The *modifier* component is an optional field that specifies the size of the *argument*.

| <i>modifier</i> | Meaning                                                                                                                                                                                                                                                                                                                                                                                  | Default                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| h               | When used with an integer type (b, d, i, o, x or X), the argument is a short int. When used with type u, the argument is a short unsigned int.                                                                                                                                                                                                                                           | The default size for integers is long unless otherwise specified by a <code>#pragma int directive</code> . |
| l               | When used with an integer type (b, d, i, o, x or X), the argument is a long int. When used with type u, the argument is a long unsigned int. When used with a floating-point type (e, E, f, g or G), the argument is a double rather than a float.                                                                                                                                       |                                                                                                            |
| L               | <p>When used with a floating-point type (e, E, f, g or G), the argument is a long double.</p> <p>When used with the integer types b, d, u, x or X, the argument is a long long int (64-bit).</p> <p>When used with t type, the time portion contains millisecond value (HH:MM:SS.mmm).</p> <p>When used with p type, the argument must be a far-pointer and will be output as %X:%X.</p> |                                                                                                            |

Table 8: printf Modifier Codes

**Defaults:** When no *argument-list* is supplied, only the plain text in *format* is displayed on stdout.

**Conforms to:**                      **printf**    n ANSI    n DOS    n THEOS    n POSIX  
 The types 'b' and 't' are THEOS extensions to the ANSI standard.

**See also:**                      [cprintf](#), [fprintf](#), [sprintf](#), [vfprintf](#), [vprintf](#), [vsprintf](#)

**Example:**

```

#include <stdio.h>

main()
{
    int    col;

    printf("\n 1  \"%b\" ",123);           // integers
    printf("\n 2  \"%10b\" ",123);
    printf("\n 3  \"%-5i\" ",123);
    printf("\n 4  \"%5o\" ",123);
    printf("\n 5  \"% ,#o\" ",12345678);
    printf("\n 6  \"%x\" ",12345678);
    printf("\n 7  \"%X\" ",12345678);
    printf("\n 8  \"%#x\" ",87654321);
    printf("\n 9  \"%i\" ",1234);
    printf("\n10  \"%10i\" ",123);
    printf("\n11  \"%10.5i\" ",123);
    printf("\n12  \"%-12i\" ",1234567);
    printf("\n13  \"%- ,12i\" ",1234567);
    printf("\n14  \"%-+,12i\" ",1234567);
    printf("\n15  \"% +,12i\" ",1234567);

    printf("\n\n");
    printf("16  \"%This is test %i: %n",16,&col);
    printf("%*s",30-col,"");           // align on col 30
    printf("%s\n","More text at column 30");

    printf("\n17  \"%f\" ",123.456);      // floats
    printf("\n18  \"%e\" ",123.456);
    printf("\n19  \"%E\" ",123.456);
    printf("\n20  \"%g\" ",123.456);

    printf("\n21  \"% .2f\" ",1234.5678);
    printf("\n22  \"% ,.2f|\" ",123456.789);

    printf("\n\n23  \"%Lp\" ",&col);      // display far pointer

```

**Output:**

>sample

```
1 "1111011"
2 "    1111011"
3 "123  "
4 "  173"
5 "057,060,516"
6 "bc614e"
7 "BC614E"
8 "0x5397fb1"
9 "1234"
10 "          123"
11 "        00123"
12 "1234567  "
13 "1,234,567  "
14 "+1,234,567  "
15 " 1,234,567  "

16 "This is test 16:          More text at column 30"

17 "123.456000"
18 "1.234560e+02"
19 "1.234560E+02"
20 "123.456"
21 "1234.57"
22 "123,456.79"

23 "16:FFFFFFCE0"
```



---

**putch**

Writes one character to the console display.

```
#include <conio.h>
void putch ( char c )
```

---

*c*                      »    character to display

**Operation:**        Writes one character to the console display device.

**Errors:**            No errors are detected.

**Notes:**            If a console echo file is enabled, the character is first written to that file.

If console output is suppressed, no character is displayed on the console display device (but it might still be output to the console echo file).

Control characters are translated by the console's class code.

This function is used directly or indirectly by all console output functions.

This function outputs to the console device, not stdout. Redirecting stdout has no affect on this function.

**Conforms to:**                **putch**    q ANSI    n DOS    n THEOS    q POSIX

**See also:**            [at](#), [atstr](#), [conrdy](#), [\\_conrdy](#), [cputs](#), [fputc](#), [fputl](#), [fputs](#), [fputsn](#), [fputsnl](#), [fputw](#), [putc](#), [putchar](#), [fwrite](#), [puts](#), [putw](#), [write](#)

**Example:**

```
#include <stdio.h>
#include <conio.h>
#include <charset.h>
#include <ctype.h>

void main()
{
    char    password[9];
    char    c;
    int     len = 0;

    conmask("n");                // suppress input echo

    cputs("\nEnter password: ");

    while (len<8 && c!=CR)
    {
        c = getch();             // accept character
        if (isgraph(c))
        {
            password[len++] = c;  // save character
            putch('*');           // display as asterisk
        }
    }

    cprintf("\nPassword entered is: %s.",password);
}
```

---

**Output:**

```
>test

Enter password: *****
Password entered is: abcde

>
```

## putenv, \_putenv

Define or change the value of an environment variable.

```
#include <stdlib.h>
```

```
int putenv ( char * name_val )
```

```
int _putenv ( const char * name, const char * value )
```

---

*name* » pointer to string containing environment name

*name\_val* » pointer to string containing both the environment name and the new value

*value* » pointer to string containing new environment value

**Operation:**      **putenv**      The *name\_val* must be a pointer to a string with the form of:

name=value

*name* is the name of the environment variable to be changed or added and *value* is the new value for the variable. *name* is case-insensitive.

**\_putenv**      Similar to **putenv** except the *name* and *value* are separate fields.

**Returns:**      A zero is returned when the variable is changed successfully; otherwise, a non-zero is returned.

**Notes:**      You can “delete” an environment variable by setting its value to nothing. For instance:

```
putenv( "Name=" )
```

or

```
_putenv( "Name", "" )
```

Unlike the operation under Unix and DOS, the environment value is maintained even after exiting the program. The value of an environment variable is not changed unless specifically requested or when the user logs onto another account.

**Conforms to:**

|                |        |       |         |         |
|----------------|--------|-------|---------|---------|
| <b>putenv</b>  | q ANSI | n DOS | n THEOS | n POSIX |
| <b>_putenv</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**      [getenv](#)

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
    char    old_lib_name[256] = "LIBRARY=";

    strcat(old_lib_name, getenv("LIBRARY")); // save lib name
    putenv("LIBRARY=TEST.DATFILE");          // change library

    ...

    putenv(old_lib_name);                    // set it back
}
```

## putfddate

Change the last change date for a file.

```
#include <stdlib.h>
```

```
int putfddate ( const char * filename, time_t date )
```

---

*date* » new date/time for file

*filename* » pointer to string containing file description

**Operation:** The date/time stamp in the file directory entry of *filename* is changed to *date*.

**Returns:** A success/fail indicator. A zero return value indicates success; a non-zero return value indicates failure and [errno](#) is set to a value indicating the specific error.

**Conforms to:** **putfddate** q ANSI q DOS n THEOS q POSIX

**See also:** [getfddate](#), [getfiledate](#)

**putmsg**

Write program-defined message to `stderr`, with parameter substitution.

```
#include <stdlib.h>
void putmsg ( char * string, void * args )
```

---

*args*                   »   pointer to array of string pointers  
*string*                »   pointer to message string

**Operation:**       The text of *string* is written to `stderr`.

**Notes:**           Similar to messages from the `SYSTEM.TEOSnnn.MESSAGEn` file, the message *string* is displayed using parameter substitution. Parameter substitution codes are denoted by a digit surrounded by braces, as in:

"`\{1\}`" copied to "`\{2\}`".

The usage of the *args* argument depends upon the number and type of parameter substitution codes in the message text:

- ▶ A message with no substitution codes causes *args* to be ignored.
- ▶ A message with a single substitution code of `{0}` causes *args* to be interpreted as `char *args`, that is, as a pointer to a character string.
- ▶ A message with multiple substitution codes causes *args* to be interpreted as `char *args[]`, that is, as a pointer to an array of pointers to strings.

**Substitution Rules:**   Text in the message is copied to `stderr` as-is except for text inside of a pair of curly braces. When the text inside of the curly braces is:

- `{0}`   Replaced with the text pointed to by *args*.
- `{1}`   Replaced with the first text string in the array pointed to by *args*.
- `{n}`   Replaced with the  $n^{\text{th}}$  text string in the array pointed to by *args*.
- `{n?exp-list}`   The *exp-list* is a comma-separated list of values and assignment text strings in the form *value=text*. The  $n^{\text{th}}$  text string in the array pointed to by *args* is compared to each *value* in the expression list. When a match is found, the *text* is used as the replacement value.

The last item in the *exp-list* may be an asterisk. When the asterisk is encountered, the replacement text is the original value of the  $n^{\text{th}}$  text string in the array pointed to by *args*.

Note that the *text* in the *exp-list* may not contain spaces or commas.

**Conforms to:**        **putmsg**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**           [\\_errbot](#), [errmsg](#), [getmsg](#), [fperror](#), [perror](#), [syserr](#)

**Example:**

```
#include <stdlib.h>
#include <stdio.h>

char    msg1[] = "\nPlease enter your {0}: ",
        msg2[] = "The {1} you entered was \"{2}\", Okay? ",
        fldnames[][20] = {
            "name",
            "phone number"
        },
        fields[2][40];

void
main() {
    char    *a[2];
    int     i;

    load_yn();                                // get yes/no response chars

    for (i=0; i<2; ++i) {
        putmsg(msg1, fldnames[i]); // input prompt
        gets(fields[i]);           // get response
        a[0] = fldnames[i];
        a[1] = fields[i];
        putmsg(msg2,a);            // display input
        if (!yesno())              // confirm input
            --i;
    }
    putmsg("\nThank you.\n",NULL);
}
```

---

**Output:**

>test

Please enter your name: Thomas Jefferson  
The name you entered was "Thomas Jefferson", Okay? Y

Please enter your phone number: 415 555-1234  
The phone number you entered was "415 555-1234", Okay? Y

Thank you.

### puts

Outputs a string of characters to `stdout`, appending a new-line character.

```
#include <stdio.h>
void puts ( char * string )
```

---

*string*                    »    pointer to string of characters to output

**Operation:**        The string pointed to by *string*, up to but not including the terminating null character, is output to the standard output device, `stdout`. A new-line character is then output.

**Returns:**            Success/fail indicator. A zero return indicates success; a non-zero return value indicates an error occurred.

**Errors:**             It is unlikely that an error will occur unless the `stdout` has been redirected to a device other than the console or `stdout` has been closed.

**Defaults:**          Unless redirected, `stdout` is normally the console display.

**Conforms to:**                `puts`    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [cprintf](#), [fputc](#), [fputl](#), [fputs](#), [fputsn](#), [fputsnl](#), [fputw](#), [putc](#), [putchar](#), [printf](#)



## putw

Writes an integer data value to a file stream.

```
#include <stdio.h>
int putw ( unsigned short word, FILE * file )
```

---

*file*                   »   pointer to file's fcb

*word*                   »   integer data value

**Operation:**       Write the short integer *word* to *file*. No alignment of data is presumed. That is, exactly two bytes of data are written at the current file position.

**Returns:**         Zero return indicates success. A value of -1 may be returned indicating an error.

**Errors:**          A return of -1 indicates that the write was unsuccessful because the disk is full or *file* is not open for write.

**Restrictions:**   The putw and getw functions are provided for compatibility reasons only. These functions are not portable due to the fact that the size of an integer and the byte-ordering of an integer might vary from one implementation of C to another or from one computer to another. These functions should be restricted to work files that are created and used during one session.

**Conforms to:**               **putw**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**            [fputc](#), [fputl](#), [fputs](#), [fputsn](#), [fputsnl](#), [fputw](#), [putc](#), [putchar](#), [getw](#)

### **pwverify**

Compare a password with the password defined for an account definition.

```
#include <stdlib.h>
int pwverify ( char * acc_name, char * password )
```

---

*acc\_name*           »   pointer to account name string

*password*           »   pointer to password string

**Operation:**       The value of *password* is compared to the encrypted password saved in the account definition of *acc\_name*.

**Returns:**         A true/false value indicating whether or not the *password* matches the encrypted password. A non-zero return indicates true, a zero return value indicates false.

**Notes:**           Both *acc\_name* and *password* are case-insensitive.

If *acc\_name* has no password defined, then *password* must be a null string to match.

**Conforms to:**       **pwverify**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [get\\_acc\\_name](#), [read\\_acc](#), [sch\\_acc](#)

**Example:**

In the following example, the `pwverify` function is used in two separate operations. First, it is used to make sure that the operator knows the password for the current account. They might not know the password if they are using a console that someone else logged on for them.

The second instance of the `pwverify` function is testing to determine if the operator knows the password to another account. This other account could be one that is set up just for password-verification purposes and might be an account name that is never used for actual account-logon purposes.

```
#include <stdio.h>
#include <stdlib.h>
#include <acb.h>

void main()
{
    char    pw[256];    // allocated for typing mistakes, etc.

    conmask("n");      // do not echo password typed!

    printf("\nEnter password for current account: ");
    gets(pw);
    if (pwverify(get_acc_name(getuid()),pw))
        printf("Okay\n");
    else
    {
        printf("Mismatch!\n");
        exit(200);      // exit program with special code
    }

    printf("\nEnter password to verify: ");
    gets(pw);
    if (pwverify("security",pw))
        printf("Okay\n");
    else
    {
        printf("Mismatch!\n");
        exit(201);      // exit program with special code
    }
}
```

**qsort**

Perform a “quick sort” of an array of objects.

```
#include <stdlib.h> or <search.h>
```

```
void qsort ( const void * array, size_t count, size_t size, int (*compar)() )
```

---

|               |   |                                      |
|---------------|---|--------------------------------------|
| <i>array</i>  | » | pointer to array to be sorted        |
| <i>compar</i> | » | name of comparison routine to use    |
| <i>count</i>  | » | number of members remaining in array |
| <i>size</i>   | » | size of each member in array         |

**Operation:** Perform a “quick sort” of an array starting at *array* for *count* elements, each of which is *size* bytes in length. The function *compar* is used to determine the relative order of any two items in *array*. The *array* is overwritten with the re-ordered elements.

**Notes:** The *compar* routine must be a function that uses a declaration similar to:

```
int compar(const void *memb1, const void *memb2)
```

*memb1* and *memb2* are two pointers to elements of *array*. *compar* must return an integer that is:

```
< 0 when memb1 < memb2
= 0 when memb1 = memb2
> 0 when memb1 > memb2
```

The elements of *array* are sorted in ascending order, as defined by the *compar* routine. To sort the array in decreasing order, reverse the meaning of “>” and “<” in the comparison function.

The array may be sorted by a field other than the first field of each member. The sorting criteria is exclusively controlled by the *compar* function.

**Defaults:** There is no default comparison function. However, to sort an array of strings, the [strcmp](#) function can be used.

**Conforms to:**                      **qsort**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                      [bsearch](#), [lsearch](#), [lfind](#), [sort](#)

**Example:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <crt.h>

void display_words(int);

char words[100][20]; // storage for 100 words

void
main()
{
    char word[20],
        * found;
    int nbr_words = 0;

    crt(CLEAR); // clear the screen

    while (nbr_words <= 100) {
        // get a word from operator
        at(0,20); printf("Enter your word: ");
        crt(EOL);
        gets(word);
        crt(EOS);
        if (!word[0]) // null string = end
            break;
        found = bsearch(word, words, nbr_words, 20, strcmp);
        if (!found) { // not duplicate...add
            strcpy(words[nbr_words++], word);
            qsort(words, nbr_words, 20, strcmp);
            display_words(nbr_words);
        }
        else {
            at(0,23);
            printf("Word is already in list.\b");
        }
    }
}

```

## 506 qsort

---

```
void
display_words(int count)
{
    int  col,
          row,
          rows,
          idx;
    div_t rows_div;

    crt(CLEAR);

    rows_div = div(count, 4);    // maximum words per column
    rows = rows_div.quot + (rows_div.rem ? 1 : 0);

    for (col=idx=0; col<4; ++col) {
        for (row=0; row<rows && idx<count; ++row, ++idx) {
            at(col*20, row);
            printf(words[idx]);
        }
    }
}
```

---

### Output:

>example

|          |          |           |          |
|----------|----------|-----------|----------|
| eight    | four     | seven     | thirteen |
| eighteen | fourteen | seventeen | three    |
| eleven   | nine     | six       | twelve   |
| fifteen  | nineteen | sixteen   | twenty   |
| five     | one      | ten       | two      |

Enter your word: twenty-one

***\_quitoff, \_quition***

Disable or enable the operator program-cancel operation.

```
#include <sc.h>
unsigned short _quitoff ( void )
void _quition ( void )
```

- Operation:**

***\_quitoff***

Disable the program-cancel key (Break,Q). The prior status is returned as the value of the function.

***\_quition***

Enable the program-cancel key. If it was disabled prior to this function call and the operator used the program-cancel key while it was disabled, the action programmed for SIGQUIT is performed (see [signal](#) on page 573).
- Returns:**

***\_quitoff***

True/false value. A zero indicates that the program-cancel key was already disabled; a non-zero indicates that it was enabled.

**Notes:** The proper method of disabling the program-cancel key in a program is to use these two functions, rather than using the [signal](#) function to ignore SIGQUIT actions.

```
quit_status = _quitoff();
...          // code requiring no cancels
if (quit_status)
    _quition();
```

|                     |                        |        |       |         |         |
|---------------------|------------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b><i>_quitoff</i></b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b><i>_quition</i></b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [signal](#)

**raise**

Raise a signal interrupt just as if the interrupt had occurred.

```
#include <signal.h>
int raise ( int signal )
```



---

*signal*                   »   signal value

**Operation:**       Raises the interrupt *signal*, thus causing the routine programmed to handle the event to be invoked. Routines are programmed with the [signal](#) function.

**Returns:**           A success/fail value. A zero return indicates success.

**Notes:**            The possible values for *signal* are defined in the `SIGNAL.H` file:

| <i>signal</i>                      | <i>Meaning</i>                                                                                           |
|------------------------------------|----------------------------------------------------------------------------------------------------------|
| SIGABRT <sup>A</sup>               | Abnormal program termination. Maps into SIGTERM.                                                         |
| SIGALRM                            | <a href="#">alarm</a> or <a href="#">msalarm</a> timer runout.                                           |
| SIGBOUND                           | Array-bound error used by MultiUser BASIC.                                                               |
| SIGCAN                             | Synonym to SIGINT signal.                                                                                |
| SIGCRIT                            | Input/output critical error. Maps into SIGTERM.                                                          |
| SIGDIV0                            | Divide by zero.                                                                                          |
| SIGFPE <sup>A</sup>                | Floating-point error. Maps into SIGTERM.                                                                 |
| SIGILL <sup>A</sup>                | Illegal instruction. Maps into SIGTERM.                                                                  |
| SIGINT <sup>A</sup>                |  entered by operator. |
| SIGOVF                             | Mathematical overflow or underflow occurred.                                                             |
| SIGQUIT                            | Synonym to SIGTERM signal.                                                                               |
| SIGSEGV <sup>A</sup>               | Memory segment violation (GP13 or SP12). Maps into SIGTERM.                                              |
| SIGTERM <sup>A</sup>               |  entered by operator. |
| A Conforms to ANSI specifications. |                                                                                                          |

**Conforms to:**                    **raise**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                    [alarm](#), [msalarm](#), [signal](#)



rand, srand

Generate a pseudorandom number or initialize the pseudorandom number generator.

```
#include <stdlib.h>
int rand ( void )
void srand ( unsigned seed )
```

---

*seed*                   »   seed value for pseudorandom numbers

**Operation:**        **rand**           Return the next pseudorandom integer in the range of 0 to RAND\_MAX.

**srand**         Set the starting point for generating pseudorandom numbers.

**Returns:**           **rand**           The next pseudorandom integer.

**Notes:**            To reinitialize the pseudorandom number generator, use a *seed* value of 1.

The pseudorandom number generator will return the same number each time that the program is executed unless it is seeded with a different value each time that the program is executed. The easiest method of seeding the generator with a random value is to use the [cpu\\_time](#) function.

To use random numbers in a range smaller than RAND\_MAX, use the modulo operator with the desired upper limit.

**Defaults:**        The pseudorandom number generator is initially seeded with 1.

|                     |              |        |       |         |         |
|---------------------|--------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>rand</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>srand</b> | n ANSI | n DOS | n THEOS | n POSIX |

## 510 *rand, srand*

---

**Example:** This example generates random numbers in the range of 0–51, suitable for picking one card from a deck of standard playing cards.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void
main()
{
    int    i;

    srand(cpu_time(NULL) % RAND_MAX);    // seed generator with
   // current time-of-day

    for (i=0; i<10; ++i)
        printf("\nRandom number %i = %2d",i, rand() % 52);
}
```

---

### **Output:**

>example

```
Random number 0 = 34
Random number 1 =  0
Random number 2 = 39
Random number 3 = 24
Random number 4 = 15
Random number 5 = 19
Random number 6 = 17
Random number 7 = 22
Random number 8 = 12
Random number 9 = 28
```

## read

Read a stream of bytes from a file.

```
#include <io.h> or <handle.h>
size_t read ( int fhandle, const void * buffer, size_t len )

#include <sc.h>
size_t _read ( FILE * file, const void _far * buffer, size_t len )
```

---

|                |   |                                  |
|----------------|---|----------------------------------|
| <i>buffer</i>  | » | pointer to storage for data read |
| <i>fhandle</i> | » | file handle or number            |
| <i>file</i>    | » | pointer to file's fcb            |
| <i>len</i>     | » | length of data to read, in bytes |

**Operation:** Starting at the current position pointer of *fhandle* or *file*, *len* number of bytes are read from the file and stored in *buffer*. The current position pointer is incremented for the number of bytes read.

**Returns:** The number of bytes actually read is returned. Normally this is the same as the number of bytes requested unless the end-of-file was encountered or some other type of file error occurred.

A return of 0 indicates an attempt to read while at end-of-file.

A return of -1 indicates that the read was unsuccessful because the location is locked by another user and the lock-wait-time elapsed (see below).

**Errors:** When a file error occurs, the return value is set to -1 and [errno](#) is set to EBADF.

**Notes:** The only difference between `read` and `_read` is that `read` uses a file handle argument and `_read` uses a pointer to a file control block. `read` is the DOS and UNIX-compatible form of the operation.

The principle difference between these two functions and the [fread](#) function is that [fread](#) reads from a file's input buffer if available and the `read` and `_read` read directly from the file.

New-line characters are treated as part of the data to be read. They are not converted to string terminators.

If *file* or *fhandle* is open with "r+" access, these functions honor any record or file lock placed by another user using the [filelock](#) function mechanism.

If the read is not immediately successful because the location is locked by another user these functions may wait for awhile to see if the lock is cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

| Value    | Meaning                                    |
|----------|--------------------------------------------|
| 0        | Wait until region is not locked (default). |
| 1        | Wait for 50 milliseconds.                  |
| <i>n</i> | Wait for <i>n</i> seconds.                 |

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the read request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`. This item can be examined with:

```
Scr->lock_pid
```

**Restrictions:** *buffer* must point to an area of at least *len* bytes.

These functions read data directly from the file, bypassing any input buffers associated with the file. Do not mix `read` and `_read` with `fread` on the same file.

|                     |              |        |       |         |         |
|---------------------|--------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>read</b>  | q ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>_read</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [fgetc](#), [fgetl](#), [fgets](#), [fgetsn](#), [fgetw](#), [getc](#), [getchar](#), [fread](#), [lreadk](#), [lreadn](#), [readk](#), [readn](#), [readp](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

void
main()
{
    int    data;
    struct {
        char    name[41];
        int     value;
    } record;

    if (!(data=open("data.file", O_RDONLY)))
        exit(fperror());

    lseek(data, -(sizeof record), SEEK_END); // position last rec
    read(data, &record, sizeof record);      // read last record

    printf("The last record is:\n");
    printf("%s, %d\n", record.name, record.value);

    close(data);
}
```

## read\_acc, sch\_acc

Read an account's complete definition or find an entry in an account definition that was read previously.

```
#include <acb.h>
```

```
void * read_acc ( char * acc_env, char * acc_name )
```

```
void * sch_acc ( char * acc_name, char * label )
```

---

*acc\_env* » pointer to account definition storage

*acc\_name* » pointer to account name string

*label* » pointer to account definition item label string

**Operation:** These two functions operate as a set. The `read_acc` reads an account definition into memory and the `sch_acc` searches that definition for a specific entry.

**read\_acc** Using the `SYSTEM.TEOS32.ACCOUNT` file, the complete account definition of *acc\_name* is located and read into the buffer *acc\_env*. Each item in the account definition is converted to plain text and copied to the *acc\_env* buffer in the form:

LABEL=value

The label name is always uppercase. Each line in *acc\_env* is terminated with a null and the last line is followed by null string.

All of the non-default switches and environment variable definitions for the *acc\_name* are copied to the *acc\_env* buffer.

**sch\_acc** Search the *acc\_env* buffer for a line starting with *label*=

**Returns:** **read\_acc** A pointer to *acc\_env*. When *acc\_name* cannot be found, a NULL pointer is returned.

**sch\_acc** A pointer to the location in *acc\_env* following the equal sign of LABEL=value. When *label* cannot be found, a NULL pointer is returned.

**Notes:** The values of *acc\_name* and *label* are case-insensitive.

The account definition read by the `read_acc` function includes all of the fields in the account file, including the password field if defined. The password field is always encrypted and should not be displayed because the encrypted characters are not generally displayable. They may cause problems on the display screen.

These functions are generally only used when the accounting file is used for purposes other than merely logging onto accounts. Normally, the information in the account definition is accessed with the `getenv` function, which gets user environment values that are often set by logging onto an account.

**Restrictions:** *acc\_env* should be allocated with a size sufficient to store all of the definitions in the account. Generally, 1024 should be adequate but may need to be larger depending upon the definitions used in your accounting file.

## 514 *read\_acc, sch\_acc*

---

|              |                 |        |       |         |         |
|--------------|-----------------|--------|-------|---------|---------|
| Conforms to: | <b>read_acc</b> | q ANSI | q DOS | n THEOS | q POSIX |
|              | <b>sch_acc</b>  | q ANSI | q DOS | n THEOS | q POSIX |

See also: [find\\_acc](#), [get\\_acc\\_name](#), [getenv](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <acb.h>

void main()
{
    char    acc_definition[1024];
    char    operator[9];
    char    name[40] = "\"your first name\"",
            boss[50] = "\"boss's name\"",
            coname[50] = "\"company name\"";
    char    *ptr;

    do {
        printf("\nEnter operator initials: ");
        scanf("%8s",operator);
    } while (read_acc(acc_definition, operator)==0);

    if (ptr=sch_acc(acc_definition, "first_name"))
        strcpy(name, ptr);
    if (ptr=sch_acc(acc_definition, "supervisor"))
        strcpy(boss, ptr);
    if (ptr=sch_acc(acc_definition, "company"))
        strcpy(coname, ptr);

    printf("\nHello %s. You work for %s.",name,coname);
    printf("\nHow nice. Your supervisor is %s.",boss);
}
```

---

### Output:

>example

Enter operator initials: cpw

Hello Chris. You work for THEOS Software Corporation.  
How nice. Your supervisor is Tom Jones.

>

## readk, readn, readp

Reads records from indexed or keyed organization files.

```
#include <stdio.h>
```

```
unsigned short readk ( FILE * file, void _far * key, void * record )
```

```
unsigned short readn ( FILE * file, void _far * key, void * record )
```

```
unsigned short readp ( FILE * file, void _far * key, void * record )
```

---

*file* » pointer to open file's fcb

*key* » pointer to character string

*record* » pointer to record storage buffer

**Operation:** Each of these functions reads a single record from a file opened as a keyed or indexed organization file.

**readk** *file* is searched for a record whose record key matches the value pointed to by *key*. When the record is located, the record associated with *key* is read and saved in *record*.

**readn** Using the current record pointer for *file*, the next record in the file is read. The key is saved in *key* and the record is saved in *record*.

The next record in a keyed organization file is the next physical record in the file from the last record read. The next record in an indexed organization file is the next record whose key logically follows the key of the last record read.

**readp** Using the current record pointer for *file*, the prior or previous record in the file is read. The key is saved in *key* and the record is saved in *record*.

This function cannot be used when the file's organization is keyed.

Unless the file was opened with the multilock access option, any records previously locked in this file by this user are unlocked.

**Returns:** The number of bytes read (actual length of record read).

A zero return value indicates that the record was not found in the file.

A return value of -1 indicates that the read was unsuccessful because the record is locked by another user (see below).

**Errors:** The meaning of a "record not found" condition depends upon the function used and the organization of the file.

**readk** The specific record requested does not exist in the file. For indexed files the read-next and read-previous record pointers are initialized to the next and previous logical record as if the record had been found. These read-next and read-previous record pointers can then be used by the **readn** and **readp** functions to get these next or previous records.

|              |                                                                                                                                                     |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>readn</b> | Keyed organization: The file pointer is at end-of-file. Indexed organization: There are no more records that logically follow the last record read. |
| <b>readp</b> | Indexed organization: There are no more records that logically precede the last record read.                                                        |

**Notes:** If *file* is opened with access mode “r+,” is in use by another user and that user has the desired record locked, these functions wait for that lock to be released before reading the record. When successfully read, it is automatically locked.

If the read is not immediately successful because the record is locked by another user, these functions may wait awhile to see if the lock is cleared by the other user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

| Value    | Meaning                                    |
|----------|--------------------------------------------|
| 0        | Wait until region is not locked (default). |
| 1        | Wait for 50 milliseconds.                  |
| <i>n</i> | Wait for <i>n</i> seconds.                 |

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the read request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`.

If *file* is not opened with access mode “r+,” then these functions do not check to see if the desired record is locked by another user and the record is not locked by this user. See also “Automatic Record-Locking:” on page 224.

When using the `readk` function, the key should be padded with binary zeros to the length of the allocated key length for the file. Since key fields may contain embedded binary zeros, the `readk` function may try to read a different record than expected if the key is not padded with zeros.

**Defaults:** When a keyed file is first opened, the record pointer is positioned to the start of the file. When an indexed file is first opened, the record pointer is positioned as if it pointed to the record preceding the null key record.

**Restrictions:** The areas pointed to by *key* and *record* should be allocated to at least the same size as the allocated key and record lengths for the file.

|                     |              |        |       |         |         |
|---------------------|--------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>readk</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>readn</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>readp</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [deletk](#), [fopen](#), [lreadk](#), [lreadn](#), [unlock](#), [writek](#)



**Example:** This example examines an indexed file. The operator enters a match-key and the program positions to that key's location in the file. Then, using the `readn` function, subsequent records are read from the file until a record is read whose key does not start with the match-key.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char  key[20],
      record[256],
      buf[80];
FILE  * ixfile;
int    len;

void
main()
{
    ixfile = fopen("test.file", "ri"); // open database

    if (!ixfile)                        // failure?
        exit(fperror());

    while (1) {
        printf("\nEnter key: ");      // prompt
        gets(buf);
        if (!(len = strlen(buf)))      // empty?
            break;
        strncpy(key, buf, 20);         // copy buf to key

        if (readk(ixfile, key, record) > 0) // read 1st record
            printf("%-20s %s\n", key, record);

        while (readn(ixfile, key, record) > 0) {
            if (memcmp(key, buf, len)) // match?
                break;                // no, try another
            printf("%s-20x %s\n", key, record);
        }
    }
    fclose(ixfile);                    // close file
}
```

**realloc**

Resize an existing memory buffer that was allocated with [calloc](#) or [malloc](#).

```
#include <stdlib.h> or <malloc.h>
void * realloc ( void * ptr, size_t size )
```

---

|             |   |                                         |
|-------------|---|-----------------------------------------|
| <i>ptr</i>  | » | pointer to existing allocated memory    |
| <i>size</i> | » | new size for allocated memory, in bytes |

**Operation:** The memory buffer specified by *ptr* is reallocated to the new *size*. If the buffer can not be resized in-place, a new buffer is allocated, the contents of the existing buffer are moved to the new buffer, and the old buffer is freed.

**Returns:** A pointer to the new or resized buffer.

**Errors:** If there is insufficient memory available, a NULL pointer is returned.

**Notes:** If *ptr* is a NULL pointer, this function performs an [malloc](#) operation.

If *ptr* is not NULL and *size* is zero, this function performs a [free](#) operation.

If *size* is less than the current size of the memory buffer, data is lost.

Because the heap space is dynamic, it is unlikely that there will be insufficient memory because the data segment will grow if necessary. However, always test the return value from the allocation routines to make sure that memory was actually allocated. If the return is not tested and a NULL pointer is returned, using that NULL pointer will cause a memory access error.

Use the [free](#) function to release memory allocated with this function or use the [calloc](#) or the [malloc](#) functions to reuse the memory.

There is a danger that your original memory pointer will be lost if there is insufficient memory to grow it. For example,

```
ptr = realloc(ptr, strlen(ptr)+500);
```

If there was not enough memory to satisfy the request, *ptr* will have a NULL value.

**Conforms to:** [realloc](#) n ANSI n DOS n THEOS n POSIX

**See also:** [\\_alloca](#), [calloc](#), [free](#), [\\_getmem](#), [\\_putmem](#), [malloc](#), [max\\_alloc](#), [mem\\_avail](#), [tot\\_alloc](#)

**Example:**

```
#include <stdlib.h>

void
main()
{
    union {
        double * numbers;
        char * chars;
    } storage;

    if (!(storage.numbers = calloc(200, sizeof (*storage.numbers))))
        errmsg(3, NULL);          // insufficient memory
    ...

    // finished with numbers, reuse for string

    if (!(storage.chars=realloc(storage.numbers, 1000)))
        syserr(16, 3, NULL);
    ...
}
```

**recv, recvfrom**

Receives data from a socket or receives a datagram and stores the source address.

```
#include <socket.h>

int recv ( SOCKET s, void * buffer, int len, int flags )

int recvfrom ( SOCKET s, void * buffer, int len, int flags, SOCKADDR * from,
               int * fromlen )
```

---

|                |   |                                                |
|----------------|---|------------------------------------------------|
| <i>buffer</i>  | » | pointer to storage location                    |
| <i>flags</i>   | » | value is not used                              |
| <i>from</i>    | » | pointer to location for storing source address |
| <i>fromlen</i> | » | pointer to size of the from buffer             |
| <i>len</i>     | » | length of buffer                               |
| <i>s</i>       | » | socket number                                  |

**Operation:**      **recv**      This function is used on connected datagram or stream sockets specified by the *s* parameter and is used to read incoming data.

For sockets of type `SOCK_STREAM`, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option `SO_OOBINLINE`) and out-of-band data is unread, only out-of-band data will be returned.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the excess data is lost, and `recv` returns the error `TSAEMSGSIZE`.

**recvfrom**      This function is used to read incoming data on a (possibly connected) socket and capture the address from which the data was sent.

For sockets of type `SOCK_STREAM`, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in-line reception of out-of-band data (socket option `SO_OOBINLINE`) and out-of-band data is unread, only out-of-band data will be returned. The *from* and *fromlen* parameters are ignored for `SOCK_STREAM` sockets.

For datagram sockets, data is extracted from the first enqueued datagram, up to the size of the buffer supplied. If the datagram is larger than the buffer supplied, the buffer is filled with the first part of the message, the excess data is lost, and `recvfrom` returns the error code `TSAEMSGSIZE`. If *from* is non-zero, and the socket is of type `SOCK_DGRAM`, the network address of the peer which sent the data is copied to *from*. The value pointed to by *fromlen* is initialized to the size of this structure, and is modified on return to indicate the actual size of the address stored there.

**Returns:** The number of bytes received. If the connection has been closed, it returns zero. When an error occurs, a value of `SOCKET_ERROR` is returned and a specific error code may be retrieved by calling [TSAGetLastError](#).

**Errors:** When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

| <i>Code</i>                  | <i>Meaning</i>                                                                                                 |
|------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>TSAECONNABORTED</code> | The virtual circuit was aborted due to time-out or other failure.                                              |
| <code>TSAECONNRESET</code>   | The virtual circuit was reset by the remote side.                                                              |
| <code>TSAEFAULT</code>       | The <i>fromlen</i> argument was invalid: The <i>from</i> buffer was too small to accommodate the peer address. |
| <code>TSAEINPROGRESS</code>  | A blocking THEOS Sockets operation is in progress.                                                             |
| <code>TSAEINVAL</code>       | The socket has not been bound with <code>bind</code> .                                                         |
| <code>TSAEMSGSIZE</code>     | The datagram was too large to fit into the specified buffer and was truncated.                                 |
| <code>TSAENETDOWN</code>     | The THEOS Sockets implementation has detected that the network subsystem has failed.                           |
| <code>TSAENOTCONN</code>     | The socket is not connected.                                                                                   |
| <code>TSAENOTSOCK</code>     | The descriptor is not a socket.                                                                                |
| <code>TSAEOPNOTSUPP</code>   | <code>MSG_OOB</code> was specified, but the socket is not of type <code>SOCK_STREAM</code> .                   |
| <code>TSAEWOULDBLOCK</code>  | The socket is marked as non-blocking and the receive operation would block.                                    |

**Notes:** If no incoming data is available at the socket, the `recv` and `recvfrom` calls wait for data to arrive unless the socket is non-blocking. In this case a value of `SOCKET_ERROR` is returned, with the error code set to `TSAEWOULDBLOCK`. The [select](#) call may be used to determine when more data arrives.

If the socket is of type `SOCK_STREAM` and the remote side has shut down the connection gracefully, a `recv` or `recvfrom` will complete immediately with 0 bytes received. If the connection has been abortively disconnected, a `recv` or `recvfrom` will fail with the error `TSAECONNRESET`.

The flags field value is not used in the THEOS sockets API.

**Conforms to:**

|                       |        |       |         |         |
|-----------------------|--------|-------|---------|---------|
| <code>recv</code>     | q ANSI | q DOS | n THEOS | q POSIX |
| <code>recvfrom</code> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [ioctlsocket](#), [send](#), [sendto](#), [socket](#)

**Example:** See the example for the function [socket](#).

**relock, recunlock**

Lock or unlock a record or location in a file.

```
#include <stdio.h>
unsigned short relock ( FILE * file, fpos_t location )
void recunlock ( FILE * file, fpos_t location )
```

---

*file*                   »   pointer to open file's fcb  
*location*               »   location in file

- Operation:**       **relock**       Tries to place an exclusive lock at the *location* of *file*.
- recunlock**   Release any lock set by this task at the *location* of *file*.
- Returns:**       **relock**       Success/fail code. A zero return indicates success; a non-zero return indicates that it did not place a lock on the location.
- Errors:**        The usual reason for a failure to lock a location in a file is that another task already has that location locked.
- Notes:**        The relock function tries to lock the location in *file* and, if successful, it returns immediately. If the lock is not immediately successful, the function may wait awhile to see if the lock is cleared by another user. The `scr.lock_wait` item controls the length of time of this wait. This item may have one of three values:

| Value    | Meaning                                  |
|----------|------------------------------------------|
| 0        | Wait until region is lockable (default). |
| 1        | Wait for 50 milliseconds.                |
| <i>n</i> | Wait for <i>n</i> seconds.               |

The value in `scr.lock_wait` can be examined by using the reference:

```
Scr->lock_wait
```

For instance, to set a wait limit of five seconds:

```
Scr->lock_wait = 5;
```

If the program is used on a THEOS 32 Version 4 or later system, and the lock request fails with a return value of -1 (region already locked by a user), the process number (base 1) of the user locking the region is set in `scr.lock_pid`.

The `relock` function uses the RLT (Record Lock Table) to record its locks. This is the same table that is used by the automatic record-locking mechanism used for direct, indexed and keyed files. It is **not** the same table used by the `filelock` function, and locks set with the `filelock` function are not recognized by `relock`.

The RLT is tested by all direct, indexed and keyed record-access functions prior to accessing the file. Therefore, any lock placed on a portion of a direct, indexed and keyed organization file with the `relock` function will

prevent all other users from writing to that portion of the file with the [delet](#), [ldelet](#), [lwrite](#) or [write](#) functions. If another user has the file open with “r+” access (update), it will also prevent other users from reading that portion of the file with the [lread](#), [lreadn](#), [read](#), [readn](#), [readp](#) functions.

A location successfully locked by the `reclock` function does not prevent another task from accessing that location with the byte or stream input/output functions. It only prevents other tasks from placing a lock on the location with `reclock` or with automatic record-locking. In other words, record or location-locking with the `reclock` function requires all other tasks to cooperate by using the `reclock` function before each access attempt to the file.

Locks on a file are removed with the `recunlock` function, with the [unlock](#) function or by closing the file.

**Restrictions:** These functions are suitable for locking records or locations in stream files only. If you are updating a direct, indexed or keyed organization file with byte or stream input/output functions, it is best if you lock the entire file when it is opened.

|                     |                  |        |       |         |         |
|---------------------|------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>reclock</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>recunlock</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [close](#), [fclose](#), [fcloseall](#), [filelock](#), [fopen](#), [lockf](#), [locking](#), [unlock](#)

**regcmp, regex**

Compile a regular expression search pattern or search a string using a compiled pattern.

```
#include <regex.h>
char * regcmp ( char * pattern )
char * regex ( char * expr, char * subject )
```

---

|                |   |                                      |
|----------------|---|--------------------------------------|
| <i>expr</i>    | » | pointer to compiled search pattern   |
| <i>pattern</i> | » | pointer to search pattern to compile |
| <i>subject</i> | » | pointer to object string to match    |

**Operation:**      **regcmp**      Make a copy of *pattern*, compiling any changes necessary and return a pointer to that copy. For a description of the patterns that may be specified, refer to the description of “Regular Expressions” on the following page.

**regex**      Using the compiled pattern specified by *expr*, search the *subject* string looking for the first occurrence of a substring that matches the compiled pattern.

**Returns:**          **regcmp**      A pointer to the compiled version of *pattern*.

**regex**      A pointer that is either a NULL pointer indicating that the search was unsuccessful or a pointer to the character following the first occurrence in *subject* of the pattern requested. This pointer could be used for another call to **regex** to search for the next occurrence.

A secondary value “returned” by this function is in the external variable *loc1*. When a match is found, this variable is set to point to the start of the first occurrence of the pattern in *subject*. To use this variable you must declare:

```
extern char * loc1;
```

**Notes:**            **regcmp**      Although this library function does not actually change *pattern*, you should still use it to compile the pattern before using **regex**. This will ensure that your program is portable to other implementations of C and to possible future enhancements to this implementation.

The copied pattern is not local to the **regcmp** function and can safely be used without making an additional copy of it in your program.

**Regular Expressions:**      *pattern* is actually a string expression called a **regular expression**. A regular expression is a sequence of characters consisting of **literal characters** and **metacharacters**.

**Literal characters** are characters that match in a one-for-one relationship with the text in a string. For instance, the search pattern “abc” is composed solely of literal characters. This pattern matches only when the string contains a sequence of three characters that exactly equals “a,” “b,” and “c.”



In addition to the normal ASCII characters, you may specify certain control characters as literal characters. These control characters must be specified with the following codes:

| <i>Specification</i> | <i>Meaning</i>         |
|----------------------|------------------------|
| \a                   | BELL                   |
| \b                   | Backspace              |
| \f                   | Form-feed              |
| \n                   | New-line               |
| \r                   | Return                 |
| \t                   | Tab                    |
| \v                   | Vertical tab           |
| \0                   | Null                   |
| \'                   | Single-quote character |
| \"                   | Double-quote character |

Table 9: Regular Expression Control Character Specification

A **metacharacter** is a character or group of characters that represents something else. The wild cards allowed for file specifications by most of the THEOS commands are examples of metacharacters.

The following table shows all of the metacharacters allowed in regular expressions.

| <i>Specification</i> | <i>Meaning</i>                                                                |
|----------------------|-------------------------------------------------------------------------------|
| \^                   | Anchor to start-of-line.                                                      |
| \\$                  | Anchor to end-of-line.                                                        |
| \. or \?             | Any character matches.                                                        |
| \@                   | Any letter matches (A–Z and a–z).                                             |
| \#                   | Any digit matches (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0).                          |
| \\                   | Metacharacter escape. This is how a backslant literal character is specified. |
| \[                   | Start character set.                                                          |
| \]                   | End character set.                                                            |
| \{                   | Start general repeat specification.                                           |
| \}                   | End general repeat specification.                                             |
| \*                   | Repeat zero or more times.                                                    |
| \+                   | Repeat one or more times.                                                     |

Table 10: Regular Expression Metacharacters

### ■ Regular Expression Anchors

The first two metacharacters are called **anchors**. These anchor other text to the beginning or end of the string searched. For instance, the pattern

```
"\^The"
```

means that “The” is searched for but only when it occurs at the beginning of the string. Similarly,

```
"the\$"
```

is a pattern that means search for “the” at the end of the string. Note that this last pattern will not find “the” if there is a space at the end of the string.

### ■ Regular Expression Wild Cards

There are three “wild card” metacharacters that allow you to search for matches on any character ( `\.` ), any letter ( `\@` ) and any digit ( `\#` ). For example, the pattern:

```
"THEOS\#86"
```

will match any sequence of characters starting with “THEOS,” followed by any single digit, followed by an “8” and a “6.” Thus, it will match “THEOS186,” “THEOS286,” “THEOS386,” “THEOS486,” *etc.*

### ■ Regular Expression Character Sets

Searching for one character of a set of characters is done by specifying a ***character set***. For instance, to search for a hexadecimal digit, you would specify a pattern containing:

```
"\[0123456789ABCDEFabcdef\]"
```

This pattern specifies a match on any character that is either a digit, an uppercase “A” through “F” or a lowercase “a” through “f.”

As a convenience, when specifying character sets, there are two characters that have special meaning. The hyphen or dash character ( `-` ), when specified between two characters, indicates a ***character set range***. For instance, the above pattern could have been specified with:

```
"\[0-9A-Fa-f\]"
```

which means the set of characters “0” through “9,” “A” through “F” and “a” through “f.” Ranges specified with the hyphen character refer to ranges in the ASCII collating sequence.

The hyphen does not have a special meaning when it occurs at the beginning or end of a character set specification. Thus, the pattern:

```
"\[ -0-9\]"
```

means the set of characters “dash” and the digits “0” through “9.”

The circumflex ( `^` ) is the other character that has special meaning when used in a character set specification. When the circumflex is used at the start of the character set specification, it means that it is an ***exception set***. That is, it is a specification of characters that do not match.

For instance, the pattern:

```
"\[ ^A-Z\]"
```

means a match on any character except uppercase letters. When the circumflex is used in a position other than at the start, it is merely a character included in the set.

### ■ Regular Expression Repeat Counts

A search pattern may be repeated by using the repeat metacharacters “\{” and “\}” to enclose the minimum and maximum repeat counts. For instance, the pattern:

```
" \@{\4,6\}\[ .\]"
```

means that you want a match on a space, followed by four to six letters, followed by a space or period.

It is not necessary to specify both the minimum and maximum repeat count values. A missing minimum value uses the default value of one for the minimum; a missing maximum value uses infinity for the maximum. Thus, a pattern repeat count of “\{5\}” means five or more occurrences; a pattern of “\{,5\}” means one to five occurrences.

As a convenience, there are two special metacharacters for specifying common repeat counts. The metacharacter “\\*” is a shorthand specification for the repeat count “\{0\}” – meaning zero or more occurrences. The metacharacter “\+” is a shorthand specification for the repeat count “\{1\}” – meaning one or more occurrences.

**Restrictions:** Because `loc1` is a variable defined by `regex`, it makes this a reserved identifier whenever a program references the `regex` function. Do not use this identifier as a program-defined variable if the `regex` function is used.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>regcmp</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>regex</b>  | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [\\_fmemchr](#), [\\_fstrchr](#), [\\_fstristr](#), [\\_fstrchr](#), [\\_fstrstr](#), [strchr](#), [stristr](#), [strstr](#)

### Example:

```
#include <stdio.h>
#include <string.h>
#include <crt.h>
#include <regex.h>

extern char * loc1;

void
main()
{
    char    pattern[256];
    char    *xpattern;
    char    object[256];
    char    *i,
            work[256],
            *j;

    printf("\nEnter search pattern: ");
    gets(pattern);

    xpattern = regcmp(pattern);          // compile the pattern

    printf("\nEnter object string: ");
    gets(object);

    i = object;

    printf("\n%s\n", object);

    while (i = regex(xpattern, i)) {      // repeat while found
        printf("\r");                      // position to beginning
        for (j=object; j<loc1; ++j)
            crt(RIGHT);                    // move cursor to char pos
        printf("^");                       // mark it
    }
}
```

---

### Output:

The following example locates all of the words in the string that are four or more letters in length.

>example

Enter search pattern: [A-Za-z]{4,}

Enter object string: Now is the time for all good men to come to the aid of their country.

Now is the time for all good men to come to the aid of their country.

>

## release\_dma, reserve\_dma

Reserve or release a dual-ported (direct memory access) memory region.

```
#include <dma.h>
void release_dma ( struct DMA_BLOCK * block )
void reserve_dma ( struct DMA_BLOCK * block )
```

---

*block*                   »   structure describing the dual-ported memory region

**Operation:**       **release\_dma**   The dual-ported memory region described in *block* is released.

**reserve\_dma**   The dual-ported memory region described in *block* is reserved.

**Notes:**            reserve\_dma examines the memory region, adjusts the buffer to avoid 64K boundaries and the 16MB limitation of ISA bus systems. release\_dma cleans up after the transfer and unlocks the parameters. These function must be used in the following sequence:

```
reserve_dma(block);
// dma programmed and used
release_dma(block);
```

**Restrictions:**    These functions can only be used by device-drivers operating under THEOS 32 Version 4 or above.

|                     |                    |        |       |         |         |
|---------------------|--------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>release_dma</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>reserve_dma</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**        [schedint](#)

***\_remote***

Invoke a function located in another code segment.

```
#include <driver.h>
void _remote ( unsigned int offset, unsigned int segment, ... )
```

---

|                |   |                                    |
|----------------|---|------------------------------------|
| <i>offset</i>  | » | offset to function                 |
| <i>segment</i> | » | memory segment of function         |
| ...            | » | other arguments passed to function |

**Operation:** The function at location *segment*, *offset* is called, passing any remaining arguments to it.

**Returns:** The value returned by this “function” is the value returned by the remote function called.

**Notes:** This keyword is defined in this function library reference, not because it is a function, but because it looks like a function. *\_remote* is one of the intrinsic operators built into the THEOS C language compiler.

**Restrictions:** The function called by this function must be declared as a *\_remote* function type. Declaring a function as *\_remote* tells the compiler that, when it returns, it may return to a different code segment.

**Conforms to:**            *\_remote*    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>
#include <sc.h>

int  seg;
char buffer[1024];

void
main()
{
    seg = load("module.command", 2);
    _remote(0, seg, "Now is the time", buffer);
    unload(seg);
    puts(buffer);
}

module.program:

#include <stdlib.h>
#include <string.h>

_remote void
module(char *from, char *to)
{
    while (*from) {
        *to++ = *from;           // copy char
        *to++ = ' ';             // No data segment,
```

```
        *to++ = '=';           // so copy lits one
        *to++ = ' ';           // char at a time
        itoa(to, *from++);      // convert to string
        to = strend(to);        // point to new end
        *to++ = '\n';          // start new line
    }
}
```

```
>cc example (link
```

```
>cc module (link nomain
```

---

**Output:**

```
>example
N = 78
o = 11
w = 119
  = 32
i = 105
s = 115
  = 32
t = 116
h = 104
e = 101
  = 32
t = 116
i = 105
m = 109
e = 101

>
```

### remove

Erases or removes a file from disk.

```
#include <stdio.h> or <io.h>
int remove ( char * filename )
```

---

*filename*                   »   pointer to string containing file description

**Operation:**       The file specified by *filename* is erased: All disk space previously used by the file is returned to the disk's free space map. The directory entry for the file is marked as deleted.

**Returns:**         A success/fail indicator. A return value of zero indicates success; a non-zero return value indicates failure and is the reason code.

**Errors:**         When the return value is non-zero, then *filename* was not erased. The return value is the code indicating the failure reason. Both [errno](#) and [\\_errnum](#) are set to this reason code and [\\_errarg](#) is set to *filename*.

**Notes:**          The contents of *filename* should be complete and explicit. If *filename* does not specify a path, then the current working directory is assumed. If *filename* does not specify a drive, then all drives specified in the current drive search sequence are used until the file is either found or all drives in the search sequence are examined. If *filename* does not specify a file type, then only the current default library is searched.

The `remove` function name is the ANSI name for the file erase operation. The `erase` function is the THEOS name for the operation. For portability purposes use the `remove` function name.

**Restrictions:**   You may not erase a file that is currently open by any program or task, including your own.

**Conforms to:**       **remove**    n ANSI    n DOS    n THEOS    q POSIX

**See also:**         [close](#), [erase](#), [unlink](#)



**Example:**

```
#include <stdio.h>
#include <string.h>

void
main(int argc, char * argv[])
{
    char    filename[256];
    int     rc;

    if (argc) {                // file specified?
        strcpy(filename, argv[1]);
        printf("\nThe file \"%s\" was ", filename);

        if (rc = remove(filename))
            printf("not erased. rc = %d\n", rc);
        else
            printf("erased.");
    }
}
```

---

**Output:**

```
>test some.file
```

```
The file "some.file" was erased.
```

```
>test some.file
```

```
The file "some.file" was not erased. rc = 19
```

```
>
```

**rename**

Changes the name of an existing file, library member, library or directory.

```
#include <stdio.h> or <io.h>
```

```
int rename ( char * oldname, char * newname )
```

---

*newname* » pointer to string containing new file name

*oldname* » pointer to string containing current file name

**Operation:** The file, library member, library or directory indicated by *oldname* is renamed to have *newname*.

**Returns:** A success/fail indicator. A zero return value indicates success; a non-zero return value indicates failure and [errno](#) is set to the specific error.

**Errors:** When the return value is not zero, [errno](#) may have one of the following values:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                                                               |
|-------------|--------------|--------------------------------------------------------------------------------------------------------------|
| EACCES      | 18           | Either <i>oldname</i> is protected or <i>newname</i> already exists.                                         |
| ENOENT      | 19           | <i>oldname</i> not found.                                                                                    |
|             | 12           | Drive code invalid.                                                                                          |
|             | 45           | <i>newname</i> missing or null string.                                                                       |
|             | 46           | <i>oldname</i> missing or null string.                                                                       |
|             | 47           | Attempt to rename a library or directory into a library member or a directory into a subdirectory of itself. |
|             | 52           | Account name invalid.                                                                                        |
|             | 320          | <i>oldname</i> is open.                                                                                      |
|             | 462          | File-type missing.                                                                                           |

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [fperror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:** By default, the *oldname* and *newname* refer to files in the current account in the current working directory of the current account. To specify a file that is not in the current working directory, you must include the path to the file, either relative to the current working directory or absolute from the root directory.

Normally, *oldname* refers to files owned by the current account. To rename a file owned by another account, you must specify the complete path specification to the file, including the owning account name.

```
rename("oldacct\his.file", "my.file")
```

Normally, the *newname* refers to a name in your account. To rename a file to another account, you must specify the complete path specification, including the owning account name.

```
rename("some.file", "otheraccount\new.file")
```

A file may be renamed from one account to another:

```
rename("account1\his.file", "account2\new.file")
```

**Restrictions:** You cannot rename a file, library member, library or directory that is erase, read or write-protected.

You cannot rename a file to another drive.

You cannot rename a file to a name that already exists.

When *newname* specifies a path, that path must exist. The rename function does not create subdirectories.

**Conforms to:**                **rename**    n ANSI    n DOS    n THEOS    q POSIX

---

**Example:**

```
#include <stdio.h>

void
main(int argc, char *argv[])
{
    if (argc>2) {
        if (rename(argv[1], argv[2]))
            perror();
        else
            printf("\n%s renamed to %s", argv[1], argv[2]);
    }
}
```

---

**Output:**

```
>test test.file test.newfile

TEST.FILE renamed to TEST.NEWFILE

>
```

**rewind**

Reposition the file input/output pointer to the beginning of the file.

```
#include <stdio.h>
void rewind ( FILE * file )
```

---

*file*                      »    pointer to file's fcb

**Operation:**        Sets the files input/output position pointer to the beginning of the file. This is equivalent to using:

```
(void) fseek(file, 0L, SEEK_SET)
```

except that `rewind` clears the error indicators for *file* and `fseek` does not.

**Errors:**            No errors are detected. If the `rewind` operation cannot be performed for any reason, then the function returns without any indication of this.

**Notes:**            The `rewind` operation clears the end-of-file indicator and discards any prior [ungetc](#) characters from *file*.

You may also use `rewind` to clear the keyboard input buffer. Assuming that `stdin` is still directed from the keyboard, merely `rewind(stdin)`.

Because this function does not actually read or write any data to *file*, no record or location-locking is tested.

**Conforms to:**                **rewind**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                [fseek](#), [fsetpos](#), [lseek](#), [seek](#), [\\_seek](#)

## rmdir

Removes or erases a directory.

```
#include <direct.h> or <stdio.h>
```

```
int rmdir ( char * dirname )
```

---

*dirname*           »   pointer to directory name

**Operation:**       The directory specified by *dirname* is erased. This directory must be empty.

**Returns:**         A zero when the new directory is successfully deleted. A non-zero return value indicates an error, in which case [errno](#) is set.

**Errors:**          When the return is NULL, [errno](#) is set to one of the following values:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                                                                                |
|-------------|--------------|-------------------------------------------------------------------------------------------------------------------------------|
| EACCES      | 18           | The <i>dirname</i> isn't a directory, or it is not empty, or it is the current working directory or it is the root directory. |
| ENOENT      | 19           | Path name of <i>dirname</i> not found.                                                                                        |

**Restrictions:**    The file specified by *dirname* must be the name of an empty directory.

**Conforms to:**       **rmdir**    q ANSI     n DOS     n THEOS     q POSIX

**See also:**         [erase](#), [remove](#), [unlink](#)

---

### Example:

```
#include <stdio.h>
```

```
main()
{
    if (rmdir("test")) {
        perror("Error erasing directory");
        exit(errno);
    }
    else
        printf("Directory successfully erased.\n");
}
```

---

### Output:

```
>test
Error erasing directory: File "test" not found.
```

```
>md test
```

```
>test
Directory successfully erased.
```

***\_rsvmem***

Mark an area of memory so that it does not move during memory compaction.

```
#include <sc.h>
unsigned short * _rsvmem ( long addr, long len, int code )
```

---

|             |   |                                     |
|-------------|---|-------------------------------------|
| <i>addr</i> | » | starting physical address of memory |
| <i>code</i> | » | release/reserve code                |
| <i>len</i>  | » | length of memory                    |

**Operation:** The physical region of memory starting at *addr*, for *len* bytes, is either marked as reserved or released from reserved status, depending upon the value of *code*:

| <i>code</i> | <i>Meaning</i>     |
|-------------|--------------------|
| 0           | Release the memory |
| 1           | Reserve the memory |

**Returns:** A NULL pointer is always returned.

**Notes:** This function is normally used by a device-driver for a device using DMA. Before a region of memory is used for a DMA transfer, it should be marked as reserved. After the transfer is complete, it should be released.

Periodically, the operating system compacts memory to consolidate free space. If this is done while a device is transferring to or from a DMA location, the transferred data might be destroyed. With this function a region of memory is marked as reserved so the operating system will not move that memory during memory compaction.

**Conforms to:** ***\_rsvmem***    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [\*\\_PhyAddr\*](#), [\*\\_phy\\_addr\*](#)

---

**\_saveints, \_restoreints**

Disable and conditionally re-enable interrupts.

```
#include <builtin.h>
void _restoreints ( long flag )
void _saveints ( long flag )



---


flag                » long integer "lvalue"
```

**Operation:**     **\_restoreints** This macro assumes that *flag* is the same flag value used by the **\_saveints** macro and that it contains the interrupt status at the time interrupts were disabled with the **\_saveints** macro. This flag is tested to see if interrupts were enabled when the **\_saveints** macro was used. If so, interrupts are enabled. Otherwise, no action is taken.

**\_saveints** Saves the current interrupt status flag and then disables interrupts.

**Returns:**       Although no value is actually returned by these macros, the **\_saveints** macro uses the *flag* argument as an lvalue name and the interrupt status prior to disabling interrupts is saved in that name field.

**Notes:**         These "functions" are implemented as macros.

                 It is recommended that these macros be used instead of the [intoff](#) and [inton](#) "functions" because these routines will preserve the interrupt status.

**Restrictions:**   These macros are intended for device-driver authors.

**Conforms to:**       **\_saveints**    q ANSI      q DOS      n THEOS      q POSIX

**See also:**         [intoff](#), [inton](#)

## scanf

Accept formatted fields from the stdin device.

```
#include <stdio.h>
```

```
int scanf ( const char * mask, argument ... )
```

---

|             |   |                                         |
|-------------|---|-----------------------------------------|
| <i>mask</i> | » | pointer to string containing input mask |
|-------------|---|-----------------------------------------|

|                 |   |                                                 |
|-----------------|---|-------------------------------------------------|
| <i>argument</i> | » | pointers to storage locations for items scanned |
|-----------------|---|-------------------------------------------------|

**Operation:** The mask string contains zero or more regular characters and conversion-specification characters. Data is read from the current stdin device and interpreted according to the specifications in *mask*. Each field accepted is assigned to the locations pointed to in the *argument* list.

The *mask* field controls the interpretation of the input fields. The *mask* may contain one or more of the following:

- ▶ One or more white-space characters. The white-space characters read are not saved. One white-space character in *mask* matches any number and combination of white-space characters in the input stream.
- ▶ Regular characters except for the percent sign ( % ). A regular character causes *scanf* to read the next character from stdin. If that character does not match the regular character in *mask*, *scanf* terminates. The matching characters read are not saved. Non-matching characters are left in the input stream as if they had not been read.
- ▶ Format specifications starting with the percent sign ( % ). When a format specification is encountered, the value of the input field is converted according to the specification and stored in the location pointed to by the next value in the *argument* list. An input field is defined as all characters up to the first white-space character or up to the first character that cannot be converted according to the format specification or until the field width (if specified) is reached.

**Returns:** The number of fields successfully read and assigned. The return value may be less than the number of fields in *mask*.

**Errors:** A return value of EOF means that an error or end-of-file was detected on stdin before the first conversion. A return value of zero means that no fields were assigned.

**Notes:** ■ **Format Specifications**

Format specifications always begin with a percent sign ( % ) character and are used in the sequence that they are specified in *mask*. The first specification is used for the first input field, the second specification is used for the second input field, and so on.

When there are more arguments than format specification fields, the extra arguments are evaluated but not used. If there are fewer arguments than format specifications, the results are unpredictable.

A format specification has the following form, with the components enclosed in square brackets optional:



% [\*] [width] [modifier] type

Each component of the format specification is a single letter or whole number value. The simplest form of a format specification has a percent sign followed by a single letter *type* specification. For instance: %c.

**Type** is a required component specifying the type of input field to convert (character, string or number). **Width** is an optional component indicating the maximum number of characters to read from stdin. **Modifier** is an optional component indicating the size of the argument (short, long, *etc.*).

- **Assignment Suppression**

An asterisk ( \* ) following the percent sign indicates that the input field is to be scanned but not stored. There should be no argument pointer for an assignment suppression field.

- **Type**

The *type* is required and specifies the type of argument to format.

| Type                  | Field Type | Meaning                                                                                                                                                                                                                            |
|-----------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| c                     | char *     | Single character. White-space characters that are ordinarily skipped can be read with a c type. To read the next non-white-space character, use %1c.                                                                               |
| d                     | int *      | Optionally signed, decimal integer.                                                                                                                                                                                                |
| e or E<br>f<br>g or G | float *    | Floating-point value in scientific notation: -±ddd.dxxxxde±dd. (An optional leading sign, one or more decimal digits containing a decimal point, an optional exponent (“e” or “E”) followed by an optionally signed integer value. |
| i                     | int *      | An optionally signed, decimal, hexadecimal or octal integer.                                                                                                                                                                       |
| n                     | int *      | No argument is converted. The number of characters read so far from stdin is stored in the location pointed to by the argument.                                                                                                    |
| o                     | int *      | An unsigned octal integer.                                                                                                                                                                                                         |
| p                     | void **    | A pointer value is converted in the form of xxxx:yyyy where the digits x and y are hexadecimal digits.                                                                                                                             |
| s                     | char *     | A sequence of non-white-space characters.                                                                                                                                                                                          |
| u                     | unsigned * | Unsigned decimal integer.                                                                                                                                                                                                          |
| x or X                | int *      | An unsigned hexadecimal integer.                                                                                                                                                                                                   |

Table 11: scanf Type Codes

- **Accepting Strings**

Strings are normally accepted with the “%s” specification, which accepts a sequence of characters, stopping on the first white-space character. To accept a string that is not terminated with a white-space character, use the **bracket specification**. A pair of brackets enclosing a series of characters means that a string is accepted up to the first character that doesn’t match

any of the characters in the bracketed list. For instance, “%[abcd]” means to accept a series of characters until a character is read that is not an “a,” “b,” “c” or “d.” If the first character in the set is a caret ( ^ ), then the effect is reversed: Characters are accepted up to the first character that is in the bracketed list.

A string of characters may be accepted and stored without a null terminator (“\0”). Use the specification of “%nc” where *n* is a decimal integer specifying the number of characters to accept. The ‘c’ character indicates that the *argument* is a pointer to an array of characters. The next *n* characters are read from stdin and copied to the character array. No null terminator is added.

- **Width**

The *width* is a decimal value specifying the maximum number of characters to be read from stdin. Fewer than *width* characters may be read and used if a white-space character is encountered or if a character is encountered that cannot be converted according to the format *type*.

- **Modifier**

The *modifier* component is an optional field that specifies the size of the *argument*.

| <i>modifier</i> | Meaning                                                                                                                                                                                                                                      | Default                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| h               | When used with an integer type (d, i, o or x), the argument is a short int. When used with type u, the argument is a short unsigned int.                                                                                                     | The default size for integers is long unless otherwise specified by a <code>#pragma int directive</code> . |
| l               | When used with an integer type (d, i, o or x), the argument is a long int. When used with type u, the argument is a long unsigned int. When used with a floating-point type (e, E, f, g or G), the argument is a double rather than a float. |                                                                                                            |
| L               | When used with a floating-point type (e, E, f, g or G), the argument is a long double.<br><br>When used with integer types (i, o, u, x or X), the argument is a long long (64-bit) integer.                                                  |                                                                                                            |

Table 12: scanf Modifier Codes

**Restrictions:** Each of the *arguments* must be a pointer to a variable field with a type that matches the type specifier in *mask*.

**Conforms to:**                      **scanf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                      [fscanf](#), [sscanf](#)

**Example:**

```
#include <stdio.h>

void
main()
{
    char  first[80],
          last[80];

    printf("\nEnter your first and last names: ");
    scanf("%s %s", first, last);
    printf("\nHello \"%s, %s\".\n", last, first);
}
```

---

**Output:**

>example

Enter your first and last names: Joe Friday

Hello "Friday, Joe."

>

**schedint**

Define or remove an interrupt service routine for an interrupt.

```
#include <driver.h>
```

```
void schedint ( long irq, void (* isr )(void) )
```

---

```
irq                »   interrupt request number
```

```
isr                »   name of function for interrupt service routine
```

**Operation:** The `_schedint` function is the general interface for scheduling and unscheduling interrupt service routines. The [release\\_dma](#), [reserve\\_dma](#) and [oneshot functions](#) are special calls to the `_schedint` function.

When *isr* is not a NULL pointer, then *isr* is programmed as the interrupt service routine for *irq*. This means that every time that *irq* occurs, then the current program that is executing is interrupted and *isr* is called to service the interrupt. When *isr* terminates, the interrupted program resumes.

When *isr* is a NULL pointer, then no interrupt service routine is assigned to handle *irq*.

**Defaults:** When the system is first booted, there are no interrupt service routines defined for any interrupt, although the boot-up process may schedule its own interrupt service routines.

**Restrictions:** These functions, and the [release\\_dma](#), [reserve\\_dma](#) and [oneshot functions](#), should be used in device-drivers only.

**Conforms to:** `schedint` q ANSI q DOS n THEOS q POSIX

**See also:** [oneshot functions](#), [release\\_dma](#), [reserve\\_dma](#)

## sec, sech

Compute the secant or the hyperbolic secant of an angle.

```
#include <math.h>
double sec ( double x )
double sech ( double x )
```

---

$x$                       »   floating-point angle, in radians

**Operation:**      **sec**              Compute the secant of the angle  $x$ .

**sech**            Compute the hyperbolic secant of the angle  $x$ .

**Returns:**        **sec**              The secant of the angle  $x$ .

**sech**            The hyperbolic secant of the angle  $x$ .

**Notes:**            The angle  $x$  is specified with radians, not degrees.

These functions use BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**            **sec**      q ANSI      q DOS      n THEOS      q POSIX

**sech**      q ANSI      q DOS      n THEOS      q POSIX

**See also:**            [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sin](#), [sinh](#), [tan](#), [tanh](#)

**seek, \_seek**

Sets a file's input/output position pointer to a specified location.

```
#include <stdio.h>
unsigned long seek ( FILE * file, long location, int base )
```

```
#include <sc.h>
unsigned long _seek ( FILE * file, int location, int base )
```

---

|                 |   |                              |
|-----------------|---|------------------------------|
| <i>base</i>     | » | starting point within file   |
| <i>file</i>     | » | pointer to open file's fcb   |
| <i>location</i> | » | relative position to seek to |

**Operation:**      **seek**      This function name is a synonym to the [\\_lseek](#) function.

**\_seek**      The file-position pointer for *file* is set according to *offset* and *base*. That is, the position pointer is set to *offset* bytes from *base*. *offset* is a signed value and *base* is coded to be one of three values:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                                              |
|-------------|--------------|---------------------------------------------------------------------------------------------|
| SEEK_SET    | 0            | <i>offset</i> is in bytes, relative to the beginning of the file.                           |
| SEEK_CUR    | 1            | <i>offset</i> is in bytes, relative to the current value of the position pointer.           |
| SEEK_END    | 2            | <i>offset</i> is in bytes, relative to the current end-of-file.                             |
|             | 3            | <i>offset</i> is in 512-byte blocks, relative to the beginning of the file.                 |
|             | 4            | <i>offset</i> is in 512-byte blocks, relative to the current value of the position pointer. |
|             | 5            | <i>offset</i> is in 512-byte blocks, relative to the current end-of-file.                   |

**Returns:**      Both functions return the new file position relative to the beginning of the file. A return value of -1 indicates an error occurred. On devices incapable of seeking (terminals, printers, *etc.*) the return value is undefined.

**Notes:**      You may position to any location in the file or even beyond the current end-of-file. However, you may not position to a location before the beginning of the file. An attempt to position prior to the start of the file is interpreted as a request to position to the beginning of the file.

Because these functions do not actually read or write any data to *file*, no record or location-locking is tested.

**Restrictions:** This function should be used on stream and relative-access files only.

No flush operation is performed on *file* prior to changing the position pointer. If the write buffer for *file* is dirty, you should perform a [fflush](#) operation or use the [fsetpos](#) function.

**Conforms to:**

|              |        |       |         |         |
|--------------|--------|-------|---------|---------|
| <b>seek</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_seek</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [fseek](#), [fsetpos](#), [ftell](#), [lseek](#), [rewind](#)

**select**

Determine the status of one or more sockets, waiting if necessary.

```
#include <socket.h>
```

```
long select ( int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds,  
             struct timeval * timeout )
```

---

|                  |   |                                                         |
|------------------|---|---------------------------------------------------------|
| <i>exceptfds</i> | » | pointer to set of sockets to be checked for errors      |
| <i>nfds</i>      | » | argument ignored                                        |
| <i>readfds</i>   | » | pointer to set of sockets to be checked for readability |
| <i>writefds</i>  | » | pointer to set of sockets to be checked for writability |
| <i>timeout</i>   | » | maximum time to wait or NULL for blocking operation     |

**Operation:** This function is used to determine the status of one or more sockets. For each socket, the caller may request information on read, write or error status. The set of sockets for which a given status is requested is indicated by an `fd_set` structure. Upon return, the structure is updated to reflect the subset of these sockets which meet the specified condition, and `select` returns the number of sockets meeting the conditions. A set of macros is provided for manipulating an `fd_set`. These macros are compatible with those used in the Berkeley software, but the underlying representation is completely different.

The parameter *readfds* identifies those sockets which are to be checked for readability. If the socket is currently listening, it will be marked as readable if an incoming connection request has been received, so that an [accept](#) is guaranteed to complete without blocking. For other sockets, readability means that queued data is available for reading or, for sockets of type `SOCK_STREAM`, that the virtual socket corresponding to the socket has been closed, so that a [recv](#) or [recvfrom](#) is guaranteed to complete without blocking. If the virtual circuit was closed gracefully, then a [recv](#) will return immediately with 0 bytes read; if the virtual circuit was closed abortively, then a [recv](#) will complete immediately with the error code `TSAECONNRESET`. The presence of out-of-band data will be checked if the socket option `SO_OOBINLINE` has been enabled (see [setsockopt](#)).

The parameter *writefds* identifies those sockets which are to be checked for writeability. If a socket is connecting (non-blocking), writeability means that the connection establishment is complete. For other sockets, writeability means that a [send](#) or [sendto](#) will complete without blocking. It is not specified how long this guarantee can be assumed to be valid, particularly in a multithreaded environment.

The parameter *exceptfds* identifies those sockets which are to be checked for the presence of out-of-band data or any exceptional error conditions. Note that out-of-band data will only be reported in this way if the option `SO_OOBINLINE` is `FALSE`. For a `SOCK_STREAM`, the breaking of the connection by the peer or due to `KEEPALIVE` failure will be indicated as an exception. This specification does not define which other errors will be included. If a socket is connecting (non-blocking), failure of the connect attempt is indicated in *exceptfds*.

Any of *readfds*, *writefds* or *exceptfds* may be given as `NULL` if no descriptors are of interest. Four macros are defined in the header file `SOCKET.H` for



manipulating the descriptor sets. The variable `FD_SETSIZE` determines the maximum number of descriptors in a set. (The default value of `FD_SETSIZE` is 64, which may be modified by `#defining` `FD_SETSIZE` to another value before `#including` `socket.h`.) Internally, an `fd_set` is represented as an array of `SOCKET`s; the last valid entry is followed by an element set to `INVALID_SOCKET`. The macros are:

|                                |                                                              |
|--------------------------------|--------------------------------------------------------------|
| <code>FD_CLR(s, *set)</code>   | Removes the descriptor <i>s</i> from set.                    |
| <code>FD_ISSET(s, *set)</code> | Non-zero if <i>s</i> is a member of the set, zero otherwise. |
| <code>FD_SET(s, *set)</code>   | Adds descriptor <i>s</i> to set.                             |
| <code>FD_ZERO(*set)</code>     | Initializes the set to the NULL set.                         |

The parameter *timeout* controls how long the `select` may take to complete. If *timeout* is a null pointer, `select` will block indefinitely until at least one descriptor meets the specified criteria.

Otherwise, *timeout* points to a struct `timeval` which specifies the maximum time that `select` should wait before returning. If the *timeval* is initialized to `{0, 0}`, `select` will return immediately; this is used to “poll” the state of the selected sockets. If this is the case, then the `select` call is considered non-blocking and the standard assumptions for non-blocking calls apply. For example, the blocking hook must not be called, and the THEOS Sockets implementation must not yield.

**Returns:** The total number of descriptors which are ready and contained in the `fd_set` structures, or 0 if the time limit expired, or `SOCKET_ERROR` if an error occurred. If the return value is `SOCKET_ERROR`, [TSAGetLastError](#) may be used to retrieve a specific error code.

**Errors:** When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

| <i>Code</i>                 | <i>Meaning</i>                                                                       |
|-----------------------------|--------------------------------------------------------------------------------------|
| <code>TSAEINVAL</code>      | The <i>timeout</i> value is not valid.                                               |
| <code>TSAENETDOWN</code>    | The THEOS Sockets implementation has detected that the network subsystem has failed. |
| <code>TSAENOTSOCK</code>    | One of the descriptor sets contains an entry which is not a socket.                  |
| <code>TSAEINPROGRESS</code> | A blocking THEOS Sockets call is in progress.                                        |

**Conforms to:** `select` q ANSI q DOS n THEOS q POSIX

**See also:** [accept](#), [connect](#), [recv](#), [recvfrom](#), [setsockopt](#), [send](#), [sendto](#)

**semaphore functions**

These functions maintain and access semaphores in your set of tasks or in a remote set of tasks.

```
#include <semaphore.h>
unsigned short rsema ( int pid, int sema_nbr )
unsigned short rsemaname ( int pid, char * sema_name )
unsigned short rsemares ( int pid, int sema_nbr )
unsigned short rsemaset ( int pid, int sema_nbr )
void rsemawait ( int pid, int sema_nbr )

unsigned short sema ( int sema_nbr )
unsigned short semaphore ( const char _far * sema_name )
unsigned short semares ( int sema_nbr )
unsigned short semaset ( int sema_nbr )
void semawait ( int sema_nbr )

#include <timer.h>
void timer ( int sema_nbr, int type, long msec )
```

---

|                  |   |                                             |
|------------------|---|---------------------------------------------|
| <i>msec</i>      | » | time in milliseconds                        |
| <i>pid</i>       | » | process number of remote task               |
| <i>sema_name</i> | » | pointer to string containing semaphore name |
| <i>sema_nbr</i>  | » | semaphore number                            |
| <i>type</i>      | » | coded type of timer                         |

**Operation:** Semaphores are named flags that are used to signal true/false, ready/not ready, *etc.* between two cooperating tasks. Remote semaphores are semaphores used to signal between tasks operating in different processes (users).

|                  |                                                                                                                                                                                                                   |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>rsema</b>     | Gets the current status of a remote semaphore numbered <i>sema_nbr</i> in the task group executing in user process <i>pid</i> .                                                                                   |
| <b>rsemaname</b> | Gets the semaphore number of a remote semaphore named <i>sema_name</i> in the task group executing in user partition <i>pid</i> .                                                                                 |
| <b>rsemares</b>  | Clears the status of a remote semaphore numbered <i>sema_nbr</i> in the task group executing in user process <i>pid</i> .                                                                                         |
| <b>rsemaset</b>  | Sets the status of a remote semaphore numbered <i>sema_nbr</i> in the task group executing in user process <i>pid</i> .                                                                                           |
| <b>rsemawait</b> | Waits for a remote semaphore numbered <i>sema_nbr</i> in the task group executing in user process <i>pid</i> to be set. Once the semaphore is set and before control exits this function, the semaphore is reset. |
| <b>sema</b>      | Gets the current status of a local semaphore <i>sema_nbr</i> .                                                                                                                                                    |

|                  |                                                                                                                                                                                                                                                           |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>semaphore</b> | Gets the semaphore number of a local semaphore named <i>sema_name</i> . If <i>sema_name</i> has never been catalogued, then it is added and a number is assigned to it.                                                                                   |
| <b>semares</b>   | Clears the status of a local semaphore numbered <i>sema_nbr</i> .                                                                                                                                                                                         |
| <b>semaset</b>   | Sets the status of a local semaphore numbered <i>sema_nbr</i> .                                                                                                                                                                                           |
| <b>semawait</b>  | Waits for a local semaphore numbered <i>sema_nbr</i> to be set. Once <i>sema_nbr</i> is set and before control exits this function, <i>sema_nbr</i> is reset.                                                                                             |
| <b>timer</b>     | Any current timers for <i>semaphore</i> are cleared and <i>semaphore</i> is reset. A timer is then programmed for <i>semaphore</i> using <i>msec</i> and <i>type</i> . When the timer is activated, the <i>semaphore</i> is set and the timer is cleared. |

**Returns:** The value of a semaphore status is zero when it is cleared and non-zero when it is set.

|                  |                                                                                                                                                                 |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>rsema</b>     | The status of the semaphore.                                                                                                                                    |
| <b>rsemaname</b> | The semaphore number of the remote semaphore. A return value of -1 indicates that <i>sema_name</i> is not catalogued in the remote task.                        |
| <b>rsemares</b>  | The status of the semaphore prior to clearing it.                                                                                                               |
| <b>rsemaset</b>  | The status of the semaphore prior to setting it.                                                                                                                |
| <b>sema</b>      | The status of the the semaphore.                                                                                                                                |
| <b>semaphore</b> | The semaphore number of your semaphore. A return value of -1 indicates that the semaphore could not be catalogued because all 64 semaphores are already in use. |
| <b>semares</b>   | The status of the semaphore prior to clearing it.                                                                                                               |
| <b>semaset</b>   | The status of the semaphore prior to setting it.                                                                                                                |

**Notes:** Each set of tasks (a task, its child tasks and its parent tasks) shares a single set of 64 semaphore names and numbers. A set of tasks use the *sema*, *semaphore*, *semares*, *semaset* and *semawait* functions to maintain and access these semaphores. The *rsema*, *rsemaphore*, *rsemares*, *rsemaset* and *rsemawait* functions are used by one set of tasks to access the semaphores of another set of tasks.

The expected usage of remote semaphores is for the control of shared memory resources. For example, a scheduling system might be started as a background task in the morning. It would define some shared memory and put its process number in the shared memory. Semaphores used by this task determine when an event has occurred that it needs to service.

Other users could then query this shared memory to find the process number of the scheduling system and then signal it with its remote semaphores about jobs that it wants done by the scheduling system.

## 552 semaphore functions

---

- rsemawait** Although processing of your program is suspended during the wait, it may be awakened by signal processing operations. Refer to the [alarm](#), [signal](#) and [msalarm](#) function descriptions.
- semawait** Although processing of your program is suspended during the wait, it may be awakened by signal processing operations. Refer to the [alarm](#), [signal](#) and [msalarm](#) function descriptions.
- timer** The *type* argument is a coded value. Possible *type* codes and their meanings are:

| <i>type</i> | <i>Meaning</i>                                                                                                                                     |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| 0           | Deactivate timer for <i>semaphore</i> .                                                                                                            |
| 1           | <i>msec</i> is an elapsed amount of time specification.                                                                                            |
| 2           | Synchronize <i>msec</i> with time-of-day. For instance, an <i>msec</i> of 60000 will set a timer for the next change of minute in the time-of-day. |
| 3           | <i>msec</i> is a specific time-of-day specification.                                                                                               |

**Defaults:** The initial status of a newly catalogued semaphore is off or reset.

**Restrictions:** There is a maximum of 64 semaphores for each set of tasks.

Semaphore names must be valid identifiers. That is, they must start with a letter or underscore and contain only letters, digits and underscores. Semaphore names are case-insensitive and only the first eight characters of a semaphore name are used for identification.

Although multiple timers can be in effect at one time, only one timer per *semaphore* number is supported.

**Conforms to:**

|                  |        |       |         |         |
|------------------|--------|-------|---------|---------|
| <b>rsema</b>     | q ANSI | q DOS | n THEOS | q POSIX |
| <b>rsemaname</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>rsemares</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>rsemaset</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>rsemawait</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>sema</b>      | q ANSI | q DOS | n THEOS | q POSIX |
| <b>semaphore</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>semares</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>semaset</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>semawait</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>timer</b>     | q ANSI | q DOS | n THEOS | q POSIX |

**Example:**

```

#include <semaphore.h>
#include <process.h>

int    ready;

void main()
{
    int    child;

    ready = semaphore("ready");    // catalogue ready semaphore

    if (child = fork())            // spawn subtask
    {
        semawait(ready);            // wait for child
        ...                        // do prelim parent stuff
    }
    else
    {
        subtask();                // do child task stuff
        exit();
    }
    ...                            // remainder of parent stuff
}

void
subtask(void)
{
    ...                            // do prelim stuff
    semaset(ready);                // tell parent okay
    ...
}

#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
#include <peek.h>
#include <sc.h>

void
main()
{
    int    rpid,
           msg_wait;
    unsigned    schedule;           // selector for shared memory

    schedule = shared("schedule",4096);    // find shared memory
    rpid = _peekb(schedule, (char *)0);    // get schedule system pid

    if (rpid)
    {
        msg_wait = rsemaname(rpid,"msg_wait"); // get sema #

        lockset(0, schedule);            // make sure exclusive use
        ...                               // copy our message to schedule system
        lockres(0, schedule);            // remove lock
    }
}

```

## 554 semaphore functions

---

```
    rsemaset(rpid, msg_wait); // alert schedule system
}
else // no shared memory = no schedule system
{
    printf("\nSchedule system is not present.\n");

    lockset(0, schedule); // lock, just in case
    if (_peekb(schedule, (char *)0)==0) // still not there?
        shared("schedule", 0); // deallocate
    lockres(0, schedule);
}
}

#include <timer.h>
#include <semaphore.h>

void
main()
{
    int timeout,
        noon;

    timeout = semaphore("timeout"); // catalog timeout
    noon = semaphore("noon"); // catalog noon

    while (1) {
        timer(noon, 3, 4320000); // timer set for 12 noon
        timer(timeout, 1, 300000); // timer set for 5 min

        while (!(sema(noon) || sema(timeout))) {
            ... // do stuff when timers haven't occurred
        }
        if (sema(noon)) { // noon?
            ... // do noon stuff
            timer(noon, 3, 4320000); // reschedule
        }
        if (sema(timeout)) { // 5-minute timer?
            ...
            timer(timeout, 1, 300000); // reschedule
        }
    }
}
```

## send, sendto

Sends data to a socket or sends a datagram and stores the destination address.

```
#include <socket.h>

int send ( SOCKET s, void * buffer, int len, int flags )
int sendto ( SOCKET s, void * buffer, int len, int flags, SOCKADDR * to,
            int tolen )
```

---

|               |   |                                                     |
|---------------|---|-----------------------------------------------------|
| <i>buffer</i> | » | pointer to data to be transmitted                   |
| <i>flags</i>  | » | value is not used                                   |
| <i>len</i>    | » | length of data in buffer                            |
| <i>s</i>      | » | socket number                                       |
| <i>to</i>     | » | pointer to location for storing destination address |
| <i>tolen</i>  | » | pointer to size of the <i>to</i> buffer             |

**Operation:**      **send**      This function is used on connected datagram or stream sockets and is used to write outgoing data to a socket.

For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets. If the data is too long to pass as one unit through the underlying protocol, the error `TSAEMSGSIZE` is returned and no data is transmitted. Note that the successful completion of a `send` does not indicate that the data was successfully delivered.

If buffer space is not available within the transport system to hold the data to be transmitted, `send` will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking `SOCK_STREAM` sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The `select` call may be used to determine when it is possible to send more data.

**sendto**      This function is used on datagram or stream sockets and is used to write outgoing data on a socket.

For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets. If the data is too long to pass as one unit through the underlying protocol, the error `TSAEMSGSIZE` is returned and no data is transmitted. Note that the successful completion of a `sendto` does not indicate that the data was successfully delivered.

`sendto` is normally used on a `SOCK_DGRAM` socket to send a datagram to a specific peer socket identified by the *to* parameter. On a `SOCK_STREAM` socket, the *to* and *tolen* parameters are ignored; in this case the `sendto` is equivalent to `send`.

To send a broadcast (on a `SOCK_DGRAM` only), the address in the *to* parameter should be constructed using the special IP address `INADDR_BROADCAST` (defined in `SOCKET.H`), together with the intended port number. It is generally inadvisable for a broadcast datagram to exceed the size at which

fragmentation may occur, which implies that the data portion of the datagram (excluding headers) should not exceed 512 bytes.

If buffer space is not available within the transport system to hold the data to be transmitted, `sendto` will block unless the socket has been placed in a non-blocking I/O mode. On non-blocking `SOCK_STREAM` sockets, the number of bytes written may be between 1 and the requested length, depending on buffer availability on both the local and foreign hosts. The `select` call may be used to determine when it is possible to send more data.

**Returns:** The number of bytes sent. If the connection has been closed, it returns zero. When an error occurs, a value of `SOCKET_ERROR` is returned and a specific error code may be retrieved by calling [TSAGetLastError](#).

**Errors:** When the return is `SOCKET_ERROR`, the possible error codes returned by [TSAGetLastError](#) may be:

| <i>Code</i>      | <i>Meaning</i>                                                                                                                           |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| TSAEACCES        | The requested address is a broadcast address but the appropriate flag was not set.                                                       |
| TSAEADDRNOTAVAIL | The specified address is not available from the local machine.                                                                           |
| TSAEAFNOSUPPORT  | Addresses in the specified family cannot be used with this socket.                                                                       |
| TSAECONNABORTED  | The virtual circuit was aborted due to timeout or other failure.                                                                         |
| TSAECONNRESET    | The virtual circuit was reset by the remote side.                                                                                        |
| TSAEDESTADDRREQ  | A destination address is required.                                                                                                       |
| TSAEFAULT        | The <i>buffer</i> or <i>to</i> len argument was invalid; the <i>to</i> buffer was too small to accommodate the peer address.             |
| TSAEINPROGRESS   | A blocking THEOS Sockets operation is in progress.                                                                                       |
| TSAEINVAL        | The socket has not been bound with <code>bind</code> .                                                                                   |
| TSAEMSGSIZE      | The socket is of type <code>SOCK_DGRAM</code> and the datagram is larger than the maximum supported by the THEOS Sockets implementation. |
| TSAENETDOWN      | The THEOS Sockets implementation has detected that the network subsystem has failed.                                                     |
| TSAENETRESET     | The connection must be reset because the THEOS Sockets implementation dropped it.                                                        |
| TSAENETUNREACH   | The network can't be reached from this host at this time.                                                                                |
| TSAENOBUFS       | The THEOS Sockets implementation reports a buffer deadlock.                                                                              |



| <i>Code</i>    | <i>Meaning</i>                                                              |
|----------------|-----------------------------------------------------------------------------|
| TSAENOTCONN    | The socket is not connected.                                                |
| TSAENOTSOCK    | The descriptor is not a socket.                                             |
| TSAEOPNOTSUPP  | MSG_OOB was specified, but the socket is not of type SOCK_STREAM.           |
| TSAEWOULDBLOCK | The socket is marked as non-blocking and the receive operation would block. |

**Conforms to:**

|               |        |       |         |         |
|---------------|--------|-------|---------|---------|
| <b>send</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>sendto</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [recv](#), [recvfrom](#), [socket](#)

**Example:** See the example for the function [socket](#).

**setbuf, setvbuf**

Allocate or assign an input/output buffer for an open file.

```
#include <stdio.h>
void setbuf ( FILE * file, char * buffer )
int setvbuf ( FILE * file, char * buffer, int type, size_t len )
```

---

|               |   |                            |
|---------------|---|----------------------------|
| <i>buffer</i> | » | pointer to buffer area     |
| <i>file</i>   | » | pointer to file's fcb      |
| <i>len</i>    | » | size of buffer to assign   |
| <i>type</i>   | » | type of buffering for file |

**Operation:**      **setbuf**      This function is identical in operation to either of the following two function calls:

```
setvbuf(file, buffer, _IOFBF, BUFSIZ)
```

```
fbuf(file, buffer, BUFSIZ)
```

It allocates or assigns a buffer to the open file specified by *file*. Full input and output buffering will be performed for accesses to the file.

**setvbuf**      An input/output buffer of size *len* is assigned to or allocated for the open file specified by *file*.

The *type* argument specifies the type of buffering to be performed for accesses to this file.

| <i>type</i> | Meaing                                                                                                                                                                                                    |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| _IONBF      | No buffering. All accesses to the file will be done by reading or writing directly from or to the disk image of the file.                                                                                 |
| _IOLBF      | Line buffering.                                                                                                                                                                                           |
| _IOFBF      | Full buffering. All accesses to the file will be done by first using the buffered data from the last access of the file. Only when the requested information is not in the buffer is the disk image used. |

When *type* is not *\_IONBF* and *buffer* is a NULL pointer, a buffer is allocated and assigned to *file*.

When *type* is not *\_IONBF* and *buffer* is not a NULL pointer, it is assumed to be a pointer to a buffer allocated by your program and that it is at least *len* bytes in size. In this situation, *buffer* is merely assigned to *file*.

**Returns:**      **setbuf**      No value is returned.

**setvbuf**      A zero is returned if the operation was performed successfully. A non-zero return indicates that *file* is not an open file or that it already has a buffer assigned to it.

**Defaults:** If the [fbuf](#), `setbuf` or `setvbuf` functions are not used before a file is accessed, a buffer size of 256 will automatically be allocated.

**Restrictions:** An input/output buffer can only be set between the time that a file is opened and the first access to the file.

Once a buffer is allocated for an open file, it cannot be reallocated without first closing the file and reopening it.

|                     |                |        |       |         |         |
|---------------------|----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>setbuf</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>setvbuf</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [fbuf](#)

---

**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE * in;

    if (!(in=fopen("my.file", "r")))
        exit(ferror());
    setvbuf(in, NULL, _IOFBF, 4*1024); // allocate a 4K buffer
    ...
}
```

### SetBypass, ClrBypass

Sets or clears the Session Manager bypass status.

```
#include <wmapi.h>
int ClrBypass ( void )
int SetBypass ( void )
```

**Operation:**      **ClrBypass**    Session Manager bypass status is cleared.

**SetBypass**    Set Session Manager bypass mode.

**Returns:**          Both functions return the prior Session Manager bypass status: 0 indicates it was not in bypass status; non-zero indicates that it was.

**Notes:**            Session Manager bypass refers to the ability to bypass the Session Manager control of multiple sessions on one console and multiple windows on one session. When in normal or clear bypass mode, characters are only displayed on the console if the session is the active session and the selected window is in update-on mode.

When Session Manager bypass mode is set, the Session Manager control of windows and sessions is bypassed. Characters are displayed on the console just as though the console was a single session, single window display. Characters are not saved by the Session Manager in this mode and, when bypass mode is cleared, cannot be “refreshed.”

**Defaults:**        When a console is first started, it is not in bypass mode. The bypass mode when the current program begins execution is dependent upon the state that the last program left it in.

|                     |                  |        |       |         |         |
|---------------------|------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>ClrBypass</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>SetBypass</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**        [IsBypass](#), [wRepaint](#)

---

## setjmp

Save the current state of a program for subsequent usage by a [longjmp](#) function call.

```
#include <setjmp.h>
int setjmp ( jmp_buf env )
```

---

*env*                   »   buffer variable to hold environment

**Operation:**       Save the current stack environment and instruction pointer in the *env* structure.

**Returns:**         A zero.

When this function returns due to a [longjmp](#) function call using the saved *env*, the return value is the *value* argument from the [longjmp](#) function call, or, if the *value* argument is zero, the return value is set to one.

**Notes:**           The setjmp function call saves the stack and return environment for subsequent use by a [longjmp](#) function call. The purpose is for the setjmp and [longjmp](#) functions to provide a way to execute a non-local goto operation.

**Restrictions:**   If a [system](#) function is used after the setjmp, your program should reset the setjmp prior to the [system](#) function call. The [system](#) function may change the program's data and code segment contents.

**Conforms to:**       **setjmp**    n ANSI      n DOS      n THEOS    n POSIX

**See also:**         [longjmp](#)

---

**Example:**         See "Example:" on page 408.

**setlocale**

Define or query the locale.

```
#include <locale.h>
```

```
char * setlocale ( int category, const char * locale )
```

---

*category*               »   coded locale of interest

*locale*                 »   pointer to string specifying name of locale

**Operation:**       The locale *category* is set to *locale*.

If *locale* is a NULL pointer (not a pointer to a null string), no change is made. Only the current locale name is returned.

**Returns:**         A pointer to the current locale name after any change is made.

**Errors:**         If *category* or *locale* is invalid, a NULL pointer is returned and no locale is changed.

**Notes:**         There are six *category* codes that may be used. The names for these categories are defined in `LOCALE.H`.

| <i>category</i> | <i>Meaning</i>                                                                                                                                                                                                                                                                 |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LC_ALL          | All categories.                                                                                                                                                                                                                                                                |
| LC_COLLATE      | Affects the behavior of <a href="#">strcoll</a> and <a href="#">strxfrm</a> .                                                                                                                                                                                                  |
| LC_CTYPE        | Affects the character testing functions ( <a href="#">is character functions</a> )                                                                                                                                                                                             |
| LC_MONETARY     | Defines the formatting used for the formatted output of monetary values and some of the data returned by <a href="#">localeconv</a> .                                                                                                                                          |
| LC_NUMERIC      | Defines the decimal point character used for the formatted output routines ( <a href="#">printf</a> , <i>etc.</i> ), for data conversion routines ( <a href="#">atof</a> , <a href="#">scanf</a> , <i>etc.</i> ) and some of the data returned by <a href="#">localeconv</a> . |
| LC_TIME         | Affects the behavior of <a href="#">strftime</a> .                                                                                                                                                                                                                             |

The values used in the *locale* string depend upon the various categories.

### ■ LC\_ALL

The LC\_ALL category is used to reset all categories at one time. Valid locales include:

| <i>locale</i> | <i>meaning</i>                                                                       |
|---------------|--------------------------------------------------------------------------------------|
| "C"           | The standard C locale for all categories. Use the standard, 7-bit ASCII definitions. |
| ""            | The default locale for all categories, which is the standard C locale.               |

## ■ LC\_COLLATE

The LC\_COLLATE category specifies the collating sequence used by the `strcoll` and `strxfrm` functions. Valid locales include:

| <i>locale</i> | <i>meaning</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "C"           | <p>The standard C locale.</p> <p>With this locale in effect, characters have the same collating sequence as their value. The sequence for the displayable characters is:</p> <p>!"#\$%&amp;'()*+,-./0123456789:;&lt;=&gt;?@ABCDEFGHIJKLMN O P Q R S T U V W X Y Z [\]^_`a b c d e f g h i j k l m n o p q r s t u v w x y z {   } ~ &lt;line graphics symbols&gt; Ä ä å à Á é ê ë è ì í î ï Ò ó ô õ ò Û ü û ú ù Ç ç Ñ ñ Æ æ Å å ß ÿ ; ¢ £ ¥ Pt ¢ ¼ ½ ¯ § · º</p>                                                                                                                                                                                                                                                                                                                                                                  |
| ""            | <p>The default locale, which is the standard C locale.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| "L"           | <p>Casemode-insensitive collating sequence. All uppercase and lowercase characters collate as if they were all uppercase characters. International characters are not affected by this setting. The sequence for the displayable characters is:</p> <p>!"#\$%&amp;'()*+,-./0123456789:;&lt;=&gt;?@AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz[\]^_`{   } ~ &lt;line graphics symbols&gt; Ä ä å à Á é ê ë è ì í î ï Ò ó ô õ ò Û ü û ú ù Ç ç Ñ ñ Æ æ Å å ß ÿ ; ¢ £ ¥ Pt ¢ ¼ ½ ¯ § · º</p>                                                                                                                                                                                                                                                                                                                                  |
| "T"           | <p>International collating sequence. All of the international letter characters collate into their expected position according to the alphabet. Other symbols are grouped together for a more consistent sorting sequence. The sequence the displayable characters is:</p> <p>!"#\$%&amp;'()*+,-./:;&lt;=&gt;?@{   } ~ [\]^_` ; ¢ £ ¥ Pt ¢ ¼ ½ ¯ § · º 0 1 2 3 4 5 6 7 8 9 Ä Å Æ Å Ä Ç Ç D E É F G H I J K L M N Ñ O Ö P Q R S ß T U Ü V W X Y Z a ä å à á æ å b c ç d e ë ê ë è f g h i ï î ï j k l m n ñ o ö ô õ ò p q r s t u ü û ú ù v w x y ÿ z &lt;line graphics symbols&gt;</p> <p>The collating sequence above is not exact because the comparison functions treat many of the characters as equivalents. For instance, the characters "e ë ê é è" are treated as if they are all greater than "d" but less than "f."</p> |





### ■ LC\_NUMERIC

The LC\_NUMERIC category determines which character (period or comma) is used to separate the whole number portion from the fractional portion of a floating-point number when formatted with the [printf](#) and similar functions. The opposite character is used to separate the groups of three digits in the whole number portion when the “,” specifier is used. Valid locales include:

| <i>locale</i> | <i>meaning</i>                                      | <i>Number format</i>                                 |
|---------------|-----------------------------------------------------|------------------------------------------------------|
| “C”           | The standard C locale.                              | Format as specified by DECIMAL environment variable. |
| “”            | The default locale, which is the standard C locale. |                                                      |
| “.”           | Use the period character as the separator.          | 123,456.78                                           |
| “,”           | Use the comma character as the separator.           | 123.456,78                                           |

### ■ LC\_TIME

The LC\_TIME category determines how dates are formatted with the [strftime](#) function. Valid locales include:

| <i>locale</i> | <i>meaning</i>                                      | <i>Date format</i>                                    |
|---------------|-----------------------------------------------------|-------------------------------------------------------|
| “C”           | The standard C locale.                              | Format as specified by DATEFORM environment variable. |
| “”            | The default locale, which is the standard C locale. |                                                       |
| “A”           | Use the “American” date format.                     | mm/dd/yyyy                                            |
| “E”           | Use the “European” date format.                     | dd-mm-yyyy                                            |
| “I”           | Use the “International” date format.                | yyyy.mm.dd                                            |

**Defaults:** The default locale for all categories is the standard C locale, which might be dependent upon the current value of environment variables.

**Restrictions:** The pointer returned is a pointer to static storage within the `setlocale` function and is overwritten on each subsequent call to `setlocale`. If this information is used in your program, it should make a copy of the string.

**Conforms to:** `setlocale`    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [localeconv](#), [strcoll](#)

**`_setprty, _set_slice`**

Set the current task priority or time slice allocated to the task.

```
#include <sc.h>
void _setprty ( int priority )
void _set_slice ( int msec )
```

---

*msec*                   »   time slice, in milliseconds  
*priority*               »   priority number

**Operation:**        **`_setprty`**   Set the priority of this task. A task's priority determines whether it gets activated or not. When one task's time slice is exhausted, the system finds the next, highest priority task to activate. For instance, a program with a priority of 4 will not receive any time until there are no tasks with priorities of 7, 6 and 5 waiting.

**`_set_slice`**   Sets the time slice allocated to this task. When a program is activated, it will receive this amount of processor time unless it a) gives up its time slice voluntarily, b) performs some operation that cannot be performed at this time because a resource is not available, or c) a resource becomes available that was needed by a higher priority task.

**Notes:**            Priority levels are in the range of 0 to 7. Slice times are in the range of 1 to 500 milliseconds.

                  Changing the time slice is a system-wide action because the same time slice is used for all tasks.

**Defaults:**        The default priority for a program is 4; the default slice time is defined in the system configuration file as maintained by the SYSGEN command.

**Conforms to:**        **`_setprty`**   q ANSI     q DOS     n THEOS    q POSIX  
                      **`_set_slice`**   q ANSI     q DOS     n THEOS    q POSIX

**See also:**           [yield, \\_pre\\_empty, \\_snu, \\_yield](#)

---

**Example:**

```
#include <stdio.h>
#include <sc.h>

void main()
{
    _setprty(2);        // this program is low priority
    ...
}
```

## setsockopt

Set a socket option.

```
#include <socket.h>
```

```
int setsockopt ( SOCKET s, int level, int optname, void * optval, int optlen )
```

---

|                |   |                                             |
|----------------|---|---------------------------------------------|
| <i>level</i>   | » | level at which option defined               |
| <i>optlen</i>  | » | size of the value to be set                 |
| <i>optname</i> | » | socket option number to be set              |
| <i>optval</i>  | » | pointer to location containing value to set |
| <i>s</i>       | » | socket number                               |

**Operation:** setsockopt sets the current value for a socket option associated with a socket of any type, in any state. Although options may exist at multiple protocol levels, this specification only defines options that exist at the uppermost “socket” level. Specifically, the THEOS Sockets API only allows the SOL\_SOCKET level to be set with this function.

**Returns:** A zero. When an error is detected, SOCKET\_ERROR is returned and the specific error code may be retrieved by using [TSAGetLastError](#).

**Errors:** When the return is SOCKET\_ERROR, the possible error codes returned by [TSAGetLastError](#) may be:

| <i>Code</i>    | <i>Meaning</i>                                                                                                                                                                                                                                  |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TSAENETDOWN    | The THEOS Sockets implementation has detected that the network subsystem has failed.                                                                                                                                                            |
| TSAEFAULT      | The <i>namelen</i> argument is incorrect.                                                                                                                                                                                                       |
| TSAEINPROGRESS | A blocking THEOS Sockets call is in progress.                                                                                                                                                                                                   |
| TSAEINVAL      | <i>level</i> is not valid or the information in <i>optval</i> is not valid.                                                                                                                                                                     |
| TSAENETRESET   | Connection has timed out when SO_KEEPALIVE is set.                                                                                                                                                                                              |
| TSANOPROTOOPT  | The option is unknown or unsupported. In particular, SO_BROADCAST is not supported on sockets of type SOCK_STREAM, while SO_ACCEPTCONN, SO_CONTLINGER, SO_KEEPALIVE, SO_LINKER and SO_OOINLINE are not supported on sockets of type SOCK_DGRAM. |
| TSAENOTCONN    | Connection has been reset when SO_KEEPALIVE is set.                                                                                                                                                                                             |
| TSAENOTSOCK    | The descriptor <i>s</i> is not a socket.                                                                                                                                                                                                        |

**Notes:** Options affect socket operations, such as whether expedited data is received in the normal data stream, whether broadcast messages may be sent on the socket, *etc.* There are two types of socket options: Boolean options that enable or disable a feature or behavior, and options which require an integer value or structure.

To enable a Boolean option, *optval* points to a non-zero integer. To disable the option, *optval* points to an integer equal to zero. *optlen* should be equal to `sizeof(int)` for Boolean options. For other options, *optval* points to the integer or structure that contains the desired value for the option, and *optlen* is the length of the integer or structure.

SO\_LINGER controls the action taken when unsent data is queued on a socket and a [closesocket](#) is performed. See [closesocket](#) for a description of the way in which the SO\_LINGER settings affect the semantics of [closesocket](#). The application sets the desired behavior by creating a `struct linger` (pointed to by the *optval* argument) with the following elements:

```
struct linger {
    int     l_onoff;
    int     l_linger;
}
```

To enable SO\_LINGER, the application should set *l\_onoff* to a non-zero value, set *l\_linger* to 0 or the desired timeout (in seconds), and call `setsockopt`. To enable SO\_DONTLINGER (i.e. disable SO\_LINGER), *l\_onoff* should be set to zero and `setsockopt` should be called.

By default, a socket may not be bound (see [bind](#)) to a local address which is already in use. On occasions, however, it may be desirable to “re-use” an address in this way. Since every connection is uniquely identified by the combination of local and remote addresses, there is no problem with having two sockets bound to the same local address as long as the remote addresses are different. To inform the THEOS Sockets implementation that a [bind](#) on a socket should not be disallowed because the desired address is already in use by another socket, the application should set the SO\_REUSEADDR socket option for the socket before issuing the [bind](#). Note that the option is interpreted only at the time of the [bind](#): It is therefore unnecessary (but harmless) to set the option on a socket that is not to be bound to an existing address, and setting or resetting the option after the [bind](#) has no effect on this or any other socket.

An application may request that the THEOS Sockets implementation enable the use of “keep-alive” packets on TCP connections by turning on the SO\_KEEPALIVE socket option. A THEOS Sockets implementation need not support the use of keep-alives. If it does, the precise semantics are implementation-specific but should conform to section 4.2.3.6 of RFC 1122: Requirements for Internet Hosts -- Communication Layers. If a connection is dropped as the result of “keep-alives,” the error code TSAENETRESET is returned to any calls in progress on the socket, and any subsequent calls will fail with TSAENOTCONN.

THEOS Sockets suppliers are encouraged (but not required) to supply output debug information if the SO\_DEBUG option is set by an application. The mechanism for generating the debug information and the form it takes are beyond the scope of this manual. The following options are sup-

ported for `setsockopt`. The Type identifies the type of data addressed by *optval*.

| <i>Value</i>  | <i>Type</i>         | <i>Meaning</i>                                                                                                                                |
|---------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| SO_BROADCAST  | BOOL                | Allow transmission of broadcast messages on the socket.                                                                                       |
| SO_DEBUG      | BOOL                | Record debugging information.                                                                                                                 |
| SO_DONTLINGER | BOOL                | Don't block close waiting for unsent data to be sent. Setting this option is equivalent to setting SO_LINGER with <i>l_onoff</i> set to zero. |
| SO_DONTROUTE  | BOOL                | Don't route: Send directly to interface.                                                                                                      |
| SO_KEEPAIVE   | BOOL                | Send keep-alives.                                                                                                                             |
| SO_LINGER     | struct LINGER FAR * | Linger on close if unsent data is present.                                                                                                    |
| SO_OOBINLIN   | BOOL                | Receive out-of-band data in the normal data stream.                                                                                           |
| SO_RCVBUF     | int                 | Specify buffer size for receives.                                                                                                             |
| SO_REUSEADDR  | BOOL                | Allow the socket to be bound to an address which is already in use.                                                                           |
| SO_SNDBUF     | int                 | Specify buffer size for sends.                                                                                                                |

BSD options not supported for `getsockopt` include:

| <i>Value</i>  | <i>Type</i> | <i>Meaning</i>                             |
|---------------|-------------|--------------------------------------------|
| SO_ACCEPTCONN | BOOL        | Socket is listening.                       |
| SO_ERROR      | int         | Get error status and clear.                |
| SO_RCVLOWAT   | int         | Receive low water mark.                    |
| SO_RCVTIMEO   | int         | Receive timeout.                           |
| SO_SNDLOWAT   | int         | Send low water mark.                       |
| SO_SNDTIMEO   | int         | Send timeout                               |
| SO_TYPE       | int         | The type of the socket (e.g. SOCK_STREAM). |
| IP_OPTIONS    |             | Set options in IP header.                  |
| TCP_MAXSEG    | int         | Get TCP maximum segment size.              |

**Conforms to:**      **setsockopt**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**      [bind](#), [closesocket](#), [getsockopt](#), [ioctlsocket](#), [socket](#)

**sgetl, sputl**

Get or put a long integer value from or to a buffer or string.

```
#include <stdlib.h>
long sgetl ( void * buffer )
long sputl ( long value, void * buffer )
```

---

*buffer*                   »   pointer to buffer or string  
*value*                   »   long integer value

**Operation:**       **sgetl**       Gets the long integer stored in *buffer*.

**sputl**       Puts the long integer *value* into *buffer*.

**Returns:**       Both functions return the long integer value that was either transferred from the buffer (sgetl) or transferred to the buffer (sputl).

**Notes:**        These functions transfer long integer values to or from a buffer in a machine-independent manner. That is, the storage sequence of the bytes comprising the value are stored in a manner that is independent of the normal storage sequence for a particular CPU. These functions are used as a set to maintain long integers in a buffer that is read from or to external storage and used on other computer systems.

|                     |              |        |       |         |         |
|---------------------|--------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>sgetl</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>sputl</b> | q ANSI | q DOS | n THEOS | q POSIX |

## shared

Find or allocate a region of named, shared memory in the system memory pool.

```
#include <stdlib.h>
unsigned short shared ( const char _far * name, size_t len )
```

---

*len*                   »   size of memory requested  
*name*                 »   name of shared memory

**Operation:**   A case-insensitive search is performed for a shared memory region with an assigned name of *name*. The actions performed depend upon the result of this search and the value of *len*:

| <i>name</i> | <i>len</i> | Action                                                 |
|-------------|------------|--------------------------------------------------------|
| Found       | > 0        | Location returned.                                     |
|             | 0          | Memory deallocated.                                    |
|             | -1         | Location returned.                                     |
| Not found   | > 0        | Memory allocated to <i>len</i> size, cleared to zeros. |
|             | 0          | Zero returned.                                         |
|             | -1         | Zero returned.                                         |

**Returns:**   When *len* is non-zero, the segment address of the memory is returned.

There is no indication whether the memory was allocated by this call to **shared** or whether a previous, other user's call to **shared** caused the allocation.

**Errors:**   A zero value is returned when *len* is positive, a match was not found and there is insufficient memory available to satisfy the request.

**Notes:**   Shared memory segments are accessible by any task that knows the segment's name.

It is the responsibility of the applications using shared memory to devise and implement their own management of the memory segment. This pertains to allocation and deallocation and the memory segment's data integrity. One standard method might be to use the first few bytes of the segment as a table indicating its size, usage counter and current activity.

The [semaphore functions](#) can also be utilized for intertask communication and control of this shared memory.

The [\\_lockres](#), [\\_lockset](#), [\\_lockshare](#), [\\_locktest](#), [\\_lockwait](#) functions should be utilized to ensure data integrity.

When a match is found with an existing shared memory segment, the size might be different from your requested size. The [\\_getlimit](#) function can be used to determine the actual size of the shared memory.

## 572 *shared*

---

- Defaults:** When first allocated, shared memory is cleared to zeros.
- Shared memory segments remain in memory until a specific request is made to deallocate them or until the system is restarted.
- Restrictions:** Only the first 16 characters of *name* are used.
- Once a shared memory region is allocated, its size cannot be changed.
- The results are unpredictable if a shared memory segment is deallocated while another task or user is still utilizing it. The probable result will be an address error the next time that the other task tries to access the deallocated memory.
- The -1 value for *len* is only supported on THEOS 32 Version 4 or above.
- Conforms to:**                **shared**    q ANSI    q DOS    n THEOS    q POSIX
- See also:**                [\\_getmem](#), [\\_putmem](#)

---

**Example:**                See the [\\_mem\\_grow](#) example.



## signal

Specify how interrupt signals are handled.

```
#include <signal.h>
void * signal ( int signal, void (*func)(int) )
```

---

*func*                   »   name of function to invoke  
*signal*                »   signal value

**Operation:**       *signal* specifies the interrupt signal value and *func* specifies the action to be taken when the signal occurs. This information is saved and used when the event occurs.

**Returns:**         The prior value for *signal*.

**Errors:**          A return value of -1 indicates an error and [errno](#) is set to EINVAL.

**Notes:**          The possible values for *signal* are defined in the SIGNAL.H file:

| <i>signal</i>                                 | <i>Meaning</i>                                                 |
|-----------------------------------------------|----------------------------------------------------------------|
| SIGABRT <sup>A</sup>                          | Abnormal program termination. Maps into SIGTERM.               |
| SIGALRM                                       | <a href="#">alarm</a> or <a href="#">msalarm</a> timer runout. |
| SIGBOUND                                      | Array-bound error used by MultiUser BASIC.                     |
| SIGCAN                                        | Synonym to SIGINT signal.                                      |
| SIGCRIT                                       | Input/output critical error. Maps into SIGTERM.                |
| SIGDIV0                                       | Divide by zero.                                                |
| SIGFPE <sup>A</sup>                           | Floating-point error. Maps into SIGTERM.                       |
| SIGILL <sup>A</sup>                           | Illegal instruction. Maps into SIGTERM.                        |
| SIGINT <sup>A</sup>                           | <a href="#">Break</a> , <a href="#">C</a> entered by operator. |
| SIGOVF                                        | Mathematical overflow or underflow occurred.                   |
| SIGQUIT                                       | Synonym to SIGTERM signal.                                     |
| SIGSEGV <sup>A</sup>                          | Memory segment violation (GP13 or SP12). Maps into SIGTERM.    |
| SIGTERM <sup>A</sup>                          | <a href="#">Break</a> , <a href="#">Q</a> entered by operator. |
| <sup>A</sup> Conforms to ANSI specifications. |                                                                |

The value of *func* may be the name of a function that you want invoked when the event occurs or it may be one of the standard actions as defined in SIGNAL.H:

| <i>func</i> | <i>Meaning</i>                                                                                                                                                                                   |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SIG_DFL     | Perform default action for signal. Except for SIGQUIT and the signals that are mapped into SIGQUIT, the default is to ignore the event. The default for SIGQUIT is to exit with return code 254. |
| SIG_ERR     | Ignore the event. The process generating the event receives an invalid signal call.                                                                                                              |
| SIG_IGN     | Ignore the event.                                                                                                                                                                                |

When a user-supplied function is specified for a signal and that signal occurs, the signal is set to SIG\_DFL and then the function is invoked. Upon normal exit from that function, the calling process resumes execution immediately following the point at which it received the interrupt signal.

The user-supplied function should expect one argument: the value of *signal* that caused its invocation.

**Defaults:** The default for all signals except SIGQUIT is to ignore the event. The default for SIGQUIT is to exit with return code 254.

**Restrictions:** User-supplied signal-handling functions should be coded to perform their task quickly, just as if they were an interrupt-service routine.

**Conforms to:** **signal**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [alarm](#), [msalarm](#), [raise](#)

---

**Example:**

```
#include <signal.h>

void cancel(int sig)
{
    ...                // code to handle Break,C

    signal(SIGCAN, cancel);
}

void
main()
{
    signal(SIGCAN, cancel);    // trap break,c entry
    ...
}
```

## sin, sinh

Compute the sine or hyperbolic sine of an angle.

```
#include <math.h>
double sin ( double x )
double sinh ( double x )
```

---

*x*                      »   floating-point angle, in radians

**Operation:**      **sin**              Compute the sine of the angle *x*.

**sinh**            Compute the hyperbolic sine of the angle *x*.

**Returns:**        **sin**              The sine of the angle *x*.

**sinh**            The hyperbolic sine of the angle *x*.

**Errors:**            When *x* is too large to produce meaningful results, a zero is returned and [errno](#) is set to ERANGE.

**Notes:**            The sine of an angle is always in the range of  $-1$  to  $+1$ .

This function uses BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                      **sin**      n ANSI      n DOS      n THEOS      n POSIX  
                                              **sinh**      n ANSI      n DOS      n THEOS      n POSIX

**See also:**            [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [tan](#), [tanh](#)

**skipsp**

Find the next non-white space character in a string.

```
#include <stdlib.h>
```

```
char * skipsp ( char * string )
```

---

*string*                    »    pointer to string containing text

**Operation:**            Using the [isspace](#) function, *string* is scanned until [isspace](#) reports false.

**Returns:**             A pointer to the first non-white space character in *string*.

If *string* contains only white space characters, the value returned is a pointer to the string-terminating character.

**Conforms to:**                **skipsp**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                [isspace](#)

## **sleep, sleep\_msec, sleep\_sec, sleep\_until**

Suspend execution of the program for a period of time.

```
#include <timer.h>
void sleep ( long msec )
```

```
#include <stdlib.h>
void sleep_msec ( unsigned msecs )
void sleep_sec ( unsigned secs )
void sleep_until ( unsigned time_of_day )
```

---

|                    |   |                                            |
|--------------------|---|--------------------------------------------|
| <i>msec</i>        | » | number of milliseconds to sleep            |
| <i>msecs</i>       | » | number of milliseconds to sleep            |
| <i>secs</i>        | » | number of seconds to sleep                 |
| <i>time_of_day</i> | » | time of day in milliseconds to sleep until |

**Operation:**     **sleep**     The execution of your program is suspended until *msecs* milliseconds of time has elapsed.

**sleep\_msec** The execution of your program is suspended until *msec* milliseconds of time has elapsed.

**sleep\_sec**     The execution of your program is suspended until *secs* seconds of time has elapsed.

**sleep\_until**   The execution of your program is suspended until the system clock is equal to *time\_of\_day*. The *time\_of\_day* is specified as the number of milliseconds since midnight. If that time has already passed, the program sleeps until that time of the next day.

**Notes:**             Some useful values for usage with the **sleep** and **sleep\_msec** functions:

| <i>msecs</i> | <i>Interval of Time</i> |
|--------------|-------------------------|
| 1,000        | 1 second                |
| 60,000       | 1 minute                |
| 300,000      | 5 minutes               |
| 1,800,000    | 30 minutes              |
| 3,600,000    | 1 hour                  |
| 86,400,000   | 1 day                   |

**sleep**             An interrupt from one of the functions [alarm](#), [clock](#), [delay](#), [msalarm](#), [signal](#) or a semaphore-awakening action will awaken a sleeping program.

**sleep\_msec**     The sleep time is translated into a time-of-day and passed to **sleep\_until**.

## 578 *sleep, sleep\_msec, sleep\_sec, sleep\_until*

---

Will not wake up until the interval has elapsed. Interrupts will still be processed but the program will return to the sleep state.

**sleep\_sec** The sleep time is translated into milliseconds and passed to sleep\_msec, which will pass it to sleep\_until.

Will not wake up until the interval has elapsed. Interrupts will still be processed but the program will return to the sleep state.

**sleep\_until** Will not wake up until the interval has elapsed. Interrupts will still be processed but the program will return to the sleep state.

**Restrictions:** Care should be taken with large values of *msec*, *msecs* or *secs*. It is possible to suspend your program for very long periods of time (multiple days).

|                     |                    |        |       |         |         |
|---------------------|--------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>sleep</b>       | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>sleep_msec</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>sleep_sec</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>sleep_until</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [alarm](#), [clock](#), [delay](#), [msalarm](#), [signal](#), [semaphore](#) functions

## socket

Create a socket.

```
#include <socket.h>
SOCKET socket ( int af, int type, int protocol )
```

---

*af* » address format specification  
*protocol* » protocol to be used with the socket  
*type* » type specification for the new socket

**Operation:** socket allocates a socket descriptor of the specified address family, data type and protocol, as well as related resources. If a *protocol* is not specified (i.e. equal to 0), the default for the specified connection mode is used. Only a single protocol exists to support a particular socket type using a given address format. The protocol number to use is unique to the “communication domain” in which communication is to take place.

**Returns:** A descriptor referencing the new socket. If an error occurs, INVALID\_SOCKET is returned and a specific error code may be retrieved by calling [TSAGetLastError](#).

**Notes:** The following *type* specifications are supported:

| <i>Code</i>    | <i>Meaning</i>                                                                                                                                                 |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SOCK_STREAM    | Provides sequenced, reliable, two-way connection-based byte streams with an out-of-band data transmission mechanism. Uses TCP for the Internet address family. |
| SOCK_DGRAM     | Supports datagrams, which are connectionless, unreliable buffers of a fixed length. Uses UDP for the Internet address family.                                  |
| SOCK_RAW       | Raw protocol interface.                                                                                                                                        |
| SOCK_RDM       | Reliably-delivered message.                                                                                                                                    |
| SOCK_SEQPACKET | Sequenced packet stream.                                                                                                                                       |

Sockets of type SOCK\_STREAM are full-duplex byte streams. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a [connect](#) call. Once connected, data may be transferred using [send](#) and [recv](#) calls. When a session has been completed, a [closesocket](#) must be performed. Out-of-band data may also be transmitted as described in [send](#) and received as described in [recv](#).

The communications protocols used to implement a SOCK\_STREAM ensure that data is not lost or duplicated. If data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken and subsequent calls will fail with the error code set to TSAETIMEDOUT.

SOCK\_DGRAM sockets allow sending and receiving of datagrams to and from arbitrary peers using [sendto](#) and [recvfrom](#). If such a socket is connected

to a specific peer, datagrams may be sent to that peer with [send](#) and may be received from (only) this peer using [recv](#).

The following *protocol* specifications are supported:

| <i>Code</i>  | <i>value</i> | <i>Meaning</i>                 |
|--------------|--------------|--------------------------------|
| IPPROTO_IP   | 0            | Default, IP protocol.          |
| IPPROTO_ICMP | 1            | Control message protocol.      |
| IPPROTO_TCP  | 6            | Transmission control protocol. |
| IPPROTO_UDP  | 17           | User datagram protocol         |
| IPPROTO_RAW  | 255          | Raw IP packet.                 |

Refer to the `SOCKET.H` header file for information about additional, pre-defined names for socket numbers, functions, options, *etc.*

#### Errors:

When the return is `INVALID_SOCKET`, the possible error codes returned by [TSAGetLastError](#) may be:

| <i>Code</i>        | <i>Meaning</i>                                                                                  |
|--------------------|-------------------------------------------------------------------------------------------------|
| TSAEAFNOSUPPORT    | The specified address family is not supported.                                                  |
| TSAEINPROGRESS     | A blocking THEOS Sockets call is in progress.                                                   |
| TSAEMFILE          | The queue is empty upon entry to <a href="#">accept</a> and there are no descriptors available. |
| TSAENETDOWN        | The THEOS Sockets implementation has detected that the network subsystem has failed.            |
| TSAENOBUFFS        | No buffer space is available.                                                                   |
| TSAEPROTONOSUPPORT | The specified protocol is not supported.                                                        |
| TSAEPROTOTYPE      | The specified protocol is the wrong type for this socket.                                       |
| TSAESOCKTNOSUPPORT | The specified socket type is not supported in this address family.                              |

**Conforms to:**                      **socket**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                      [accept](#), [bind](#), [connect](#), [getsockname](#), [getsockopt](#), [ioctlsocket](#), [listen](#), [recv](#), [recvfrom](#), [select](#), [send](#), [sendto](#), [setsockopt](#)



**Example:**

ECHOSERVER.C program:

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <socket.h>

SOCKET      tcp_sock,
            cli_sock;
SOCKADDR_IN serv_addr,
            cli_addr;
int         cli_len;

void process(void) {
    int  len;
    char buf[80];

    while (1) {          // read data until connection is broken

        len = recv(cli_sock, buf, sizeof(buf), 0);
        if (len == 0)          // normal disconnect?
            return;
        if (len == SOCKET_ERROR) {    // socket error?
            switch (TSAGetLastError()) {
                case TSAECONNABORTED:
                case TSAECONNRESET:
                    return;
                default:
                    fprintf(stderr, "recv() %s\n",
                        TSAStrError(TSAGetLastError()));
                    return;
            }
        }
        send(cli_sock, buf, len, 0); // echo the data
    }
}

main(void) {

    // get a TCP socket

    tcp_sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (tcp_sock == SOCKET_ERROR) {
        fprintf(stderr, "socket() %s\n",
            TSAStrError(TSAGetLastError()));
        exit(1);
    }

    // fill in info and bind

    bzero(&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(1234);
    if (bind(tcp_sock, (SOCKADDR*)&serv_addr, sizeof(serv_addr))
        == SOCKET_ERROR) {
```

```
        fprintf(stderr, "bind() %s\n",
            TSAStrError(TSAGetLastError()));
        exit(1);
    }

    // set up a listen

    if (listen(tcp_sock, 5) == SOCKET_ERROR) {
        fprintf(stderr, "listen() %s\n",
            TSAStrError(TSAGetLastError()));
        exit(1);
    }

    // wait for connection

    while (1) {
        cli_len = sizeof(cli_addr);
        cli_sock = accept(tcp_sock, (SOCKADDR*)&cli_addr, &cli_len);
        if (cli_sock == SOCKET_ERROR) {
            fprintf(stderr, "accept() %s\n",
                TSAStrError(TSAGetLastError()));
            exit(1);
        }

        // launch task to do the echo

        switch (fork()) {
            case 0:                // child
                closesocket(tcp_sock); // close parent socket
                process();           // process the echo
                closesocket(cli_sock); // close my socket
                killtask(getpid());   // kill myself

            default:               // parent
                closesocket(cli_sock); // close my socket
                break;

            case -1:              // error
                fprintf(stderr, "Cannot fork() new process\n");
                exit(1);
        }
    }
}
```

ECHOCLIENT.C program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <socket.h>

char        *host;
SOCKET      cli_sock;
SOCKADDR_IN serv_addr;
IN_ADDR     host_ip;
HOSTENT     *hp;
```

```

extern void process(void);

main(int argc, char *argv[]) {
    // get host name

    switch (argc) {
        case 1: // none, use self
            host = "localhost";
            break;
        case 2: // save the name
            host = argv[1];
            break;
        default: // too many args
            fprintf(stderr, "Syntax: ECHOCLIENT hostname\n");
            exit(1);
    }

    if (isalpha(host[0])) { // name or ip address?
        hp = gethostbyname(host); // lookup the name
        if (!hp) { // not found?
            fprintf(stderr, "Cannot resolve host: %s\n", host);
            exit(1);
        }
        host_ip = *(IN_ADDR*)hp->h_addr; // get ip address
    }
    else
        host_ip.s_addr = inet_addr(host); // ip addr from cmd

    // get a socket

    cli_sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (cli_sock == SOCKET_ERROR) { // failure?
        fprintf(stderr, "ECHOCLIENT socket() %s\n",
            TSAStrError(TSAGetLastError()));
        exit(1);
    }

    // fill in the fields and do connect

    bzero(&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET; // family = Internet
    serv_addr.sin_addr.s_addr = host_ip.s_addr; // server IP addr
    serv_addr.sin_port = htons(1234); // server port
    if (connect(cli_sock, (SOCKADDR*)&serv_addr, sizeof(serv_addr))
        == SOCKET_ERROR) {
        fprintf(stderr, "ECHOCLIENT connect() %s\n",
            TSAStrError(TSAGetLastError()));
        exit(1);
    }

    // do the work

    process();

    // finished, close the socket and exit

    closesocket(cli_sock);
    exit(0);
}

```

```
void process(void) {
    int    len,
           sndlen,
           rcvlen;
    char   buf[1024];

    while (1) {

        // if stdin is console, put a prompt of ">"
        if (iscon(stdin))
            printf(">");

        // read a line from stdin

        if (!fgets(buf, sizeof(buf), stdin))
            break;                // end of file, break loop
        sndlen = strlen(buf);      // compute length

        // send data to server

        if (send(cli_sock, buf, sndlen, 0) == SOCKET_ERROR) {
            fprintf(stderr, "ECHOCLIENT send() %s\n",
                    TSAStrError(TSAGetLastError()));
            exit(1);
        }

        // if stdout is display, put "<"
        if (iscon(stdin) && iscon(stdout))
            printf("<");

        // read response

        for (len=0; len<sndlen; len += rcvlen) {
            rcvlen = recv(cli_sock, buf, sndlen-len, 0);
            if (rcvlen == SOCKET_ERROR) {
                fprintf(stderr, "ECHOCLIENT recv() %s\n",
                        TSAStrError(TSAGetLastError()));
                exit(1);
            }
            buf[rcvlen] = 0;        // mark end of string
            printf("%s", buf);      // display on stdout
        }
    }
}
```

**Output:**

This first example uses keyboard entry (stdio not redirected).

```
>echoclient randd.theos-software.com
>This is the first line
<This is the first line
>Last line (then I will press F4 to close)
<Last line (then I will press F4 to close)

>
```

This second example uses keyboard entry (stdin is redirected).

```
>echoclient randd.theos-software.com < dt.c
#include <stdio.h>
#include <time.h>
#include <timer.h>

main() {
    time_t t;
    char d[9];

    printf("sc_time %s\n", gettime(d));
    time(&t);
    printf("localtime %s",asctime(localtime(&t)));
    printf("gmtime %s",asctime(gmtime(&t)));
    printf("daylight = %d\n",daylight);
    printf("timezone = %d\n",timezone);
}

>
```

**sort**

Sort an array of pointers to objects.

```
#include <stdlib.h> or <search.h>
void sort ( void * array, size_t count, int (*compar)() )
```

---

|               |   |                                         |
|---------------|---|-----------------------------------------|
| <i>array</i>  | » | pointer to array of pointers to objects |
| <i>compar</i> | » | name of comparison routine to use       |
| <i>count</i>  | » | number of members remaining in array    |

**Operation:** This function operates similar to the [qsort](#) function except that the objects are not rearranged when sorted, only the pointers to the objects are rearranged.

Perform a “quick sort” of the objects pointed to by the array of pointers in *array*. The function *compar* is used to determine the relative order of any two items pointed to in *array*. The *array* is overwritten with the reordered pointers.

**Notes:** The *compar* routine must be a function that uses a declaration similar to:

```
int compar(const void *memb1, const void *memb2)
```

*memb1* and *memb2* are two values from *array* which are pointers to the objects to compare. *compar* must return an integer that is:

```
< 0 when memb1 < memb2
= 0 when memb1 = memb2
> 0 when memb1 > memb2
```

The elements of *array* are sorted in ascending order, as defined by the *compar* routine. To sort the array in decreasing order, reverse the meaning of “>” and “<” in the comparison function.

The array may be sorted by a field other than the first field of each member. The sorting criteria is exclusively controlled by the *compar* function.

**Conforms to:** **sort** q ANSI q DOS n THEOS q POSIX

**See also:** [bsearch](#), [lsearch](#), [lfind](#), [qsort](#)

## spawn functions

Start another program as a subtask of the current program.

```
#include <process.h>
```

```
int spawnl ( int mode, char * program, char * arg0, ... char * argn, NULL )
```

```
int spawnlp ( int mode, char * path, char * arg0, ... char * argn, NULL )
```

```
int spawnv ( int mode, char * program, char * argv[] )
```

```
int spawnvp ( int mode, char * path, char * argv[] )
```

---

|                       |                                                                       |
|-----------------------|-----------------------------------------------------------------------|
| <i>arg0, ... argn</i> | » list of character pointers to command-line arguments                |
| <i>argv</i>           | » pointer to an array of character pointers of command-line arguments |
| <i>mode</i>           | » flag defining execution mode of parent program                      |
| <i>path</i>           | » character pointer to program path name                              |
| <i>program</i>        | » character pointer to program name                                   |

**Operation:** Each of these functions loads and executes another program as a subtask (child) of the current (parent) program. The *mode* argument specifies the actions taken by the parent during and after the child is spawned. Possible values for *mode* are:

| <i>mode</i> | Description                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| P_NOWAIT    | Parent program continues execution concurrently with child process.                                                  |
| P_OVERLAY   | Child process overlays parent process, destroying the parent. This is the same as if an EXEC function had been used. |
| P_WAIT      | Parent program suspends operation until child process terminates.                                                    |

These value names are defined in PROCESS.H.

These four functions operate similarly, with the following differences:

**spawnl** The *program* argument must be a complete specification for the program file. It must specify a file-name, file-type and, if appropriate, member-name. The file-drive need not be specified if it can be found in the normal drive search sequence.

Command-line arguments to *program* are specified as a list of pointers, one to each argument in sequence. The list is terminated with a NULL pointer.

**spawnlp** The *path* may be a complete specification or a simple program name. When *path* is a simple name, the normal search sequence for programs is used, first on the system drive and then on all other drives in the drive search sequence. The search sequence used is similar to that used by the CSI when searching for command names. It differs in that the synonym/abbreviation file is not used and no search is made for EXEC programs.

Similar to [execl](#), command-line arguments to *path* are specified as a list of pointers, one to each argument in sequence. The list is terminated with a NULL pointer.

**spawnv** Similar to [execl](#), *program* is a complete specification for the program file.

Command-line arguments to *program* are specified as a pointer to an array of character pointers to each argument. The array is terminated with a NULL pointer.

**spawnvp** Similar to [execvp](#), *path* is a complete specification or a simple program name.

Similar to `spawnv`, command-line arguments to *path* are specified as a pointer to an array of pointers to characters strings. The array is terminated with a NULL pointer.

**Returns:** The return value of a spawn function call depends upon the *mode* used. Synchronous spawns (*mode* is `P_WAIT`) return the exit status of the child program. The return of an asynchronous spawn (*mode* is `P_NOWAIT`) is the child process ID.

There is no return when the *mode* is `P_OVERLAY`.

**Errors:** A return value of -1 indicates an error and that the child process did not start. The [errno](#) variable is set to one of the following values to indicate the cause of the error:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>                                                         |
|-------------|--------------|------------------------------------------------------------------------|
| EINVAL      | 428          | <i>mode</i> argument value invalid.                                    |
| ENOENT      | 19           | The <i>program</i> or <i>path</i> not found.                           |
| ENOMEM      | 3            | Insufficient memory or no available process id to start child process. |

**Notes:** Files that are open when a spawn call is made remain open in the child process. The child process also inherits the environment of the parent.

Signal settings are not preserved in the child process. They are reset to the default signals.

You should flush (using [flush](#) or [flushall](#)) any open stream files prior to invoking a spawn function.

**Restrictions:** The list of command-line arguments or the array of command line arguments must be terminated with a NULL pointer.

**Conforms to:**

|                |        |       |         |         |
|----------------|--------|-------|---------|---------|
| <b>spawnl</b>  | q ANSI | n DOS | n THEOS | q POSIX |
| <b>spawnlp</b> | q ANSI | n DOS | n THEOS | q POSIX |
| <b>spawnv</b>  | q ANSI | n DOS | n THEOS | q POSIX |
| <b>spawnvp</b> | q ANSI | n DOS | n THEOS | q POSIX |

**See also:** [abort](#), [atexit](#), [\\_atexit\\_clear](#), [\\_atexit\\_remove](#), [exec](#) functions, [exit](#), [flush](#), [flushall](#), [fflush](#), [fork](#), [forktask](#), [system](#)



**Example:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern void process(char *);

void main(int argc, char **argv) {
    int    i,
           count=0;
    char   *fn,
           **list = NULL;

    diropen(argv[1]);           // open the directory
    while (fn = dirread()) {    // get next item
        if (!(count % 50))      // need to grow list?
            list = realloc(list+50*sizeof(char **));
        list[count++] = strdup(fn); // save the name
    }
    dirclose();                 // close the directory

    sort(list, count, strcmp); // sort the list of names

    for (i=0; i<count; ++i) {  // one at a time
        fn = list[i];          // get name
        process(fn);            // go do something to file
    }
}
```

**sprintf**

Write formatted characters, strings, literal text and numbers to a string buffer.

```
#include <stdio.h>
int sprintf ( char * string, char * format [, argument-list ] )
```

---

|                      |   |                                        |
|----------------------|---|----------------------------------------|
| <i>string</i>        | » | pointer to a string buffer             |
| <i>format</i>        | » | formatting control mask                |
| <i>argument-list</i> | » | optional arguments to format and print |

**Operation:**        Format the *arguments* using the *format* mask and write to *string*.

**Returns:**         The number of characters written.

**Notes:**           This function operates identically to the [printf](#) function described on page 484 except that the resulting formatted string is written to *string* rather than to stdout.

**Conforms to:**        **sprintf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [cprintf](#), [fprintf](#), [printf](#), [vfprintf](#), [vprintf](#), [vsprintf](#)

---

**Example:**         See also the [printf](#) examples.

```
#include <stdio.h>

main()
{
    char work[256];

    sprintf(work, "$%,.2f",123456.789); // format as fixed point
    printf("\nThe dollar amount is: %s",work);
}
```

---

**Output:**

>sample

The dollar amount is: \$123,456.79

## sqrt

Compute the square root of a value.

```
#include <math.h>
double sqrt ( double x )
```

---

*x*                      »   floating-point value

**Operation:**        Compute the square root of *x*.

**Returns:**           The square root of *x*. A zero is returned if *x* is negative.

**Errors:**            When *x* is negative, [errno](#) is set to EDOM.

**Notes:**            This function uses BCD or IEEE arithmetic, depending upon the current `#pragma float bcd` or `#pragma float ieee` directive.

**Conforms to:**                **sqrt**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [exp](#), [log](#), [log2](#), [log10](#), [pow](#)

---

### Example:

```
#include <stdio.h>
#include <math.h>

void
main()
{
    double x,
           x_root;

    printf("\nEnter numeric value: ");
    scanf("%lf",&x);

    x_root=sqrt(x);

    if (errno==EDOM)
        printf("Negative value entered\n");
    else
        printf("The square root of %f is %f\n",x,x_root);
}
```

---

### Output:

>example

```
Enter numeric value: 123.456
The square root of 123.456000 is 11.111076
```

>

**sscanf**

Accept formatted data from a string.

```
#include <stdio.h>
```

```
int sscanf ( char * string, const char * mask, argument... )
```

---

|             |   |                                         |
|-------------|---|-----------------------------------------|
| <i>mask</i> | » | pointer to string containing input mask |
|-------------|---|-----------------------------------------|

|               |   |                                 |
|---------------|---|---------------------------------|
| <i>string</i> | » | pointer to string to be scanned |
|---------------|---|---------------------------------|

|                    |   |                                                 |
|--------------------|---|-------------------------------------------------|
| <i>argument...</i> | » | pointers to storage locations for items scanned |
|--------------------|---|-------------------------------------------------|

**Operation:** Identical in operation to the [scanf](#) function except that the input source is *string*, not stdin. Data is read from *string* and converted according to the format specifications in *mask*. The converted data is saved in the locations pointed to by the *argument* list.

**Returns:** The number of fields successfully converted and assigned. The return value includes only the fields read and assigned.

**Errors:** A return value of EOF means that an attempt was made to read past the end-of-string was detected before the first conversion. A return value of zero means that no fields were assigned.

**Conforms to:**                    **sscanf**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                    [fscanf](#), [scanf](#)

## ***\_std, \_cld assembly functions***

These “functions” generate in-line code to change the direction flag used by the CPU.

```
#include <builtin.h>
void _cld ( void )
void _std ( void )
```

**Operation:** Sets or clears the direction flag used by the CPU for multibyte instructions.

**\_cld** Clear the direction flag.

**\_std** Set the direction flag.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the `call` and `ret` instructions or the inconvenience of the `#asm` and `#endasm` directives.

All of the [far memory functions](#) and the [memory functions](#) are affected by the CPU direction flag set or cleared by these functions.

**Defaults:** The normal or default mode of the direction flag is cleared.

**Restrictions:** These functions are intended for use by device-driver authors.

The direction flag should not be set for long periods of time. The THEOS C library functions assume that the flag is cleared.

|                     |             |        |       |         |         |
|---------------------|-------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_cld</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_std</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [far memory functions](#), [memory functions](#)

**store memory assembly functions**

This group of “functions” generates in-line code to store or copy a byte, short integer or long integer to a specified memory area.

```
#include <driver.h>
void _stosb ( void _far * far_offset, char bvalue, size_t count )
void _stosd ( void _far * far_offset, long lvalue, size_t count )
void _stosw ( void _far * far_offset, int ivalue, size_t count )
```

---

|                   |   |                               |
|-------------------|---|-------------------------------|
| <i>bvalue</i>     | » | byte value                    |
| <i>count</i>      | » | number of objects to store    |
| <i>far_offset</i> | » | pointer to remote data object |
| <i>ivalue</i>     | » | short integer value           |
| <i>lvalue</i>     | » | long integer value            |

**Operation:** These functions store a value to a series of memory locations in a specified memory segment.

**\_stosb** Stores *count* copies of the single-byte value of *bvalue* starting in the location *far\_offset*.

**\_stosd** Stores *count* copies of the long integer value of *lvalue* starting in the location *far\_offset*.

**\_stosw** Stores *count* copies of the short integer value of *ivalue* starting in the location *far\_offset*.

**Returns:** No value is returned by these functions.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive usage of the call and ret instructions.

The argument *far\_offset* is a void pointer. This means that the functions can store whatever type of object desired to any location. For instance, a *\_stosw* function can store a word value (double-byte value) to a location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program’s code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_stosb</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_stosd</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_stosw</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [subtract memory assembly functions](#), [xor memory assembly functions](#)

**string functions**

This group of functions manipulates null-terminated strings. They can copy, append, create, duplicate, compare, initialize, *etc.* strings in your data space.

```
#include <string.h>
char * strcat ( char * string1, const char * string2 )
char * strchr ( const char * string, char c )
int strcmp ( const char * string1, const char * string2 )
char * strcpy ( char * string1, const char * string2 )
size_t strcspn ( const char * string1, const char * string2 )
int strdcmp ( const char * string1, const char * string2 )
char * strdup ( const char * string )
char * strend ( const char * string )
int streq ( const char * string1, const char * string2 )
int stricmp ( const char * string1, const char * string2 )
int strieq ( const char * string1, const char * string2 )
char * stristr ( const char * string1, const char * string2 )
size_t strlen ( const char * string )
char * strlwr ( char * string )
char * strmake ( char * string1, const char * string2, size_t len )
int strmatch ( const char * string, const char * mask )
char * strncat ( char * string1, const char * string2, size_t len )
int strncmp ( const char * string1, const char * string2, size_t len )
char * strncpy ( char * string1, const char * string2, size_t len )
int strneq ( const char * string1, const char * string2, size_t len )
int strnicmp ( const char * string1, const char * string2, size_t len )
char * strnset ( char * string, char c, size_t len )
char * strpbrk ( const char * string1, const char * string2 )
char * strrchr ( const char * string, char c )
char * strset ( char * string, char c )
size_t strspn ( const char * string1, const char * string2 )
char * strstr ( const char * string1, const char * string2 )
char * strtok ( char * string1, const char * string2 )
char * strtrim ( char * string )
char * strupr ( char * string )
```

|                            |   |                                            |
|----------------------------|---|--------------------------------------------|
| <i>c</i>                   | » | character value to set string to           |
| <i>len</i>                 | » | length of string to use                    |
| <i>mask</i>                | » | pointer to string containing match pattern |
| <i>string</i>              | » | pointer to string                          |
| <i>string</i> <sub>1</sub> | » | pointer to first string                    |
| <i>string</i> <sub>2</sub> | » | pointer to second string                   |

**Operation:**      **strcat**      The contents of *string*<sub>2</sub> are copied to the end of *string*<sub>1</sub>.

**strchr**      *string* is searched for the first occurrence of the character value *c*.



|                |                                                                                                                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>strcmp</b>  | The contents of <i>string</i> <sub>1</sub> are compared with the contents of <i>string</i> <sub>2</sub> . A greater than, equal to, or less than result is returned.                                                                                |
| <b>strcpy</b>  | The contents of <i>string</i> <sub>2</sub> are copied to <i>string</i> <sub>1</sub> , replacing the prior contents of <i>string</i> <sub>1</sub> .                                                                                                  |
| <b>strcspn</b> | Similar to the function <code>strpbrk</code> , the contents of <i>string</i> <sub>1</sub> are searched for any occurrence of a character that is also in <i>string</i> <sub>2</sub> . The null-terminating character is not included in the search. |
| <b>strdcmp</b> | This function is identical in operation to the <code>strcmp</code> function unless there are ASCII decimal digits in the strings.                                                                                                                   |

When *string*<sub>1</sub> and *string*<sub>2</sub> contain ASCII decimal digits starting in the same relative location, the strings will compare as if the portions containing the decimal digits were the same length, with zeros added to pad the shorter string of digits to the same length as the other string's string of digits. For instance:

| Original set | with <code>strcmp</code> : | with <code>strdcmp</code> : |
|--------------|----------------------------|-----------------------------|
| a1xxx        | a101xxx                    | a1xxx                       |
| a83xxx       | a11xxx                     | a11xxx                      |
| a11xxx       | a1xxx                      | a83xxx                      |
| a101xxx      | a83xxx                     | a101xxx                     |
| b15xxx       | b100xxx                    | b15xxx                      |
| b100xxx      | b15xxx                     | b100xxx                     |

|                |                                                                                                                                                                                                                                          |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>strdup</b>  | Using the <code>malloc</code> function, an area of memory is allocated sufficient to hold a copy of <i>string</i> . The contents of <i>string</i> , along with its terminator, are then copied to this new area.                         |
| <b>strend</b>  | The null-terminating character of <i>string</i> is located.                                                                                                                                                                              |
| <b>streq</b>   | The contents of <i>string</i> <sub>1</sub> are compared with the contents of <i>string</i> <sub>2</sub> . Only equality or inequality is determined.                                                                                     |
| <b>stricmp</b> | A “case-insensitive” comparison is performed. The lowercased contents of <i>string</i> <sub>1</sub> are compared with the lowercased contents of <i>string</i> <sub>2</sub> . A greater-than, equal-to, or less-than result is returned. |
| <b>strieq</b>  | A “case-insensitive” comparison is performed between the contents of <i>string</i> <sub>1</sub> and the contents of <i>string</i> <sub>2</sub> . Only equality or inequality is determined.                                              |
| <b>stristr</b> | A “case-insensitive” substring search is performed. The lowercased contents of <i>string</i> <sub>1</sub> are searched for the first occurrence of the lowercased contents of <i>string</i> <sub>2</sub> .                               |
| <b>strlen</b>  | Determines the current length of <i>string</i> , not including the null-terminating character.                                                                                                                                           |
| <b>strlwr</b>  | Converts the uppercase letters in <i>string</i> to their lowercase equivalents. International letters such as Á are also converted.                                                                                                      |

**strmake** Copies *len* number of characters from *string*<sub>2</sub> to *string*<sub>1</sub>, adding the null-terminating character if necessary. Trailing space characters are trimmed from *string*<sub>1</sub>.

**strmatch** Compares the contents of *string* to the *mask* specification.

*mask* may contain literal characters or wild card character specifications. Literal characters in *mask* must match exactly with characters in *string*. Wild card characters in *mask* match “generically” with characters in *string*.

**Table 13: Wild Card Characters**

| Char | Meaning                                    |
|------|--------------------------------------------|
| ?    | Any character matches.                     |
| @    | Any alphabetic character matches.          |
| #    | Any numeric digit matches.                 |
| *    | Any number of any characters match.        |
| *@   | Any number of alphabetic characters match. |
| *#   | Any number of numeric digits match.        |

The mask specifications are analyzed to determine if the contents of string match the specifications. For instance:

```
strmatch(ssn, "###-##-####")
```

This function call tests the contents of the field *ssn* against the mask field. The match is true if *ssn* contains exactly three digits followed by a dash, followed by two digits followed by a dash, and then four digits. If it contains any more characters at the end or any other place in the string, or uses characters other than dashes to separate the digits, the match will be false.

When using the asterisk wild card specification, be careful how it is specified. For instance, a specification of:

```
strmatch(field, "*@c")
```

asks for a match on any series of letters terminated with the letter “c.” Note that if there is any character following the first occurrence of a “c,” this match will be false:

```
strmatch("abcdefghc", "*@c")
```

**strncat** Up to *len* characters of *string*<sub>2</sub> are copied to the end of *string*<sub>1</sub> and the null-terminating character is added if necessary. When *string*<sub>2</sub> is shorter than *len* characters, the length of *string*<sub>2</sub> is used in place of *len*.

**strncmp** Up to *len* characters of the contents of *string*<sub>1</sub> are compared with *len* characters of the contents of *string*<sub>2</sub>. A greater than, equal to, or less than result is returned.

**strncpy** Up to *len* characters of *string*<sub>2</sub> are copied to *string*<sub>1</sub>. When *string*<sub>2</sub> is shorter than *len* characters, *string*<sub>1</sub> is padded with

null characters up to length *len*. When the length of *string<sub>2</sub>* is equal to or greater than *len* characters, no null character is copied to *string<sub>1</sub>*.

- strneq** Up to *len* characters of the contents of *string<sub>1</sub>* are compared with the contents of *string<sub>2</sub>*. Only equality or inequality is determined.
- strnicmp** A “case-insensitive” comparison is performed. Up to *len* lower-cased characters of the contents of *string<sub>1</sub>* are compared with the lowercased contents of *string<sub>2</sub>*. A greater-than, equal-to, or less-than result is returned.
- strnset** Up to *len* characters of the contents of *string* are set to the value *c*. When *string* is shorter than *len* characters, the length of *string* is used in place of *len*.
- strpbrk** Similar to the function *strcspn*, the contents of *string<sub>1</sub>* are searched for any occurrence of a character that is also in *string<sub>2</sub>*. The null-terminating character is not included in the search.
- The difference between this function and *strcspn* is that *strpbrk* returns a pointer to the matching character and *strcspn* returns the index of the matching character.
- strrchr** *string* is searched for the last occurrence of the character value *c*.
- strset** The contents of *string* are set to the value *c*.
- strspn** This is the opposite of the function *strcspn*. The contents of *string<sub>1</sub>* are searched for any occurrence of a character that is not in *string<sub>2</sub>*. The null-terminating character is not included in the search. In other words, this function determines the initial length of *string<sub>1</sub>* that consists solely of characters in *string<sub>2</sub>*.
- strstr** The contents of *string<sub>1</sub>* are searched for the first occurrence of the substring *string<sub>2</sub>*.
- strtok** Finds the next “token” in *string<sub>1</sub>*.

The contents of *string<sub>2</sub>* define the characters that delimit tokens. A token is defined as a series of one or more characters that are not delimiting characters. For instance, the words in a typical English sentence can be treated as tokens if the delimiting characters are defined as “.,”.

The *string<sub>1</sub>* argument may be a pointer to a string of characters or it may be a NULL pointer. When *string<sub>1</sub>* is a character pointer, *strtok* saves this pointer in an internal variable. When *string<sub>1</sub>* is a NULL pointer, *strtok* uses the previously saved pointer.

Each time that `strtok` is called it searches *string<sub>1</sub>*. First it skips any leading delimiters identified by *string<sub>2</sub>* and remembers this first non-delimiting character location. Then it searches for the next delimiter and changes it to a null terminator. The internal variable is then set to point to the character following this location.

Subsequent calls to `strtok` (using a NULL pointer for *string<sub>1</sub>*) cause `strtok` to use its saved pointer to the string. It repeats the same sequence as above: Skipping leading delimiters, remembering the location of the first non-delimiting character and changing the next delimiting character to a null-terminator.

**strtrim**      The contents of *string* are “trimmed” of all leading and trailing spaces and embedded multiple spaces are reduced to single spaces.

**strupr**      Converts the lowercase letters in *string* to their uppercase equivalents. International letters such as á are also converted.

**Returns:**      **strcat**      A pointer to *string<sub>1</sub>*.

**strchr**      A pointer to the first occurrence of *c* in *string*. A return of a NULL pointer indicates that *c* cannot be found in *string*.

**strcmp**      A signed, integer value.

| Comparison            | Return value |
|-----------------------|--------------|
| $string_1 < string_2$ | < 0          |
| $string_1 = string_2$ | 0            |
| $string_1 > string_2$ | > 0          |

**strcpy**      A pointer to *string<sub>1</sub>*.

**strcspn**      The length of the portion of *string<sub>1</sub>* that contains no characters found in *string<sub>2</sub>*.

A return value of zero indicates that the first character of *string<sub>1</sub>* is one of the characters in *string<sub>2</sub>*.

A return value of the length of *string<sub>1</sub>* indicates that none of the characters in *string<sub>1</sub>* is also in *string<sub>2</sub>*.

**strdcmp**      A signed, integer value, similar to `strcmp`.

**strdup**      A pointer to the copy of *string*. If a copy could not be made due to insufficient memory, a NULL pointer is returned.

**strend**      A pointer to the null-terminating character of *string*.

**streq**      A true/false value. A non-zero value indicates that the two strings are equal; a zero value indicates that they are unequal. (Note that this is the opposite result of a `strcmp` comparison.)

**stricmp**      A signed, integer value, similar to `strcmp`.

A position is equal when the two strings are equal or if they have the same letter independent of its case. When a position is unequal, the lowercase form of each position is compared. Thus, “a” is less than “b” and “B” even though the value of “a” is actually greater than the value of “B.”

|                 |                                                                                                                                                                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>stricmp</b>  | A true/false value. A non-zero value indicates that, ignoring any case-mode differences, the two strings are equal; a zero value indicates that they are unequal.                                                                                                                  |
| <b>stristr</b>  | A pointer to the start of the first occurrence of the lowercase form of <i>string</i> <sub>2</sub> that was found in the lowercase form of <i>string</i> <sub>1</sub> . A NULL pointer is returned when <i>string</i> <sub>2</sub> cannot be found in <i>string</i> <sub>1</sub> . |
| <b>strlen</b>   | The current length of the contents of <i>string</i> .                                                                                                                                                                                                                              |
| <b>strlwr</b>   | A pointer to <i>string</i> .                                                                                                                                                                                                                                                       |
| <b>strmake</b>  | A pointer to <i>string</i> <sub>1</sub> .                                                                                                                                                                                                                                          |
| <b>strmatch</b> | A true/false value. A non-zero value indicates that <i>string</i> matches the <i>mask</i> specifications; a zero value indicates that it does not.                                                                                                                                 |
| <b>strncat</b>  | A pointer to <i>string</i> <sub>1</sub> .                                                                                                                                                                                                                                          |
| <b>strncmp</b>  | A signed, integer value, similar to <i>strcmp</i> .                                                                                                                                                                                                                                |
| <b>strncpy</b>  | A pointer to <i>string</i> <sub>1</sub> .                                                                                                                                                                                                                                          |
| <b>strneq</b>   | A true/false value. A non-zero value indicates that the two strings are equal for the specified length; a zero value indicates that they are unequal. (Note that this is the opposite result of a <i>strncmp</i> comparison.)                                                      |
| <b>strnicmp</b> | A signed, integer value, similar to <i>strcmp</i> .<br><br>See also comment about <i>stricmp</i> return.                                                                                                                                                                           |
| <b>strnset</b>  | A pointer to <i>string</i> .                                                                                                                                                                                                                                                       |
| <b>strpbrk</b>  | A pointer to the first character in <i>string</i> <sub>1</sub> that is also contained in <i>string</i> <sub>2</sub> . A NULL pointer indicates that no character in <i>string</i> <sub>1</sub> matched any of the characters in <i>string</i> <sub>2</sub> .                       |
| <b>strrchr</b>  | A pointer to the <u>last</u> occurrence of <i>c</i> in <i>string</i> . A return of a NULL pointer indicates that <i>c</i> cannot be found in <i>string</i> .                                                                                                                       |
| <b>strset</b>   | A pointer to <i>string</i> .                                                                                                                                                                                                                                                       |
| <b>strspn</b>   | The length of the portion of <i>string</i> <sub>1</sub> that contains only characters found in <i>string</i> <sub>2</sub> .                                                                                                                                                        |

A return value of zero indicates that the first character of *string*<sub>1</sub> is not one of the characters in *string*<sub>2</sub>.

|                      |                                                                                                                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                      | A return value of the length of <i>string<sub>1</sub></i> indicates that all of the characters in <i>string<sub>2</sub></i> are also in <i>string<sub>2</sub></i> .                                                                               |
| <b>strstr</b>        | A pointer to the first occurrence of the substring <i>string<sub>2</sub></i> in <i>string<sub>1</sub></i> . A NULL pointer indicates that <i>string<sub>2</sub></i> cannot be found in <i>string<sub>1</sub></i> .                                |
| <b>strtok</b>        | A pointer to the next token found in <i>string1</i> . (If <i>string<sub>1</sub></i> was a NULL pointer, it returns a pointer to the next token found in the last usage of <i>strtok</i> using a non-NULL pointer for <i>string<sub>1</sub></i> .) |
|                      | A return of a NULL pointer indicates that there are no more tokens in <i>string<sub>1</sub></i> .                                                                                                                                                 |
| <b>strtrim</b>       | A pointer to the trimmed <i>string</i> .                                                                                                                                                                                                          |
| <b>strupr</b>        | A pointer to <i>string</i> .                                                                                                                                                                                                                      |
| <b>Notes:</b>        |                                                                                                                                                                                                                                                   |
| <b>strcat</b>        | The contents of <i>string</i> are modified.                                                                                                                                                                                                       |
| <b>strcpr</b>        | The contents of <i>string</i> are modified.                                                                                                                                                                                                       |
| <b>strdup</b>        | The copy of <i>string</i> is created using the <a href="#">malloc</a> function. When the program no longer needs the copy, use the <a href="#">free</a> function to deallocate the storage.                                                       |
| <b>stricmp</b>       | The two strings are not modified by the case-insensitive comparison.                                                                                                                                                                              |
| <b>stristr</b>       | The two strings are not modified by the case-insensitive search.                                                                                                                                                                                  |
| <b>strlwr</b>        | The contents of <i>string</i> are modified. Uses the current locale setting for international characters.                                                                                                                                         |
| <b>strmake</b>       | The contents of <i>string<sub>1</sub></i> are modified.                                                                                                                                                                                           |
| <b>strncat</b>       | The contents of <i>string<sub>1</sub></i> are modified.                                                                                                                                                                                           |
| <b>strncpy</b>       | The contents of <i>string<sub>1</sub></i> are modified.                                                                                                                                                                                           |
| <b>strnset</b>       | The contents of <i>string</i> are modified.                                                                                                                                                                                                       |
| <b>strset</b>        | The contents of <i>string</i> are modified.                                                                                                                                                                                                       |
| <b>strtok</b>        | The contents of <i>string<sub>1</sub></i> are modified by the operation of this function. (Some delimiting characters are changed to null-terminating characters.)                                                                                |
|                      | Multiple, contiguous delimiters are treated as one delimiter.                                                                                                                                                                                     |
| <b>strtrim</b>       | The contents of <i>string</i> are modified.                                                                                                                                                                                                       |
| <b>strupr</b>        | The contents of <i>string</i> are modified. Uses the current locale setting for international characters.                                                                                                                                         |
| <b>Restrictions:</b> | All of these functions operate on null-terminated strings.                                                                                                                                                                                        |

**strcat** The contents of *string<sub>2</sub>* are added directly to the end of *string<sub>1</sub>*. Make sure that the allocation for *string<sub>1</sub>* is sufficient to contain both strings.

**strcpy** If *string<sub>2</sub>* and *string<sub>1</sub>* overlap, the resulting string may be erroneous. When *string<sub>2</sub>* is greater than *string<sub>1</sub>*, the copy is performed correctly; when *string<sub>1</sub>* is greater than *string<sub>2</sub>*, the process of copying the characters in *string<sub>2</sub>* may replace subsequent characters in *string<sub>2</sub>*.

For example:

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
string1 string2

```

Result: ghijklmnopklmnopqrstuvwxyz

```

      abcdefghijklmnopqrstuvwxyz
      ↑      ↑
string2 string1

```

Result: abcdefabcdefabcdqrstuvwxyz

Also, make sure that the allocation for *string<sub>1</sub>* is sufficient to contain the copy of *string<sub>2</sub>*.

**strmake** Make sure that the allocation for *string<sub>1</sub>* is sufficient to contain *len* number of characters and the null-terminating character. See also the restrictions for the *strcpy* function.

**strncat** *len* characters of *string<sub>2</sub>* are added directly to the end of *string<sub>1</sub>*. Make sure that the allocation for *string<sub>1</sub>* is sufficient to contain added characters.

**strncpy** See restrictions for *strcpy* function.

#### Conforms to:

|                 |        |       |         |         |
|-----------------|--------|-------|---------|---------|
| <b>strcat</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strchr</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strcmp</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strcpy</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strcspn</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strdcmp</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>strdup</b>   | q ANSI | n DOS | n THEOS | q POSIX |
| <b>strend</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>streq</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>stricmp</b>  | q ANSI | n DOS | n THEOS | n POSIX |
| <b>strieq</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>stristr</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>strlen</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strlwr</b>   | q ANSI | n DOS | n THEOS | q POSIX |
| <b>strmake</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>strmatch</b> | q ANSI | q DOS | n THEOS | q POSIX |

## 604 *string functions*

---

|                 |        |       |         |         |
|-----------------|--------|-------|---------|---------|
| <b>strncat</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strncmp</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strncpy</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strneq</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>strnicmp</b> | q ANSI | n DOS | n THEOS | q POSIX |
| <b>strnset</b>  | q ANSI | n DOS | n THEOS | q POSIX |
| <b>strpbrk</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strrchr</b>  | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strset</b>   | q ANSI | n DOS | n THEOS | q POSIX |
| <b>strspn</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strstr</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strtok</b>   | n ANSI | n DOS | n THEOS | n POSIX |
| <b>strtrim</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>strupr</b>   | q ANSI | n DOS | n THEOS | q POSIX |

**See also:** [far memory functions](#), [far string functions](#), [memory functions](#)

---

**Examples:** In addition to the examples that follow, refer to the other examples in this manual. These string functions are possibly the most common type of function and are used extensively in almost all C language programs.

---

**Example 1:** This example uses `strlen` in a function that pads a string with spaces to a specified length.

---

```
// Special Library: RPAD
//
// rpad(char *string, int len)
//
// Adds spaces, if necessary, to the end of string, forcing the
// length to be at least len number of bytes. If the string is
// already len number of bytes or more in length, then no action
// is taken.

#include <string.h>

char *
rpad(char *dest, int len)
{
    char * ptr;
    int l;

    l = strlen(dest);

    if (l < len)
    {
        ptr = dest + l;           // point to end of string

        l = len - l;              // compute number of blanks to add

        while (l--)
            *ptr++ = ' ';        // add a space

        *ptr = '\0';              // mark new end of string
    }
}
```



```
    }  
    return dest;  
}
```

**Example 2:** This example illustrates the `strtok` function by using it to extract the tokens from a C language source program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
main(int argc, char *argv[])
{
    char    *delimiters = " ,;{}()[]*\"'!%^&-+=|\\:;<>/?#@\\r\\n\\t";
    char    *token;
    char    *record;
    FILE    *infile;

    if (argc<2)                                // file name missing?
        syserr(46, 46, NULL);

    if (!(infile=fopen(argv[1], "rt"))) // can't open file?
        exit(fperror());

    record = malloc(1024);
    token = malloc(256);

    while (fgets(record, 1024, infile))        // get next record
    {
        token = strtok(record, delimiters); // get 1st token of line
        while (token) {                        // repeat all tokens in line
            printf("%s\\n", token);
            token = strtok(NULL, delimiters);
        }
    }

    fclose(infile);
}
```

## string assembly functions

This group of “functions” generates in-line code that manipulates null-terminated byte strings.

```
#include <builtin.h>
void _strcpy ( void * to, void * from )
void * _strend ( void * string )
unsigned long _strlen ( void * string )
```

---

|               |   |                               |
|---------------|---|-------------------------------|
| <i>from</i>   | » | pointer to source string      |
| <i>string</i> | » | pointer to string             |
| <i>to</i>     | » | pointer to destination string |

- Operation:**
- \_strcpy** Copies the null-terminated string pointed to by *from* to the location pointed to by *to*.
  - \_strend** Locates the null-terminator for *string*.
  - \_strlen** Counts the number of bytes in *string* up to the null-terminator.
- Returns:**
- \_strend** A pointer to the null-terminator of *string*.
  - \_strlen** The length of *string*.
- Errors:** Because these functions are actually in-line code, no error checking is performed.
- Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.
- The arguments *from*, *to* and *string* are void pointers. This means that the functions can use whatever type of object desired.
- Restrictions:** These functions are intended for device-driver authors.
- Conforms to:**
- |                |        |       |         |         |
|----------------|--------|-------|---------|---------|
| <b>_strcpy</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_strend</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_strlen</b> | q ANSI | q DOS | n THEOS | q POSIX |
- See also:** [memory functions](#), [string functions](#)

## strcoll

Compare two strings using the current locale collating sequence.

```
#include <string.h>
int strcoll ( char * string1, char * string2 )
```

---

*string*<sub>1</sub>               »   pointer to first string  
*string*<sub>2</sub>               »   pointer to second string

**Operation:**       Using the collating sequence specified by the LC\_COLLATE locale (see “setlocale” on page 562), *string*<sub>1</sub> is compared with *string*<sub>2</sub>.

**Returns:**         A signed, integer value.

| Comparison                                              | Return value |
|---------------------------------------------------------|--------------|
| <i>string</i> <sub>1</sub> < <i>string</i> <sub>2</sub> | < 0          |
| <i>string</i> <sub>1</sub> = <i>string</i> <sub>2</sub> | 0            |
| <i>string</i> <sub>1</sub> > <i>string</i> <sub>2</sub> | > 0          |

**Conforms to:**               **strcoll**   n ANSI    n DOS    n THEOS   q POSIX

**See also:**               [localeconv](#), [setlocale](#), [strcmp](#)

## strerror

Get message text from system message file.

```
#include <string.h>
char * strerror ( int num )
```

---

*num*                   »   number of message from SYSTEM.TEOS*nnn*.MESSAGE*n* file

**Operation:**       The message text from SYSTEM.TEOS*nnn*.MESSAGE*n* file is retrieved with parameter substitution performed using the reserved field `_errarg`.

**Returns:**         A pointer to the message text.

**Notes:**          The message text is saved in an array that is local to the `strerror` function.

The static array used by this function is overwritten each time that the function is used. You should copy or use this message text immediately to ensure that the proper message text is being used.

**Conforms to:**       **strerror**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**         [clearerr](#), [eof](#), [\\_errbot](#), [errmsg](#), [feof](#), [ferror](#), [file\\_err](#), [fperror](#), [getmsg](#), [perror](#), [putmsg](#), [syserr](#)

---

### Example:

```
#include <stdio.h>
#include <string.h>

void
main()
{
    char    tofmsg[80],
           eofmsg[80];
    int     tof,
           eof;

    strcpy(tofmsg, strerror(123));           // get top-of-file msg
    strcpy(eofmsg, strerror(122));           // get end-of-file msg

    ...
    if (tof)
        printf("%s\n", tofmsg);
    else if (bof)
        printf("%s\n", eofmsg);
    ...
}
```

**strftime**

Converts the time in a time structure into a formatted string.

```
#include <time.h>
size_t strftime ( char * string, size_t maxsize, char * format, struct tm * timeptr )
```

*format*  
*maxsize*  
*string*  
*timeptr*

» pointer to format control string  
» maximum length of string generated  
» pointer to storage location for generated string  
» pointer to time structure

**Operation:** The *format* string contains zero or more directives and literal characters. A directive consists of the percent character ( % ), followed by a single character specifying the element of *timeptr* to be substituted. As each character or directive in the *format* string is processed, it is either copied to the destination *string* (literal characters) or the appropriate time element is converted and substituted (directive).

Conversion stops when *maxsize* characters have been copied to the *string* or when the end of the *format* string is encountered.

Invalid directives in *format* are ignored.

**Returns:** When the length of the resulting *string* including the null terminator is less than or equal to *maxsize*, the actual length is returned. Otherwise, a zero is returned indicating an error and the contents of *string* are indeterminate.

**Notes:** The *format* string contains literal characters and translation directives.

| Char | Time Element Substitution                                                                                      |
|------|----------------------------------------------------------------------------------------------------------------|
| %a   | Abbreviated weekday name (Sun, Mon, <i>etc.</i> ) <sup>† A</sup>                                               |
| %A   | Full weekday name (Sunday, Monday, <i>etc.</i> ) <sup>† A</sup>                                                |
| %b   | Abbreviated month name (Jan, Feb, <i>etc.</i> ) <sup>† A</sup>                                                 |
| %B   | Full month name (January, February, <i>etc.</i> ) <sup>† A</sup>                                               |
| %c   | Date and time in standard DATEFORM format. For instance, DATEFORM 1 produces: MM/DD/YY HH:MM:SS <sup>‡ A</sup> |
| %C   | Date and time in current locale's long format, using “%J %B %Y %H:%M” <sup>U</sup>                             |
| %d   | Day number of month (01-31) <sup>A</sup>                                                                       |
| %D   | Date in format: MM/DD/YY <sup>U</sup>                                                                          |
| †    | The SYSTEM.TEOS32.MESSAGES file is used to get the actual names.                                               |
| ‡    | The date format is determined by the DATEFORM environment variable.                                            |
| A    | Conforms to ANSI minimal specifications.                                                                       |
| U    | Extensions to ANSI added by UNIX system V.                                                                     |

Table 14: *strftime* Directives

| Char | Time Element Substitution                                                                |
|------|------------------------------------------------------------------------------------------|
| %e   | Day number of month (1-31) with single digits preceded by a space character <sup>U</sup> |
| %h   | Abbreviated month name (Jan, Feb, <i>etc.</i> ), synonym to “%b” <sup>U</sup>            |
| %H   | Hour number, 24-hour clock (00-23) <sup>A</sup>                                          |
| %I   | Hour number, 12-hour clock (01-12) <sup>A</sup>                                          |
| %j   | Julian day number of year (001-366) <sup>A</sup>                                         |
| %J   | Day number of month (1-31) with single digits not preceded by a space or zero character  |
| %k   | Hour number (0-23) with single digits preceded by a space character <sup>U</sup>         |
| %l   | Hour number (1-12) with single digits preceded by a space character <sup>U</sup>         |
| %m   | Month number (01-12) <sup>A</sup>                                                        |
| %M   | Minute number (00-59) <sup>A</sup>                                                       |
| %n   | Same as a “\n” <sup>U</sup>                                                              |
| %p   | AM or PM <sup>† A</sup>                                                                  |
| %r   | Time in standard 12-hour format, using “%I:%M:%S %p” <sup>U</sup>                        |
| %R   | Time in standard 24-hour format, using “%H:%M” <sup>U</sup>                              |
| %S   | Second number (00-59) <sup>A</sup>                                                       |
| %t   | Same as a “\t” <sup>U</sup>                                                              |
| %T   | Time in standard 24-hour format: HH:MM:SS <sup>U</sup>                                   |
| %U   | Week number of year, Sunday is 1st day of week (00-52) <sup>A</sup>                      |
| %w   | Weekday number (0=Sunday through 6=Saturday) <sup>A</sup>                                |
| %W   | Week number of year, Monday is 1st day of week (00-52) <sup>A</sup>                      |
| %x   | Date in standard format: MM/DD/YY <sup>‡ A</sup>                                         |
| %X   | Time in standard 24-hour format: HH:MM:SS <sup>A</sup>                                   |
| %y   | Year number in century (00-99) <sup>A</sup>                                              |
| %Y   | Year number (1900-2038) <sup>A</sup>                                                     |
| %Z   | Timezone name (the <a href="#">tzset</a> function is called) <sup>A</sup>                |
| %%   | Literal percent character <sup>A</sup>                                                   |
| †    | The SYSTEM.TEOS32.MESSAGE <i>n</i> file is used to get the actual names.                 |
| ‡    | The date format is determined by the DATEFORM environment variable.                      |
| A    | Conforms to ANSI minimal specifications.                                                 |
| U    | Extensions to ANSI added by UNIX system V.                                               |

Table 14: strftime Directives

**Restrictions:** This function only recognizes dates in the range of Jan 1, 1900 through Jan 18, 2038.

**Conforms to:** **strftime**    n ANSI    n DOS    n THEOS    n POSIX

**See also:** [asctime](#), [clock](#), [ctime](#), [difftime](#), [gmtime](#), [localtime](#), [mktime](#), [setlocale](#), [time](#), [tzset](#)

## 612 *strftime*

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void
main(void) {
    time_t  utc;           // UTC time
    char    buffer[80];
    char    tz[20] = "";   // TZ env value

    time(&utc);            // get current UTC time
    if (getenv("tz"))
        strcpy(tz, getenv("tz")); // get current TZ value

    tzset();               // set global time variables

    if (daylight)
        printf("Daylight/summer time supported.\n");
    else
        printf("Daylight/summer time not supported.\n");

    printf("Timezone names are: %s\n                %s\n",
           tzname[0], tzname[1]);
    printf("Local time is %ld seconds %s UTC time\n\n",
           timezone, timezone > 0 ? "before" : "after");

    strftime(buffer, 80, "%A, %B %d, %Y, at %I:%M %p %Z",
              localtime(&utc));
    printf("Local time is %s\n\n", buffer);

    strftime(buffer, 80, "%A, %B %d, %Y, at %H:%M", gmtime(&utc));
    printf("UTC time is %s\n\n", buffer);

    putenv("tz=EST5EDT"); // Set to Eastern Time
    tzset();

    strftime(buffer, 80, "%A, %B %d, %Y, at %I:%M %p %Z",
              localtime(&utc));
    printf("Eastern time is %s\n\n", buffer);

    strcat(strcpy(buffer, "TZ="), tz); // get real tz setting
    putenv(buffer);                    // reset
}
```

---

### Output:

```
Daylight/summer time supported.
Timezone names are: PST
                    PDT
Local time is 28,800 seconds before UTC time

Local time is Tuesday, March 19, 1996, at 10:43 am PST

UTC time is Tuesday, March 19, 1996, at 18:43

Eastern time is Tuesday, March 19, 1996, at 01:43 pm EST
```



## strtod, strtol, strtoul

Convert the ASCII string representation of a floating-point value to a numeric field.

```
#include <stdlib.h>
```

```
double strtod ( const char * string, char ** endptr )
```

```
long strtol ( const char * string, char ** endptr, int base )
```

```
unsigned long strtoul ( const char * string, char ** endptr, int base )
```

---

*base* » numeric base to convert from

*endptr* » address of variable that will point to end of number string

*string* » pointer to string

**Operation:**     **strtod**     The first non-white space character in *string* is interpreted as the start of a floating-point number specification. This value is converted into a floating-point value and returned as the value of the function.

*string* is expected to point to a string that looks like:

$$[whitespace][sign][digits][.digits] \left[ \begin{matrix} d \\ D \\ e \\ E \end{matrix} \right] [sign][digits]$$

**strtol**     The first non-white space character in *string* is interpreted as the start of integer specification. This value is converted into a long integer value and returned as the value of the function.

*string* is expected to point to a string that looks like:

$$[whitespace][sign][0] \left[ \begin{matrix} x \\ X \end{matrix} \right] [digits]$$

**strtoul**     The first non-white space character in *string* is interpreted as the start of an unsigned integer specification. This value is converted into an unsigned long integer value and returned as the value of the function.

$$[whitespace][0] \left[ \begin{matrix} x \\ X \end{matrix} \right] [digits]$$

Each of these functions stops scanning *string* when a character is found that cannot be part of a number. In the case of **strtol** and **strtoul**, it may also stop scanning when a digit is found that is greater than base. When *endptr* is not a NULL pointer, each of these functions sets *endptr* to point to the first character that stopped the scan.

**Returns:**     **strtod**     The floating-point value found in *string*, unless the value would cause an overflow, in which case `±HUGE_VAL` is returned. If the number cannot be converted, zero is returned.

**strtol**     The long integer value found in *string*, unless the value would cause an overflow, in which case `LONG_MAX` or `LONG_MIN` is returned. If the number cannot be converted, zero is returned.

## 614 *strtod, strtol, strtoul*

---

**strtoul**      The unsigned long integer value found in *string*, unless the value would cause an overflow, in which case `ULONG_MAX` is returned. If the number cannot be converted, zero is returned.

**Errors:**      If the conversion causes an overflow or underflow, [errno](#) is set to `ERANGE`.

**Notes:**      If *base* is 0, the initial characters of *string* are used to determine the base. If the first character is “0” and the second character is not “x” or “X,” then the string is interpreted as an octal integer; otherwise it is interpreted as a decimal number. If the first character is “0” and the second character is “x” or “X,” then the string is interpreted as a hexadecimal integer.

If the first character is “1” through “9,” then the string is interpreted as a decimal integer. The letters “a” through “z” or “A” through “Z” are assigned the values 10 through 35. Only letters whose assigned values are less than *base* are permitted.

|                     |                |        |       |         |         |
|---------------------|----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>strtod</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>strtol</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>strtoul</b> | n ANSI | n DOS | n THEOS | q POSIX |

**See also:**      [atof](#), [atoi](#), [atol](#), [atoll](#), [sscanf](#)

## strxfrm

Transform one string of characters into another by using the current locale-specific collating sequence.

```
#include <string.h>
size_t strxfrm ( char * to, const char * from, size_t len )
```

---

|             |   |                                   |
|-------------|---|-----------------------------------|
| <i>from</i> | » | pointer to source string          |
| <i>len</i>  | » | number of characters to translate |
| <i>to</i>   | » | pointer to destination buffer     |

**Operation:** The string pointed to by *from* is transformed into a new string and stored at *to*. No more than *len* characters of *from* are transformed.

The transformation of the characters is made using the information in the LC\_COLLATE locale (see “setlocale” on page 562).

**Returns:** The length of the transformed string. This should be the same as *len*.

**Notes:** After the transformation, a call to [strcmp](#) with the transformed strings yields the same results as a call to [strcoll](#) with untransformed strings.

**Conforms to:** **strxfrm** n ANSI n DOS n THEOS q POSIX

**See also:** [localeconv](#), [setlocale](#), [strcoll](#)

**subtract memory assembly functions**

This group of “functions” generates in-line code to subtract a byte, short integer or long integer from a specified memory area or from your program’s code segment.

```
#include <builtin.h>

void _subb ( void _far * far_offset, char bval )
void _subd ( void _far * far_offset, long lval )
void _subw ( void _far * far_offset, int wval )

void _subcsb ( void * offset, char bval )
void _subcsd ( void * offset, long lval )
void _subcsw ( void * offset, int wval )
```

---

*bval*                   »   byte value to subtract  
*far\_offset*           »   pointer to remote data object  
*lval*                   »   long integer value to subtract  
*offset*                »   pointer to data item in code segment  
*wval*                   »   word value (short integer) to subtract

**Operation:**        **\_subb**       Subtracts the value of *bval* from the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_subd**       Subtracts the value of *lval* from the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_subw**       Subtracts the value of *wval* from the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_subcsb, \_subcsd, \_subcsw** These functions perform the same operation as **\_subb**, **\_subd** and **\_subw** except that the memory segment is predefined to be your program’s code segment alias. These three functions can only be used in device-drivers because other programs do not have an alias to their code segment and they will generate a general protection error.

**Returns:**           No value is returned by these functions. The values are subtracted directly from the values in the locations specified.

**Notes:**            These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can subtract whatever type of object desired from any location. For instance, a **\_subw** function can subtract a word value (double-byte value) from a location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:**    These functions are intended for device-driver authors.

The nucleus and ucb memory regions are not generally accessible without proper memory access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

**Conforms to:**

|                |        |       |         |         |
|----------------|--------|-------|---------|---------|
| <b>_subb</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_subd</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_subcsb</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_subcsd</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_subcsw</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>_subw</b>   | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**

[add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [xor memory assembly functions](#)

**swab**

Copy pairs of bytes from one location to another, swapping adjacent bytes.

```
#include <stdlib.h>
void swab ( char * from, char * to, int count )
```

---

|              |   |                               |
|--------------|---|-------------------------------|
| <i>count</i> | » | number of bytes to copy       |
| <i>from</i>  | » | pointer to source buffer      |
| <i>to</i>    | » | pointer to destination buffer |

**Operation:** Consecutive pairs of bytes in *from* are copied to *to*. As each pair of bytes is copied their relative positions are swapped. For instance:

```
char from[] = "abcdefghij";
char to[10];

swab(from, to, 6);
```

The *to* buffer will now contain the sequence “badcfe.”

The copy and swap are terminated when *count* bytes have been transferred. When *count* is an odd value, then *count*-1 bytes are transferred. When *count* is negative or zero, no bytes are transferred.

**Notes:** The bytes transferred to the *to* buffer are not null-terminated.

**Conforms to:**                    **swab**    q ANSI    n DOS    n THEOS    n POSIX

## swapbits

Reverses the bits in an integer value (four byte).

```
#include <stdlib.h>
int swapbits ( int value )
```

---

*value*                    »   integer value

**Operation:**      The sequence of the value of the bits in the four bytes of *value* are reversed. For instance:

```
swapbits(0x12345678);
```

This function call returns the value 0x90092c48. The original value of “00010010001101000101011001111000” was reversed to “00011110011010100010110001001000.”

**Returns:**          The *value* with the bits in reverse sequence.

**Notes:**            This function might be used in a device-driver that is sending or receiving data from a serial port that requires the bits to be passed in reverse sequence.

**Conforms to:**            **swapbits**      q ANSI      q DOS      n THEOS      q POSIX

**\_sync**

Synchronize the disk cache with the disk volumes.

```
#include <driver.h>
void _sync ( int code )
```

---

*code*                    »    control code for disk caching

**Operation:**        This function synchronizes the disk cache with the disk volumes by writing all dirty buffers to disk. It also sends a sync command to caching disk controllers to force them to sync.

The *code* is used to interface to the disk caching mechanism, allowing you to control how caching is performed.

**Notes:**            Values for *code* include:

| <i>Name</i>       | <i>Value</i> | <i>Meaning</i>                     |
|-------------------|--------------|------------------------------------|
| WRITEBACK_ON      | -17768       | Enable directory write caching.    |
| WRITEBACK_OFF     | 1            | Disable directory write caching.   |
| WRITEBACK_RESTART | -1107        | Re-enable directory write caching. |
| CACHE_OFF         | -1           | Disable disk caching.              |
| SYNC_CMD          | 0            | Perform disk cache sync only.      |

The names listed above are not defined in any header file and are listed for convenience only.

**Restrictions:**    You should only use the *code* value of zero. The other values can be used but are only intended for the BOOTER3 process during system start-up or by the CACHE utility command.

Disk caching can only be enabled with the CACHE utility command or during system start-up if it has been specified in the system configuration file.

**Conforms to:**            **\_sync**    q ANSI    q DOS    n THEOS    q POSIX



## syserr

Display a system message on `stderr` with parameter substitution and exit.

```
#include <stdlib.h>
```

```
void syserr ( int retcode, int num, void * args )
```

---

*args* » pointer to array of string pointers

*num* » number of message from `SYSTEM.TEOSnnn.MESSAGE` file

*retcode*»return code for program exit

**Operation:** Using the `errmsg` function, message *num* is displayed on `stderr`, with parameter substitution using *args*. After the message is written, the `exit` function is called to exit your program with a return code of *retcode*.

**Returns:** There is no return from this function: The program is exited.

**Notes:** Familiarize yourself with the message text because the usage of the *args* argument depends upon the number and type of parameter substitution codes in the message text:

- ▶ A message with no substitution codes causes *args* to be ignored.
- ▶ A message with one substitution code of `{0}` causes *args* to be interpreted as `char *args`, that is, as a pointer to a character string.
- ▶ A message with multiple substitution codes causes *args* to be interpreted as `char *args[]`, that is, as a pointer to an array of pointers to strings.

**Conforms to:** **syserr** q ANSI q DOS n THEOS q POSIX

**See also:** [clearerr](#), [eof](#), [\\_errbot](#), [errmsg](#), [exit](#), [feof](#), [ferror](#), [file\\_err](#), [fperror](#), [perror](#), [putmsg](#), [strerror](#)

## 622 syserr

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    char    filename[256];

    printf("\nEnter name of file to erase: ");
    gets(filename);

    if (!access(filename, 0))
    {
        errmsg(48, filename);
        if (yesno())
        {
            remove(filename);
            errmsg(49, filename);
        }
    }
    else
        syserr(1, 19, filename);
}
```

---

### Output:

```
>test

Enter name of file to erase: bad.filename
File "bad.filename" not found.

>test

Enter name of file to erase: sample.file
Ok to erase "sample.file" (Yes,No,All) Y
"sample.file" erased.

>
```

## system

This function passes a command-string to the CSI for execution, and then returns to this program.

```
#include <stdlib.h> or <process.h>
int system ( const char _far * cmdstr )
```

---

*cmdstr*                    »    command string to execute

**Operation:** All subtasks of this program are killed. The contents of the current program's data segment and all open file pointers are saved on the work drive in a temporary file. The *cmdstr* is then passed to the command string interpreter for execution. Upon completion of the execution of the program or EXEC specified in *cmdstr*, control returns to this program with the data segment restored from the temporary save file.

**Returns:** The return code of the command executed in *cmdstr*.

**Errors:** If the *cmdstr* cannot be executed for some reason, the command string interpreter will handle the error in the same manner as if the command had been typed by the operator at the CSI prompt.

**Notes:** If the first character of *cmdstr* is ">" the *cmdstr* is displayed on the console.

Other than command-line arguments specified in *cmdstr*, there is no communication between the current program and the called program.

This function is similar to the [csi](#) function except the system function does return to the current program and the [csi](#) function does not.

**Conforms to:**                    **system**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**                    [csi](#)

---

### Example:

```
#include <stdlib.h>
#include <stdio.h>

main()
{
    int    filecnt;

    filecnt = system("filelist my.library.* (notype");
    printf("\nNumber of files in my.library is %d\n", filecnt);
}
```

---

### Output:

```
>test

Number of files in my.library is 24

>
```

**typeahead buffer functions**

This group of functions maintains and accesses a type-ahead buffer for a device-driver.

```
#include <driver.h>
void ta_alloc ( UCB * ucb, unsigned len )
void ta_free ( UCB * ucb )
unsigned short ta_get ( UCB * ucb )
void ta_put ( UCB * ucb, char c )
unsigned short ta_rdy ( UCB * ucb )
void ub_alloc ( UCB * ucb, void _far * far_offset )
void ub_free ( UCB * ucb )
```

---

|                   |   |                                      |
|-------------------|---|--------------------------------------|
| <i>c</i>          | » | character code to add to buffer      |
| <i>far_offset</i> | » | pointer to remote memory buffer      |
| <i>len</i>        | » | size of buffer                       |
| <i>ucb</i>        | » | unit control block number for device |

**Operation:** A type-ahead buffer is a fifo (first-in, first-out) buffer used by device-drivers supporting relatively slow-speed input devices, such as the keyboard, a serial communications port, *etc.*

A device-driver may have one of three types of type-ahead buffer: none, standard or user. The standard type-ahead buffer is enabled and allocated with the `ta_alloc` function; the user type-ahead buffer is enabled with the `ub_alloc`. The allocation of a user buffer is done by the calling program prior to using the `ub_alloc` function.

**ta\_alloc** Allocates a *len* sized standard type-ahead buffer for the device-driver associated with unit control block number *ucb*.

If there is insufficient memory to allocate the buffer, then no buffering is done for this device.

**ta\_free** Deallocates the standard type-ahead buffer associated with unit control block number *ucb*.

**ta\_get** Gets the next character available from the standard or user type-ahead buffer associated with unit control block number *ucb*. If no character is available, this function waits for the driver to read a character from the device.

**ta\_put** Puts or adds the character *c* to the standard or user type-ahead buffer associated with unit control block number *ucb*.

**ta\_rdy** Returns the status of the standard or user type-ahead buffer associated with unit control block number *ucb*.

**ub\_alloc** Allocates a *len* sized type-ahead buffer for the device-driver associated with unit control block number *ucb*.

**ub\_free** Disables the user type-ahead buffer associated with unit control block number *ucb*. If there was a standard buffer associated with this device, then it is automatically re-enabled.

|                 |                 |                                                                                                                                          |
|-----------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Returns:</b> | <b>ta_get</b>   | Returns the character from the standard or user type-ahead buffer.                                                                       |
|                 | <b>ta_rdy</b>   | The status of the standard or user type-ahead buffer. The status of a type-ahead buffer is the number of characters saved in the buffer. |
| <b>Notes:</b>   | <b>ta_alloc</b> | The memory selector for the standard buffer is saved in <code>ucb-&gt;iobuff1</code> .                                                   |
|                 | <b>ta_free</b>  | Clears the <code>ucb-&gt;iobuff1</code> , thus disabling the standard buffer.                                                            |
|                 | <b>ta_put</b>   | The device-driver should check before using this function to ensure that there is space available in the buffer for this new character.  |
|                 | <b>ub_alloc</b> | The memory location for the user buffer is saved in <code>ucb-&gt;ubuf_seg</code> and <code>ucb-&gt;ubuf_ofs</code> .                    |

A device may have only one type-ahead buffer enabled at any one time. Although both a standard and a user buffer may be defined, the `ta_get`, `ta_put` and `ta_rdy` functions will always use the user buffer if it is defined. When there is no user buffer defined, then these functions will use the standard buffer.

The `ta_alloc` should be done during device initialization. The interrupt service routine function should use `ta_put` to add characters to the buffer; the device's read function should use the `ta_get` to retrieve the characters. These functions do their own locking and buffer management. `ta_put` automatically drops the console input through `_con_tran`.

**Defaults:** Unless the `ta_alloc` or `ub_alloc` function is used, a device does not have a type-ahead buffer, and the `ta_rdy` and `ta_get` will always return a zero and `ta_put` will generate an address error fault.

**Restrictions:** These functions are intended for device-driver authors and should not be used by application programs. Use the [fifo functions](#) instead.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>ta_alloc</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>ta_free</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>ta_get</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>ta_put</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>ta_rdy</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>up_alloc</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>up_free</b>  | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [fifo functions](#)

**tan, tanh**

Compute the tangent or hyperbolic tangent of an angle.

```
#include <math.h>
double tan ( double x )
double tanh ( double x )
```

---

*x*                      »   floating-point angle, in radians

**Operation:**        **tan**            Compute the tangent of the angle *x*.

**tanh**          Compute the hyperbolic tangent of the angle *x*.

**Returns:**           **tan**            The tangent of the angle *x*.

**tanh**          The hyperbolic tangent of the angle *x*.

**Restrictions:**    The value of *x* should not be an odd multiple of  $\pi/2$  radians as these values are outside this function's domain and cause **tan** to return meaningless values.

**Conforms to:**                **tan**        n ANSI        n DOS        n THEOS        n POSIX

**tanh**        n ANSI        n DOS        n THEOS        n POSIX

**See also:**            [acos](#), [acot](#), [acsc](#), [asec](#), [asin](#), [atan](#), [atan2](#), [cos](#), [cosh](#), [cot](#), [coth](#), [csc](#), [csch](#), [sec](#), [sech](#), [sin](#), [sinh](#)

**tell, \_tell**

Gets a file's position pointer.

```
#include <io.h>
unsigned long tell ( int file_handle )

#include <sc.h>
unsigned long _tell ( FILE * file )
```

---

*file*                   »   pointer to open file's fcb  
*file\_handle*           »   open file number or handle

**Operation:**       This function is identical in operation to the [ftell](#) function described on page 261 except that [errno](#) is not set.

- tell**           The *file\_handle* argument is converted to a file pointer and the *\_tell* function is called.
- \_tell**         The current input/output position of *file* is determined, relative to the beginning of the file.

**Returns:**        The current file position.

A return of a negative value indicates an error occurred.

When *file* or *file\_handle* is on a device incapable of seeking, such as a terminal or printer, the return value is undefined.

**Errors:**         When the return value is negative, the only cause of error is an invalid handle or the file is not currently open.

|                     |              |        |       |         |         |
|---------------------|--------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>tell</b>  | q ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>_tell</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**        [fgetpos](#), [file\\_err](#), [fseek](#), [ftell](#), [lseek](#)

**Example:**        See “Example:” on page 262.

**tempnam, tmpfile, tmpnam, \_tmpnam\_drive**

Create a unique, temporary file name.

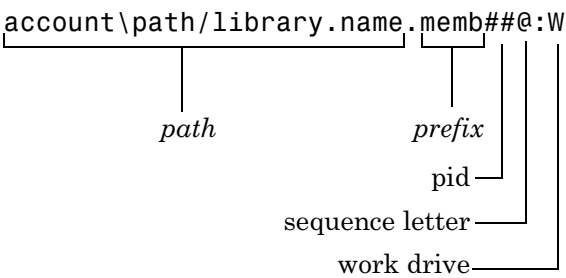
```
#include <stdio.h>
char * tempnam ( char * path, char * prefix )
FILE * tmpfile ( void )
char * tmpnam ( char * string )
void _tmpnam_drive ( int lub )
```

---

|               |   |                                                             |
|---------------|---|-------------------------------------------------------------|
| <i>lub</i>    | » | logical unit number for disk drive                          |
| <i>path</i>   | » | pointer to string containing path or library name           |
| <i>prefix</i> | » | pointer to string containing prefix for file or member name |
| <i>string</i> | » | pointer to buffer for temporary file name                   |

**Operation:**      **tempnam**      Using the value in *path*, appends a period and the value in *pre-*  
*fix*, and then adds a unique identifier consisting of a two-digit  
process number and a single-letter sequence number. The  
entire string is terminated with a colon and the drive code des-  
ignation for the current work drive. For instance:

tempnam("/private/my.library.", "work")  
generates a file name of "/private/my.library.work01A:S".



**tmpfile**      Using the tmpnam function, a unique temporary file name is  
created and the file is opened using **fopen** with read and write  
access.

**tmpnam**      Using the value defined for P\_tmpdir, a unique file name is gen-  
erated by appending a two-digit process number and a single-  
letter sequence number. The entire string is terminated with a  
colon and the drive letter specified in the last call to  
\_tmpnam\_drive.

This generated file name is then stored in the string pointed to  
by the *string* argument. If *string* is a NULL pointer, the file  
name is not saved in that variable.

**\_tmpnam\_drive**      Sets the drive code used by the tmpnam function.

**Returns:**      **tempnam**      A pointer to the generated file name. If the file name already  
exists, a NULL pointer is returned.



**tmpfile** A pointer to the newly opened file's fcb. If the file cannot be opened for some reason, a NULL pointer is returned.

**tmpnam** A pointer to the generated file name. If the file name already exists, a NULL pointer is returned.

**Notes:** **tempnam** The value of *path* does not necessarily have to contain an actual path. It is merely used as the major prefix for the generated file name. You could use a *path* value that is a NULL string. When *path* is a NULL, string the default name of SYSTEM is used.

If *path* contains a drive code specification, it is omitted from the generated file name. If *path* is terminated with a period, it is omitted from the generated file name (the period is not duplicated).

Similarly, the value of *prefix* can be any string that you want appended to the end of the value of the *path* string. It may also be NULL. When *prefix* is a NULL string, the default name of WORK is used.

Only the first four characters of *prefix* are used.

**tmpfile** The file created by this function is automatically deleted when it is closed or when the program exits normally.

**Defaults:** **\_tmpnam\_drive** The default value for the drive code used by the tmpnam function is "S."

**Restrictions:** **tempnam** The sequence letter used by this function is in the same series as the sequence letter used by the mktemp function when its mask contains a ##@ or ###@ pattern.

**tmpnam** When *string* is not a NULL pointer, it should point to a character array of at least L\_tmpnam bytes. L\_tmpnam is defined in STDIO.H file.

When *string* is a NULL pointer, the generated file name is in an array internal to the tmpnam function. This internal array is reused each time that tmpnam is called.

**Conforms to:**

|                      |        |       |         |         |
|----------------------|--------|-------|---------|---------|
| <b>tempnam</b>       | q ANSI | n DOS | n THEOS | n POSIX |
| <b>tmpfile</b>       | n ANSI | n DOS | n THEOS | n POSIX |
| <b>tmpnam</b>        | n ANSI | n DOS | n THEOS | n POSIX |
| <b>_tmpnam_drive</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [mktemp](#)

## testarg

Compare a string to a keyword in the standard keywords file.

```
#include <stdlib.h>
```

```
int testarg ( const char * string, int keynum )
```

---

*string*                   »   pointer to string to be compared

*keynum*                 »   number of keyword in SYSTEM.TEOS32.KEYWORD*n* file

**Operation:**       *string* is compared to the token in the keywords file specified by *keynum*.

**Returns:**       A true/false value: A non-zero return indicates that the two strings do match; a zero return value indicates that the strings do not match.

**Notes:**        The tokens in the keywords file have a minimum spelling count. *string* must be at least as long as this count. In order for *string* to match the keyword, *string* must match, character for character, each of the characters in the keyword token.

The characters in *string* must be uppercase characters. Since this function is normally used to test command-line option specifications, and command-line arguments are normally folded to uppercase by the CSI, this is not normally a consideration.

This function must access the keywords file using the [getkey](#) function. That function opens the file and leaves it open. When the program is finished accessing the keywords file with this function, or with other calls to [getkey](#), it should close the file with the [keyclose](#) function.

**Conforms to:**        **testarg**    q ANSI      q DOS      n THEOS      q POSIX

**See also:**        [getkey](#), [keyclose](#), [matcharg](#)

---

### Example:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void
main(int argc, char * argv[])
{
    char            typekey[9],
                  outdev[5];
    int            i = 0;
    FILE           * out;

    out = stdout;    // default output is console

    --argc;

    while (i < argc && !streq(argv[i], "("))    // search for "("
    {
        ...        // process leading args
        ++i;
    }
}
```

```

while (++i <= argc)    // process options
{
    if (testarg(argv[i],4) ||          // PRINTER1 ?
        testarg(argv[i],8) ||          // PRINT1 ?
        testarg(argv[i],12))           // PRT1 ?
    {
        strcpy(outdev,"prt1");
        out = NULL;
    }
    else if (testarg(argv[i],5) ||      // PRINTER2 ?
        testarg(argv[i],9) ||          // PRINT2 ?
        testarg(argv[i],13))           // PRT2 ?
    {
        strcpy(outdev,"prt2");
        out = NULL;
    }
    else if (testarg(argv[i],6) ||      // PRINTER3 ?
        testarg(argv[i],10) ||         // PRINT3 ?
        testarg(argv[i],14))           // PRT3 ?
    {
        strcpy(outdev,"prt3");
        out = NULL;
    }
    else if (testarg(argv[i],7) ||      // PRINTER4 ?
        testarg(argv[i],11) ||         // PRINT4 ?
        testarg(argv[i],15))           // PRT4 ?
    {
        strcpy(outdev,"prt4");
        out = NULL;
    }
    else if (testarg(argv[i],2))        // TYPE ?
        ;
    else
        syserr(27, 27, argv[i]);      // Invalid option
}

keyclose();

...

if (!out)
{
    if (!(out=fopen(outdev,"w")))
        exit(fperror());
}

...
}

```

**testhead**

Control and display headings when listing to stdout.

```
#include <stdlib.h>
int testhead ( char * header )
```

---

*header*                   »   pointer to page heading string

**Operation:** Increments an internal line counter. When the value of this line counter is greater than the depth of the stdout display page, a [pagewait](#) is performed and a new display page is started with a formatted heading containing the contents of *header*, the date and time, and the current page number.

**Returns:** The character entered by the operator in response to the [pagewait](#) request.  
If page-waits are suppressed, a zero is returned.

**Notes:** The [getll](#) and [getpl](#) functions are used to determine the display size of the stdout device.

This function assumes that every display line output to stdout is preceded by a call to this function. When the function is first called, it saves the current date and time. This saved date and time is used when formatting all subsequent headers. Thus, even though several seconds may elapse between the display of the first header and the last, all headers will use the same date and time stamp.

The page heading is formatted using a [printf](#) format string containing:

```
"%s%*s%s %s\n\n", header, padding, " ", date, pagestr
```

*header* is the string supplied by the argument *header*. This string can be different each time that *testhead* is called.

*padding* is a computed value that reflects the number of spaces required to right justify the remainder of the heading on the stdout display.

*date* is the saved date and time stamp string. This is generated by a call to [strftime](#) with a format of:

```
"%d %b %Y %I:%M%p"
```

This corresponds to a date string such as "04 Jul 1996 10:20am ."

*pagestr* is the word "Page" followed by the current page number. The word "Page" is language-specific because it is read from the `sys-TEM.TEOS32.MESSAGE` file, message number 424.

Initially, the page number is zero. The page number is incremented by one and the line counter is reset prior to displaying the heading. To reset these internal counters, call *testhead* with a NULL argument.

**Restrictions:** The page numbers are limited to the size of an int.

**Conforms to:**                   **testhead**    q ANSI       q DOS       n THEOS       q POSIX

**See also:**                   [getll](#), [getpl](#), [pagewait](#), [strftime](#)

**Example:**

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main(int argc, char *argv[])
{
    FILE    * infile;
    char    filename[256],
           head[64],
           record[256];

    if (argc<2)    // no filename specified?
        syserr(25, 25, NULL);    // error: file name missing

    strcpy(filename, argv[1]);
    if (testwild(filename))    // wild cards specified?
        syserr(110, 110, NULL);    // error: wild cards not permitted

    if (!(infile=fopen(filename, "r")))
        exit(fperror());    // error on open

    strcpy(filename, getfn(filename));    // get the full file name
    longfname(filename, head, 50);    // shorten, if necessary

    while (fgets(record, 256, infile)) {    // repeat for entire file
        testhead(head);    // test if need head
        fputs(record, stdout);    // display a record
    }
    pagewait();    // release last page

    fclose(infile);
}

```

**Output:**

CLIENTS\ABC\DOC\FILES\CUSTOMER.MASTER:S      15 Nov 1996 11:20am Page 1

| CUSTOMER.MASTER |                 |       |                          | Indexed<br>RecLen 420 |
|-----------------|-----------------|-------|--------------------------|-----------------------|
| Nbr             | Field-name      | T Len | Description              |                       |
| Key             | Customer code.. | A 6   | A99999                   |                       |
| 1               | Name.....       | A 30  | Custome name             |                       |
| 2               | Address.....    | A 30  | Street address           |                       |
| 3               | Address2.....   | A 30  | Street address, line two |                       |

...

## testwild

Test a string to see if there are any wild card specifications included.

```
#include <stdlib.h>
```

```
int testwild ( char * filename )
```

---

*filename*                   »   pointer to string containing possible file name specification

**Operation:**       The string pointed to by *filename* is searched for any occurrence of a wild card specification. Wild cards recognized by this function include: \*, ? # and @.

**Returns:**        A true/false value. A true or non-zero value means that *filename* contains at least one wild card character; a zero value means that there are no wild card characters in *filename*.

**Notes:**         Most of the file-related functions cannot use file name specifications containing wild cards. The `testwild` function can be used to identify an improper file specification or it can be used to identify a situation where different code is used to process the wild card specification. For example, the [directory search functions](#) and the [find\\_first](#), [find\\_next](#) functions are designed to use file name specifications containing wild cards.

**Conforms to:**        **testwild**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [directory search functions](#), [find\\_first](#), [find\\_next](#)

---

### Example:

```
#include <stdid.h>
#include <stdlib.h>
```

```
void
main(int argc, char *argv[])
{
    if (testwild(argv[1]))     // wild cards specified?
        syserr(110, 110, NULL);     // exit with message

    ...
}
```

---

## time

Determines and returns the current UTC time.

```
#include <time.h>
time_t time ( time_t * timer )
```

---

*timer*                   »   pointer to storage location for calendar time value

**Operation:**       Using the current system date and time and the global *timezone* and *daylight* variables, the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC is computed. If *timer* is not NULL, this value is copied to it.

**Returns:**         The time and date expressed as the number of seconds since January 1, 1970.

**Errors:**          No errors are detected.

**Notes:**          This function calls [tzset](#).

**Restrictions:**   Current dates before January 1, 1970 or after February 6, 2106, cannot be computed.

**Conforms to:**                **time**    n ANSI    n DOS    n THEOS    n POSIX

**See also:**           [asctime](#), [clock](#), [difftime](#), [gmtime](#), [localtime](#), [mktime](#), [strftime](#), [tzset](#)

---

**Example:**         Refer to the [strftime](#) example on page 612.

**toascii, tolower, \_tolower, toupper, \_toupper**

These functions convert a character.

```
#include <ctype.h>
int toascii ( int c )
int tolower ( int c )
int _tolower ( int c )
int toupper ( int c )
int _toupper ( int c )
```

---

*c*                      »    character to convert

- Operation:**
- toascii**      The value *c* is converted to an ASCII character by masking off all but the lower 7 bits.
  - tolower**      The value *c* is tested and, if it is an uppercase alphabetic character value, it is converted to lowercase.
  - \_tolower**      The value *c* is converted by adding the value 32.
  - toupper**      The value *c* is tested and, if it is a lowercase alphabetic character value, it is converted to uppercase.
  - \_toupper**      The value *c* is converted by subtracting the value 32.

**Returns:**      The converted value of *c*. If *c* is not converted, the original value is returned.

**Errors:**      No errors are detected by these functions.

**Notes:**      The functions *toascii*, *\_tolower* and *\_toupper* are implemented as in-line macros. To use a real function, rather than a macro, use `#define _CTYPE_NO_MACRO` prior to `#include <ctype.h>`.

The functions *\_tolower* and *\_toupper* should only be used on values of *c* that are known to be uppercase or lowercase, as appropriate. These functions do not test the value *c*; they merely add or subtract 32 from it.

The functions *tolower* and *toupper* recognize and convert international characters by using the current locale. Refer to the description of the function [setlocale](#).

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>toascii</b>  | q ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>tolower</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>_tolower</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>toupper</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>_toupper</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**      [isatty](#), [setlocale](#), [strlwr](#), [strupr](#)



**Example:**

```
#include <stdio.h>
#include <ctype.h>

void main() {
    char    string[80];    // buffer for input string
    char    * sptr;        // work pointer
    int     force = 1;      // flag for folding characters

    printf("\nEnter string: ");
    gets(string);          // accept string from operator

    lowercase(string);      // first, fold string to lower

    for (sptr=string; *sptr; ++sptr)    {
        if (isalpha(*sptr))    {
            if (force) {
                *sptr = toupper(*sptr);    // upcase letter
                force = 0;    // reset flag
            }
        }
        else
            force = 1;    // then next char is folded
    }
    printf("%s\n",string); // display modified string
}
```

---

**Output:**

>test

Enter string: NOW IS THE TIME FOR ALL GOOD MEN.  
Now Is The Time For All Good Men.

**topen**

Open a tape file for read or write access.

```
#include <stdio.h>
FILE * topen ( char * filename, char * mode, unsigned reclen, size_t blocklen )
```

---

|                 |   |                           |
|-----------------|---|---------------------------|
| <i>blocklen</i> | » | tape block length         |
| <i>filename</i> | » | pointer to tape file name |
| <i>mode</i>     | » | access mode               |
| <i>reclen</i>   | » | file record length        |

**Operation:** *filename* is opened on the tape device.

When *mode* is “r,” “rb” or “rt,” an existing file is opened. The tape is rewound to the start and searched for a file header specifying the requested *filename*. If the “l” code is used, no rewind is performed and the tape must be already positioned to the desired file header.

When *mode* is “w,” “wb” or “wt,” a new file is opened. The tape is searched for the end-of-tape mark and a new file header is written replacing the tape mark. If the “l” code is used, no searching is performed and the tape must already be positioned to a file header or tape mark. The header or mark is replaced with a new file header for this new file.

The *blocklen* and *reclen* values are written to the file header if the file is opened for “w” access. *blocklen* is also used to allocate the size of the buffer used for read and write access.

**Returns:** A pointer to the opened file’s file control block (fcb). If the file cannot be opened, a NULL pointer is returned and [errno](#) is set to the error code describing the reason for the failure.

**Errors:** When the file cannot be opened and a NULL pointer is returned, [errno](#) is set to one of the following codes:

| <i>Name</i> | <i>Value</i> | <i>Meaning</i>           |
|-------------|--------------|--------------------------|
| ENODEV      | 13           | Tape not attached.       |
| ENOENT      | 19           | File not found.          |
| ENOMEM      | 3            | Insufficient memory.     |
| EMFILE      | 15           | Too many open files.     |
|             | 14           | Invalid access mode.     |
|             | 161          | Tape input/output error. |
|             | 163          | Tape mark encountered.   |

The variables [\\_errnum](#) and [\\_errarg](#) are also set by this function. This means that the [fpererror](#), [strerror](#), *etc.* functions can be used to report the error.

**Notes:** If a tape device other than TAPE1 is desired, specify it in the file name description. For instance, “some.file:tape2.”

The character string *mode* may contain one or more of the following codes:

| code                                                                                 | Meaning                                                                    |
|--------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| One of the following codes <u>must</u> be used to specify the access type.           |                                                                            |
| r                                                                                    | Open an existing file for read access.                                     |
| w                                                                                    | Open a new file for write access.                                          |
| One of the following codes can be added to specify the type of the data in the file. |                                                                            |
| b                                                                                    | File is a binary stream (no record marks).                                 |
| t                                                                                    | File is a text stream.                                                     |
| The following code may be added to specify that tape is not rewind.                  |                                                                            |
| l                                                                                    | Do not rewind tape. The tape must be prepositioned to a tape header label. |

**Restrictions:** When used on a system with a version of THEOS prior to Version 4, `topen` only allows 50 files to be open at any one time.

**Conforms to:** `topen` q ANSI q DOS n THEOS q POSIX

**See also:** [fopen](#)

**tot\_alloc**

Compute the total amount of memory currently in use in your program's data segment.

```
#include <malloc.h>
size_t tot_alloc ( void )
```

**Operation:** The heap space is analyzed and the total of all allocated blocks of memory is computed.

**Returns:** The total amount of memory allocated in your program's data segment.

**Notes:** The size returned by this function refers only to the space allocated by the memory allocation functions ([malloc](#), *etc.*). Space in the data segment used by constants and variables allocated by the compiler and the space used by the stack are excluded.

**Conforms to:** **tot\_alloc** q ANSI q DOS n THEOS q POSIX

**See also:** [\\_alloca](#), [calloc](#), [free](#), [malloc](#), [max\\_alloc](#), [mem\\_avail](#), [realloc](#)

## TSAGetLastError, TSASetLastError, TSAStrError, h\_errno

Get or set the THEOS Sockets API last error code.

```
#include <socket.h>
int h_errno ( void )
int TSAGetLastError ( void )
void TSASetLastError ( int err_code )
char * TSAStrError ( int err_code )
```

---

*err\_code*                   »   error code value to set

**Operation:**       **h\_errno**       Synonym name to the TSAGetLastError function.

**TSAGetLastError** Gets the error code result of the last THEOS Socket function or the last code set by TSASetLastError.

**TSASetLastError** Sets the error code that is retrieved by TSAGetLastError.

**TSAStrError** Returns a pointer to a message string matching the THEOS Sockets error number *err\_code*.

**Returns:**       **h\_errno**       Synonym name to the TSAGetLastError function.

**TSAGetLastError** The last error code set by a particular THEOS Sockets API function or the TSASetLastError function.

**TSAStrError** A pointer to the message string. A NULL pointer is returned if *err\_code* is not a valid THEOS Sockets error code.

**Notes:**       **TSAGetLastError** When a particular THEOS Sockets API function indicates that an error has occurred, this function should be called to retrieve the appropriate error code.

For portability, an application should use TSAGetLastError immediately after the THEOS Sockets API function which failed.

**TSASetLastError** Any subsequent THEOS Sockets routine called by the application will override the error code as set by this routine.

|                     |                        |        |       |         |         |
|---------------------|------------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>h_errno</b>         | q ANSI | q DOS | n THEOS | p POSIX |
|                     | <b>TSAGetLastError</b> | q ANSI | q DOS | n THEOS | p POSIX |
|                     | <b>TSASetLastError</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>TSAStrError</b>     | q ANSI | q DOS | n THEOS | q POSIX |

**Example:**       See the example for the function [socket](#).

## TWS functions

This group of functions provides a means for a program using a THEOS WorkStation to control some of the attributes of that workstation.

```
#include <tws.h>

void TWS_disconnect ( BOOL flag )
void TWS_disconnect_now ( void )
int TWS_execute ( char * prog_name, BOOL wait_flag,
    TWS_WINDOW_TYPE window_type )
void TWS_focus ( void )
void TWS_maximize ( void )
void TWS_minimize ( void )
void TWS_ontop ( BOOL flag )
void TWS_receive ( char * remote_file, char * local_file,
    TWS_ACTION2 action2, TWS_ACTION1 action1 )
void TWS_restore ( void )
void TWS_send ( char * local_file, char * remote_file,
    TWS_ACTION2 action2, TWS_ACTION1 action1 )
void TWS_title ( char * title )
void TWS_user_focus ( BOOL flag )
```

---

|                    |   |                                                             |
|--------------------|---|-------------------------------------------------------------|
| <i>action1</i>     | » | 8-bit coded value specifying action options                 |
| <i>action2</i>     | » | 8-bit coded value specifying additional action options      |
| <i>flag</i>        | » | yes/no (non-zero/zero) valued flag                          |
| <i>local_file</i>  | » | pointer to path and name of THEOS Server file               |
| <i>prog_name</i>   | » | pointer to string containing command-line of program to run |
| <i>remote_file</i> | » | pointer to path and name of TWS client file                 |
| <i>title</i>       | » | pointer to string containing title text                     |
| <i>wait_flag</i>   | » | yes/no (non-zero/zero) valued flag                          |
| <i>window_type</i> | » | value indicating normal, maximized or minimized window      |

**Operation:**     **TWS\_disconnect** Enables or disables the ability for the user to disconnect the workstation manually. When *flag* is zero, the user is prevented from disconnecting the workstation; when *flag* is non-zero, the user may disconnect the workstation manually.

**TWS\_disconnect\_now** Disconnects this workstation. The process of disconnecting requires that the current program is exited, the user is logged off, the user process is stopped and the intranet connection between the two computers for this user is terminated.

**TWS\_execute** Forces the client workstation to execute another program in another window. The *wait\_flag* and *window\_type* arguments specify options controlling the window display on the client

workstation and the action of this program while the command is executed on the client workstation.

| <i>wait_flag</i> |                                                                                                                                                     |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Value            | Meaning                                                                                                                                             |
| zero             | Tells the THEOS NetServer to not wait for the completion of the program launched on the client. Control returns immediately to the calling program. |
| non-zero         | Tells the THEOS NetServer to wait for completion of the program launched on the client before returning control to this calling program.            |

| <i>window_type</i> |                                               |
|--------------------|-----------------------------------------------|
| Value              | Meaning                                       |
| NORMAL_WINDOW      | The window is opened in its “normal” size.    |
| MAXIMIZED_WINDOW   | The window is opened as a full-screen window. |
| MINIMIZED_WINDOW   | The window is opened as an icon.              |

**TWS\_focus** Selects the window used by this program as the active window. If the window is currently minimized, a TWS\_restore operation is performed first.

**TWS\_maximize** The window used by the client workstation is maximized to its full size and is selected as the focus or active window.

**TWS\_minimize** The window used by the client workstation is minimized to an icon and another window is selected as the focus.

**TWS\_ontop** Enables or disables the window “on top” feature. When *flag* is zero, the “on top” feature is disabled; when *flag* is non-zero, the “on top” feature is enabled.

When “on top” is enabled, the window used by the client workstation will be displayed on top of all other windows, even when the window is not the active window.

**TWS\_receive** Transfers the *remote\_file* from the client workstation to the THEOS NetServer, giving it the name *local\_file*. The operation options for display of file names transferred and for what to do

when the file already exists on the THEOS Server are specified in *action1* and *action2*.

| <i>action1</i>      |                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Value               | Meaning                                                                                                                                                                                                                                                                                                                                                         |
| EXIST_ABORT         | When destination file exists, abort the transfer. This source file and any remaining source files are not copied to the destination system.                                                                                                                                                                                                                     |
| EXIST_OLDER         | The source file is copied only if its file date is older than the destination file. (If the destination file does not exist, the source is always copied.)                                                                                                                                                                                                      |
| EXIST_QUERY         | When destination file exists, ask the operator if it is okay to replace it with the source file.                                                                                                                                                                                                                                                                |
| EXIST_REPLACE       | The destination file is always replaced by the source file.                                                                                                                                                                                                                                                                                                     |
| EXIST_SKIP          | When destination file exists, do not replace it with the source file. The source file is not copied in this situation.                                                                                                                                                                                                                                          |
| EXIST_SUBDIR        | If the source file specification is a subdirectory name or it is a wild card specification that may include subdirectories, all matching subdirectories and all of their member files are copied to the destination system.                                                                                                                                     |
| EXIST_TRANSLATE_OEM | <p>When the source file is an ASCII stream file, the record terminators are translated to the receiving system's standard.</p> <p>When the receiving system is the THEOS NetServer, each CR, LF sequence found in the source file is translated to CR; when the receiving system is a DOS-based client, each CR in the source file is translated to CR, LF.</p> |



| <i>action2</i>       |                                                                                                                                                                          |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Value                | Meaning                                                                                                                                                                  |
| EXIST_MOVE           | This action code is ignored.                                                                                                                                             |
| EXIST_TRANSLATE_ANSI | Translates characters between the THEOS code set and the Windows code set. Primarily applies to the interpretation of multinational characters such as á, ñ, <i>etc.</i> |
| EXIST_NOTYPE         | Do not display the results of each file transferred.                                                                                                                     |

**TWS\_restore** The window used by the client workstation is restored to the size used prior to the last maximize or minimize window operation.

**TWS\_send** Transfers the *local\_file* from the THEOS NetServer to the client workstation, giving it the name *remote\_file*. The operation options for display of file names transferred and for actions to take when the file already exists on the workstation are specified in *action1* and *action2*. Refer to their descriptions under the TWS\_receive function description.

**TWS\_title** Sets the title text displayed for the window on the client workstation.

**TWS\_user\_focus** Enables or disables the ability of the user to select another window on the client workstation. When *flag* is zero, the user is prevented from selecting another window. When *flag* is non-zero, the user may select another window.

**Returns:** **TWS\_execute** Success/fail indicator. A zero return value indicates success; a non-zero return is the failure code, as defined by the program executed or the client operating system.

**Notes:** When using TWS\_execute, TWS\_send or TWS\_receive, any path names in the command line text *prog\_name* or *remote\_file* are normalized to the client environment. That is, the subdirectory character “/” used for the THEOS environment is converted to the “\” used in the Windows environment. Also, drive-code specifications are moved from the end-of-a file specification (as used in THEOS) to the beginning of the path (as used by Windows).

**Restrictions:** These functions may only be used in a program on a system using the THEOS NetServer version of the operating system and only when executed in a user process whose console is a THEOS WorkStation.

The length limit of the file name or path components is eight characters.

**Conforms to:**

|                           |        |       |         |         |
|---------------------------|--------|-------|---------|---------|
| <b>TWS_disconnect</b>     | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_disconnect_now</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_execute</b>        | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_focus</b>          | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_maximize</b>       | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_minimize</b>       | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_ontop</b>          | q ANSI | q DOS | n THEOS | p POSIX |

## 646 *TWS functions*

---

|                       |        |       |         |         |
|-----------------------|--------|-------|---------|---------|
| <b>TWS_receive</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_restore</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_send</b>       | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_title</b>      | q ANSI | q DOS | n THEOS | q POSIX |
| <b>TWS_user_focus</b> | q ANSI | q DOS | n THEOS | q POSIX |

---

## tzset

Set the global time zone values.

```
#include <time.h>
void tzset ( void )
```

**Operation:** The `tzset` function sets the global variables *daylight*, *timezone* and *tzname* according to the current TZ environment variable value. These variables have the following values after `tzset` is executed.

- daylight**     A true/false value indicating whether or not daylight-savings time is supported. A zero value means that it isn't; a value of one indicates that it is. This variable is set to one when the TZ environment variable has a name defined for daylight-savings time.
- timezone**    Value indicating the number of seconds that must be added to UTC time to arrive at local time. Positive values indicate that the local time zone is west of the prime meridian; negative values indicate that you are east of the prime meridian.
- tzname**       This two-element array contains the names for the standard and daylight-savings time zones. The values for this array are copied from the TZ environment variable.

**Notes:** The TZ environment variable is set by the boot-up process using information from the system configuration file, by account environment definitions, or by the SET command. Functions:

The functions [ctime](#), [localtime](#), [mktime](#) and [strftime](#) (when the %Z directive is used) call this `tzset` function.

**Defaults:** When TZ is undefined, the values for the variables default to `daylight=0`, `timezone=0`, `tzname[0]=""` and `tzname[1]=""`. This corresponds to UTC.

**Conforms to:**                      **tzset**     q ANSI     n DOS     n THEOS     n POSIX

**See also:**                      [asctime](#), [ctime](#), [gmtime](#), [localtime](#), [mktime](#), [strftime](#), [time](#)

---

**Example:**                      Refer to the [strftime](#) example on page 612.

### **uncache, \_uncache**

Clear the processor's L1 and L2 memory cache.

```
#include <driver.h>
void uncache ( unsigned selector )

#include <sc.h>
void _uncache ( unsigned selector )
```

---

*selector*                    »    memory selector

**Operation:**        Mark the *selector* memory region as uncachable.

**Notes:**            When a memory region is defined with the `make_sel` or `_make_sel` functions and that memory will be used as dual-ported memory (DMA), this `uncache` or `_uncache` function should be used immediately following the `make_sel` or `_make_sel` function call.

**Restrictions:**    This function is intended for device-driver authors. Selectors to dual-ported memory should always have this function performed on them.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>uncache</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>_uncache</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**        [make\\_sel](#), [\\_make\\_sel](#)

## ungetc

Puts a character onto a file's input stream.

```
#include <stdio.h>
```

```
int ungetc ( int c, FILE * file )
```

---

*c*                      »    character to “unget”

*file*                    »    pointer to file's fcb

**Operation:**        The character *c* is placed in the file's input stream. It will be the next character returned by subsequent read operation on *file*.

The end-of-file flag for *file* is cleared.

**Returns:**            When successful, the character *c* is returned. When the `ungetc` is unsuccessful, an EOF is returned.

**Errors:**            Probable causes for an EOF return value include: *file* is invalid, the file is not open for reading, no read operation has been performed on *file* yet, or there is no room in the *file*'s input buffer to store *c*.

**Notes:**            An attempt to `ungetc` an EOF character is ignored.

The `ungetc` operation has no effect on the actual contents of the file. Only the file's input buffer is affected.

The value of the file positioning pointers is unspecified until all pushed-back characters are read or discarded.

**Restrictions:**     *file* must be open for reading and there must have been at least one character read from the file stream prior to using `ungetc`.

**Conforms to:**        `ungetc`    n ANSI    n DOS    n THEOS    n POSIX

**See also:**            [fgetc](#), [getc](#), [getch](#), [getchar](#), [putc](#), [putch](#), [putchar](#)

### ungetch

Place a character in the console's input stream.

```
#include <conio.h> or <stdio.h>
int ungetch ( char c )
_____
c                »   character to "unget"
```

**Operation:** If there is space available in the console's input buffer, the character *c* is placed at the head of this buffer. The next input from the console (or `stdin` if `stdin` is `conin`) will be this character.

**Returns:** The character *c*. If the buffer was full and the "unget" operation could not be performed, an EOF is returned.

**Notes:** The character *c* does not have to be a character that was actually read from the console.

**Restrictions:** Attempting to "unget" an EOF character is ignored.

**Conforms to:** `ungetch` q ANSI n DOS n THEOS q POSIX

**See also:** [ungetc](#)

---

#### Example:

```
#include <stdio.h>
#include <func_key.h>
#include <conio.h>

int chk_char(void)
{
    char    c;

    c = getch();    // get next char
    if (c==QUIT)    // is it the QUIT key?
        return c;  // yes--return it
    ungetch(c);     // no--put back
    return NULL;    // return nothing (0)
}
```

## unlink

Erases a file from disk.

```
#include <io.h> or <stdio.h>
int unlink ( char * filename )
```

---

*filename*           »   pointer to string containing file description

**Operation:**       The file specified by *filename* is erased: All disk space previously used by the file is returned to the disk's free space map. The directory entry for the file is marked as deleted.

**Returns:**         A success/fail indicator. A return value of zero indicates success; a return value of -1 indicates failure.

**Errors:**         When the return value is non-zero, then *filename* was not erased. The return value is the code indicating the failure reason. Both [errno](#) and [\\_errnum](#) are set to this reason code and [\\_errarg](#) is set to *filename*.

**Notes:**         The contents of *filename* should be complete and explicit. If *filename* does not specify a path, then the current working directory is assumed. If *filename* does not specify a drive, then all drives specified in the current drive search sequence are used until the file is either found or all drives in the search sequence are examined. If *filename* does not specify a file type, then only the current default library is searched.

The unlink function name is the POSIX name for the file-erase operation. The [remove](#) function is the ANSI name for the operation and [erase](#) is the THEOS name. The difference between unlink and [remove](#) is in their return values: unlink returns a success/failure value while [remove](#) returns a success/failure code.

The unlink function is declared in both the `STDIO.H` and the `IO.H` header files. However, the declarations differ between the two headers. When a program includes the `IO.H` file, the references to the unlink function will refer to the actual unlink function. When `STDIO.H` is included but `IO.H` is not included, references to the unlink function are translated into references to the [remove](#) function.

**Restrictions:**   You may not erase a file that is currently open in any program or task, including your own.

**Conforms to:**       **unlink**    q ANSI    n DOS    n THEOS    n POSIX

**See also:**         [close](#), [erase](#), [remove](#)

## 652 *unlink*

---

### Example:

```
#include <stdio.h>
#include <string.h>
#include <io.h>
#include <errno.h>

void
main(int argc, char * argv[])
{
    char    filename[256];

    if (argc) {                // file specified?
        strcpy(filename, argv[1]);
        printf("\nThe file \"%s\" was ",filename);

        if (unlink(filename))
            printf("not erased. errno = %d\n", errno);
        else
            printf("erased.");
    }
}
```

---

### Output:

>test some.file

The file "some.file" was erased.

>test some.file

The file "some.file" was not erased. errno = 19

>



## unlock

Remove all record locks on a file.

```
#include <stdio.h>
void unlock ( FILE * file )
```

---

*file*                   »   pointer to open file's FCB

**Operation:**       All record locks on *file* are removed.

**Notes:**           Records are locked with either the [reclock](#) function (stream and direct files), [filelock](#), or with the automatic record-locking mechanism enabled when a direct, indexed or keyed file is opened with update mode.

**Defaults:**       Record locks are also removed when *file* is closed.

**Restrictions:**   *file* must be open.

**Conforms to:**       **unlock**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**       [filelock](#), [reclock](#), [recunlock](#)

**utoa**

Converts an unsigned integer value to an ASCII string.

```
#include <stdlib.h>
char * utoa ( char * string, unsigned value )
```

---

*string*                   »   pointer to storage area

*value*                    »   value to convert

**Operation:**       The integer *value* is converted into a string, similar to the actions of the [sprintf](#) function with the format string “%u.” The converted string is stored in the location pointed to by *string*.

**Returns:**         A pointer to *string*.

**Conforms to:**         **utoa**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [sprintf](#)

## **va\_arg, va\_dcl, va\_end, va\_start**

Access the arguments of a variable argument list.

```
#include <stdarg.h>                // ANSI & THEOS versions
type va_arg ( va_list arg_ptr, type )
void va_end ( va_list arg_ptr )
void va_start ( va_list arg_ptr, void * prev_param )

#include <varargs.h>                // UNIX versions
va_alist void *_valast, ...
type va_arg ( va_list arg_ptr, type )
va_dcl
void va_end ( va_list arg_ptr )
void va_start ( va_list arg_ptr )
```

---

*arg\_ptr*           »   pointer to list of arguments  
*prev\_param*       »   pointer to last argument prior to variable arg list  
*type*             »   type of argument to get, i.e., int, char, long, etc.

**Operation:**       These functions are implemented as macros and two different versions of the macros are provided for compatibility with UNIX and ANSI standards. For a description of the macros in the `VARARGS.H` header, refer to your UNIX programming manual. These macros are similar to, but not interchangeable with, the macros defined in the `STDARG.H` header file.

**va\_arg**           Retrieves a value of *type* from the location pointed to by *arg\_ptr*. The pointer in *arg\_ptr* is incremented by the size of *type* so that it points to the next argument in the list of arguments passed to the function.

The *va\_arg* macro may be used any number of times in a function to retrieve the arguments in its variable argument list.

**va\_end**           Reset the pointer *arg\_ptr* to NULL.

**va\_start**         Set the *arg\_ptr* pointer to the first optional argument in the list of arguments passed to the function. *prev\_param* is the name of the last required parameter passed to the function.

**Notes:**           These macros are used in functions that receive a variable number of arguments in their calling sequence. It is assumed that there are a fixed number of required arguments followed by a variable number of optional arguments.

For instance, the declaration of `myfct` might be:

```
myfct(char *fn, ...)
```

This means that `myfct` requires a single argument that is a pointer to a character and may also be passed any number of additional arguments. It might be called with:

```
myfct(filename, arg1, arg2, arg3)
```

## 656 *va\_arg, va\_dcl, va\_end, va\_start*

---

In the myfct code, the `va_start`, `va_end` and `va_arg` macros would be used to access the optional arguments (`arg1`, `arg2` and `arg3` in this case).

**Restrictions:** You must know the type of each of the optional arguments because it is required by the `va_arg` macro when it calculates the size of each argument.

The `va_start` macro must be used prior to using `va_arg`. It initializes the pointer to point to the first of the list of arguments.

There must be at least one required argument preceding the first optional argument in the calling sequence of a function that uses these macros.

|                     |                        |        |       |         |         |
|---------------------|------------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | (UNIX) <b>va_alist</b> | q ANSI | q DOS | n THEOS | n POSIX |
|                     | <b>va_arg</b>          | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>va_end</b>          | n ANSI | n DOS | n THEOS | n POSIX |
|                     | (UNIX) <b>va_dcl</b>   | q ANSI | q DOS | n THEOS | n POSIX |
|                     | <b>va_start</b>        | n ANSI | n DOS | n THEOS | q POSIX |
|                     | (UNIX) <b>va_start</b> | q ANSI | q DOS | n THEOS | n POSIX |

**See also:** [vfprintf](#), [vprintf](#), [vsprintf](#)

---

### Example:

```
#include <stdio.h>
#include <stdarg.h>

int average(int, ...);

void
main()
{
    printf("\nThe average of 1,2,3,4,5,6,7 is %i\n",
        average(1,2,3,4,5,6,7,-1));
}

int
average(int first, ...)
{
    int    val,
           count,
           total = first;
    va_list ptr;

    va_start(ptr, first);    // initialize var arg list

    for (count=1; (val=va_arg(ptr, int))!=-1; ++count) {
        total += val;
    }
    va_end(ptr);            // reset var arg

    return total/count;
}
```

---

### Output:

>example

The average of 1,2,3,4,5,6,7 is 4

## vdi functions

This group of functions provide access to the Virtual Device Interface, allowing a program to draw objects on a graphics output device.

```
#include <vdi.h>
void vdi ( VDIPB * vpb )
void vdialpha ( VDIPB * vpb )
void vdiiarc ( VDIPB * vpb, int x, int y, int radius, int ang1, int ang2 )
void vdiibar ( VDIPB * vpb, int xll, int yll, int xur, int yur )
void vdicircle ( VDIPB * vpb, int x, int y, int radius )
void vdiclear ( VDIPB * vpb )
void vdiclose ( VDIPB * vpb )
void vdigraph ( VDIPB * vpb )
void vdiline ( VDIPB * vpb, int x1, int y1, int x2, int y2 )
void vdimark ( VDIPB * vpb, int x, int y )
VDIPB * vdiopen ( int number )
void vdi pie ( VDIPB * vpb, int x, int y, int radius, int ang1, int ang2 )
void vdisetfc ( VDIPB * vpb, int color )
void vdisetfs ( VDIPB * vpb, int style )
void vdisetlc ( VDIPB * vpb, int color )
void vdisetlh ( VDIPB * vpb, int size )
void vdisetls ( VDIPB * vpb, int style )
void vdisetmc ( VDIPB * vpb, int color )
void vdisetmh ( VDIPB * vpb, int size )
void vdisetms ( VDIPB * vpb, int style )
void vdisetta ( VDIPB * vpb, int angle )
void vdisettc ( VDIPB * vpb, int color )
void vdisetth ( VDIPB * vpb, int size )
void vdisettp ( VDIPB * vpb, int path )
void vdisetts ( VDIPB * vpb, int style )
void vditext ( VDIPB * vpb, int x, int y, char * string )
```

|                        |   |                                |
|------------------------|---|--------------------------------|
| <i>ang<sub>1</sub></i> | » | starting angle of arc          |
| <i>ang<sub>2</sub></i> | » | ending angle of arc            |
| <i>angle</i>           | » | angle of text                  |
| <i>color</i>           | » | color code                     |
| <i>num</i>             | » | vdi device number              |
| <i>path</i>            | » | path direction                 |
| <i>radius</i>          | » | length of radius               |
| <i>size</i>            | » | height value                   |
| <i>style</i>           | » | style code                     |
| <i>string</i>          | » | pointer to text                |
| <i>vpb</i>             | » | pointer to vdi structure       |
| <i>x</i>               | » | x value                        |
| <i>x<sub>1</sub></i>   | » | x value for line start         |
| <i>x<sub>2</sub></i>   | » | x value for line end           |
| <i>x<sub>ll</sub></i>  | » | x value for lower-left corner  |
| <i>x<sub>ur</sub></i>  | » | x value for upper-right corner |
| <i>y</i>               | » | y value                        |
| <i>y<sub>1</sub></i>   | » | y value for line start         |
| <i>y<sub>2</sub></i>   | » | y value for line end           |
| <i>y<sub>ll</sub></i>  | » | y value for lower-left corner  |
| <i>y<sub>ur</sub></i>  | » | y value for upper-right corner |

**Operation:**      **vdi**      This is the primitive routine that is used by all of the other vdi functions (except *vdlopen*). The *vpb* structure describes the action to be performed and identifies the device to use.

Although *vdi* is the function used by the other functions described here, and it is the one that performs the actual drawing or initialization operation, the operations performed by this function are described in each of the following function descriptions.

Normally the *vdi* function is not used because its capabilities are more easily accessed by using the specific function for the task.

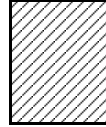
**vdialpha**      Sets the graphics device to its text mode. This is the opposite operation of the *vdigraph* function. See the section on restrictions for important comments.

**vdia**      Draws an arc on the open graphics device *vpb*. The center of this arc is at *x,y* coordinates, it has a radius of *radius* and the end-points of the arc are defined by the two angles *ang<sub>1</sub>* and *ang<sub>2</sub>*. The angles are specified in tenths of degrees. Therefore, a specification of 450 is an angle of 45°.



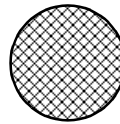
The line of the arc is solid and is drawn using the color last defined by the `vdisetlc` (line color) function or its default value.

**vdibar** Draws an enclosed rectangle on the open graphics device *vdp*. The sides of this rectangle are parallel to the edges of the device. The lower-left corner of the rectangle is at  $x_{ll}, y_{ll}$  coordinates and the upper-right corner of the rectangle is at  $x_{ur}, y_{ur}$  coordinates.



The rectangle is drawn using solid lines with the color last defined by the `vdisetlc` (line color) function or its default value. The interior of the rectangle is filled according to the parameters established with the last usage of the `vdisetfc` (fill color) and `vdisetfs` (fill style) functions or their default values.

**vdicircle** Draws an enclosed circle on the open graphics device *vdp*. The center of this circle is at  $x, y$  coordinates and it has a radius of *radius*.



The circle is drawn using a solid line with the color last defined by the `vdisetlc` (line color) function or its default value. The interior of the circle is filled according to the parameters established with the last usage of the `vdisetfc` (fill color) and `vdisetfs` (fill style) functions or their default values.

**vdiclear** The vdi device is cleared, similar to performing a form feed on a text device.

**vdiclose** Closes the vdi device *vdp*. A `vdiclear` is always performed prior to closing the device. After a graphics device is closed it is in alphabetic or text mode, if appropriate.

**vdigraph** Sets the graphics device to its graphics mode. This is the opposite operation of the `vdialpha` function.

This function is not normally used because all of the vdi functions force the device into graphics mode if needed.

**vdiline** Draws a line on the open graphics device *vdp*. The end points of this line are defined by the  $x_1, y_1$  and  $x_2, y_2$  coordinates.



The style, color and width of the line are drawn using the parameters established with the last usage of the `vdisetls` (line style), `vdisetlc` (line color) and `vdisetlh` (line height) functions, or their default values.

**vdimark** Draws a mark on the open graphics device *vdp*. The location of this mark is defined by the *x,y* coordinate.



The shape, color and size of this mark is drawn using the parameters established with the last usage of the **vdisetms** (mark style), **vdisetmc** (mark color) and **vdisetmh** (mark height) functions, or their default values.

**vdioopen** Opens the attached graphics device named *VDI*number.

**vdipie** Draws a wedge or “pie slice” on the graphics device *vdp*. The center of this wedge is at *x,y* coordinates, it has a radius of *radius* and the end points of the arc are defined by the two angles *ang*<sub>1</sub> and *ang*<sub>2</sub>. The angles are specified in tenths of degrees. Therefore, a specification of 450 is an angle of 45°.



The wedge is drawn using solid lines, with the color last defined by the **vdisetlc** (line color) function, or its default value. The interior of the wedge is filled according to the parameters established with the last usage of the **vdisetfc** (fill color) and **vdisetfs** (fill style) functions, or their default values.

**vdisetfc** Sets the fill color used by the functions **vdibar**, **vdicircle** and **vdipie** to fill the enclosed areas.

| Code | Color | Code | Color   |
|------|-------|------|---------|
| 0    | Black | 4    | Red     |
| 1    | Blue  | 5    | Magenta |
| 2    | Green | 6    | Yellow  |
| 3    | Cyan  | 7    | White   |

**vdisetfs** Sets the fill style used by the functions **vdibar**, **vdicircle** and **vdipie** to fill the enclosed areas.

| Code | Style                | Sample |
|------|----------------------|--------|
| 0    | Hollow               |        |
| 1    | Solid                |        |
| 2    | Vertical lines       |        |
| 3    | Horizontal lines     |        |
| 4    | 45° diagonals        |        |
| 5    | 135° diagonals       |        |
| 6    | Cross hatch          |        |
| 7    | Diagonal cross hatch |        |

Table 15: Fill Styles



- vdisetlc** Sets the line color used by other functions to draw lines or borders around objects. The color codes are the same as those used by [vdisetfc](#).
- vdisetlh** Sets the line height used by other functions to draw lines or borders around objects.
- vdisetls** Sets the line style used by other functions to draw lines or borders around objects.

| Code | Style          | Sample      |
|------|----------------|-------------|
| 1    | Solid          | —————       |
| 2    | Short dashes   | - - - - -   |
| 3    | Dotted         | . . . . .   |
| 4    | Dash, dot      | - . - . -   |
| 5    | Long dashes    | - - - - -   |
| 6    | Dash, dot, dot | - . . - . . |
| 7    | Small dots     | .....       |

Table 16: Line Styles

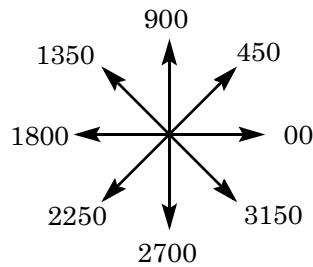
- vdisetmc** Sets the marker color used by the [vdimark](#) function to draw a marker. The color codes are the same as those used by [vdisetfc](#).
- vdisetmh** Sets the marker height used by the [vdimark](#) function to draw a marker.
- vdisetms** Sets the marker style used by the [vdimark](#) function to draw a marker.

| Code | Style     | Sample |
|------|-----------|--------|
| 0    | Dot       | •      |
| 1    | Plus sign | +      |
| 2    | Asterisk  | ✱      |
| 3    | Circle    | ○      |
| 4    | X         | ×      |

Table 17: Marker Styles

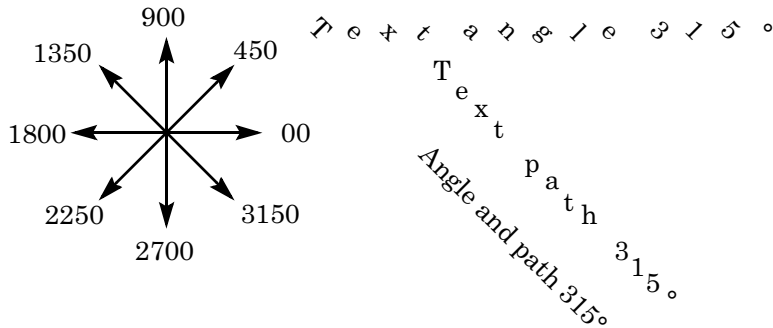
- vdisetta** Sets the text angle used by the [vditext](#) function to draw the text characters. The *angle* value is the requested text angle in tenths of a degree. Therefore, a text angle of 45° is specified with a value of 450.

The text angle refers to the orientation of each character displayed. Compare with the text path.



- vdissetc** Sets the text color used by the `vditext` function to draw text. The color codes are the same as those used by `vdissetfc`.
- vdisseth** Sets the text height used by the `vditext` function to draw text. The size argument should be a low-valued integer with 0 being the smallest size.
- vdissetp** Sets the text path used by the `vditext` function to draw the text characters. The *angle* value is the requested text path in tenths of a degree. Therefore, a text path of 45° is specified with a value of 450.

The text path refers to the orientation of the sequence of characters displayed. Compare with the text path:



**vdissets** Sets the text style used by the `vditext` function to draw the text characters.

| Code | Style                    |
|------|--------------------------|
| 1    | Normal                   |
| 2    | Double-wide              |
| 3    | Double-high              |
| 4    | Double-wide, double-high |

**vditext** Draws text on the open graphics device `vdp`. The text pointed to by *string* is displayed, starting at coordinate *x,y*.

Each character of the text is drawn using the angle defined by the last usage of `vdisseta`. The path of the characters is determined by the angle defined with the last usage of `vdissetp`. The text is drawn using the color last defined by the `vdissetc` (text color) function or its default value. The height and style of the text are specified by the parameters established with the last usage of the `vdisseth` (text height) and `vdissets` (text style) functions, or their default values.

**Returns:** **vdioopen** When the graphics device is opened successfully, a pointer to a VDIPB structure is returned. A null pointer is returned when the open operation fails.

**Errors:** If any errors are detected during an operation, the operation is ignored.

**Notes:** **vdialpha** You can display graphical text on a graphics device by using the `vditext` function.

**vdiaarc** The shape of the arc might not be the arc of a true circle, depending upon the ratio of the horizontal and vertical resolution of the device and the ability of the vdi device-driver to scale these ratios into a uniform scale.

**vdibar** Unless the fill style is hollow, the entire area of the rectangle is filled by this function: Any shapes, figures or text that were drawn in this region prior to this function are overlaid. For a graphics screen, this means that these prior graphics will probably be destroyed; for a graphics printer/plotter, they may be over-printed.

For consistent and reliable output, draw the filled areas before drawing other objects that should exist inside the filled area.

**vdicircle** Same as `vdibar` above.

The shape of the circle might not be a true circle, depending upon the ratio of the horizontal and vertical resolution of the device and the ability of the vdi device-driver to scale these ratios into a uniform scale.

**vdiclear** A clear operation is performed automatically by the `vdiclose` function.

Some graphics devices perform a clear operation when the `vdialpha` function is used.

**vdioopen** The value returned by a successful open operation is the value used by all of the other vdi functions.

**vdipie** Unless the fill style is hollow, the entire area of the wedge is filled by this function: Any shapes, figures or text that were drawn in this region prior to this function are overlaid. For a graphics screen, this means that these prior graphics will probably be destroyed; for a graphics printer/plotter, they may be over-printed.

For consistent and reliable output, draw the filled areas before drawing other objects that should exist inside the filled area.

The shape of the arc of the wedge might not be the arc of a true circle, depending upon the ratio of the horizontal and vertical resolution of the device and the ability of the vdi device-driver to scale these ratios into a uniform scale.

**vdisetfs** More fill styles may be available on specific graphic devices. The styles listed here are the minimum that are always available.

**vdisetlh** All graphical devices have at least one line height available (one unit). More heights may be available on some devices.

**vdisetls** The styles documented here are the minimum styles that a graphics device supports. Specific devices may support additional line styles.

**vdisetmh** There are at least 11 marker heights available on all vdi devices. More may be available on some. The minimum marker heights available include: 400, 500, 600, 700, 900, 1000, 1200, 1300, 1400, 1600 and 1700 VDI units.

When the marker style is 1 (dot), marker height does not apply.

**vdisetms** The styles documented here are the minimum styles that a graphics device supports. Specific devices may support additional marker styles.

**vdisetta** Graphics devices do not normally support all possible text angles. However, almost all devices will support text angles that are multiples of 45°. Therefore, you should restrict the text angles used to 0, 45, 90, 135, 180, 225, 270 and 315 degrees.

**vdisetth** At a minimum, graphics devices support one text height. Other text heights may be supported depending upon the capabilities of the device and the vdi device-driver.

**vdisetts** Many graphics devices do not support multiple text styles. At a minimum, only style 1 (normal) is supported.

**Defaults:** When a graphics device is first opened, the following attributes are in effect until changed by one of the `vdiset` functions:

| Attribute          | Default value |
|--------------------|---------------|
| Graphics/Text mode | Graphics      |
| Fill color         | Black         |
| Fill style         | Solid         |
| Line color         | Black         |
| Line height        | 1             |
| Line style         | 1 (solid)     |
| Marker color       | Black         |
| Marker height      | 100           |
| Marker style       | 1 (dot)       |
| Text angle         | 0             |
| Text color         | Black         |
| Text height        | 1             |
| Text path          | 0             |
| Text style         | 1 (normal)    |

**Restrictions:** **vdialpha** This routine is useful only when the graphics device is also a normal output device such as a graphics console or printer.

Many graphics devices cannot support alphabetic mode while there are graphics displayed. For those types of devices the transition from graphics to alphabetic mode causes a close operation that form-feeds or clears the displayed page.

**vdioopen** The value of *number* must be in the range 1–4 and the corresponding device (VDI1, VDI2, VDI3 or VDI4) must be attached.

**Conforms to:**

|                  |        |       |         |         |
|------------------|--------|-------|---------|---------|
| <b>vdi</b>       | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdialpha</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdiaarc</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdibar</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdicircle</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdiclear</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdiclose</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdigraph</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdiline</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdimark</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdioopen</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdipie</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetfc</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetfs</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetlc</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetlh</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetls</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetmc</b>  | q ANSI | q DOS | n THEOS | q POSIX |

|                 |        |       |         |         |
|-----------------|--------|-------|---------|---------|
| <b>vdisetmh</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetms</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetta</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisettc</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetth</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisettp</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vdisetts</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>vditext</b>  | q ANSI | q DOS | n THEOS | q POSIX |

---

**Example:**

```
#include <stdlib.h>
#include <time.h>
#include <vdi.h>

void
main()
{
    VDIPB      *v;

    if (!(v = vdiopen(1)))          // open VDI1
        syserr(253, 253, NULL);

    vdisetlc(v, 7);                 // lines are white
    vdisetfc(v, 1);                 // fill color is blue
    vdisetfs(v, 1);                 // fill style is solid
    vdipie(v, 20000, 20000, 10000, 0, 450); // 45 degree slice

    vdisetfc(v, 6);                 // fill color is yellow
    vdisetfs(v, 2);                 // fill style is vertical lines
    vdipie(v, 20000, 20000, 10000, 450, 1350); // 90 degree slice

    vdisetfc(v, 3);                 // fill color is cyan
    vdisetfs(v, 6);                 // fill style is cross hatch
    vdipie(v, 20000, 20000, 10000, 1350, 3600); // remainder

    sleep(5000);

    vdiclose(v);                   // finished with device
}
```

## fprintf, printf, vsprintf

Create formatted output on a file, stdout or a string buffer, using a variable argument list.

```
#include <stdio.h>

int fprintf ( FILE * file, const char * format, va_list arg )
int printf ( char * format, va_list arg )
int vsprintf ( char * buffer, char * format, va_list arg )
```

---

|               |   |                                  |
|---------------|---|----------------------------------|
| <i>arg</i>    | » | pointer to a list or arguments   |
| <i>buffer</i> | » | pointer to output buffer         |
| <i>file</i>   | » | pointer to open file's fcb       |
| <i>format</i> | » | pointer to format control string |

**Operation:** These three functions are identical in operation to the functions [fprintf](#), [printf](#) and [sprintf](#), respectively, except that the list of arguments to format are not provided directly in the function call, but rather as a pointer to an array of pointers to the arguments. These functions utilize the [va\\_arg](#) function capabilities to access the individual arguments in the [va\\_list](#).

**fprintf** The values pointed to by the *arg* are formatted according to the specifications in *format* and output to *file*.

**printf** The values pointed to by the *arg* are formatted according to the specifications in *format* and output to stdout.

**vsprintf** The values pointed to by the *arg* are formatted according to the specifications in *format* and output to the location *buffer*.

**Returns:** All three functions return the number of characters successfully output to their respective devices or memory buffer. A return of a negative value indicates that an error occurred.

**Notes:** The *arg* value should be initialized with the [va\\_start](#) function and it may have already been partially processed with the [va\\_arg](#) function.

These functions do not use the [va\\_end](#) function.

If *file* has been opened with “r+” access, [fprintf](#) checks to see if the requested location in the file is locked by another user with the [filelock](#) function. If it is locked, the function waits for the lock to be released before reading the data. [fprintf](#) does not check for locks by other users placed with automatic record-locking or the [reclock](#) function.

If *file* was not opened with “r+” access, [fprintf](#) does not check for locks by other users.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>fprintf</b>  | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>printf</b>   | n ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>vsprintf</b> | n ANSI | n DOS | n THEOS | n POSIX |

**See also:** [fprintf](#), [printf](#), [sprintf](#), [va\\_arg](#), [va\\_dcl](#), [va\\_end](#), [va\\_start](#)

**Example:**

```
#include <stdio.h>
#include <stdarg.h>

void
example(char * text, char * format, ...)
{
    va_list arg_ptr;                // pointer to arguments

    va_start(arg_ptr, format);      // set up arg pointer

    printf(text);                   // output leading text

    vprintf(format, arg_ptr);       // output remainder

    va_end(arg_ptr);                // cleanup
}
```



***\_wait, \_unwait***

Suspend operation of a process until an interrupt occurs.

```
#include <sc.h>
void _wait ( void )
void _unwait ( int pid )
```

---

*pid*                   »   process number

**Operation:**        ***\_unwait***       The process identified by *pid* is awakened if it was waiting for an interrupt to occur. Its “wait for interrupt” flag is cleared.

***\_wait***        The current process is suspended until the next interrupt occurs or until another process uses the *\_unwait* function with the partition number of this process. The “wait for interrupt” flag in the Process Control Block (PCB) is set.

**Notes:**            Interrupts occur constantly and when a process is awakened by an interrupt, it may not be the interrupt desired. The program should test what it needs to and, if it is not the desired interrupt, use the *\_wait* function again.

**Restrictions:**    These functions are intended for device-driver programs.

**Conforms to:**        ***\_unwait***    q ANSI     q DOS     n THEOS    q POSIX  
                          ***\_wait***    q ANSI     q DOS     n THEOS    q POSIX

**wChoice, wChoiceFuncKeys, wChoiceSelect, wChoiceTimeout**

Display a menu of choices in a window and allow the operator to select one.

```
#include <wmapi.h>
int wChoice ( int window, int count, char * items[], int begin, int * hot_keys,
             int prompt_win, char * prompts[], int * grayed, void (*help_func)(int line) )
void wChoiceFuncKeys ( wCHOICE_FUNC_KEYS * keys )
int wChoiceSelect ( void )
void wChoiceTimeout ( int duration )
```

---

|                   |   |                                                          |
|-------------------|---|----------------------------------------------------------|
| <i>begin</i>      | » | item number to start with                                |
| <i>count</i>      | » | total number of items to display                         |
| <i>duration</i>   | » | response time limit, in milliseconds                     |
| <i>grayed</i>     | » | pointer to list of items to display in reduced intensity |
| <i>help_func</i>  | » | name of help function to use when <b>F1</b> pressed      |
| <i>hot_keys</i>   | » | pointer to list of hot-key character positions           |
| <i>items</i>      | » | pointer to array of item text string pointers            |
| <i>keys</i>       | » | pointer to wCHOICE_FUNC_KEYS structure                   |
| <i>prompts</i>    | » | pointer to array of pointers to prompt text strings      |
| <i>prompt_win</i> | » | window number for prompt text display                    |
| <i>window</i>     | » | window number for choice display                         |

**Operation:** These four functions are related. The **wChoice** function is the primary function and the other three set conditions for its operation or report its result.

**wChoice** Using the open window number *window*, the list of choice strings specified with *items* is displayed, one item per line, with a leading space before each item.


*count* specifies how many of the *items* text strings are to be used.

The four arrays *items*, *hot\_keys*, *prompts* and *grayed* are all related by their indices. That is, the first item in *hot\_keys* is the hot-key character position of the first item in *items*; the first item in *grayed* is the yes/no flag for graying the first item in *items*; and the first item in *prompts* is the prompt text for the first item in *items*.

The function of these four arrays can be disabled by specifying their values with NULL.

If the *grayed* array indicates that an item is grayed (non-zero in the corresponding index), the *items* text is displayed in reduced intensity and cannot be selected.


If the *hot\_keys* array has a non-zero value for an item, that character position of the *items* text is displayed with the underline attribute. When an item has a hot-key specified, the operator can select the item by entering that character.

If two or more items have the same hot-key character value specified and the operator enters that character, the next item with that character as its hot-key is positioned to but not selected. The operator must select the item with the  key. If there is only one item that uses a particular hot-key character value, that item is selected when the operator enters that hot-key. You should not mix unique hot-keys with non-unique hot-keys because the operation of the choice window will be confusing to the operator.

If the *prompts* array has a text string defined for an item, that text string is displayed in *prompt\_win* whenever the operator points to the item text.

The *begin* argument specifies the index of the *items* text string that is highlighted initially.

If the *window*'s height is less than *count*, a scroll bar is displayed on the right side of the frame and the *items* text scrolls within the window. If the *window*'s width is less than the longest text item, the text is truncated at the right edge of the window, even if *window* has `WCLIP_WRAP` set.

Once the choice list is displayed in window, the operator can choose one of the items by positioning to it with the arrow keys or the space-bar and then selecting it with the  key. See the notes section about other keys that can be used.

**wChoiceFuncKeys** This function defines a list of function keys that can be used while a `wChoice` function operates. *keys* is a pointer to a `WCHOICE_FUNC_KEYS` structure.

This structure contains a list of keystroke values and the values that they should be translated to. During the operation of the `wChoice` function, if the operator presses any of the keys listed, the translate-to value will be returned as the value of `wChoice`.


**wChoiceSelect** Returns the index (base 0) of the item pointed to from the last call to `wChoice`.

**wChoiceTimeout** Defines the time limit, in milliseconds, allowed for a response to the `wChoice` function.

When *duration* is non-zero, a count-down clock is started when `wChoice` is called and, during the processing of `wChoice`, every time that a keystroke is entered by the operator. When there has been no keyboard activity for *duration* milliseconds, `wChoice` exits with a special return value.

## 672 *wChoice*, *wChoiceFuncKeys*, *wChoiceSelect*, *wChoiceTimeout*

**Returns:**            **wChoice**    Several different types of values can be returned, depending upon the cause of the exit from the function.




| Return Value | Meaning                                                                                                                                    |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| > 0          | Okay. Value is index of item selected, base 1.                                                                                             |
| 0            | Okay, operator cancelled with  key.                     |
| -1           | Error, <i>window</i> is open but it does not have a frame.                                                                                 |
| -2           | Error, <i>window</i> is not open.                                                                                                          |
| < -2         | Okay but timeout occurred. Absolute value of return value minus 2 is the index of the item (base 1) pointed to at the time of the timeout. |







**wChoiceSelect** Index of item (base 0) pointed to at the time when last *wChoice* function call exited.

**Notes:**            **wChoice**    When the text of each of the *items* displays, it is automatically displayed with a leading space in addition to the actual contents of the text string.


Empty string-item text displays as a horizontal line. This type of item cannot be selected and is intended as a separator between related items.

While the choice window is active, the operator may use any of the following keys for choice positioning and selection. There may be other keys available, depending upon the values in the *hot\_keys* array. *wChoice* uses a “highlight bar” to identify the item that is currently positioned to.

| Key                                                                                 | Meaning                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Move the highlight bar to the previous item in the list. When the highlight bar is at the top of the choice window, the items in the display are scrolled down one line and the prior item is displayed at the top. When the highlight bar is at the top of the choice list, no action is taken. Blank items (shown as horizontal bars) are skipped.    |
|  | Moves the highlight bar to the next item in the list. When the highlight bar is at the bottom of the choice window, the items in the display are scrolled up one line and the next item is displayed at the bottom. When the highlight bar is at the bottom of the choice list, no action is taken. Blank items (shown as horizontal bars) are skipped. |
|  | Positions the highlight bar to the first item in the list. If this first item is not currently displayed in the window, the contents of the choice window are repainted with the first item at the top of the window.                                                                                                                                   |

| Key                                                                               | Meaning                                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Positions the highlight bar to the last item in the list. If this last item is not currently displayed in the window, the contents of the choice window are repainted with the last item at the bottom of the window. |
|  | The previous page or window-full of choices is displayed and the top item in that display is highlighted. If the current display is the first page of the choice list, the top item is highlighted.                   |
|  | The next page or window-full of choices is displayed. If the current display already includes the last item of the choice list, the bottom item is highlighted.                                                       |
|  | Selects the currently highlighted item. The index of this highlighted item is set to the return value for the function.                                                                                               |
|  | The function is exited with a return value of zero.                                                                                                                                                                   |
|  | This key is ignored unless the <i>help_func</i> argument is defined. When it is defined and this key is pressed, the specified function is invoked, giving it the index value of the currently highlighted item.      |



The mouse device can also be used to select items. Merely point to the desired item and left-click the mouse. If the desired item is not displayed, the scroll-bar on the right can be used by left-clicking on the up or down arrows displayed there. Clicking the mouse when the mouse pointer is positioned outside of the choice window acts as if the operator had pressed .

The help feature of this function can be disabled by using `NULL` as the argument for the help function name.

The prompt message display feature can be disabled by using `0` or `NULL` as the argument for *prompt\_win*. The *prompts* array can also be specified as `NULL` in this situation.

Upon exit from the function, *window* is not removed or closed. Also, *window* is the currently active window when `wChoice` exits. Your program should save the window number that was active prior to using this function so that it can be reselected.

**wChoiceFuncKeys** Since the normal return of `wChoice` is a positive or negative value representing the index of the choice selected by the operator, it is recommended that the translate-to values defined in the *keys* structure be defined as either large positive values or large negative values so they can be easily distinguished by the operator selecting one of the valid choices offered.

The standard definition of the `wCHOICE_FUNC_KEYS` structure provides for an array of ten definitions of keystroke values.

## 674 *wChoice*, *wChoiceFuncKeys*, *wChoiceSelect*, *wChoiceTimeout*

---

The function key selection capability can be disabled by using NULL as the argument to this function.

**Defaults:** During the normal operation of *wChoice*, all on-key trapping is disabled. To use on-keys you must define the keys with the *wChoiceFuncKeys* function prior to calling *wChoice*.

**Restrictions:** *wChoice* *window* must be open and it must have a frame defined.

If *prompt\_win* is not zero, it must be the number of an open window.

The arrays *items*, *prompts*, *grayed* and *hot\_keys* should all be allocated to at least *count* in size.

|                     |                        |        |       |         |         |
|---------------------|------------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>wChoice</b>         | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>wChoiceFuncKeys</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>wChoiceSelect</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>wChoiceTimeout</b>  | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [wClose](#), [wCloseAll](#), [wColor](#), [wFrame](#), [wMenuBar](#), [wOpen](#), [wRemove](#), [wTitle](#)

---

**Example:** This example displays a list of six items on the screen using a choice window. The *wChoiceTimeout* function is used to provide default selection if no keyboard activity occurs for a period of five seconds.

```
#include <stdio.h>
#include <wmapi.h>

void help_dummy(int);

void
main(void)
{
    int    ret_val,
           win_in_use,
           next_win,
           choice_win,
           prompt_win;

    char * items[7] = {
        "Deposit",
        "Check",
        "Service charge",
        "",
        "Adjustment",
        "Interest earned",
        "Transaction fee"
    };

    int items_hot[7] = { 1, 1, 1, 0, 1, 1, 13 };
    int items_grayed[7] = { 0, 0, 0, 0, 0, 0, 0 };

    char * prompts[7] = {
        "Deposit of cash, coin or checks to bank account",
        "A check drawn on the bank account",
        "Bank service charges",
        ""
    };
```

```
        "Adjustment due to bank or posting error",
        "Interest earned on deposits",
        "Item transaction fee"
    };

    wCount(&win_in_use, &next_win);
    prompt_win = next_win;
    wOpen(prompt_win, 0, 23, 78, 1);    // define prompt window
    wColor(prompt_win, wWHITE, wGREEN, wWHITE, wBLUE);

    wCount(&win_in_use, &next_win);
    choice_win = next_win;
    wOpen(choice_win, 20, 5, 22, 7);
    wClip(choice_win, wCLIP_WRAP);
    wColor(choice_win, wWHITE, wRED, wWHITE, wBLUE);
    wTitle(choice_win, " Transaction Types ", wTITLE_TOP,
            wTITLE_CENTERED, 0x4f);
    wFrame(choice_win, wFRAME_SINGLE, wSHADOW_RIGHT, 0x4f);

    wChoiceTimeout(5000);                // allow up to 5 seconds

    ret_val = wChoice( choice_win, 7, items, 0, items_hot,
                      NULL, prompts, items_grayed, NULL);
    wClose(choice_win);
    wClose(prompt_win);

    printf("Value returned      = %i\n",ret_val);

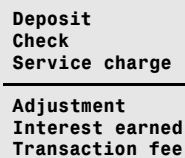
    ret_val = wChoiceSelect();
    printf("Item positioned to = %i\n",ret_val);

    wFinish();
}
```

---

**Output:**

>example



|                 |
|-----------------|
| Deposit         |
| Check           |
| Service charge  |
| Adjustment      |
| Interest earned |
| Transaction fee |

## wClear

Clears a window to repetitive characters (like spaces).

```
#include <wmapi.h>
```

```
int wClear ( int window, int c )
```

---

*c* » character to clear window to

*window* » window number

**Operation:** The contents of *window* are set to the character *c*, for its entire width and depth.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:** *window* does not have to be the active window. However, if *window* is not the active window, it is `wUPDATE_HIDDEN` or is `wUPDATE_OFF`, the effect of this `wClear` is not seen until *window* is refreshed with `wRefresh` or is selected with `wUPDATE_ON` status.

The character value *c* is not translated by the operating system or the console class code.

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:** `wClear` q ANSI q DOS n THEOS q POSIX

**See also:** [crt](#)

---

### Example:

```
...
name_win = next_win();           // get next window # avail

wOpen(name_win, 8, 4, 24, 9);    // define window
wColor(name_win, wBLACK, wWHITE, wWHITE, wWHITE); // its colors
wFrame(name_win, wFRAME_RAISED, wSHADOW_RIGHT, NULL); // its frame
wInvert(name_win, wINVERT_COMPLEMENT); // its invert
wClear(name_win, '*');           // clear it
...
```



## wClip

Sets the text-clipping mode of a window.

```
#include <wmapi.h>
int wClip ( int window, wCLIP_MODE clip )
```

---

*clip*                   »   text clip code  
*window*                »   window number

**Operation:**       The text-clipping status of *window* is set to *clip*.

**Returns:**         Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**         A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 25           | Clip error. The value of <i>clip</i> is invalid.                                                                |

**Notes:**         Values for wCLIP\_MODE are defined in WMAPI.H and include:

| Name           | Value | Meaning                                                                                                                                                                                                                          |
|----------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| wCLIP_WRAP     | 0     | Characters that would display beyond the right edge of the window wrap to the start of the next line. When wrapping from the last line of the window, the contents of the window are scrolled up to make room for one more line. |
| wCLIP_TRUNCATE | 1     | Characters that would display beyond the right edge of the window are truncated.                                                                                                                                                 |

Changing the text-clipping status of a window does not change the display of text already in the window.

Setting wCLIP\_WRAP doesn't allow you to position outside of a window's boundaries. It only takes effect when you are properly positioned within a window and then display more text than will fit on the line.

**Defaults:**       A newly opened window has a text-clipping status of wCLIP\_TRUNCATE.

**Restrictions:**   *window* must be an open window (it does not have to be selected).

Window 0 always has wCLIP\_WRAP and its text-clipping status cannot be changed.

**Conforms to:**       wClip    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [wOpen](#)

**wClose, wCloseAll**

Close a window or close all open windows.

```
#include <wmapi.h>
int wClose ( int window )
void wCloseAll ( void )
```

---

*window*                   »   window number

**Operation:**       **wClose**       *window* is closed, removed from the display and the memory used by the window is freed.

**wCloseAll**   All windows (except 0) are closed, removed from the display and the memory used by those windows is freed.

**Returns:**       **wClose**       Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**        A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return | Meaning                                                                                                           |
|--------|-------------------------------------------------------------------------------------------------------------------|
| 1      | Window not open.                                                                                                  |
| 6      | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager.   |
| 19     | Invalid operation on window 0. Window 0 cannot be resized, moved, have a frame or shadow and it cannot be closed. |

**Notes:**        Closing the active window causes the underlying window to become the active window. Refer to the discussion of the “Window Refresh Sequence” in the [wOrder](#) function.

Good programming practices dictate that a program close all windows that it has opened when it is finished using the windows.

**THEOS 32**        A program using the `wCloseAll` function on a system running THEOS 32  
**Version 4:**      Version 4 will close only the windows opened by the current task and those windows opened by child tasks. Windows opened by sibling tasks or parent tasks are not closed with the `wCloseAll` function. The [wFinish](#) function will close all windows.

**Defaults:**       Windows are closed by the [wFinish](#) function.

**Restrictions:**   *window* must be an open window (it does not have to be selected). Window 0 cannot be closed.

**Conforms to:**       **wClose**       q ANSI       q DOS       n THEOS       q POSIX  
                  **wCloseAll**   q ANSI       q DOS       n THEOS       q POSIX

**See also:**        [wFinish](#), [wOrder](#)

---

**Example:**        See [wOpen](#) example on page 721.

## wColor

This function specifies the text and background colors for a window.

```
#include <wmapi.h>

int wColor ( int window, wCOLORS fg_color, wCOLORS bg_color,
             wCOLORS rfg_color, wCOLORS rbg_color )
```

---

|                  |   |                                                       |
|------------------|---|-------------------------------------------------------|
| <i>bg_color</i>  | » | background color code                                 |
| <i>fg_color</i>  | » | foreground color code                                 |
| <i>rbg_color</i> | » | reverse video background color code                   |
|                  |   | <i>rfg_color</i> »reverse video foreground color code |
| <i>window</i>    | » | window number                                         |

**Operation:** The normal video text and background colors are set to *fg\_color* and *bg\_color*. The reverse video text and background colors are set to *rfg\_color* and *rbg\_color*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:** Values for wCOLORS are defined in WMAPI.H and include:

| Name     | Value |
|----------|-------|
| wBLACK   | 0     |
| wBLUE    | 1     |
| wGREEN   | 2     |
| wCYAN    | 3     |
| wRED     | 4     |
| wMAGENTA | 5     |
| wYELLOW  | 6     |
| wWHITE   | 7     |

*window* does not have to be the active window.

After the colors are set for *window*, a `wClear(window, ' ')` is performed to clear the interior of *window* to spaces of the requested background color.

## 680 *wColor*

---

**Defaults:** When a window is initially opened with the `wopen` function, the colors are defined by the console's session number, the window number and the colors defined for the various sessions on the console. (See `SETUP CRT` command in the *THEOS System Reference* manual.)

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:** **wColor** q ANSI q DOS n THEOS q POSIX

**See also:** [crtcolor](#), [wOpen](#)

---

### Example:

```
...
name_win = next_win();           // get next window # avail

wOpen(name_win, 8, 4, 24, 9);    // define window
wColor(name_win, wBLACK, wWHITE, wWHITE, wWHITE);    // its colors
wFrame(name_win, wFRAME_RAISED, wSHADOW_RIGHT, NULL); // its frame
wInvert(name_win, wINVERT_COMPLEMENT);    // its invert
...
```

## wCopy

Copies and pastes a rectangular region of text from one window to another.

```
#include <wmapi.h>
```

```
int wCopy ( int from_win, int from_col, int from_row, int width, int height,  
            int to_win, int to_col, int to_row )
```

---

|                 |   |                              |
|-----------------|---|------------------------------|
| <i>from_col</i> | » | starting column number       |
| <i>from_row</i> | » | starting row number          |
| <i>from_win</i> | » | window number of source      |
| <i>height</i>   | » | number of rows to copy       |
| <i>to_col</i>   | » | destination column number    |
| <i>to_row</i>   | » | destination row number       |
| <i>to_win</i>   | » | window number or destination |
| <i>width</i>    | » | number of columns to copy    |

**Operation:** The rectangular block of text in *from\_win* is copied to *to\_win*.

The upper-left corner of the block of text in *from\_win* is identified by *from\_col* and *from\_row*. The block extends to the right for *width* columns and down for *height* rows. This block of text is copied to *to\_win* and overlays the rectangular region with an upper-left corner of *to\_col* and *to\_row*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 16           | Copy error. The region specified is not wholly contained with the window.                                       |

**Notes:** *from\_win* and *to\_win* may be the same window number and the from and to areas may overlap.

The copied text includes its color and video attributes (blink, reverse video, etc.).

*from\_col*, *from\_row*, *to\_col* and *to\_row* refer to their respective window origins. That is, they are base 0 values relative to their window's upper-left corner.

The difference between **wCopy** and **wTake** is that **wCopy** copies the region without altering it and **wTake** moves the region, leaving the source area blank.

## 682 *wCopy*

---

**Restrictions:**    *from\_win* and *to\_win* must be open windows.

The block of text must be a rectangle that is within the boundaries of *from\_win* and it must fit in *to\_win* starting at the location specified.

**Conforms to:**                    **wCopy**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [wTake](#)

## wCount

Determines the maximum number of windows supported, currently in use, and the next window number available.

```
#include <wmapi.h>
```

```
int wCount ( int * in_use, int * next )
```

---

|               |   |                                                |
|---------------|---|------------------------------------------------|
| <i>in_use</i> | » | pointer to storage of number of windows opened |
|---------------|---|------------------------------------------------|

|             |   |                                                    |
|-------------|---|----------------------------------------------------|
| <i>next</i> | » | pointer to storage of next window number available |
|-------------|---|----------------------------------------------------|

**Operation:** The number of windows currently in use is determined and saved in the location pointed to by *in\_use*. The lowest, unused window number is determined and saved in the location pointed to by *next*.

**Returns:** The maximum number of windows supported by Session Manager.

**Notes:** When all available windows are in use, the value of *next* is set to zero.

**Conforms to:** **wCount** q ANSI q DOS n THEOS q POSIX

**See also:** [wVer](#)

---

**Example:** See also [wOpen](#) example on page 721.

```
#include <stdio.h>
```

```
#include <wmapi.h>
```

```
int
next_win(void)
{
    int next;
    int in_use;

    wCount(&in_use, &next);
    if (next==0) {
        fputs("\nNo windows available.", stderr);
        exit(1);
    }
    return next;
}
```

**wcstombs, wcsrten, wctomb**

These functions copy a wide-character string (*wcs*) to a multi-byte string (*mbs*) or a wide-character (*wc*) to a multi-byte character (*mb*).

```
#include <stdlib.h>
size_t wcstombs ( char * mbstring, const wchar_t * wcs, size_t count )
size_t wcsrten ( wchar_t * wcs )
int wctomb ( char * mbstring, wchar_t wc )
```

---

|                 |   |                                             |
|-----------------|---|---------------------------------------------|
| <i>count</i>    | » | number of characters in a multi-byte string |
| <i>mbstring</i> | » | pointer to multi-byte character             |
| <i>wc</i>       | » | pointer to wide character                   |
| <i>wcs</i>      | » | pointer to wide character array             |

**Operation:**      **wcstombs**      Converts the sequence of wide characters pointed to by *wcs* into multi-byte characters using the current LC\_CTYPE locale. The multi-byte characters are stored in consecutive locations in *mbstring*.

Conversion stops when a null wide character is encountered or when *count* bytes have been copied to *mbstring*.

**wcsrten**      Determine the number of wide characters in the array *wcs*.

**wctomb**      When *mbstring* is a NULL pointer, this function initializes the multi-byte functions to the current LC\_CTYPE locale category. Subsequent calls with a NULL pointer argument merely test to see if multi-byte translations are supported.

Convert the wide character pointed to by *wc* into a multi-byte character and store the converted character at *mbstring*.

**Returns:**            **wcstombs**      The number of bytes used in *mbstring*.

**wcsrten**      The number of wide characters in the *wcs* array.

**wctomb**      A true/false value when *mbstring* is a NULL pointer. A non-zero return indicates that multi-byte encodings are supported in the current LC\_CTYPE locale; a zero return indicates that they are not.

Returns the number of bytes used for the converted multi-byte character when *mbstring* is not a NULL pointer.

**Errors:**            These functions return a -1 if an invalid multi-byte character or an invalid wide character is encountered.

**Notes:**            The only LC\_CTYPE locale supported is the default C-type, which does not support multi-byte characters. These functions are provided for ANSI conformance.

**Conforms to:**            **wcstombs**      n ANSI      n DOS      n THEOS      n POSIX  
                              **wcsrten**      q ANSI      q DOS      n THEOS      q POSIX  
                              **wctomb**      n ANSI      n DOS      n THEOS      n POSIX

**See also:**            [mbten](#), [mbstowcs](#), [mbtowc](#)



## wEdit

Displays a text array in a window and allows the operator to edit that text.

```
#include <wmapi.h>
```

```
void wEdit ( int window, int count, char * lines[] )
```

---

*count* » number of items in *lines* to edit

*lines* » pointer to array of text strings to edit

*window* » window number

**Operation:** *window* is selected as the active window and the first *count* lines of text from *lines* is displayed in *window* and the operator may edit these strings in a manner similar to a general-purpose text editor.

The appearance of the window during the operation of wEdit depends upon the presence of a frame for *window* and whether or not that frame has a title defined. When *window* has a frame and *count* is greater than the window height, a scroll bar is displayed on the right side of the frame to indicate the relative position of the cursor in *lines*. When *window* has a frame but does not have a title defined, the edit status for wrap mode, input case mode and insert/replace mode is displayed in the lower-right corner of the frame.

**Notes:** While the edit window is active, the operator may use any of the following keys for positioning and editing.

| Key |   | Meaning                                                                                                                                        |
|-----|---|------------------------------------------------------------------------------------------------------------------------------------------------|
|     |   | Exit without saving.                                                                                                                           |
|     | + | Exit without saving.                                                                                                                           |
|     | + | Save and then exit.                                                                                                                            |
|     | + | Save without exiting.                                                                                                                          |
|     | + | Undo all changes since start or last save. This command reformats the <i>lines</i> according to the current wrap mode.                         |
|     | + | Move up one line, scrolling the display if necessary.                                                                                          |
|     | + | Move down one line, scrolling the display if necessary.                                                                                        |
|     | + | Move left one character. If at the start of current line, move to end of prior line. If on first character of first line, do nothing.          |
|     | + | Move right one character. If at right edge of window, move to start of next line. If at right edge of window and on the last line, do nothing. |
|     | + | Move to start of the current line.                                                                                                             |

| Key                |               | Meaning                                                                                                                                                                                                                             |
|--------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Home</b>        | <b>Ctrl+T</b> | Move to the start of the current line. <b>Home</b> , <b>Home</b> moves to the start of the top line in the window. <b>Home</b> , <b>Home</b> , <b>Home</b> moves to the start of the first line of <i>lines</i> .                   |
| <b>EndLine</b>     | <b>Ctrl+E</b> | Move to the end of current line.                                                                                                                                                                                                    |
| <b>End</b>         | <b>Ctrl+Y</b> | Move to the end of the current line. <b>End</b> , <b>End</b> moves to the end of the last line in the window. <b>End</b> , <b>End</b> , <b>End</b> moves to the end of the last line of <i>lines</i> .                              |
| <b>InsLine</b>     | <b>Ctrl+V</b> | Insert a blank line above the current line and move to the start of the inserted line.                                                                                                                                              |
| <b>DelLine</b>     | <b>Ctrl+^</b> | Delete line that cursor is on, pulling up remainder of array.                                                                                                                                                                       |
| <b>Enter ↵</b>     | <b>Ctrl+M</b> | Always inserts a new line and positions to the start of the new line. When the cursor is not at the end of a line of text, the text is split at the current location with the text to the right of the cursor forming the new line. |
| <b>← Backspace</b> |               | Deletes the character to the left of the cursor. If wrap mode is enabled, the paragraph is rewrapped.                                                                                                                               |
| <b>Delete</b>      | <b>Ctrl+Z</b> | Delete the character under the cursor. If wrap mode is enabled, the paragraph is rewrapped.                                                                                                                                         |
| <b>PageUp</b>      | <b>Ctrl+B</b> | Display the prior page of text array.                                                                                                                                                                                               |
| <b>PageDn</b>      | <b>Ctrl+P</b> | Display the next page of text array.                                                                                                                                                                                                |
| <b>Case</b>        | <b>Ctrl+C</b> | Toggle input case mode.                                                                                                                                                                                                             |
| <b>Tab ⇥</b>       | <b>Ctrl+I</b> | Insert spaces to advance cursor to next eight-column tab stop.                                                                                                                                                                      |
| <b>Erase</b>       | <b>Ctrl+N</b> | Erase to end of current line.                                                                                                                                                                                                       |
| <b>Insert</b>      | <b>Ctrl+R</b> | Toggle insert/replace mode. For most terminals the cursor shape changes to a blinking underline when in replace mode and a blinking block when in insert mode.                                                                      |
| <b>WordFwd</b>     | <b>Ctrl+X</b> | Advance to the start of the next “word.”                                                                                                                                                                                            |
| <b>WordBack</b>    | <b>Ctrl+U</b> | Back up to the start of the “word.” If at the start of a “word,” then back up to the start of the prior “word.”                                                                                                                     |
| <b>Find</b>        | <b>Ctrl+D</b> | Wrap the current paragraph.                                                                                                                                                                                                         |
| <b>SchFwd</b>      | <b>Ctrl+W</b> | Toggle wrap/no-wrap mode.                                                                                                                                                                                                           |
| <b>Transpose</b>   | <b>Ctrl+O</b> | Transpose or swap the two characters under the cursor.                                                                                                                                                                              |

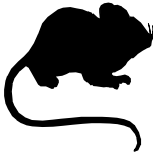
The text-clipping mode for *window* sets the initial status of the text wrap mode for the text in *window*. If `wCLIP_TRUNCATE` is set, the text is displayed one string per line, truncated to the *window* width. If `wCLIP_WAP` is set, the strings are displayed in “wrap” mode.

In **wrap mode**, as much of a string is displayed on a line as can fit. Words in the string that cannot fit on the line are moved to the next line, which is also wrapped if necessary. This initial display is slightly different from the “wrap paragraph” editing command that can be used by the operator.

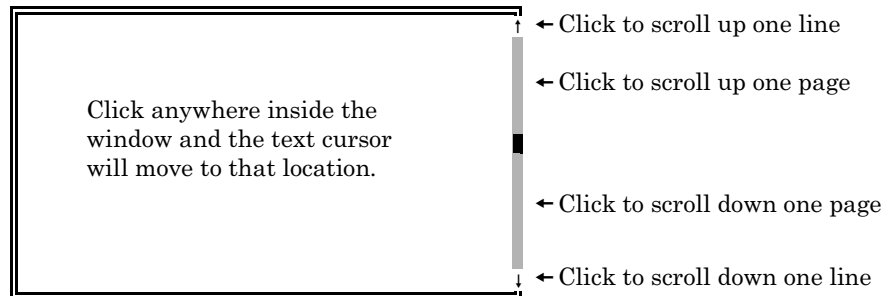
With the **wrap paragraph** command, the current line and following lines are wrapped, fitting as many words on a line as can fit in the *window* width. When there is space available at the end of the line, words from the line following are “pulled up” to fill out the line. This process continues until a paragraph break is encountered. A paragraph break is a blank text line.

With wrap mode disabled, inserting characters or tabs pushes characters to the right, possibly beyond the right edge of *window*. When this occurs, those characters are lost. With wrap mode enabled, the words pushed off the right side are wrapped to the next line.

wEdit always restores the active window upon exit. If *window* is selected prior to using wEdit, it will be the selected window after the function terminates.



A mouse can be used to position the cursor to a particular character location. It may also be used to scroll the display up or down.



**Defaults:** The initial status of the word-wrap feature is set according to the *window* clip status; the initial status of the insert/replace feature is set to INSERT; and the initial status of the input case mode is set to the current status of the [conmask](#) input case setting.

**Restrictions:** *window* must be open.

The array *lines* must be allocated to at least *count* size.

**Conforms to:** [wEdit](#)    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [wEnter](#), [wEnterFirst](#)

**wEnter, wEnterFirst**

Accept and edit a text string in a window.

```
#include <wmapi.h>
int wEnter ( int window, int row, int col, int width, char * string, int max_len,
            void (*help_func)(void) )
void wEnterFirst ( wENTER_FLAG flag )
```

---

|                  |   |                                                       |
|------------------|---|-------------------------------------------------------|
| <i>col</i>       | » | column number of the start of text entry              |
| <i>flag</i>      | » | indicator for initial text keep or replace            |
| <i>help_func</i> | » | pointer to function to handle help request            |
| <i>lines</i>     | » | pointer to array of text strings to edit              |
| <i>max_len</i>   | » | maximum length of text to accept                      |
| <i>row</i>       | » | row number of the text entry                          |
| <i>string</i>    | » | pointer to initial text string and storage for result |
| <i>width</i>     | » | display width for text entry                          |
| <i>window</i>    | » | window number                                         |

**Operation:**      **wEnter**      This function displays the initial value of *string* and allows the operator to edit or add text to *string*. The operation of this function depends upon whether *window* refers to an existing, open window or not.

When *window* is not the number of an open window, **wEnter** opens it with the following calls:

```
wOpen(window, row, col, width, 1);
wFrame(window, wFRAME_NONE, wSHADOW_NONE);
```



The color for this new window is the default colors for that window number on this console/session. Upon termination of the **wEdit** function, *window* is closed and the prior active window is selected.

When *window* is the number of an open window, **wEnter** selects it, positions to *col,row* inside *window* and performs its text editing operation. Upon termination of the **wEnter** function, *window* is not closed but the prior active window is selected.


**wEnter**'s text-editing operation is performed by first displaying the existing value of *string* at the designated location and then accepting new text or editing existing text in *string*.

Note that only *width* amount of *string* is displayed at any one time: If *max\_len* is greater than *width*, then only *width* amount of *string* is displayed at any one time. The value of *string* scrolls left/right as the operator positions to various parts of the text or enters more text than *width* characters.

















**wEnterFirst** Specifies the action taken by **wEnter** when the first character entered is a text character. See notes, below.

**Returns:** **wEnter** The character value of the key used by the operator to terminate the text entry. For instance, a return of 13 indicates  was used; a return of 27 indicates  was used; *etc.*

**Notes:** Values for WENTER\_FLAG are defined in WMAPI.H and include:

| <i>Name</i>   | <i>Value</i> | <i>Meaning</i>                                                                                                                                                                                                                                                                                                                                                 |
|---------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| wENTER_KEEP   | 0            | The existing value of <i>string</i> is kept, even when the first character entered is a text character. To replace the value of <i>string</i> , you must either delete the characters individually or use the  key to erase all of the characters to the end of the string. |
| wENTER_DELETE | 1            | If the first character entered is a text character, the existing value of <i>string</i> is deleted and replaced with that character.                                                                                                                                                                                                                           |

While the edit window is active, the operator may use any of the following keys for positioning and editing.

| <b>Key</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <b>Meaning</b>                                                                                                                                                                                       |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <br>                                                                                                                                                                                                                                                                                                                                                              | Exit without saving. A NULL string is always returned.                                                                                                                                               |
| <br><br><br><br><br> | Save and then exit.                                                                                                                                                                                  |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Move left one character. If the new column position is less than <i>col</i> , scroll the string right one character position. If on the first character of the string, do nothing.                   |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Move right one character. If the new column position is greater than <i>col+width</i> , scroll the string left one character position. If after the last character of the string, do nothing.        |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Move to the start of the string.                                                                                                                                                                     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Move to the end of the string.                                                                                                                                                                       |
|  Backspace                                                                                                                                                                                                                                                                                                                                                                                                                                          | Delete the character to the left of the cursor.                                                                                                                                                      |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Delete the character under the cursor.                                                                                                                                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Erase to the end of the string.                                                                                                                                                                      |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                     | If the <i>help_func</i> is defined (not NULL), calls that function and then returns to the wEdit function to continue accepting editing changes. If <i>help_func</i> is defined as NULL, do nothing. |

The mouse is not enabled during the operation of this function.

**690 *wEnter, wEnterFirst***

---

**Restrictions:**     *string* must be allocated to at least *max\_len* bytes.

|                     |                    |        |       |         |         |
|---------------------|--------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>wEnter</b>      | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>wEnterFirst</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**       [wEdit](#), [wOpen](#)

## **wFinish**

Terminates window operations for your session.

```
#include <wmapi.h>
void wFinish ( void )
```

**Operation:** Window 0 is selected, all other windows are closed.

**Notes:** This function closes all windows for your session, even windows not opened by this task or its children.

**Conforms to:** **wFinish** q ANSI q DOS n THEOS q POSIX

**See also:** [wClose](#), [wCloseAll](#)

**wFrame**

Define the frame and shadow style of an open window.

```
#include <wmapi.h>
int wFrame ( int window, wFRAME_TYPE frame, wSHADOW_TYPE shadow,
             wATTRIBUTE attr )
```

---

|               |   |                              |
|---------------|---|------------------------------|
| <i>attr</i>   | » | display attribute code       |
| <i>frame</i>  | » | frame style code             |
| <i>shadow</i> | » | window shadow direction code |
| <i>window</i> | » | window number                |

**Operation:** The frame style of *window* is set to *frame* and its shadow style is set to *shadow*. The color of the frame is set according to *attr*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                            |
|--------------|--------------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                   |
| 4            | Size error. The size of the window, including any frame and shadow, must fit within the virtual screen boundaries. |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager.    |
| 19           | Invalid operation on window 0. Window 0 cannot have a frame or shadow.                                             |
| 20           | Frame error. Invalid wFRAME_TYPE or wSHADOW_TYPE value.                                                            |

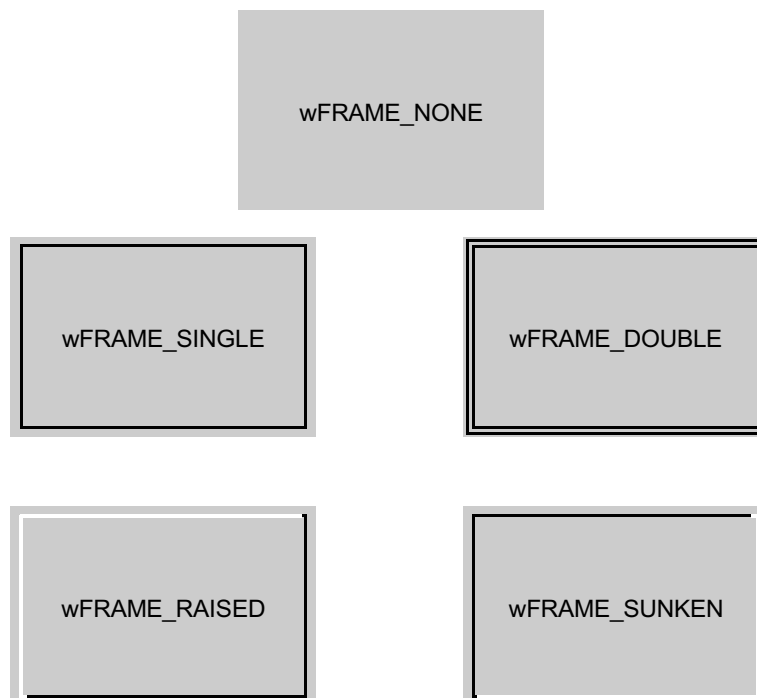
**Notes:** ■ **wFRAME\_TYPE**

Values for wFRAME\_TYPE are defined in WMAPI.H and include:

| Name          | Value | Meaning                                               |
|---------------|-------|-------------------------------------------------------|
| wFRAME_NONE   | 0     | No frame around window.                               |
| wFRAME_SINGLE | 1     | Single-line frame around window.                      |
| wFRAME_DOUBLE | 2     | Double-line frame around window.                      |
| wFRAME_RAISED | 3     | Single-line frame around window with “raised” effect. |
| wFRAME_SUNKEN | 4     | Single-line frame around window with “sunken” effect. |



These codes correspond to the following frame styles:

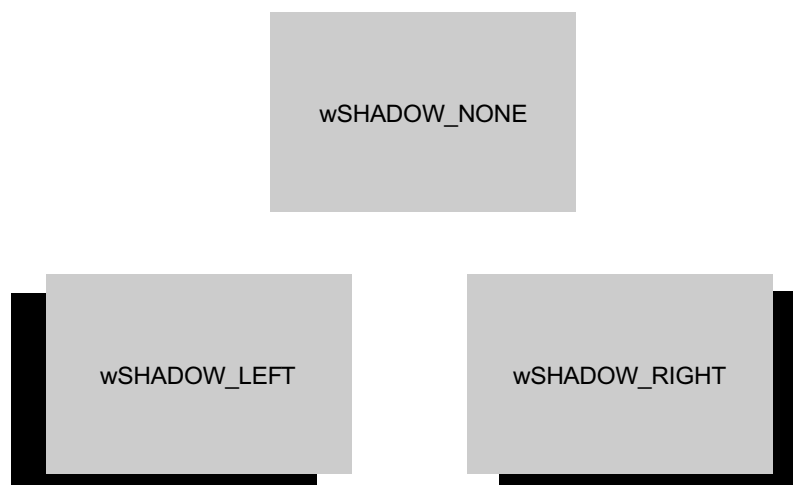


#### ■ wSHADOW\_TYPE

Values for wSHADOW\_TYPE are defined in WMAPI.H and include:

| <i>Name</i>   | <i>Value</i> | <i>Meaning</i>             |
|---------------|--------------|----------------------------|
| wSHADOW_NONE  | 0            | No drop-shadow on frame.   |
| wSHADOW_RIGHT | 1            | Drop-shadow on right side. |
| wSHADOW_LEFT  | 2            | Drop-shadow on left side.  |

These codes correspond to the following shadow styles:



It is recommended that wSHADOW\_LEFT not be used for consistency and portability reasons.

■ **wATTRIBUTE**

The *attr* field used for window-frame definitions (and window-title definitions) is a bit-mapped value. It describes the colors used for the foreground and background and the video display attributes to apply to the frame. The bit meanings are listed in `WMAPI.H`.

```
typedef struct wATTRIBUTE {
    unsigned reserved      : 6;
    unsigned underline     : 1;
    unsigned reverse       : 1;
    unsigned blink         : 1;
    unsigned bg_color      : 3;
    unsigned intensity     : 1;
    unsigned fg_color      : 3;
} wATTRIBUTE;
```

A NULL or zero value for *attr* means that the currently defined window colors are used for the frame.

**Defaults:** A newly opened window has `wFRAME_DOUBLE` and `wSHADOW_NONE` set. The color attributes of the frame match the default colors for the window.

**Restrictions:** *window* must be open.

The position of *window* must be such that adding a frame or a shadow would not cause the frame or shadow to be placed outside the bounds of the virtual screen.

Do not use the reserved bits of the `wATTRIBUTE` structure.

**Conforms to:**            **wFrame**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [wOpen](#), [wTitle](#)

---

**Example:**

```
...
name_win = next_win();           // get next window # avail

wOpen(name_win, 8, 4, 24, 9);    // define window
wColor(name_win, wBLACK, wWHITE, wWHITE, wWHITE); // its colors
wFrame(name_win, wFRAME_RAISED, wSHADOW_RIGHT, 0); // its frame
wInvert(name_win, wINVERT_COMPLEMENT); // its invert
...
```

---

## wGetActive

Get the console's currently active session number.

```
#include <wmapi.h>
int wGetActive ( void )
```

- Operation:** Retrieves the console display/keyboard session number that is currently active.
- Returns:** The active session number. This value is base zero.
- Notes:** Session-switching refers to the ability of one console display, keyboard and mouse to be shared between two or more separate user processes and programs. This function gets the session number that is currently displayed.
- The specific session returned by this function is needed if the program needs to force this program's session to be the active session (see [wSwitchTo](#) on page 741) and then later wants to switch back to the session that was active prior to the forced session-switch.
- Defaults:** When the console is not capable of session-switching (see [IsSession](#) on page 373), the return value from this function will always be zero.
- Conforms to:** **wGetActive**    q ANSI    q DOS    n THEOS    q POSIX
- See also:** [IsActive](#), [IsSession](#), [wGetSess](#), [wSwitch](#), [wSwitchTo](#)

---

**Example:** Refer to example for [wSwitch](#), [wSwitchTo](#).

**wGetSess**

Get this program's console session number.

```
#include <wmapi.h>
int wGetSess ( void )
```

**Operation:** The console display/keyboard session number used by this program is retrieved.

**Returns:** The session number used by this program. This value is base zero.

**Notes:** Session-switching refers to the ability of one console display and keyboard to be shared between two or more separate user processes and programs. This function gets the session number used by this program.

The specific session returned by this function is needed if the program needs to force this program's session to be the active session (see [wSwitchTo](#) on page 741).

**Defaults:** When the console is not capable of session-switching (see [IsSession](#) on page 373), the return value from this function will be zero, the same value returned when the program is using the first session of a session-capable console.

**Conforms to:** **wGetSess** q ANSI q DOS n THEOS q POSIX

**See also:** [IsActive](#), [IsSession](#), [wGetActive](#), [wSwitch](#), [wSwitchTo](#)

---

**Example:**

```
#include <stdio.h>
#include <wmapi.h>

main()
{
    printf("\nConsole is %session-capable\n",
        IsSession() ? "" : "not ");
    printf("Program is %scurrently the active session\n",
        IsActive() ? "" : "not ");
    printf("Current session number is %i\n",wGetSess());
}
```

---

**Output:**

>example

```
Console is session-capable
Program is currently the active session
Current session number is 3
```

>

## wGetStat

Tests whether a specific window number is open or not.

```
#include <wmapi.h>
int wGetStat ( int window )
```

---

*window*                   »   window number

**Operation:**       The open/closed status of *window* is determined and returned.

**Returns:**        A true/false value. A non-zero value indicates that *window* is currently open; a zero value indicates that it is not open.

**Conforms to:**        **wGetStat**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**        [wStatus](#)

## wGetStr

Get a string of characters displayed in a window.

```
#include <wmapi.h>
int wGetStr ( int window, char * string, int len )
```

---

|               |   |                                     |
|---------------|---|-------------------------------------|
| <i>len</i>    | » | maximum number of characters to get |
| <i>string</i> | » | pointer to storage for text string  |
| <i>window</i> | » | window number                       |

**Operation:** The text characters displayed in *window*, starting at its current cursor position, are copied to *string*. The maximum number of characters copied is the lesser of:

- ▶ The value of *len*
- ▶ The number of columns remaining in *window* from the cursor location to the right edge of the window.

*string* is always null-terminated.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 3            | Insufficient memory.                                                                                            |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 28           | String error.                                                                                                   |

**Notes:** Only text characters are transferred with this function. The color and video attributes of the text are ignored.

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:** **wGetStr** q ANSI q DOS n THEOS q POSIX

**See also:** [wCopy](#), [wGetTitle](#)

## wGetTitle

Copies the title text of a window to a character array.

```
#include <wmapi.h>
```

```
int wGetTitle ( int window, char * string )
```

---

*string*                   »   pointer to buffer to receive window title

*window*                   »   window number

**Operation:**       The title text of *window* is copied to *string*. If window does not have a title, *string* is set to the NULL string.

**Returns:**         Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**         A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                         |
| 3            | Insufficient memory.                                                                                                     |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager.          |
| 19           | Invalid operation on window 0. Window 0 cannot be resized, moved, have a frame, shadow or title and it cannot be closed. |
| 28           | String error.                                                                                                            |
| 29           | No frame. Titles are displayed in the frame area of a window.                                                            |

**Notes:**         Only text characters are transferred with this function. The color and video attributes of the text are ignored.

**Restrictions:**   *window* must be an open window (it does not have to be selected).

**Conforms to:**       **wGetTitle**    q ANSI      q DOS      n THEOS      q POSIX

**See also:**         [wFrame](#), [wGetStr](#), [wTitle](#)

### wGetWin

Get the currently selected window number.

```
#include <wmapi.h>
int wGetWin ( void )
```

**Operation:** The window number of the currently selected window is determined.

**Returns:** The active window number, base zero.

**Defaults:** If no other windows are in use, window number zero is returned.

**Conforms to:** **wGetWin** q ANSI q DOS n THEOS q POSIX

**See also:** [wSelect](#)



## WhoAml

Get “Who Am I” information about the current console.

```
#include <whoami.h>
void WhoAml ( WHOAMI * whoami )
```

---

*whoami*               »   pointer to WHOAMI structure

**Operation:**       Extracts the required information from the operating system and network, and fills in the WHOAMI structure pointed to by *whoami*.

**Notes:**           The WHOAMI structure is defined in the WHOAMI.h header file:

```
typedef struct WHOAMI {
    short  pid;                // process number (base zero)
    short  ucb_num;            // display UCB number (base one)
    char   dev_num;            // display device number (base one)
    char   sess_num;           // session number (base one)
    char   session_count;      // number of sessions
    char   device_name[16];    // name of display device
    char   net_client_type;     // network connection:
                                // 0=none
                                // 1=windows client
                                // 2=THEOS client
    long   net_client_address;  // remote network address
    char   net_client_name[32]; // remote network name
    short  net_client_ucb_num;  // remote network disp UCB
} WHOAMI;
```

**Conforms to:**       **WhoAml**   q ANSI    q DOS    n THEOS   q POSIX

**See also:**          [devnames functions](#), [getlub](#), [\\_getpid](#), [getpid](#), [getucb](#), [GetUCB](#)

---

**Example:**          See the example used in the [devnames functions](#) description.

**wInvert**

The normal video/reverse video status of a window is set for monochrome displays.

```
#include <wmapi.h>
int wInvert ( int window, WINVERT_TYPE invert_mode )
```

---

*invert\_mode*           »   monochrome invert code  
*window*                »   window number

**Operation:**       Sets *invert\_mode* for *window*.

**Returns:**         Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**         A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                        |
|--------------|----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                               |
| 6            | Window number error. Window numbers must be in the range of one to the maximum allowed by the Session Manager. |
| 26           | Invert error. Invalid WINVERT_TYPE value.                                                                      |

**Notes:**         Values for WINVERT\_TYPE are defined in WMAPI.H and include:

| Name               | Value | Meaning                                                                   |
|--------------------|-------|---------------------------------------------------------------------------|
| WINVERT_NORMAL     | 0     | Normal video is to have white or light-colored text on a dark background. |
| WINVERT_COMPLEMENT | 1     | Normal video is to have black or dark-colored text on a light background. |

The video invert mode is applicable on monochrome displays only. When the console is a color or grayscale display, the invert status is ignored.

When WINVERT\_COMPLEMENT is set, the normal video and reverse video attributes are swapped. The window is displayed with dark characters on a light background. Areas of the interior of a window that have not had any text displayed in it are displayed with a light background.

It is good programming practice to set the invert mode for all windows used by a program, even if it is to be used on a color display. The user might change their console to a monochrome display for some reason and, if the invert mode is set, the windows can still be differentiated.

**Defaults:**       When a new window is opened, its invert status is set according to its window number. Even-numbered windows have their status set to WINVERT\_NORMAL; odd-numbered windows have their status set to WINVERT\_COMPLEMENT; .

**Restrictions:**     *window* must be an open window (it does not have to be selected).

**Conforms to:**             **wInvert**     q ANSI     q DOS     n THEOS     q POSIX

**See also:**             [wOpen](#)

---

**Example:**

```
...
name_win = next_win();           // get next window # avail

wOpen(name_win, 8, 4, 24, 9);    // define window
wColor(name_win, wBLACK, wWHITE, wWHITE, wWHITE); // its colors
wFrame(name_win, wFRAME_RAISED, wSHADOW_RIGHT, NULL); // its frame
wInvert(name_win, wINVERT_COMPLEMENT); // its invert
...
```

## wLocate

Determine the cursor location for a designated window.

```
#include <wmapi.h>
```

```
int wLocate ( int window, int * col, int * row )
```

---

*col* » pointer to storage of cursor-column number

*row* » pointer to storage of cursor-row number

*window* » window number

**Operation:** The current location of the cursor in *window* is determined and returned in the locations pointed to by *col* and *row*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:** The values returned in *col* and *row* are relative to the window origin, not the screen origin.

The screen display is not affected by this function.

**Defaults:** The initial cursor location for a newly opened window is 0,0.

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:** **wLocate** q ANSI q DOS n THEOS q POSIX

**See also:** [wStatus](#)

## wMenuBar


Display a menu bar window and allow the operator to select one of the menu items.

```
#include <wmapi.h>
int wMenuBar ( int window, int count, char * text[], int begin, int * hot_keys,
               int prompt_win, char * prompt[], wLEAVE leave, int * leave_items,
               int * grayed, void (*help_func)(int line) )
```

---

|                    |   |                                                          |
|--------------------|---|----------------------------------------------------------|
| <i>begin</i>       | » | item number to start with                                |
| <i>count</i>       | » | total number of items to display                         |
| <i>grayed</i>      | » | pointer to list of items to display in reduced intensity |
| <i>help_func</i>   | » | name of help function to use when <b>F1</b> pressed      |
| <i>hot_keys</i>    | » | pointer to list of hot-key character positions           |
| <i>leave</i>       | » | default display action code on exit                      |
| <i>leave_items</i> | » | pointer to list of specific display action codes on exit |
| <i>prompt</i>      | » | pointer to array of pointers to prompt text strings      |
| <i>prompt_win</i>  | » | window number for prompt-text display                    |
| <i>text</i>        | » | pointer to array of item-text strings                    |
| <i>window</i>      | » | window number for choice display                         |

**Operation:** This is a rather complex function with many arguments. Some of the arguments control how the function behaves when it starts, some control how it behaves when it exits. Most of the arguments tell the function how the data is displayed during the operation of the menu bar selection.


A **menu bar** is a horizontal, single-line list of menu items that the operator can select by positioning to the item and pressing **Enter** .

The *window* argument specifies the window to use for displaying the list of text items pointed to by *text*. The value of *count* is the number of items in *text* that are used this time by wMenuBar.

The arguments *begin*, *hot\_keys* and *grayed* tell wMenuBar how the items are displayed initially. *begin* specifies the index of the *text* item that is positioned-to at the start of operation. *hot\_keys* specifies the character location for each of the items of *text* that should be highlighted with an underline attribute. These hot-key characters can be used by the operator to position to and select an item by merely typing the hot-key character.

*grayed* specifies the indices of the *text* items that are displayed in reduced intensity. Grayed items can be positioned to (for display of the prompt text or the help text) but cannot be selected.

*prompt\_win* and *prompts* define the window to be used for the display of prompt text and the text for each of the items in *text*. If *prompt\_win* is NULL, then no prompt text is displayed.

*leave* and *leave\_items* define the status of the window display when wMenuBar exits back to your program. *leave* specifies the default display status when the function is exited without selecting any of the items (entry of **Esc** ). *leave\_items* is a pointer to an array of leave specifications. These

values specify the display status when a specific item is selected and the function exits.

Finally, *helpfunc* is the name of the function that you want to handle operator help requests. This function is called when the operator presses **F1**. The function is given the index of the item positioned to when the help request was made. If *helpfunc* is NULL, this feature is disabled.

Once the menu is displayed in *window*, the operator can choose one of the items by positioning to it with the arrow keys or the space-bar and then selecting it with the **Enter** key. See the notes section about other keys that can be used.

**Returns:**

Several different types of values can be returned, depending upon the cause of the exit from the function.

| Return Value | Meaning                                                                                                                                    |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| > 0          | Okay. Value is index of item selected, base 1.                                                                                             |
| 0            | Okay, operator cancelled with <b>Esc</b> key.                                                                                              |
| -2           | Error, <i>window</i> is not open.                                                                                                          |
| < -2         | Okay but timeout occurred. Absolute value of return value minus 2 is the index of the item (base 1) pointed to at the time of the timeout. |

**Errors:**







As noted in return values.

**Notes:**

Values for *wLEAVE* are defined in *WMAPI.H* and include:

| Name                     | Value | Meaning                                                       |
|--------------------------|-------|---------------------------------------------------------------|
| <i>wLEAVE_HIGH</i>       | 0     | Upon exit, selected item is highlighted in reverse video.     |
| <i>wLEAVE_CLEAR</i>      | 1     | Upon exit, the menu bar window is cleared.                    |
| <i>wLEAVE_DIM</i>        | 2     | Upon exit, all items are dimmed.                              |
| <i>wLEAVE_NOHIGH</i>     | 3     | Upon exit, selected item is not highlighted in reverse video. |
| <i>wLEAVE_CLEAR_EXIT</i> | -1    | The menu bar window is cleared at once.                       |

While the menu window is active, the operator may use any of the following keys for item positioning and selection. There may be other keys available, depending upon the values in the *hot\_keys* array. *wMenuBar* uses a “highlight bar” to identify the item that is currently positioned to. This highlight bar is the item text displayed in reverse video.

| Key                                                                               | Meaning                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | Move the highlight bar to the previous item in the list. When the highlight bar is at the beginning of the item list, no action is taken.                                                                  |
|  | Moves the highlight bar to the next item in the list. When the highlight bar is at the end of the item list, no action is taken.                                                                           |
|  | Positions the highlight bar to the first item in the list.                                                                                                                                                 |
|  | Positions the highlight bar to the last item in the list.                                                                                                                                                  |
|  | Selects the currently highlighted item. One is added to the index of this highlighted item and is returned as the value for the function.                                                                  |
|  | This key is ignored unless the <i>help_func</i> argument is defined. When it is defined and this key is pressed, the specified function is invoked with the index value of the currently highlighted item. |

The mouse is not enabled during the operation of this function.

The help feature of this function can be disabled by using NULL as the argument for the help function name.

The prompt message display feature can be disabled by using 0 or NULL as the argument for *prompt\_win*. The *prompts* array must still be provided and it must be allocated with at least *count* elements.

Upon exit from the function, *window* is not removed or closed. Also, *window* is the currently active window when wMenuBar exits. Your program should save the window number that was active prior to using this function so that it can be reselected.

**Restrictions:** *window* must be an open window (it does not have to be selected).

There must be at least *count* number of items in the arrays *text*, *hot\_keys*, *prompt*, *leave\_items* and *grayed*.

**Conforms to:** wMenuBar q ANSI q DOS n THEOS q POSIX

**See also:** wChoice, wChoiceFuncKeys, wChoiceSelect, wChoiceTimeout

**Example:**

```
#include <stdio.h>
#include <wmapi.h>

void menubar_help(int);

void
main()
{
    int menu_win,
        prompt_win,
        prior_win,
        index;
    char * menu_items[6] = {      // wMenuBar items
        " File ",
        " Edit ",
        " Window ",
        " Options ",
        " Tools ",
        " Help "
    };
    int hot_keys[6] = { 2, 2, 2, 2, 2, 2 };
    char * prompts[6] = {
        " Load/Save file and other file operations ",
        " Text editing command menu ",
        " Window open/close command menu ",
        " Set various operating options ",
        " Invoke tools ",
        " Help system "
    };
    int leave_items[6] = {
        wLEAVE_HIGH,
        wLEAVE_DIM,
        wLEAVE_NOHIGH,
        wLEAVE_CLEAR,
        wLEAVE_HIGH,
        wLEAVE_DIM
    };
    int grayed_items[6] = { 0, 0, 1, 0, 1, 0 };

    prior_win = wGetWin();          // save current win number
    menu_win wOpen(OPEN_ANY, 0, 0, 80, 1);  // define menu window
    wColor(menu_win, wWHITE, wBLUE, wWHITE, wBLACK);
    wFrame(menu_win, wFRAME_NONE, wSHADOW_NONE, 0);

    prompt_win wOpen(OPEN_ANY, 0, 23, 80, 1); // prompt msg window
    wColor(prompt_win, wWHITE, wGREEN, wWHITE, wMAGENTA);
    wFrame(prompt_win, wFRAME_NONE, wSHADOW_NONE, 0);

    wSelect(menu_win, wUPDATE_TOP);
    index = wMenuBar(menu_win, 6, menu_items, index, hot_keys,
        prompt_win, prompts, wLEAVE_HIGH, leave_items,
        grayed_items, menubar_help);

    wSelect(menu_win, wUPDATE_TOP);      // make sure menu win displayed
    wSelect(prior_win, wUPDATE_ON);      // activate prior window
}
```



```

        pagewait();                // wait, test purpose only
        wClose(prompt_win);        // close the windows I opened
        wClose(menu_win);
        printf("\nreturn value = %i\n",index);
    }

void
menubar_help(int item)
{
    int prior_win,
        help_win,
        help_frame_win,
        tot_win;
    int width,
        height,
        start_col,
        start_row;

    prior_win = wGetWin();
    GetScreenSize(&width, &height);

    start_col = ((width-40)/2)-1;    // compute start loc for help
    start_row = ((height-5)/2)-1;    // so that it is centered

    help_frame_win = wOpen(OPEN_ANY, start_col-2, start_row-1,
        44+4, 5+4);
    wColor(help_frame_win, wWHITE, wGREEN, wBLACK, wGREEN);
    wFrame(help_frame_win, wFRAME_RAISED, wSHADOW_RIGHT, 0);
    wSelect(help_frame_win, wUPDATE_ON);

    help_win = wOpen(OPEN_ANY, start_col, start_row, 44, 7);
    wColor(help_win, wWHITE, wGREEN, wBLACK, wGREEN);
    wFrame(help_win, wFRAME_SUNKEN, wSHADOW_NONE, 0);
    wSelect(help_win, wUPDATE_ON);

    at(1,1);    // display help text
    printf("This is the help text for item # %i",item);

    pagewait();    // wait for operator to release

    wSelect(prior_win, wUPDATE_ON); // select prior window
    wClose(help_win); // close help window
    wClose(help_frame_win); // and frame
}

```

### wMouse functions

These functions enable or disable the mouse device or report on the last mouse event.

```
#include <wmapi.h>
void wMouseDisable ( void )
void wMouseEnable ( short * mouse_signal, int mask )
int wMouseHit ( int win, wMOUSE_BUTTON * button, int * col, int * row,
               wMOUSE_FRAME * frame )
void wMouseRead ( wMOUSE_BUTTON * button, int * col, int * row, long * time )
void wMouseState ( BYTE_CTL * ctl )
```

---

|                     |   |                                                      |
|---------------------|---|------------------------------------------------------|
| <i>button</i>       | » | pointer to button-pressed-type return value          |
| <i>col</i>          | » | pointer to mouse cursor column return value          |
| <i>ctl</i>          | » | pointer to a BYTE_CTL control structure              |
| <i>frame</i>        | » | pointer to mouse cursor window frame location return |
| <i>mask</i>         | » | button codes for mouse events to trap                |
| <i>mouse_signal</i> | » | true/false indicator for mouse event occurrence      |
| <i>row</i>          | » | pointer to mouse cursor row return value             |
| <i>time</i>         | » | time, in milliseconds, that mouse event occurred     |
| <i>win</i>          | » | window number of interest                            |

**Operation:**      **wMouseDisable** Tells the system to ignore all mouse activity. This is the default state for all programs.

**wMouseEnable** Enables the mouse. The *mask* field defines the specific mouse events that will be detected. This *mask* field can be set by “ORing” wMOUSE\_BUTTON values. For instance, if a program wants to allow only left and right button clicks, then it would use a *mask* value of:

wMOUSE\_LEFT\_CLICK | wMOUSE\_RIGHT\_CLICK

All other mouse events (all middle button activity, left and right button dragging and double clicks) are ignored.

To specify that all mouse events are to be detected, use a *mask* value of -1.

When an enabled mouse event occurs, the location *mouse\_signal* is set to a non-zero value.

**wMouseHit** This function has two basic modes of operation. When *win* is a zero or positive value, this function tests the last enabled mouse event to see if it occurred when the mouse was positioned in the open window number *win*. A return value of -1 indicates that the last mouse event did not occur in that window.

When *win* is -1 the return value is the window number of the window where the last mouse event occurred.

When the last mouse event did occur when the mouse was positioned in window number *win*, or when *win* is -1, the fields *button*, *col*, *row* and *frame* are set to reflect the event and location of that last mouse event.

**wMouseRead** Similar to the wMouseHit except that the position of the last mouse event is always returned. This position is relative to the physical console screen origin, not any specific window.

**wMouseState** The BYTE\_CTL structure pointed to by *ctl* is filled in with the current mouse status including enable/disable, button mask and signal variable location. Specifically:

ctl.c\_word1 contains enable/disable status  
 ctl.c\_word2 contains the current *mask*, as set by the last wMouseEnable call  
 ctl.c\_seg:ctl.c\_buf is the address of the *mouse\_signal* as set by the last wMouseEnable call

**Returns:** **wMouseHit** Returns the window number of the last mouse event. A return value of -1 indicates that the last mouse event was not in the value specified in *win*.

**Errors:** No errors are detected or reported by these functions.

**Notes:** Values for wMOUSE\_BUTTON are defined in WMAPI.H and include:

| <i>Name</i>         | <i>Value</i> | <i>Meaning</i>                                                        |
|---------------------|--------------|-----------------------------------------------------------------------|
| wMOUSE_RIGHT_DOWN   | 1            | Depress right mouse button.                                           |
| wMOUSE_MID_DOWN     | 2            | Depress middle mouse button.                                          |
| wMOUSE_LEFT_DOWN    | 4            | Depress left mouse button.                                            |
| wMOUSE_RIGHT_CLICK  | 8            | Depress and release right mouse button.                               |
| wMOUSE_MID_CLICK    | 16           | Depress and release middle mouse button.                              |
| wMOUSE_LEFT_CLICK   | 32           | Depress and release left mouse button.                                |
| wMOUSE_RIGHT_DCLICK | 64           | Depress and release right mouse button two times, quickly.            |
| wMOUSE_MID_DCLICK   | 128          | Depress and release middle mouse button two times, quickly.           |
| wMOUSE_LEFT_DCLICK  | 256          | Depress and release left mouse button two times, quickly.             |
| wMOUSE_RIGHT_DRAG   | 512          | Depress right mouse button and move the mouse while it is depressed.  |
| wMOUSE_MID_DRAG     | 1024         | Depress middle mouse button and move the mouse while it is depressed. |
| wMOUSE_LEFT_DRAG    | 2048         | Depress left mouse button and move the mouse while it is depressed.   |

## 712 *wMouse functions*

---

Values for `wMOUSE_FRAME` are defined in `WMAPI.H` and include:

| <i>Name</i>                            | <i>Value</i> | <i>Meaning</i>                                    |
|----------------------------------------|--------------|---------------------------------------------------|
| <code>wMOUSE_FRAME_TOP</code>          | 4            | Mouse position on top frame element.              |
| <code>wMOUSE_FRAME_BOTTOM</code>       | 5            | Mouse position on bottom frame element.           |
| <code>wMOUSE_FRAME_LEFT</code>         | 6            | Mouse position on left frame element.             |
| <code>wMOUSE_FRAME_RIGHT</code>        | 7            | Mouse position on right frame element.            |
| <code>wMOUSE_FRAME_TOP_LEFT</code>     | 10           | Mouse position at top-left frame corner.          |
| <code>wMOUSE_FRAME_TOP_RIGHT</code>    | 11           | Mouse position at top-right frame corner.         |
| <code>wMOUSE_FRAME_BOTTOM_LEFT</code>  | 12           | Mouse position at bottom-left frame corner.       |
| <code>wMOUSE_FRAME_BOTTOM_RIGHT</code> | 13           | Mouse position at bottom-right frame corner.      |
| <code>wMOUSE_SCROLL_BUTTON</code>      | 16           | Mouse position on scroll button.                  |
| <code>wMOUSE_SCROLL_UP</code>          | 18           | Mouse position on scroll-up button.               |
| <code>wMOUSE_SCROLL_DOWN</code>        | 19           | Mouse position on scroll-down button.             |
| <code>wMOUSE_SCROLL_ABOVE</code>       | 20           | Mouse position on scroll bar above scroll button. |
| <code>wMOUSE_SCROLL_BELOW</code>       | 21           | Mouse position on scroll bar below scroll button. |

**Defaults:** The initial state of every program is `wMouseDisable`. Also, programs using `wMouseEnable` will automatically invoke the `wMouseDisable` function through the `atexit` process.

**Restrictions:** No mouse activity is detected unless the `wMouseEnable` is set, and then only the events identified with that function's *mask*.

**Conforms to:**

|                      |        |       |         |         |
|----------------------|--------|-------|---------|---------|
| <b>wMouseDisable</b> | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wMouseEnable</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wMouseHit</b>     | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wMouseRead</b>    | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wMouseState</b>   | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [HasMouse](#)

**Example:**

This first example is a simple program that allows entry of name and address information. The mouse can be used to position to any of the fields or, using the right mouse button, a single-line help text can be displayed for any of the entry fields.

```
#include <stdio.h>
#include <stdlib.h>
#include <wmapi.h>
#include <crt.h>
#include <func_key.h>

#define MAX_I 5

short fields_x[MAX_I] = {20,20,20,45,55}; // input column
short fields_y[MAX_I] = {3,5,7,7,7}; // input row
short fields_len[MAX_I] = {40,40,15,2,10}; // input length
short labels_x[MAX_I] = {10,10,10,38,50}; // field label pos
char *labels[MAX_I] = { // field labels
    "Name",
    "Address",
    "City",
    "State",
    "Zip" };
char *fields[MAX_I]; // field contents
char *fields_help[MAX_I] = { // field help text
    "Enter name of person",
    "Enter street address",
    "Enter city name",
    "Enter state code",
    "Enter zip+4 code" };

void init_fields(void);
void display_screen(void);
void display_help(int);
extern char * linputu(char *, char *);
extern int linp;
extern char * rpad(char *, int);

void
main()
{
    short          mouse_signal;
    wMOUSE_BUTTON  mouse_mask = wMOUSE_LEFT_DOWN | wMOUSE_RIGHT_DOWN;
    wMOUSE_BUTTON  mouse_button;
    int            mouse_col,
                  mouse_row;
    long           mouse_time;

    int            i;

    init_fields(); // allocate field storage
    display_screen(); // display fields on screen

    wMouseEnable(&mouse_signal, mouse_mask);

    for (i=0; i<MAX_I; ++i) // repeat for each fields
    {
        at(fields_x[i],fields_y[i]);
        linputu(fields[i], rpad(fields[i], fields_len[i]));

        at(1, getpl(27)-1); // clear any help text
        crt(EOL);
    }
}
```

```
        if (linp==QUIT)
            break;

        if (mouse_signal)    // mouse event?
        {    // see if mouse points to one of the fields

            int    j;

            wMouseRead(&mouse_button, &mouse_col, &mouse_row,
                &mouse_time);
            for (j=0; j<MAX_I; ++j)
            {
                if (mouse_row == fields_y[j]    // matches row??
                    && mouse_col >= fields_x[j]    // matches col??
                    && mouse_col < fields_x[j]+fields_len[j])
                {
                    break;    // we have a match!!!!
                }
            }
            if (j < MAX_I)
            {
                if (mouse_button==wMOUSE_LEFT_DOWN)
                    i = j-1;    // point to that field
                else if (mouse_button==wMOUSE_RIGHT_DOWN)
                {
                    display_help(j);
                    --i;    // repeat current field
                }
            }
            else
            {
                --i;    // invalid mouse pos, ignore
                crt(BEL);
            }

            mouse_signal = 0;    // reset flag
        }
    }

    if (linp!=QUIT)
        pagewait();

    at(1, getpl(27)-2);

    wMouseDisable();
}

void
init_fields(void)    // allocate field storage
{
    int i;

    for (i=0; i<MAX_I; ++i)
    {
        fields[i] = malloc(fields_len[i]);    // allocate string
        fields[i][0] = 0;    // make null string
    }
}

void
display_screen(void)    // display field labels
{
    int i;
```

```

    crt(CLEAR);                // clear the screen

    for (i=0; i<MAX_I; ++i)
    {
        at(labels_x[i],fields_y[i]);
        printf(labels[i]);
        at(fields_x[i]-1,fields_y[i]);
        putchar('[');
        at(fields_x[i]+fields_len[i],fields_y[i]);
        putchar(']');
    }
}

void
display_help(int f)    // display help text for field f
{
    at(1, getpl(27)-1);    // at bottom of screen
    crt(EOL);
    printf(fields_help[f]);
}

```

This second example is a “progress bar” display function. It uses two windows, one for the progress bar display and another for a “cancel button” display. The `wMouseState` function is used to get and save the mouse status prior to using the function. This status is restored when the progress bar display is finished. The `wMouseHit` function is used to determine if the operator clicked on the cancel button.

```

// Progress bar display
//
// void progress(int type, int value, char *action)
//
// type = 1 initialize, value is total, action is title
// type = 2 update, value is current, action ignored
// type = 3 close, value ignored, action ignored
//
// A return value of -1 indicates that the cancel button was used

#include <stdio.h>
#include <stdlib.h>
#include <wmapi.h>
#include <string.h>
#include <conio.h>

int
progress(int type, int value, char *action)
{
    static int total;
    static int prog_win,
               cancel_button;
    static char bar[] = {
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,
        27,219,27,219,27,219,27,219,27,219,0 };
    static BYTE_CTL prior_state;
    int return_state = 0;

```

```
int m_row, m_col;
short mouse_signal;
wMOUSE_BUTTON m_button;
wMOUSE_FRAME m_frame;

switch (type)
{
    case 1:                                // initialize
    {
        int x,y;

        wMouseState(&prior_state); // Save current mouse state

        x = (getll(27)-59)/2;    // compute location of window
        y = getpl(27)-11;

        total = value;          // save total value
        value = 0;

        // define and create window for progress bar display
        prog_win = wOpen(-1, x, y, 57, 7);
        wColor(prog_win, wWHITE, wCYAN, wBLACK, wWHITE);
        wInvert(prog_win, wINVERT_COMPLEMENT);
        wFrame(prog_win, wFRAME_RAISED, wSHADOW_RIGHT, 0x3f);
        wTitle(prog_win, " Progress ", wTITLE_TOP,
                wTITLE_CENTERED, 0x3f);
        wSelect(prog_win, wUPDATE_ON);

        at(1,1);
        printf(action);

        // define and create "Cancel" button display
        cancel_button = wOpen(-1, x+24, y+5, 8, 1);
        wColor(cancel_button, wWHITE, wCYAN, wCYAN, wWHITE);
        wInvert(cancel_button, wINVERT_COMPLEMENT);
        wFrame(cancel_button, wFRAME_RAISED, wSHADOW_NONE, 0x3f);
        wSelect(cancel_button, wUPDATE_TOP);
        at(1,0);
        printf("Cancel");

        // enable mouse for selecting the "Cancel" button
        wMouseEnable(&mouse_signal, wMOUSE_LEFT_CLICK);

        // drop into update section with value = 0
    }
    case 2:                                // update
    {
        int        percent;
        int        done, remain;
        char        c;

        percent = value*100 / total;
        done = min(percent / 2, 50); // limit to 50 units
        remain = 50 - done;

        conmask("n");                // turn off input echo

        wSelect(prog_win, wUPDATE_ON);
        at(1,3);                    // position to progress bar location
        printf("%*.*s%c%*.*s%c %3.1d%%",
                done*2, done*2, bar,
                14, remain*2, remain*2, bar,
                15, percent);
        if (mouse_signal || (conrdy() && (c=getch())==13))
```



```
    {
        // mouse used or ENTER pressed

        if (c==13 || wMouseHit(cancel_button,
                                &m_button, &m_col, &m_row, &m_frame))
        { // mouse click on cancel button or ENTER pressed
            wFrame(cancel_button, WFRAME_SUNKEN,
                    WSHADOW_NONE, 0x3f);
            wRefresh(cancel_button);
            return_state = -1;
        }
        mouse_signal = 0;
        while(conrdy())
            getch();          // clear type-ahead
    }
    break;
}
case 3:          // close
{
    wClose(cancel_button);
    wClose(prog_win);

    conmask("e");          // enable input echo

    if (prior_state.c_word1) // mouse was enabled prior ?
    {
        wMouseEnable(x, prior_state.c_word2);
    }
    else
        wMouseDisable();

    break;
}
default:
    ;
}
return return_state;
}
```

## wMove

Change the displayed position of a window.

```
#include <wmapi.h>
int wMove ( int window, int to_col, int to_row )
```

---

|               |   |                          |
|---------------|---|--------------------------|
| <i>to_col</i> | » | column number to move to |
| <i>to_row</i> | » | row number to move to    |
| <i>window</i> | » | window number            |

**Operation:** The window origin (upper-left corner) of *window* is moved to *to\_col*, *to\_row*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                                      |
|--------------|------------------------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                             |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager.              |
| 9            | Position error. Similar to size error. The window, with its frame and shadow, must fit within the virtual screen boundaries. |
| 19           | Invalid operation on window 0. Window 0 cannot be resized, moved, have a frame or shadow, and it cannot be closed.           |

**Notes:** If *window* is the active window and has `wUPDATE_ON` set, the screen is refreshed to reflect the new location of window. If *window* is not active or does not have `wUPDATE_ON` set, the screen is not refreshed.

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:** **wMove**    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [wOpen](#), [wRefresh](#), [wSave](#), [wSaveSize](#), [wRestore](#)

## wOnKey

Enables or disables On-Key trapping.

```
#include <wmap.h>
int wOnKey ( ON_KEY_TAB * on_key_ptr )
```

---

*on\_key\_ptr*           »   pointer to On Key table struture

**Operation:**     The *on\_key\_ptr* may be either a pointer to an ON\_KEY\_TAB structure or a NULL pointer. When *on\_key\_ptr* is a pointer to a structure, the wOnKey function enables On-Key trapping for the keys defined in that structure. When *on\_key\_ptr* is a NULL pointer, wOnKey disables On-Key trapping.

**Returns:**       One of three possible return values indicating whether or not the On-Key trapping capabilities are present for your console:

| <i>Name</i>            | <i>Value</i> | <i>Meaning</i>                                    |
|------------------------|--------------|---------------------------------------------------|
| ON_KEY_IOCTL_INSTALLED | 19248        | On-Key trapping is available on the console       |
| ON_KEY_IOCTL_ERROR     | 19249        | The ON_KEY_TAB contents are invalid.              |
|                        | 0            | On-Key trapping is not available on this console. |

**Errors:**       No errors are detected except for the presence or absence of the On-Key trapping capabilities of your console.

**Notes:**        The names of the keys that may be trapped are defined in the header file ON\_KEY.H. This header file also defines the structure ON\_KEY\_TAB.

```
typedef struct ON_KEY_TAB {
    enum ON_KEY_TOKENS cur_tok;     // set by driver
    unsigned long tok_time;         // time token was entered
    short tok_count;                // number of tokens defined
    enum ON_KEY_TOKENS tok_tab[];   // trappable tokens
} ON_KEY_TAB;
```

When defining the tok\_tab array, be sure to fill in the tok\_count to reflect the number of tokens defined.

When the operator presses a key matching one of the defined and trappable tokens, the console driver will set the cur\_tok item to that key's value and it will place a CR (carriage return, value 13) in the input buffer. This allows any console input routine to be terminated when the trapped On-Key is pressed.

**Defaults:**     When a program is initially started, On-Key trapping is disabled.

**Restrictions:** Unlike a BASIC program using the ON KEY statement, when a key is pressed that is currently being trapped with the wOnKey function, no routine in your program is automatically invoked. It is the responsibility of your program to periodically check to see if a trappable key was pressed (test the cur\_tok item) and to perform whatever type of action desired.

After your program detects and processes the event, it is responsible for clearing the `cur_tok` item in the `ON_KEY_TAB` structure.

There is no method available for detecting whether or not On-Key trapping is enabled or to determine the current `ON_KEY_TAB` structure being used for trapping. That is, if one function enables On-Key trapping and then invokes another function that sets its own On-Key trapping, the original key definitions are not knowable nor can they be re-enabled without telling `wOnKey` where the structure is that defines the keys.

**Conforms to:** `wOnKey` q ANSI q DOS n THEOS q POSIX

**See also:** [wChoice](#), [wChoiceFuncKeys](#), [wChoiceSelect](#), [wChoiceTimeout](#)

---

**Example:**

```
...
ON_KEY_TAB *on_keys;
...
// enable on-key traps for F1, F9, Tab and Shift+Tab

// first, allocate memory for 4 trap definitions
on_keys = malloc(sizeof(ON_KEY_TAB)+4*sizeof(short));
on_keys->cur_tok = 0;      // init current on-key token
on_keys->tok_count = 4;    // four key trap definitions
on_keys->tok_tab[0] = ON_KEY_TOK_F1;
on_keys->tok_tab[1] = ON_KEY_TOK_TAB;
on_keys->tok_tab[2] = ON_KEY_TOK_TAB+ON_KEY_SHIFT_MODIFIER;
on_keys->tok_tab[3] = ON_KEY_TOK_F9;

// enable trapping for these keys
wOnKey(on_keys);
...
c = getchar();           // accept input from operator
c = chk_on_keys(c, on_keys); // was an ON KEY pressed?
...

wOnKey(NULL);          // disable on key trapping
free(on_keys);           // release memory
...

int
chk_on_keys(int c, ON_KEY_TAB *on_key_ptr)
{
    int    work;

    if (work=on_key_ptr->cur_tok) { // on-key pressed?
        on_key_ptr->cur_tok = 0;    // clear status
        c = work;                  // set as return value
    }
    return c;                      // return orig char or on-key
}
```

## wOpen

Opens a new window.

```
#include <wmapi.h>
```

```
int wOpen ( int window, int col, int row, int width, int height )
```

---

|               |   |                                                        |
|---------------|---|--------------------------------------------------------|
| <i>col</i>    | » | column number of top left corner of interior of window |
| <i>height</i> | » | number of lines inside window                          |
| <i>row</i>    | » | row number of top left corner of interior of window    |
| <i>width</i>  | » | number of columns inside window                        |
| <i>window</i> | » | window number to open                                  |

**Operation:** A new window is opened. This window has an upper-left corner at *col*, *row*. It is *width* columns wide and *height* rows deep.

*window* may either be a specific window number, in which case that is the number of the window opened, or it may be the special value `wOPEN_ANY`. When *window* equals `wOPEN_ANY`, the next available window number is used for this new window.

**Returns:** When the value of *window* is not equal to `wOPEN_ANY`, then a success or failure code is returned. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

When the value of *window* equals `wOPEN_ANY`, then the return value will be the actual window number used. In this situation, a return value of zero indicates that there was a problem opening the window. Most likely, the problem is that there are no more window numbers available. See the [wCount](#) function.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------|
| 3            | Insufficient memory.                                                                                                             |
| 4            | Size error. The size of the window, including any frame and shadow, must fit within the virtual screen boundaries.               |
| 6            | Window number error. Window numbers must be in the range of one to the maximum allowed by the version of Session Manager in use. |
| 19           | Invalid operation on window 0. Window 0 cannot be resized, moved, have a frame or shadow, and it cannot be closed.               |

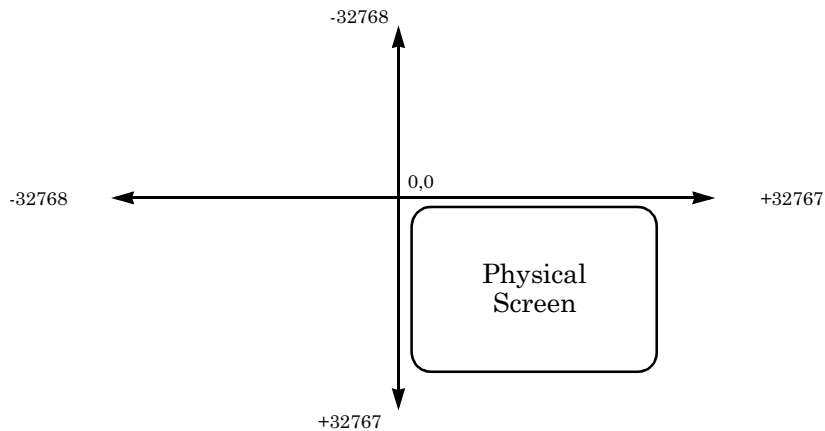
**Notes:** The constant `wOPEN_ANY` is defined in the `WMAPI.H` header file. It has a value of -1.

This function only creates the window. It does not cause it to be displayed on the console. Use the other windowing functions to further define the attributes of the window (frame style, title, etc.) and then use the [wSelect](#) function to activate and display the window.

The size of a window may be as small as a single character or as large as 255 characters by 255 lines.

**Virtual Screen:** The windows defined and used with THEOS exist in a virtual screen. The *virtual screen* is an imaginary screen that contains the physical screen seen on your console.

The dimensions of this virtual screen are -32,768 to +32,767 columns and rows.

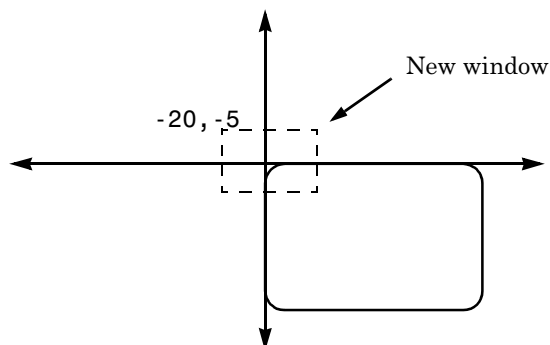


A window may be defined anywhere on this virtual screen. If a portion of the window lies within the boundaries of the physical screen, then that portion can be displayed on your console. The size of the physical screen is defined by the attached page width and page depth. Frequently, this is 80 columns by 24 rows. The upper-left corner of the physical screen is column zero, row zero.

For example, a window created with the function:

```
wOpen(1, -20, -5, 40, 10);
```

has a location in the virtual screen of:



Only the portion of the window from its column 20 through 39 and its line 5 through 9 are visible. If the window has a frame or shadow, only the portions of the frame and shadow that lie within the boundaries of the physical screen are displayed.

Although this window is positioned partially outside of the physical screen, it may be selected just as any window can be selected. Text may be displayed in the window and the operator may type characters into the window, even though the characters may not be visible due to the window's position on the virtual screen.

**Defaults:** When a window is opened with this function, it has the following attributes set by default:

```
Cursor at: 0, 0
wFRAME_DOUBLE
wSHADOW_NONE
wTITLE_NONE
wCLIP_TRUNCATE
wINVERT_NORMAL
wUPDATE_OFF
```

The initial, default colors used for a new window are the colors defined for one of the sessions of this console. Specifically, the default colors for the window will be the colors associated with this console's session number  $\text{mod}(\text{window}, 8)+1$ . For instance, window number 2 will have the colors defined for session number 3, window number 12 will have the colors defined for session number 5, *etc.*

**Restrictions:** When the value of *window* is not equal to `wOPEN_ANY`, then *window* must be an unused window number. *window* may not be zero because window zero is always open.

The values for *col* and *row* must be valid integer values (-32,768 through +32,767). That is, the window origin must be in the virtual console display space.

The values of *col+width* and *row+height* must be less than +32,767. That is, the window must be wholly contained in the virtual console display space.

In addition, since the default for the frame style is `wFRAME_DOUBLE`, and a frame requires one column on the left, one column on the right, one row above and one row below the window, the values for *col*, *row*, *height* and *depth* must allow for this frame in the virtual console display space.

**Conforms to:**                    `wOpen`    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [wClip](#), [wClose](#), [wCloseAll](#), [wColor](#), [wFinish](#), [wFrame](#), [wInvert](#), [wSelect](#), [wTitle](#)

Additional information about windows can be found in the *THEOS System Reference Manual* and in the *THEOS MultiUser BASIC Programmer's Guide*. (MultiUser BASIC is implemented with the Session Manager C functions described in this *THEOS C Standard Function Library* manual.)

### Example:

```
...
name_win = wOpen(OPEN_ANY, 8, 4, 24, 9);           // define window
wColor(name_win, wBLACK, wWHITE, wWHITE, wWHITE); // its colors
wFrame(name_win, wFRAME_RAISED, wSHADOW_RIGHT, 0); // its frame
wInvert(name_win, wINVERT_COMPLEMENT); // its invert

cal_win = wOpen(OPEN_ANY, 10, 6, 20, 6);           // define window
wColor(cal_win, wWHITE, wWHITE, wBLACK, wWHITE);  // its colors
wFrame(cal_win, wFRAME_SUNKEN, wSHADOW_NONE, 0);   // its frame
wInvert(cal_win, wINVERT_COMPLEMENT); // its invert
...
```



## wOrder

Gets or sets the display order for open windows.

```
#include <wmapi.h>
int wOrder ( char list[], WORDER_DIRECTION in_out )
```

---

*in\_out*                   »   get or put window order  
*list*                     »   array of window numbers, in sequence

**Operation:** Depending upon the value of *in\_out*, the display order for all open windows is either set or retrieved.

When *in\_out* has a value of `WORDER_GET` (value 0), the current display order for the windows is copied to *list*.

When *in\_out* has a value of `WORDER_PUT` (value 1), the display order for windows is set according to the sequence of window numbers defined in *list*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open. One or more of the window numbers in <i>list</i> is the number of an unopened window.          |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 12           | Select error. Invalid <code>WORDER_DIRECTION</code> value.                                                      |

**Notes:** Values for `WORDER_DIRECTION` are defined in `WMAPI.H` and include:

| Name                    | Value | Meaning                                                    |
|-------------------------|-------|------------------------------------------------------------|
| <code>WORDER_GET</code> | 0     | Copy the current window refresh sequence to <i>list</i> .  |
| <code>WORDER_PUT</code> | 1     | Use <i>list</i> to define the new window refresh sequence. |

The *list* array contains the window numbers in the sequence that the windows are displayed. A value of 255 (0xff) in this *list* array is used as a filler for unopened or never selected window numbers.

The display of windows with `wUPDATE_MODE` of `wUPDATE_TOP` or `wUPDATE_HIDDEN` are affected by this function. That is, a window that has a `wUPDATE_MODE` of `wUPDATE_HIDDEN` is displayed if its window number is in the *list* array.

Similarly, a window with `wUPDATE_TOP` is no longer displayed on top of other windows if its window number is in the *list* array at a position other than the end (before the first 255 code). However, a window's `wUPDATE_TOP` status is remembered and, the next time that any window is selected, all `wUPDATE_TOP` windows will revert to their top-display status.

The `wOrder` function is most often used to restore the refresh sequence of windows after some unknown resequencing. For instance:

```
wOrder(list, wORDER_GET); // get current sequence
// some windows are selected and used
...
wOrder(list, wORDER_PUT); // restores prior sequence
```

**Window  
Refresh  
Sequence**

The window refresh sequence refers to the display sequence for windows. When window positions and dimensions overlap, the refresh sequence determines which windows display on top of other windows. The refresh sequence is defined by this `wOrder` function or, more commonly, by the [wSelect](#) function.

When a window is selected with [wSelect](#) and a `wUPDATE_MODE` of `wUPDATE_ON`, it is placed at the top (end) of the refresh sequence but underneath any `wUPDATE_TOP` windows. When a window is selected with [wSelect](#) and a `wUPDATE_MODE` of `wUPDATE_HIDDEN`, it is removed from the refresh sequence. A window selected with [wSelect](#) and a `wUPDATE_MODE` of `wUPDATE_TOP` is placed at the top (end) of the refresh sequence after all other windows.

A window that has been opened but has never been selected with [wSelect](#) is not in the refresh sequence unless the `wOrder` function is used to define its display order.

**Restrictions:**

The *list* array should be allocated with at least as many positions as there are possible windows.

You should reselect the active window after a `wOrder` function call. Its `wUPDATE_ON` / `wUPDATE_OFF` status might be changed by this function.

**Conforms to:**

**wOrder**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**

[wClose](#), [wCloseAll](#), [wOpen](#), [wRefresh](#), [wRemove](#), [wSelect](#)

**Example:**

```

#include <stdio.h>
#include <stdlib.h>
#include <wmapi.h>

void
main()
{
    int win1 = 1,
        win2 = 2,
        win3 = 3,
        i;
    char order[32];

    if (wOpen(win1, 5, 5, 30, 10)           // open 3 windows
        || wOpen(win2, 28, 14, 30, 10)
        || wOpen(win3, 11, 4, 30, 10)) {
        wSelect(0, wUPDATE_ON);
        printf("Cannot open all required windows.\n");
        exit(0);
    }
    wSelect(win1, wUPDATE_HIDDEN);           // 1st window hidden
    at(1, 1);
    printf("This is window number 1");

    wSelect(win2, wUPDATE_TOP);              // 2nd window on top
    wClip(win2, wCLIP_WRAP);
    at(1, 1);

    printf("This is window number 2");

    wSelect(win3, wUPDATE_ON);              // 3rd window normal
    at(1, 9);
    printf("This is window number 3");

    wSelect(win2, wUPDATE_TOP);              // get and display order
    wOrder(order, wORDER_GET);
    at(1, 3);
    for (i=0; i<6; ++i) {
        printf("%i ",order[i]);
    }
    pagewait();

    order[1] = 3;                           // change refresh sequence
    order[2] = 2;
    order[3] = 1;

    wOrder(order, wORDER_PUT);
    wOrder(order, wORDER_GET);              // get and display order
    at(1, 4);
    for (i=0; i<6; ++i) {
        printf("%i ",order[i]);
    }
    wRepaint();                             // repaint due to wORDER_PUT
    pagewait();

    wSelect(0, 1);
    wCloseAll();
}

```

## wRefresh

Display a window and its contents without making it active.

```
#include <wmapi.h>
int wRefresh ( int window )
```

---

*window*                   »   window number to refresh

**Operation:**       The window and its contents are displayed on the screen.

**Returns:**         Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**         A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:**         The window is not made the active window.

*window* is placed at the top (end) of the window refresh sequence (see page 726) but before all windows with `wUPDATE_MODE` of `wUPDATE_TOP` and before the active window if the active window has `wUPDATE_ON`.

The `wUPDATE_MODE` for *window* is not changed.

**Defaults:**       The display of all other displayed windows that are at least partially visible are refreshed.

**Restrictions:**   *window* must be an open window (it does not have to be selected).

A window with `wUPDATE_HIDDEN` mode cannot be refreshed.

**Conforms to:**       **wRefresh**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**         [wOrder](#), [wRepaint](#), [wSelect](#)

## wRemove

Erase the display of a window from the screen.

```
#include <wmapi.h>
int wRemove ( int window )
```

---

*window*                    »    window number

**Operation:**        The display of *window* is removed from the screen. All regions of the screen underlying this window are updated to reflect this removed window.

**Returns:**            Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**            A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:**            The *window* is not closed, it is merely removed from the display and the window refresh sequence (see page 726). The contents of *window* are not affected.

A removed window is not redisplayed on the screen unless it is selected with `wUPDATE_ON` or `wUPDATE_TOP`, or it is refreshed with [wRefresh](#).

Removing the active window causes the window number immediately beneath this window in the window refresh sequence to be selected as the active window.

**Restrictions:**     *window* must be an open window (it does not have to be selected).

**Conforms to:**        **wRemove**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [wOrder](#), [wRefresh](#), [wSelect](#)

## wRepaint

Redisplay all windows using the current window refresh sequence.

```
#include <wmapi.h>
int wRepaint ( void )
```

**Operation:** The console display is updated using the windows defined in the window refresh sequence (see page 726).

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:** The wRepaint operation can be manually invoked by using **Break**, **F9**.

wRepaint is normally only used after Session Manager bypass mode is used. For instance, a program's display is interrupted by a subtask that uses bypass mode to display some critical information on the screen. When bypass mode is cleared, the wRepaint function can be used to restore the screen display to the image that was displayed prior to using bypass mode.

**Conforms to:**            **wRepaint**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**            [SetBypass](#), [ClrBypass](#), [wOrder](#), [wRefresh](#), [wSelect](#)

## write

Write a stream of bytes to a file.

```
#include <io.h> or <handle.h>
size_t write ( int fhandle, const void _far * buffer, size_t len )
```

```
#include <sc.h>
size_t _write ( FILE * file, const void _far * buffer, size_t len )
```

---

|                |   |                          |
|----------------|---|--------------------------|
| <i>buffer</i>  | » | pointer to storage area  |
| <i>fhandle</i> | » | file handle or number    |
| <i>file</i>    | » | pointer to file's fcb    |
| <i>len</i>     | » | number of bytes to write |

**Operation:** *len* number of bytes from *buffer* are written to the file associated with *fhandle* or *file*, starting at the file's current position pointer. The position pointer is adjusted to reflect the number of bytes written.

**Returns:** Both of these functions return the number of bytes actually written. Normally, this will be the same as *len*.

A return of a value less than *len* indicates an error was encountered during the write operation.

A return of -1 indicates that the write was unsuccessful.

**Notes:** The principle difference between *write* and *\_write* is that *write* uses a file handle argument and *\_write* uses a pointer to a file control block.

The principle difference between these two functions and the [fwrite](#) function is that [fwrite](#) will write to a file's output buffer if available and the *write* and *\_write* write directly to the file. For this reason, do not mix [fwrite](#) and *write* or *\_write* calls to the same file.

|                     |               |        |       |         |         |
|---------------------|---------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>write</b>  | q ANSI | n DOS | n THEOS | n POSIX |
|                     | <b>_write</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [fwrite](#)

**writek**

Writes a record to a keyed or indexed access file.

```
#include <stdio.h>
unsigned short writek ( FILE * file, void _far * key, void * record )
```

---

|               |   |                                  |
|---------------|---|----------------------------------|
| <i>file</i>   | » | pointer to open file's fcb       |
| <i>key</i>    | » | pointer to character string      |
| <i>record</i> | » | pointer to record storage buffer |

**Operation:** The contents of *record* and *key* are written to *file*.

**Returns:** The *file*'s allocated record length.

A zero return value indicates that the record could not be written.

**Errors:** When the return value is zero, the record was not written for some reason. Possible reasons include: *file* not opened as keyed or indexed-access file with write access.

**Notes:** The *key* and *record* fields of a keyed or indexed file are not C strings. That is, it is not necessarily a null-terminated sequence of characters. Instead, it is a fixed-length sequence of characters that may have null characters embedded within. If the record and key lengths are unknown, use the [fcntl](#) function to get these lengths.

The allocated key and record lengths for *file* are used when writing the data pointed to by *key* and *record*. That is, if the allocated record length is 64 then 64 bytes of *record* will be written even if *record* was declared to be larger or smaller than 64.

If *file* has been opened with access "r+" but without specifying the access modifier "m," then writing a record releases any other record locks set by this user for this file.

If *file* is in use by another user and that user has the record locked, the *writek* function waits for that lock to be released before writing the record.

**Restrictions:** This function may only be used with a file opened with keyed or indexed organization and write or update access.

**Conforms to:** **writek** q ANSI q DOS n THEOS q POSIX

**See also:** [deletk](#), [readk](#), [readn](#), [readp](#)



**wSave, wSaveSize, wRestore**

Save or restore a window and its contents to or from a buffer.

```
#include <wmapi.h>
unsigned wRestore ( int window, char * save_data, unsigned offset,
    unsigned length )
unsigned wSave ( int window, char * save_data, unsigned offset,
    unsigned length )
unsigned wSaveSize ( int window )
```

---

|                  |   |                                      |
|------------------|---|--------------------------------------|
| <i>length</i>    | » | number of bytes in buffer to use     |
| <i>offset</i>    | » | offset within <i>save_data</i>       |
| <i>save_data</i> | » | pointer to buffer for window storage |
| <i>window</i>    | » | window number                        |

**Operation:**

**wRestore** Copies the information from memory location specified by *save\_data* to the contents of *window* starting at *offset*.

**wSave** Copies the contents of *window* starting at *offset* to the memory location specified by *save\_data*.

**wSaveSize** Computes the number of bytes needed to save the contents of *window*.

The size computed is three times the width and height of *window* because each display position requires two bytes of data for the video attributes and color.

**Returns:**

**wRestore** The number of bytes successfully restored.

**wSave** The number of bytes successfully saved.

**wSaveSize** The number of bytes needed to save *window*.

**Errors:** When a zero is returned by any of these functions, then the operation failed. Normally, this will be due to *window* not being the number of an open window.

**Notes:** The *offset* field is normally a zero. A non-zero *offset* would be used only when the contents of a window are saved or restored in sections, as would be the case if the memory buffer were smaller than the window content size.

**Restrictions:** *window* must be an open window (it does not have to be selected).

**Conforms to:**

|                  |        |       |         |         |
|------------------|--------|-------|---------|---------|
| <b>wRestore</b>  | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wSave</b>     | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wSaveSize</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [wStatus](#), [wTake](#)

**Example:** This example creates a window, saves it to disk, and then retrieves that window and displays it at a different location. These two operations are not normally performed in the same program. In most situations, a program restores a window that was saved by another program.

When the window is restored only some of the attributes that are known to apply to the window are redefined. The other attributes (clip, invert, *etc.*) could be redefined by extracting the information from the status array. The title text is not saved or restored because this window does not use a title. If a title were defined, the [wGetTitle](#) and [wTitle](#) would be used to get the title and to restore it.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <wmapi.h>

void
main()
{
    int      win = 1,
            win1 = 2;
    char *   saveit;
    char *   restoreit;
    wSTAT_ARRAY wstatus;
    unsigned savesize,
            restoresize,
            retval;
    FILE *   savefile;
    wATTRIBUTE frame_color;

    if (wOpen(win, 5, 5, 35, 5)) {
        fprintf(stderr, "\nCannot open required window");
        exit(255);
    }

    wColor(win, wWHITE, wWHITE, wBLACK, wWHITE);
    frame_color.bg_color = frame_color.fg_color = wWHITE;
    frame_color.intensity = 1;
    wFrame(win, wFRAME_RAISED, wSHADOW_RIGHT, frame_color);
    wSelect(win, wUPDATE_ON);

    at(10, 0);
    printf("SAMPLE Program");
    at(4, 1);
    printf("Version 1.0 1 January 1997");
    at(1, 3);
    printf("Copyright 1997 by ABC Corporation");
    at(7, 4);
    printf("All rights reserved");

    // Save the window in buffer, then transfer to file

    // First, allocate buffer

    savesize = (wSaveSize(win)+1)*9/4;
    saveit = malloc(savesize);
```

```
// Get window status

wStatus(win, &wstatus);

// Copy window contents to buffer

retval = wSave(win, saveit, 0, savesize);

wClose(win);                      // finished with window

// Save status and buffer in disk file

if (!(savefile = fopen("sample.copyrite", "w")))
    exit(fperror());
fwrite(&wstatus, 1, sizeof(wstatus), savefile);
fwrite(saveit, 1, savesize, savefile);
fclose(savefile);

// Retrieve window from file and display on screen

if (!(savefile = fopen("sample.copyrite", "r")))
    exit(fperror());

// First, get window status array

fread(&wstatus, 1, sizeof(wstatus), savefile);

// Next, use window size to allocate content buffer

restoreit = malloc(restoresize=wstatus.width*wstatus.height*3);

// Read content into buffer

fread(restoreit, 1, restoresize, savefile);

// Create new window

if (wOpen(win1, getll(27)-3-wstatus.width, 1,
wstatus.width, wstatus.height)) {
    fprintf(stderr, "\nCannot open required window");
    exit(255);
}
wColor(win1, wstatus.fg_color, wstatus.bg_color,
wstatus.rfg_color, wstatus.rbg_color);
wFrame(win1, wstatus.frame, wstatus.shadow, wstatus.frame_attr);

// Restore the window contents

wRestore(win1, restoreit, 0, restoresize);

wSelect(win1, wUPDATE_ON);        // display the window

pagewait();

wCloseAll();
}
```

## wScroll

Add or update a scroll bar on the right side of a window's frame.

```
#include <wmapi.h>
```

```
int wScroll ( int window, int percent )
```

---

*percent*                   »   percentage value between 0 and 100

*window*                    »   window number

**Operation:**        A ***scroll bar*** is added or updated on the open *window*. The ***scroll button*** is positioned so that it is *percent* amount between the top and bottom of the scroll bar.

**Returns:**           Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**            A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 29           | No frame. Scroll bars are displayed in the frame area of a window.                                              |

**Notes:**            The existence of a scroll bar and the position of the scroll button are not remembered by the Session Manager and are not part of the window status reported by the [wStatus](#) function.

**Restrictions:**     *window* must be open and it must have a frame.

**Conforms to:**        **wScroll**    q ANSI    q DOS    n THEOS   q POSIX

**See also:**           [wFrame](#)

## wSelect

Selects and activates a window for subsequent input and output.

```
#include <wmapi.h>
int wSelect ( int window, wUPDATE_MODE update )
```

---

*update*                   »   display update code

*window*                   »   window number

**Operation:** Window number *window* is selected as the active window and its display status is set to the *update* mode. All subsequent input from and output to the console are within the boundaries of this window.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 12           | Select error. Invalid wUPDATE_MODE value.                                                                       |

**Notes:** Values for wUPDATE\_MODE are defined in WMAPI.H and include:

| Name           | Value | Meaning                                                                                                                                                                                    |
|----------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| wUPDATE_OFF    | 0     | Window is selected but it is not refreshed and changes are not displayed at this time.                                                                                                     |
| wUPDATE_ON     | 1     | Window is selected, refreshed and changes are displayed as they are made.                                                                                                                  |
| wUPDATE_TOP    | 2     | Window is selected with wUPDATE_ON mode and its order is set to be on top of all other windows. When another window is selected without this top attribute, it will not cover this window. |
| wUPDATE_HIDDEN | 4     | Window is selected with wUPDATE_OFF mode and it is hidden from display. This window will not be displayed until it is selected with a different mode or until it is refreshed.             |

When a window is selected, the previous state of the window takes effect for the following attributes: cursor location, cursor on/off, video attributes, normal and reverse video text colors.

#### ■ Hidden mode

The `wUPDATE_HIDDEN` mode of a window causes that window to be hidden from view at all times. Specifically, a hidden window is not placed in the “refresh sequence” and is not displayed on the screen.

Hidden windows must be selected with `wUPDATE_HIDDEN` every time they are selected or they will become a visible window.

#### ■ Top mode

When a window is selected with `wUPDATE_ON` mode but not with `wUPDATE_HIDDEN` or `wUPDATE_TOP`, it is displayed on the screen on top of all other windows except those windows that were previously selected with the `wUPDATE_TOP` mode.

Selecting a window with the `wUPDATE_TOP` mode causes that window to be displayed on the screen on top of all other windows, including other windows previously selected with the `wUPDATE_TOP` mode.

#### ■ Window 0

When window number zero is selected, it covers all other windows including those marked as `wUPDATE_TOP`.

When window zero is selected and then another window is selected, and there is at least one window marked as `wUPDATE_TOP`, window zero becomes hidden. Window zero can be selected with `wUPDATE_HIDDEN` to keep it from appearing as a background to other windows.

**Defaults:** The first time that a window is selected, the cursor location is 0,0.

**Restrictions:** Window number 0 can be selected with this statement but it cannot be set to `wUPDATE_TOP` mode.

**Conforms to:** `wSelect`    q ANSI    q DOS    n THEOS    q POSIX

**See also:** [wChoice](#), [wChoiceFuncKeys](#), [wChoiceSelect](#), [wChoiceTimeout](#), [wClose](#), [wCloseAll](#), [wMenuBar](#), [wOpen](#), [wOrder](#), [wRefresh](#), [wRepaint](#)

---

**Example:** See [wOpen](#) example on page 721.

## wStatus

Get the current attributes of an open window.

```
#include <wmapi.h>
int wStatus ( int window, wSTAT_ARRAY * stat_array )
```

---

*stat\_array*           »   pointer to window status array

*window*               »   window number

**Operation:**       The attributes of *window* are retrieved and copied to the *stat\_array* structure.

**Returns:**           Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:**           A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |

**Notes:**           The wSTAT\_ARRAY type is defined in WMAPL.H file. It is a structure with the following members:

```
typedef struct wSTAT_ARRAY {
    wOPEN_STATUS open_status;
    short beg_col;
    short beg_row;
    short width;
    short height;
    short col;
    short row;
    wCURSOR_TYPE cursor;
    wFRAME_TYPE frame;
    wSHADOW_TYPE shadow;
    wTITLE_TYPE title;
    wTITLE_ALIGN title_align;
    wATTRIBUTE title_attr;
    wATTRIBUTE frame_attr;
    wCLIP_MODE clip;
    wUPDATE_MODE update;
    wATTRIBUTE attr;
    short sequence;
    wCOLORS fg_color;
    wCOLORS bg_color;
    wCOLORS rfg_color;
    wCOLORS rbg_color;
    wINVERT_TYPE invert;
} wSTAT_ARRAY;
```

The status array is read-only in the sense that changing a value in the array has no effect on the actual window. To change an attribute of a window, one or more of the functions [wClip](#), [wColor](#), [wFrame](#), [wInvert](#), [wMove](#) or [wTitle](#) must be used.

## 740 *wStatus*

---

**Conforms to:**                      **wStatus**      q ANSI      q DOS      n THEOS      q POSIX

---

**Example:**                      Refer to the example used for the [wSave](#), [wSaveSize](#), [wRestore](#) function description on page 733.



## wSwitch, wSwitchTo

Control the console's session-switching ability.

```
#include <wmapi.h>
int wSwitch ( wSWITCH_SESSION_TYPE on_off )
void wSwitchTo ( int session )
```

---

|                |   |                                  |
|----------------|---|----------------------------------|
| <i>on_off</i>  | » | Enable/disable session-switching |
| <i>session</i> | » | session number to switch to      |

**Operation:**      **wSwitch**      Enable or disable the console's session-switching capability. If the console does not support session-switching (see `IsSession` on page 373), no action is taken by this function.

When the argument is `wSWITCH_QUERY`, then the status is not changed but the current status is returned.

**wSwitchTo**      If session-switching is enabled, the display and keyboard for *session* becomes the active console session.

**Return:**      The return from `wSwitch` is always zero unless `wSWITCH_QUERY` is used, in which case the return value is the current session-switching state.

**Notes:**      Values for `wSWITCH_SESSION_TYPE` are defined in `WMAPI.H` and include:

| <i>Name</i>                  | <i>Value</i> | <i>Meaning</i>            |
|------------------------------|--------------|---------------------------|
| <code>wSWITCH_DISABLE</code> | 0            | Disable session-switching |
| <code>wSWITCH_ENABLE</code>  | 1            | Enable session-switching  |
| <code>wSWITCH_QUERY</code>   | -1           | Report current status     |

Session-switching status can only be changed by a program that is the active session (see `IsActive` on page 366).

When session-switching is disabled, then all programs using this console are prevented from switching sessions.

While session-switching is disabled, requests from the operator to switch sessions (i.e., with a `[Break]`, digit key sequence) are ignored.

The session-switching status is not reset when a program exits.

**Restrictions:**      Session-switching status can only be changed by a program that is the active session (see `IsActive` on page 366).

**Conforms to:**

|                  |        |       |         |         |
|------------------|--------|-------|---------|---------|
| <b>wSwitch</b>   | q ANSI | q DOS | n THEOS | q POSIX |
| <b>wSwitchTo</b> | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**      [IsActive](#), [IsSession](#), [wGetActive](#), [wGetSess](#)

**Example:** This function could be used to temporarily force the console to activate this program's session. It remembers the currently active session number so that it can return the console to its prior state when the program is finished with the priority condition requiring the active session display.

Note that it also reverts the session-switch capability back to the state it was in prior to activating this program's session. That way, if this program had session-switching disabled, it will remain disabled when this priority condition is finished.

```
#include <stdio.h>
#include <timer.h>
#include <types.h>
#include <wmapi.h>

Bool activate_session(int);

void
main(void)
{
    wSwitchTo(0);
    sleep(2000);
    if (activate_session(1)) {
        printf("important message");
        sleep(1000);
        activate_session(0);
    }
    else
        printf("Cannot display important message at this time.\n");
}

Bool
activate_session(int who)
{
    static int my_sess = -1,
              prior_sess,
              switch_status;

    static Bool changed = FALSE;
    Bool retval;

    if (who) {
        // activate me?
        switch_status=wSwitch(wSWITCH_QUERY);
        if (switch_status==wSWITCH_ENABLE) { // switching enabled?
            if (IsActive()) { // another session active?
                my_sess = wGetSess(); // get my session number
                prior_sess = wGetActive(); // get active session
                SwitchTo(my_sess); // make me active
            }
            wSwitch(wSWITCH_DISABLE); // disable switching
            changed = TRUE;
            retval = TRUE; // okay
        }
        else {
            changed = FALSE; // remember unchanged
            retval = FALSE; // cannot switch now
        }
    }
}
```

```
    else {
        if (changed) {
            wSwitch(wSWITCH_ENABLE);           // enable switching
            if (my_sess!=-1) {                 // activate prior ?
                wSwitchTo(prior_sess);         // and switch to prior
                my_sess=-1;                     // reset
            }
            changed = FALSE;                   // reset
        }
        retval = TRUE;
    }
    return retval;
}
```

**wTake**

Copy a region of text from one window to another.

```
#include <wmapi.h>
int wTake ( int from_win, int from_col, int from_row, int width, int height,
            int to_win, int to_col, int to_row )
```

---

|                 |   |                              |
|-----------------|---|------------------------------|
| <i>from_col</i> | » | starting column number       |
| <i>from_row</i> | » | starting row number          |
| <i>from_win</i> | » | window number of source      |
| <i>height</i>   | » | number of rows to take       |
| <i>to_col</i>   | » | destination column number    |
| <i>to_row</i>   | » | destination row number       |
| <i>to_win</i>   | » | window number or destination |
| <i>width</i>    | » | number of columns to take    |

**Operation:** The rectangular region of text in *from\_win* is defined by the upper-left corner of *from\_col*, *from\_row* for a *width* and *height* as specified. This region is moved from that location in the *from\_win* to the *to\_col*, *to\_row* of *to\_win*.

The region in *from\_win* is replaced with spaces without any attributes.

Any text and attributes that existed in *to\_win* in the copied-to location are replaced with the region of text and attributes from the region in the *from\_win*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager. |
| 16           | Copy error. The region specified is not wholly contained with the window.                                       |

**Notes:** The *from\_win* and the *to\_win* may be the same window number and the source and destination regions may overlap.

If either the *from\_win* or the *to\_win* is the active window with `wUPDATE_ON` or `wUPDATE_TOP`, it is refreshed.

**Restrictions:** Both the *from\_win* and the *to\_win* must be open although neither has to be the active window.

The rectangular region must fit within the boundaries of *to\_win*.

**Conforms to:**                    **wTake**    q ANSI    q DOS    n THEOS    q POSIX

**See also:**                    [wCopy](#), [wSave](#), [wSaveSize](#), [wRestore](#)

**wTitle**

Define the title text, position and attributes for a window.

```
#include <wmapi.h>
int wTitle ( int window, char * title, wTITLE_TYPE position,
            wTITLE_ALIGN align, wATTRIBUTE attr )
```

---

|                 |   |                        |
|-----------------|---|------------------------|
| <i>align</i>    | » | title alignment code   |
| <i>attr</i>     | » | display attribute code |
| <i>position</i> | » | title position code    |
| <i>title</i>    | » | window title text      |
| <i>window</i>   | » | window number          |

**Operation:** The title for *window* is defined using *title*, *position* and *align*. The title's color and video attributes are defined according to *attr*.

**Returns:** Success or failure code. A zero indicates success; a non-zero indicates failure and the specific reason for the failure.

**Errors:** A non-zero return value specifies the error that occurred during the operation of this function. Possible error codes and their meanings are:

| Return Value | Meaning                                                                                                            |
|--------------|--------------------------------------------------------------------------------------------------------------------|
| 1            | Window not open.                                                                                                   |
| 6            | Window number error. Window numbers must be in the range of zero to the maximum allowed by the Session Manager.    |
| 19           | Invalid operation on window 0. Window 0 cannot be resized, moved, have a frame or shadow, and it cannot be closed. |
| 27           | Title error. Invalid wTITLE_TYPE or wTITLE_ALIGN value.                                                            |
| 28           | String error.                                                                                                      |
| 29           | No frame. Titles are displayed in the frame area of a window.                                                      |

**Notes:** Multi-national and line-drawing characters may be included in the title text.

Values for wTITLE\_ALIGN are defined in WMAPI.H and include:

| Name               | Value | Meaning                          |
|--------------------|-------|----------------------------------|
| wTITLE_CENTERED    | 0     | Title text is centered in frame. |
| wTITLE_ALIGN_LEFT  | 1     | Title text is left-justified.    |
| wTITLE_ALIGN_RIGHT | 2     | Title text is right-justified.   |

Values for `wTITLE_TYPE` are defined in `WMAPI.H` and include:

| <i>Name</i>                | <i>Value</i> | <i>Meaning</i>                      |
|----------------------------|--------------|-------------------------------------|
| <code>wTITLE_NONE</code>   | 0            | Title is suppressed.                |
| <code>wTITLE_TOP</code>    | 1            | Title is displayed in frame top.    |
| <code>wTITLE_BOTTOM</code> | 2            | Title is displayed in frame bottom. |

The *attr* field has the same meaning and functions as the *attr* field in the [wFrame](#) function which is described on page 694.

**Defaults:** If *window* is the active window with `wUPDATE_ON`, the title text is displayed immediately; otherwise the new title definition is not displayed until *window* is made the active window or it is refreshed.

**Restrictions:** *window* must be open although it does not have to be the active window.

The length of *title* must be less than or equal to the window's width.

**Conforms to:** `wTitle` q ANSI q DOS n THEOS q POSIX

**See also:** [wFrame](#), [wOpen](#)

---

**Example:**

```
wATTRIBUTE attr;
...
wOpen(win, 5, 5, 30, 5);
memset(&attr, 0, sizeof(wATTRIBUTE));
attr.fg.color = wWHITE; // define as bright white on blue
attr.bg.color = wBLUE;
attr.intensity = 1;
wTitle(win, " Example Title ", wTITLE_TOP, wTITLE_CENTERED, attr);
...
```

**wVer**

This function returns the version number of the Session Manager installed on the system.

```
#include <wmapi.h>
int wVer ( void )
```

**Operation:** The Session Manager version number is retrieved and returned.

**Returns:** The Session Manager version number. A return of zero indicates that Session Manager is not installed on the system.

When a non-zero value is returned, it is the two-byte value of the Session Manager version number. The high-order byte is the version number, the low-order byte is the version modifier. For instance, a return value of 769 = 0x0301 = Session Manager Version 3.1.

**Notes:** If the Session Manager is not available (return value of zero), most of the other windowing functions will generate an error when they are used. It is best to use the wVer function to test for the presence of the Session Manager. When it is not available, the program should either exit with a message or use a different method of displaying information.

**Conforms to:**                      **wVer**    q ANSI    q DOS    n THEOS    q POSIX

---

**Example:**

```
...
if (!wVer())
    syserr(255, 548, NULL);
...
```



## extended file functions

These functions allow programs executing on THEOS Version 3.2 systems to open and use as many as 1000 files.

```
#include <xstdio.h>
int xclose ( int fhandle )
int xcreat ( char * filename, int mode )
int xdup ( int fhandle )
FILE * xfdopen ( int fhandle, char * type )
int xfileno ( FILE * file )
int xflush ( int fhandle )
int xfopen ( char * filename, char * mode )
int xlocking ( int fhandle, int mode, unsigned long size )
long xlseek ( int fhandle, long offset, int base )
int xopen ( char * filename, int mode )
size_t xread ( int fhandle, void * buffer, size_t length )
long xtell ( int fhandle )
size_t xwrite ( int fhandle, void * buffer, size_t length )
```

---

|                 |   |                                    |
|-----------------|---|------------------------------------|
| <i>base</i>     | » | starting point within file         |
| <i>buffer</i>   | » | pointer to storage area            |
| <i>fhandle</i>  | » | file handle of an open file        |
| <i>file</i>     | » | pointer to an open file's fcb      |
| <i>filename</i> | » | pointer to complete file name      |
| <i>length</i>   | » | number of bytes to write           |
| <i>mode</i>     | » | file organization mode             |
| <i>offset</i>   | » | relative position to seek to       |
| <i>size</i>     | » | length of region to lock or unlock |
| <i>type</i>     | » | coded type of open                 |

**Operation:** These functions operate just like their non-x counterparts except that they can open and access as many as 1000 files in any one task.

**Returns:** The returns from these functions are the same as their non-x counterparts. Refer to them for information about return values and error conditions detected.

**Notes:** These functions should only be used if the program will be used on a system that has THEOS Version 3.2. When used on THEOS Version 4 and above, these functions merely call the non-x form of the function, thus requiring extra overhead and processing time.

If the program does not need more than 50 files open at any one time, it is best to use the non-x form of the function.

Most of the functions except xfdopen, xfileno and xfopen can be used by including the XSTDIO.H header file after STDIO.H is included and then using the non-x form of the name. The XSTDIO.H file redefines the standard names to be the "x" form of the name.

## 750 *extended file functions*

---

To use the `xfdopen`, `xfileno` and `xopen` functions, you must include `XSTDIO.H` and use the “x” form of the name explicitly.

**Restrictions:** Do not mix the usage of these functions with the non-x form of the function.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>xclose</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xcreat</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xdup</b>     | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xfdopen</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xfileno</b>  | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xflush</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xopen</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xlocking</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xlseek</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xopen</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xread</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xtell</b>    | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>xwrite</b>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     |                 |        |       |         |         |

**See also:** [close](#), [creat](#), [dup](#), [fdopen](#), [fileno](#), [flush](#), [fopen](#), [locking](#), [lseek](#), [open](#), [read](#), [tell](#), [write](#)

## xor memory assembly functions

This group of “functions” generates in-line code to perform an XOR operation on bytes, short integers or long integers in another memory area or in your program’s code segment.

```
#include <builtin.h>
void _xorb ( void _far * far_offset, char mask )
void _xord ( void _far * far_offset, long lmask )
void _xorw ( void _far * far_offset, int imask )
```

```
void _xorcsb ( void * offset, char mask )
void _xorcsd ( void * offset, long lmask )
void _xorcsw ( void * offset, int imask )
```

---

|                   |   |                                      |
|-------------------|---|--------------------------------------|
| <i>far_offset</i> | » | pointer to remote data object        |
| <i>imask</i>      | » | 16-bit mask value                    |
| <i>lmask</i>      | » | 32-bit mask value                    |
| <i>mask</i>       | » | 8-bit mask value                     |
| <i>offset</i>     | » | pointer to data item in code segment |

**Operation:** These functions perform a bit-wise, Boolean XOR operation. That is, a bit position in the result is set to “on” if, and only if, the same bit position in one of the objects is “on” but “off” in the other object. If both objects have a bit position “on” or both have a bit position “off” the result will be an “off” bit position.

**\_xorb** XORs the value of *bval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_xord** XORs the value of *lval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_xorw** XORs the value of *wval* with the value in the location *far\_offset*. The result is placed back in that location of the remote memory area.

**\_xorcsb, \_xorcsd, \_xorcsw** These functions perform the same operation as **\_xorb**, **\_xord** and **\_xorw** except that the memory segment is pre-defined to be your program’s code segment alias. These three functions can only be used in device-drivers because other programs do not have an alias to their code segment and they will generate a general protection error.

**Returns:** No value is returned by these functions. The values are XORed directly to the values in the locations specified.

**Notes:** These “built-in” functions generate in-line code, thus avoiding the expensive use of the `call` and `ret` instructions.

The arguments *far\_offset* and *offset* are void pointers. This means that the functions can XOR whatever type of object desired to any location. For instance, an **\_xorw** function can XOR a word value (double-byte value) to a

location that is declared as a character string. Of course, the program would have to be coded so that it could handle this situation.

**Restrictions:** These functions are intended for device-driver authors.

The nucleus memory region is not generally accessible without proper memory-access permissions. Device-driver programs operate as an extension to the nucleus and thus have sufficient permission to change locations within the nucleus.

Modifying your program's code segment is not advised unless you are an advanced programmer or have the advice of an advanced programmer.

|                     |                      |        |       |         |         |
|---------------------|----------------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <code>_xorb</code>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <code>_xord</code>   | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <code>_xorcsb</code> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <code>_xorcsd</code> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <code>_xorcsw</code> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <code>_xorw</code>   | q ANSI | q DOS | n THEOS | q POSIX |

**See also:** [add memory assembly functions](#), [and memory assembly functions](#), [decrement memory assembly functions](#), [increment memory assembly functions](#), [or memory assembly functions](#), [peek memory assembly functions](#), [poke memory assembly functions](#), [store memory assembly functions](#), [subtract memory assembly functions](#)



**Example:**

```
#include <stdio.h>
#include <stdlib.h>

void
main()
{
    int    reply,
           update = 0,
           updateall = 0;

    load_yn();                // initialize yesno function

    printf("\nAre you ready to proceed? ");
    reply = yesno();        // get yes or no response

    if (reply != 1)            // response negative?
        exit(0);              // they are not ready

    printf("\nDo you want to update files? (Yes, No, All) ");
    reply = _yesno();

    if (reply == 1)            // reply was yes
        update = -1;
    elseif (reply == -2)       // reply was all
        updateall = -1;
    else
        exit(254);            // escape or no = quit
    ...
}
```

---

**Output:**

>test

Are you ready to proceed? Y

Do you want to update files? (Yes, No, All) A

## yield, \_preempt, \_snu, \_yield

Yield the CPU to the next user.

```
#include <process.h>
void yield ( void )

#include <sc.h>
void _preempt ( void )
void _snu ( void )
void _yield ( void )
```

**Operation:**      **yield**      yield, \_snu and \_yield are synonyms to each other and always yield any time remaining for your task's processing to the next task that is waiting for a "time slice."

**\_preempt**      Your process yields its time only if there is another task waiting that has higher priority than the current process.

**Notes:**      These operations differ from a sleep or other delaying operation because they ensure that other users will get a chance to execute before your program is activated again.

                 These functions are normally only used by device-drivers because many of the library functions automatically perform a yield operation when they cannot proceed.

                 The \_preempt is used by device-drivers that are waiting for some event but do not want to yield their time to a lower or equal priority task.

|                     |                 |        |       |         |         |
|---------------------|-----------------|--------|-------|---------|---------|
| <b>Conforms to:</b> | <b>_preempt</b> | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>snu</b>      | q ANSI | q DOS | n THEOS | q POSIX |
|                     | <b>yield</b>    | q ANSI | q DOS | n THEOS | q POSIX |

**See also:**      [pause](#), [delay](#), [sleep](#), [sleep\\_msec](#), [sleep\\_sec](#), [sleep\\_until](#)





# A: Character Sets

| chr | dec | oct | hex  | isalnum | isalpha | isascii | isctrl | iscsym | iscsymnum | isdis | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|-----|------|---------|---------|---------|--------|--------|-----------|-------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| NUL | 0   | 00  | 0x0  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| SOH | 1   | 01  | 0x1  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| STX | 2   | 02  | 0x2  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ETX | 3   | 03  | 0x3  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| EOT | 4   | 04  | 0x4  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ENQ | 5   | 05  | 0x5  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ACK | 6   | 06  | 0x6  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| BEL | 7   | 07  | 0x7  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| BS  | 8   | 010 | 0x8  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| TAB | 9   | 011 | 0x9  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| LF  | 10  | 012 | 0xA  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| VT  | 11  | 013 | 0xB  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| FF  | 12  | 014 | 0xC  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| CR  | 13  | 015 | 0xD  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| SO  | 14  | 016 | 0xE  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| SI  | 15  | 017 | 0xF  |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DLE | 16  | 020 | 0x10 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DC1 | 17  | 021 | 0x11 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DC2 | 18  | 022 | 0x12 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DC3 | 19  | 023 | 0x13 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DC4 | 20  | 024 | 0x14 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| NAK | 21  | 025 | 0x15 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| SYN | 22  | 026 | 0x16 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ETB | 23  | 027 | 0x17 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| CAN | 24  | 030 | 0x18 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| EM  | 25  | 031 | 0x19 |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| SUB | 26  | 032 | 0x1A |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ESC | 27  | 033 | 0x1B |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| FS  | 28  | 034 | 0x1C |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| GS  | 29  | 035 | 0x1D |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| RS  | 30  | 036 | 0x1E |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |
| US  | 31  | 037 | 0x1F |         |         |         |        |        |           |       |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct | hex  | isalnum | isalpha | isascii | iscntrl | iscsym | iscsymnum | isdisps | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|-----|------|---------|---------|---------|---------|--------|-----------|---------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
|     | 32  | 040 | 0x20 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| !   | 33  | 041 | 0x21 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| "   | 34  | 042 | 0x22 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| #   | 35  | 043 | 0x23 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| \$  | 36  | 044 | 0x24 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| %   | 37  | 045 | 0x25 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| &   | 38  | 046 | 0x26 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| '   | 39  | 047 | 0x27 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| (   | 40  | 050 | 0x28 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| )   | 41  | 051 | 0x29 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| *   | 42  | 052 | 0x2A |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| +   | 43  | 053 | 0x2B |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ,   | 44  | 054 | 0x2C |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| -   | 45  | 055 | 0x2D |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| .   | 46  | 056 | 0x2E |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| /   | 47  | 057 | 0x2F |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 0   | 48  | 060 | 0x30 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 1   | 49  | 061 | 0x31 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 2   | 50  | 062 | 0x32 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 3   | 51  | 063 | 0x33 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 4   | 52  | 064 | 0x34 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 5   | 53  | 065 | 0x35 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 6   | 54  | 066 | 0x36 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 7   | 55  | 067 | 0x37 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 8   | 56  | 070 | 0x38 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| 9   | 57  | 071 | 0x39 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| :   | 58  | 072 | 0x3A |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ;   | 59  | 073 | 0x3B |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| <   | 60  | 074 | 0x3C |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| =   | 61  | 075 | 0x3D |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| >   | 62  | 076 | 0x3E |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ?   | 63  | 077 | 0x3F |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum | isalpha | isascii | isctrl | iscsym | iscsymnum | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|------|------|---------|---------|---------|--------|--------|-----------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| @   | 64  | 0100 | 0x40 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| A   | 65  | 0101 | 0x41 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| B   | 66  | 0102 | 0x42 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| C   | 67  | 0103 | 0x43 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| D   | 68  | 0104 | 0x44 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| E   | 69  | 0105 | 0x45 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| F   | 70  | 0106 | 0x46 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| G   | 71  | 0107 | 0x47 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| H   | 72  | 0110 | 0x48 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| I   | 73  | 0111 | 0x49 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| J   | 74  | 0112 | 0x4A |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| K   | 75  | 0113 | 0x4B |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| L   | 76  | 0114 | 0x4C |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| M   | 77  | 0115 | 0x4D |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| N   | 78  | 0116 | 0x4E |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| O   | 79  | 0117 | 0x4F |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| P   | 80  | 0120 | 0x50 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| Q   | 81  | 0121 | 0x51 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| R   | 82  | 0122 | 0x52 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| S   | 83  | 0123 | 0x53 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| T   | 84  | 0124 | 0x54 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| U   | 85  | 0125 | 0x55 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| V   | 86  | 0126 | 0x56 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| W   | 87  | 0127 | 0x57 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| X   | 88  | 0130 | 0x58 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| Y   | 89  | 0131 | 0x59 |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| Z   | 90  | 0132 | 0x5A |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| [   | 91  | 0133 | 0x5B |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| \   | 92  | 0134 | 0x5C |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ]   | 93  | 0135 | 0x5D |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ^   | 94  | 0136 | 0x5E |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |
| _   | 95  | 0137 | 0x5F |         |         |         |        |        |           |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum | isalpha | isascii | iscntrl | iscsym | iscsymnum | isdisps | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|------|------|---------|---------|---------|---------|--------|-----------|---------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| '   | 96  | 0140 | 0x60 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| a   | 97  | 0141 | 0x61 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| b   | 98  | 0142 | 0x62 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| c   | 99  | 0143 | 0x63 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| d   | 100 | 0144 | 0x64 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| e   | 101 | 0145 | 0x65 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| f   | 102 | 0146 | 0x66 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| g   | 103 | 0147 | 0x67 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| h   | 104 | 0150 | 0x68 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| i   | 105 | 0151 | 0x69 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| j   | 106 | 0152 | 0x6A |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| k   | 107 | 0153 | 0x6B |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| l   | 108 | 0154 | 0x6C |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| m   | 109 | 0155 | 0x6D |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| n   | 110 | 0156 | 0x6E |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| o   | 111 | 0157 | 0x6F |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| p   | 112 | 0160 | 0x70 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| q   | 113 | 0161 | 0x71 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| r   | 114 | 0162 | 0x72 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| s   | 115 | 0163 | 0x73 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| t   | 116 | 0164 | 0x74 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| u   | 117 | 0165 | 0x75 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| v   | 118 | 0166 | 0x76 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| w   | 119 | 0167 | 0x77 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| x   | 120 | 0170 | 0x78 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| y   | 121 | 0171 | 0x79 |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| z   | 122 | 0172 | 0x7A |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| {   | 123 | 0173 | 0x7B |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 124 | 0174 | 0x7C |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| }   | 125 | 0175 | 0x7D |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| ~   | 126 | 0176 | 0x7E |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
| DEL | 127 | 0177 | 0x7F |         |         |         |         |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum                                                                                 | isalpha | isascii | isctrl | iscsym | iscsynnum | isdispc | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issynnum | isupper | isxdigit |
|-----|-----|------|------|-----------------------------------------------------------------------------------------|---------|---------|--------|--------|-----------|---------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
|     | 128 | 0200 | 0x80 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 129 | 0201 | 0x81 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 130 | 0202 | 0x82 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 131 | 0203 | 0x83 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 132 | 0204 | 0x84 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 133 | 0205 | 0x85 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 134 | 0206 | 0x86 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 135 | 0207 | 0x87 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 136 | 0210 | 0x88 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 137 | 0211 | 0x89 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 138 | 0212 | 0x8A |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 139 | 0213 | 0x8B |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 140 | 0214 | 0x8C |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 141 | 0215 | 0x8D |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 142 | 0216 | 0x8E |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 143 | 0217 | 0x8F |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 144 | 0220 | 0x90 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 145 | 0221 | 0x91 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 146 | 0222 | 0x92 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 147 | 0223 | 0x93 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 148 | 0224 | 0x94 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 149 | 0225 | 0x95 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 150 | 0226 | 0x96 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 151 | 0227 | 0x97 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 152 | 0230 | 0x98 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 153 | 0231 | 0x99 |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 154 | 0232 | 0x9A |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 155 | 0233 | 0x9B |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 156 | 0234 | 0x9C |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 157 | 0235 | 0x9D |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 158 | 0236 | 0x9E |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 159 | 0237 | 0x9F |                                                                                         |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     |     |      |      | Cells with an "T" in them are true only if the LC_CTYPE locale is set to International. |         |         |        |        |           |         |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum | isalpha | isascii | isctrl | iscsym | iscsynnum | isdispc | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issynnum | isupper | isxdigit |
|-----|-----|------|------|---------|---------|---------|--------|--------|-----------|---------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| ≥   | 160 | 0240 | 0xA0 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| Ł   | 161 | 0241 | 0xA1 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ƒ   | 162 | 0242 | 0xA2 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ƒ   | 163 | 0243 | 0xA3 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ü   | 164 | 0244 | 0xA4 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ſ   | 165 | 0245 | 0xA5 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ½   | 166 | 0246 | 0xA6 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| σ   | 167 | 0247 | 0xA7 |         |         |         |        |        |           |         |         |         |         |       |         | I      | I       |         |         |          |          |         |          |

Table 18: Character Types

## 762 Character Sets

| chr | dec | oct  | hex  | isalnum                                                                                 | isalpha | isascii | isctrl | iscsym | iscsymnum | isdisp | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|------|------|-----------------------------------------------------------------------------------------|---------|---------|--------|--------|-----------|--------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| t   | 168 | 0250 | 0xA8 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ç   | 169 | 0251 | 0xA9 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ı   | 170 | 0252 | 0xAA |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
|     | 171 | 0253 | 0xAB |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
|     | 172 | 0254 | 0xAC |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
|     | 173 | 0255 | 0xAD |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
|     | 174 | 0256 | 0xAE |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| â   | 175 | 0257 | 0xAF |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ℓ   | 176 | 0260 | 0xB0 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ƒ   | 177 | 0261 | 0xB1 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ø   | 178 | 0262 | 0xB2 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ô   | 179 | 0263 | 0xB3 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ø   | 180 | 0264 | 0xB4 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ƒ   | 181 | 0265 | 0xB5 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| φ   | 182 | 0266 | 0xB6 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| μ   | 183 | 0267 | 0xB7 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| Ω   | 184 | 0270 | 0xB8 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
| ƒ   | 185 | 0271 | 0xB9 |                                                                                         |         |         |        |        |           |        |         |         |         |       |         | I      | I       |         |         |          |          |         |          |
|     | 186 | 0272 | 0xBA |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 187 | 0273 | 0xBB |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 188 | 0274 | 0xBC |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 189 | 0275 | 0xBD |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 190 | 0276 | 0xBE |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     | 191 | 0277 | 0xBF |                                                                                         |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |
|     |     |      |      | Cells with an “I” in them are true only if the LC_CTYPE locale is set to International. |         |         |        |        |           |        |         |         |         |       |         |        |         |         |         |          |          |         |          |

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum | isalpha | isascii | isctrl | iscsym | iscsymnum | isdisp | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issymnum | isupper | isxdigit |
|-----|-----|------|------|---------|---------|---------|--------|--------|-----------|--------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|----------|---------|----------|
| Ä   | 192 | 0300 | 0xC0 | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ä   | 193 | 0301 | 0xC1 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| å   | 194 | 0302 | 0xC2 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| à   | 195 | 0303 | 0xC3 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| á   | 196 | 0304 | 0xC4 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| É   | 197 | 0305 | 0xC5 | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ë   | 198 | 0306 | 0xC6 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ê   | 199 | 0307 | 0xC7 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| è   | 200 | 0310 | 0xC8 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| é   | 201 | 0311 | 0xC9 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ĩ   | 202 | 0312 | 0xCA | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| î   | 203 | 0313 | 0xCB | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ì   | 204 | 0314 | 0xCC | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| í   | 205 | 0315 | 0xCD | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| Ö   | 206 | 0316 | 0xCE | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ö   | 207 | 0317 | 0xCF | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ô   | 208 | 0320 | 0xD0 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ò   | 209 | 0321 | 0xD1 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ó   | 210 | 0322 | 0xD2 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| Ü   | 211 | 0323 | 0xD3 | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ü   | 212 | 0324 | 0xD4 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| û   | 213 | 0325 | 0xD5 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ù   | 214 | 0326 | 0xD6 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| ú   | 215 | 0327 | 0xD7 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| Ç   | 216 | 0330 | 0xD8 | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ç   | 217 | 0331 | 0xD9 | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| Ñ   | 218 | 0332 | 0xDA | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| ñ   | 219 | 0333 | 0xDB | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| Æ   | 220 | 0334 | 0xDC | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| æ   | 221 | 0335 | 0xDD | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |
| À   | 222 | 0336 | 0xDE | I       | I       |         |        | I      | I         |        |         |         |         |       |         | I      |         |         | I       | I        | I        |         |          |
| á   | 223 | 0337 | 0xDF | I       | I       |         |        | I      | I         |        |         |         |         | I     |         | I      |         |         | I       | I        |          |         |          |

Cells with an "I" in them are true only if the LC\_CTYPE locale is set to International.

Table 18: Character Types

| chr | dec | oct  | hex  | isalnum | isalpha | isascii | isctrl | iscsym | iscsynum | isdis | isdigit | isfnsym | isgraph | ishex | islower | isocal | isprint | ispunct | isspace | issymbol | issynum | isupper | isxdigit |
|-----|-----|------|------|---------|---------|---------|--------|--------|----------|-------|---------|---------|---------|-------|---------|--------|---------|---------|---------|----------|---------|---------|----------|
| ß   | 224 | 0340 | 0xE0 | I       | I       |         |        | I      | I        |       |         |         |         | I     |         |        | I       |         |         | I        | I       |         |          |
| ı   | 225 | 0341 | 0xE1 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ı   | 226 | 0342 | 0xE2 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ø   | 227 | 0343 | 0xE3 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| £   | 228 | 0344 | 0xE4 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ¥   | 229 | 0345 | 0xE5 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ƒ   | 230 | 0346 | 0xE6 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ¤   | 231 | 0347 | 0xE7 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ¼   | 232 | 0350 | 0xE8 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ½   | 233 | 0351 | 0xE9 |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ÿ   | 234 | 0352 | 0xEA | I       | I       |         |        | I      | I        |       |         |         |         | I     |         |        | I       |         |         | I        | I       |         |          |
| §   | 235 | 0353 | 0xEB |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ¶   | 236 | 0354 | 0xEC |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
| ²   | 237 | 0355 | 0xED |         |         |         |        |        |          |       |         |         |         |       |         |        | I       | I       |         |          |         |         |          |
|     | 238 | 0356 | 0xEE |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 239 | 0357 | 0xEF |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 240 | 0360 | 0xF0 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 241 | 0361 | 0xF1 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 242 | 0362 | 0xF2 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 243 | 0363 | 0xF3 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 244 | 0364 | 0xF4 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 245 | 0365 | 0xF5 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 246 | 0366 | 0xF6 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 247 | 0367 | 0xF7 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 248 | 0370 | 0xF8 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 249 | 0371 | 0xF9 |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 250 | 0372 | 0xFA |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 251 | 0373 | 0xFB |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 252 | 0374 | 0xFC |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 253 | 0375 | 0xFD |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 254 | 0376 | 0xFE |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |
|     | 255 | 0377 | 0xFF |         |         |         |        |        |          |       |         |         |         |       |         |        |         |         |         |          |         |         |          |

Cells with an “I” in them are true only if the LC\_CTYPE locale is set to International.

Table 18: Character Types



# B: Obsolete Function Names

---

The following function names, although still valid, are obsolete and should not be used any more.

| Obsolete name | Replaced with            |
|---------------|--------------------------|
| callcsi       | <a href="#">system</a>   |
| conin         | <a href="#">getch</a>    |
| conout        | <a href="#">putch</a>    |
| display       | <a href="#">cputs</a>    |
| farcat        | <a href="#">_fstrcat</a> |
| farchr        | <a href="#">_fstrchr</a> |
| farcmp        | <a href="#">_fmemcmp</a> |
| farcpy        | <a href="#">_fmemcpy</a> |
| fareq         | <a href="#">_fstreq</a>  |
| farlen        | <a href="#">_fstrlen</a> |
| farset        | <a href="#">_fmemset</a> |
| keyin         | <a href="#">cgets</a>    |
| lowercase     | <a href="#">strlwr</a>   |
| match         | <a href="#">strmatch</a> |
| mem_ovly      |                          |
| strsave       | <a href="#">strdup</a>   |
| trim          | <a href="#">strtrim</a>  |
| upcase        | <a href="#">strupr</a>   |



# C: THEOS 32 Version 4 Functions

---

The following functions may only be used in programs that will execute on systems using THEOS 32 Version 4 or later.

- acc\_access
- acc\_maint
- \_attach
- \_detach
- \_df\_close
- \_df\_fdb
- \_df\_open
- \_df\_read
- \_df\_ucb
- dirclose
- \_dirfdb
- diropen
- dirread
- \_dirucb
- \_disk\_capacity
- find\_acc
- \_find\_first
- \_find\_next
- get\_acc\_name
- InitFileClose
- InitFileOpen
- InitFileRead
- InitFileReadInt
- InitFileReadNext
- InitFileWrite
- InitFileWriteInt
- \_ioctl\_ucb
- \_lockshare
- \_locktest
- \_lockwait
- \_mount\_fs
- pwverify
- read\_acc
- release\_dma
- reserve\_dma
- sch\_acc



# D: Logical Unit Block Numbers

---

The following table shows the logical unit block numbers (lub) for devices that may be attached to a THEOS system. These numbers and names are defined in the file LUB.H and that file should be used as the most current list of these numbers.

| LUB | Name   | LUB | Name   | LUB | Name       |
|-----|--------|-----|--------|-----|------------|
| 0   | A_DISK | 26  | CONIN  | 52  | PRT5       |
| 1   | B_DISK | 27  | CONOUT | 53  | PRT6       |
| 2   | C_DISK | 28  | PRT1   | 54  | PRT7       |
| 3   | D_DISK | 29  | PRT2   | 55  | PRT8       |
| 4   | E_DISK | 30  | PRT3   | 56  | PRT9       |
| 5   | F_DISK | 31  | PRT4   | 57  | PRT10      |
| 6   | G_DISK | 32  | COM1   | 58  | PRT11      |
| 7   | H_DISK | 33  | COM2   | 59  | PRT12      |
| 8   | I_DISK | 34  | COM3   | 60  | PRT13      |
| 9   | J_DISK | 35  | COM4   | 61  | PRT14      |
| 10  | K_DISK | 36  | TAPE1  | 62  | PRT15      |
| 11  | L_DISK | 37  | TAPE2  | 63  | PRT16      |
| 12  | M_DISK | 38  | TAPE3  |     |            |
| 13  | N_DISK | 39  | TAPE4  | 252 | PIPE_LUB   |
| 14  | O_DISK | 40  | COM5   | 254 | SPOOLER    |
| 15  | P_DISK | 41  | COM6   | 255 | UNATTACHED |
| 16  | Q_DISK | 42  | COM7   |     |            |
| 17  | R_DISK | 43  | COM8   | 26  | CONSOLE    |
| 18  | S_DISK | 44  | COM9   | 64  | MAX_LUB    |
| 19  | T_DISK | 45  | COM10  | 255 | NULL_FILE  |
| 20  | U_DISK | 46  | COM11  |     |            |
| 21  | V_DISK | 47  | COM12  |     |            |
| 22  | W_DISK | 48  | COM13  |     |            |
| 23  | X_DISK | 49  | COM14  |     |            |
| 24  | Y_DISK | 50  | COM15  |     |            |
| 25  | Z_DISK | 51  | COM16  |     |            |

Table 19: LUB Numbers and Names



# Index

---

## A

- Abbreviation
  - keyword 292, 426, 630
- Absolute value 57, 103, 176, 380
- Accept network connection 60
- Access rights, of segment 268
- Account
  - change 403
- Account command 59
- Account control block
  - account definition 513
  - get account 270
  - lock for access 59
  - password verify 502
- Account name 12
  - definition, read 513
  - environment definition 513
  - get 270
  - get current 136, 299, 324, 402
  - lock for access/maintenance 59
  - password verify 502
- Account name functions 12
- Add item
  - fifo buffer 197
  - type-ahead buffer 624
- Add memory functions 67
- Add string to array of strings 411
- Allocate
  - fifo buffer 197
  - type-ahead buffer 624
  - user type-ahead buffer 624
- Allocation
  - determining file's 199
  - memory 70, 104, 301, 424
    - changing size of 437, 518
    - currently allocated 640
    - maximum 428
    - maximum system 432
  - memory, freeing 244, 301
  - shared memory 571
    - freeing 571
- And memory functions 71
- Appending
  - to end of file 223
- Arccosecant 66
- Arccosine 64
- Arccotangent 65
- Arcsecant 75
- Arcsine 76
- Arctangent 82
- Arctangent of ratio 82
- ASCII
  - character test 362

- convert to 636
- Assembly-language instructions,
  - adding 77
- Attribute
  - test printer 333
- Automatic record locking 224

---

## B

- Base64
  - convert to binary 88
  - convert to long integer 55
- BASIC string
  - convert to C string 91
- Baud rate
  - convert from code 92
  - convert to code 92
  - get 272
- BCD
  - convert to IEEE 93
- Binary
  - convert to Base64 88
- Bind
  - name to socket 95
- Break-Q
  - disable 507
  - enable 507
- Buffer
  - file I/O allocation 558
- Buffer manipulation functions 13
- Buffers 13–14
  - compare 216–217, 433–434
  - copy 216–217, 433–434, 618
    - reverse sequence 217, 434
  - fifo 13
    - add item 197
    - allocate 197
    - check for item 197
    - deallocate 197
    - get next item 197
  - file I/O allocation 177, 558
  - file I/O, free 178
  - initialize contents 434
  - initialize contents to zero 94
  - locate character in 216, 433
  - memory 14
  - type-ahead 13, 624
    - add item 624
    - allocate 624
    - check for item 624
    - deallocate 624
    - get next item 624
  - user 624
- Buffers, far
  - initialize contents 217

**Bold** page numbers indicate a primary reference to the subject; *italic* numbers indicate an example.

- Buffers, user
  - type-ahead
    - allocate 624
    - deallocate 624
- Built-in functions
  - absolute value 58
  - add memory 67
  - and memory 71
  - CPU register access 268
  - decrement memory 138
  - direction flag 593
  - increment memory 338
  - input port 340
  - move memory 450
  - or memory 463
  - output port 466
  - peek memory 470
  - poke memory 476
  - push and pop flags 481
  - store memory 594
  - string 607
  - subtract memory 616
  - xor memory 751
- Bypass Session Manager
  - clear 560
  - set 560

---

**C**

- C string
  - convert to BASIC string 101
- Cache
  - synchronize 620
- Catalogue
  - semaphore 550–551
- Ceiling 105
  - See also:* Floor
- Change
  - account 403
  - file
    - date and time 200
    - file size 200
    - growth factor 200
    - key length 200
    - ownership 200
    - protection codes 200
    - record length 200
  - file date 285, 497
  - file name 534
  - I/O position 250, 252, 413, 536, 546
  - size of allocated memory 518
  - size of allocated memory segment 437
  - window display order 725
  - window location 718
- Character
  - input 273, 753
    - unget 649–650
  - output 493
  - ready, console 119

- search in string 255–256, 596, 599
- Character classification functions 15
- Character conversion functions 15
- Characters
  - "is" types 562
  - classification 15
  - convert to ASCII 636
  - convert to lowercase 636
  - convert to uppercase 636
  - test alphabeticc 362
  - test alphanumeric 362
  - test ASCII 362
  - test control 362
  - test decimal digit 362
  - test displayable 362–363
  - test file name 362
  - test hexadecimal digit 363
  - test lowercase 363
  - test octal digit 363
  - test punctuation 363
  - test symbol 362–363
  - test uppercase 363
  - test white-space 363
- Checksum
  - generate 126
- Child task
  - See:* Multitask
- Choice list 670
  - timeout 671
- Class code
  - input translation 120
- Class codes
  - PC Term 371
- Clear
  - semaphore 550–551
  - window to character 676
- Close
  - device names file 143
  - file 111, 178
  - initialization file 344
  - keywords file 376
  - message file 452
  - network socket 112
  - window 678
- Code segment
  - get alias selector 268
  - get selector 268
- Codes
  - baud rate 92
  - cursor-control 128
  - error, clearing 109, 233
  - line-drawing 128
  - mouse button 711
  - return 56, 173, 621
  - screen-editing 128
- Color
  - attribute, setting 130



- background 679
    - foreground 679
  - window frame 692
- Command name
  - abbreviation
    - Command name
    - synonym 114
- Compare
  - case-insensitive 217, 434
  - contents of far memory buffers
    - 216–217
  - contents of memory buffers 94,
    - 433–434
  - dictionary order 216, 433
  - for equality 217, 434
  - for equality, case-insensitive
    - 217, 434
  - string to keyword, with abbrevi-
    - ation 630
  - string to mask 598
  - two strings 255, 597–599
    - case-insensitive 256, 597, 599
    - with abbreviation 426
  - two strings for equality 256, 597
    - case-insensitive 256, 597
  - two strings, with numbers 256,
    - 597
- Compute
  - physical address 473
  - size of data item 456
  - window save size 733
- Concatenate
  - strings 255, 596, 598
- Configuration file functions 31
- Connect
  - network socket 117
- Console
  - capabilities 330
  - character ready 119
  - class code 274
  - color 130
  - cursor control 135
  - display 125
  - display, formatted 123
  - input 273, 753
    - casemode 115
    - character 273, 753
      - unset 650
    - control character 115
    - conversion 115
    - echo 115, 125
    - EXEC stack 115, 119, 296
    - Input
      - from console 106, 296
    - string 106, 296
    - translate 120
    - translation 120
    - unset 650
    - yes/no reply 753
  - line-length 298
    - get 298
  - menu display 439
  - output
    - character 493
    - line-length 313
    - Output
      - to console 493
    - page heading 632
    - page size 313
    - page wait 115
    - page wait 467
    - page-length 298
      - get 298
    - position cursor 80
    - test if 369
    - test if PC Term 371
    - test if VGA 374
    - video attribute 128, 330
  - Console I/O functions 16
  - Console name
    - get 701
  - Constants
    - math 35
  - Convert
    - character to ASCII 636
    - character to lowercase 636
    - character to uppercase 636
    - console input 115
  - Copy
    - far memory buffer contents
      - 216–217
      - reverse sequence 217
    - memory buffer contents 94,
      - 433–434, 618
      - reverse sequence 434
    - region of window 681, 744
    - string 255, 597–598, 607
    - string to end of string 255, 596,
      - 598
    - window from string 733
    - window to string 733
  - Cosecant 132
  - Cosine 121
  - Cotangent 122
  - CRC16
    - generate 126
  - CRC32
    - generate 126
  - Create
    - direct-access file 182
    - directory 442
    - file 127
    - indexed-access file 182
    - keyed-access file 182
    - library 421
    - network socket 60
    - new network socket 579

- CREATE environment variable 127, 182
  - \_CTYPE\_NO\_MACRO 363, 636
  - Cursor
    - current position in window 704
    - hide/show 135
    - positioning 80
    - shape 135
  - Cursor-control codes 128
- 
- D**
- Data conversion
    - absolute value of complex 103
    - absolute value of float 176
    - absolute value of integer 57–58
    - absolute value of long integer 380
    - Base64 to binary 88
    - Base64 to long 55
    - BASIC string to string 91
    - BCD to IEEE 93
    - binary to Base64 88
    - C string to BASIC string 101
    - float to string 160, 184, 263, 266
    - fractional portion 448
    - host-byte order to network-byte order 334
    - IEEE to BCD 93
    - integer portion 361, 448
    - integer to string 375, 379, 415
    - integer to 3-byte integer 415
    - integer to 2-byte integer 415
    - long integer to machine independent 570
    - long integer to 3-byte integer 415
    - machine independent to long integer 570
    - mult-byte character to wide-character 430
    - multi-byte string to wide-character string 430
    - network "dotted" to network-byte order 342
    - network-byte order to host-byte order 455
    - network-byte order to network "dotted" 342
    - numeric string to double 613
    - numeric string to float 85
    - numeric string to integer 85
    - numeric string to long integer 85, 613
    - numeric string to unsigned long integer 613
    - string to lowercase 597
    - string to uppercase 600
    - string transformation 615
    - 3-byte integer to long 102
    - 3-byte integer to long integer 378
  - time structure to formatted string
  - time structure to string 73
  - time structure to time value 446
  - time value to string 134
  - time value to time structure 329, 393
  - transpose bits of integer 619
  - 2-byte integer to long 337
  - unsigned long to string 654
  - wide-character string to multi-byte string 684
  - wide-character to multi-byte 684
- Data conversion functions 17
- Data segment
  - get selector 268
- Date
  - day number 610
  - month name 610
    - abbreviated 610
  - weekday name 610
    - abbreviated 610
- Date and time
  - compute difference between two time values 149
  - convert time structure to string 73
  - convert time structure to time value 446
  - convert time value to string 134
  - convert time value to time structure 329, 393
  - convert UTC to local 393
  - current 134
  - current UTC 329
  - determine if leap year 384
  - formatted 610
  - formatted by DATEFORM 610
  - formatted in locale 610
  - get current 635
  - get current date 277
  - get current time 124, 321
  - get current time, in milliseconds 306
  - set daylight savings time flag 647
  - set time zone 647
  - standard format 610
- Date and time functions 46
- DATEFORM environment variable 277, 610
- Dates
  - formats 562
- Day number 610
- Daylight savings time 446
- Deallocate
  - fifo buffer 197

- 
- type-ahead buffer 624
  - user type-ahead buffer 624
  - Decimal-point character 562
  - Decrement memory functions 138
  - Delay processing
    - See: Wait
  - Delete file 163, 532, 651
  - Denominator, greatest common 265
  - Detach device-driver 142
  - Device control 351
  - Device name 769
    - get 417
  - Device names file
    - analyze and parse 143
    - close 143
    - open 143
    - read record 143
  - Device names file functions 32
  - Device-driver
    - attach 87
    - close 148
    - detach 87, 142
    - initialization 625
    - open 148
    - reattach 87
  - Device-driver buffers 624
  - Device-driver specific functions 19
  - Diagnostic message, display 78
  - Dictionary compare 216, 433
  - Difference
    - between two time values 149
  - Direction flag, clear or set 593
  - Directory
    - create 442
    - erase 537
  - Directory and library maintenance
    - functions 21
  - Directory search
    - deinitialize 150
    - get next 150
    - get next matching entry 210
    - initialize 150
    - initialize and find 210
  - Directory search functions 21
  - Disconnect
    - workstation 642
  - Disk
    - change 153, 449
    - formatting functions 231–232
    - get capacity 154
    - get label 327
    - synchronize cache 620
  - Disk management functions 22
  - Display
    - all windows 730
    - on console 123, 125
    - remove window 729
    - window 728
  - DMA
    - release resource 529
    - reserve resource 529
  - Duplicate
    - open file pointer 158
    - string 597
- 
- E**
- End-of-file
    - test 162, 188
  - Environment
    - account definition 513
  - Environment access functions 23
  - Environment variables
    - CREATE 127, 182
    - DATEFORM 277, 610
    - system-defined 280
    - TZ 647
  - Environments variable
    - get 280
    - put 495
  - Erase
    - directory 537
    - file 163, 532, 651
  - Error handling and exiting functions 24
  - Error message
    - diagnostic 78
    - display 166, 203, 233, 472
    - display at bottom 164
    - network 641
  - Error status
    - clear 109, 233
  - Errors
    - disk full 109
    - end-of-file 109
    - network 641
    - test file status 189, 203, 233, 472
  - EXEC stack 119
    - input 115, 296
  - EXEC variable access functions 33
  - EXEC variables
    - get 282
    - put 282
  - Execute
    - another program 170
    - force other use 227
    - program and return 623
    - program with exit 133
  - Execution
    - suspend 140, 469, 577
  - Existence
    - test file 62
  - Exit program 56, 173, 621
    - continuing in another 170
    - functions execute at 83
    - remove functions execute at 83
    - starting another 133
  - Exponent power of 2 248, 383
  - Exponential value 175

**F**

- Far pointer 473
  - create from offset and segment 443
  - get memory segment of 443
  - get offset of 443
- Far strings
  - compare two 255
    - case-insensitive 256
  - compare two for equality 256
    - case-insensitive 256
  - compare two, with numbers 256
  - concatenate 255
  - copy 255
  - copy to end 255
  - find end of 256
  - initialize to character 256
  - length of 256
  - search for character 255–256
  - search for characters in 255–256
  - search for characters not in 256
  - search for delimited substrings 257
  - search for substring 257
    - case-insensitive 256
- See also:* Strings
- FCB 187, 222, 459
  - access 180
  - get file 150
- FDB
  - get 315
  - locate 395
- Fifo buffers 13, 197, 624
  - add item 197
  - allocate 197
  - check for item 197
  - deallocate 197
  - get item 197
- File
  - change date
    - get 285
  - create 127
  - date and time
    - change 497
  - description
    - full 405
  - fine 395
  - load 387
  - test access permissions 62
  - test for existence 62
  - unload 387
- File Directory Entry
  - See:* FDB
- File handling functions 25–29
- File name
  - change 534
  - character test 362
  - flat file

- get 286
- get full 288
- normalized 453
- test wild card 634
- unique 444, 628

**Files**

- allocation 199
- appending to 223
- change date
  - get 285
- changing
  - date and time 200
  - file size 200
  - growth factor 200
  - key length 200
  - ownership 200
  - protection codes 200
  - record length 200
- close 111, 158, 178
- close all 133, 178
- creating 182
- current I/O position
  - get 194, 261, 627
  - set 250, 252
    - relative to offset 413, 546
  - set to start 536
- delete 163, 532, 651
- device number
  - get 279
- direct-access
  - automatic record locking 224
  - delete record 381
  - opening 223
  - read 409
  - read next 409
  - record length 182
- duplicate open file pointer 158
- erase 163, 532, 651
- FCB access 180
- FCB, changing 180
- growth factor
  - set 195
- handle 207
- I/O buffer
  - allocation 177, 558
  - flushing 214
  - flushing all 214
  - free 178
- I/O buffer allocation 558
- indexed-access
  - automatic record locking 224
  - creating 182
  - delete record 141
  - key length 182
  - opening 223
  - read 515
  - read next 515
  - record length 182
  - write 732

- initialization
  - close 344
  - locate and read integer item 345
  - locate and read item 344
  - locate and read next item 345
  - open 344
  - write integer item 345
  - write item 345
- keyed-access
  - automatic record locking 224
  - creating 182
  - delete record 141
  - key length 182
  - opening 223
  - read 515
  - read next 515
  - read previous 515
  - record length 182
  - write 732
- locking
  - automatic record 224
- locking region of 204
- open 628, 638
- open pipe 475
- opening 182, 187, 222, 459
- pipe
  - open 478
  - opening 224
- printing to 234, 484, 667
- read byte 190–191
- read character 190–191
- read fields 243
- read formatted input 249, 540
  - format specifications 540
- read integer 190–191
- read line 190, 312, 511
- read string 190, 312, 511
- record-locking 396, 522
- record-unlocking 522, 653
- reopening 246
- stream
  - opening 223
- SYSTEM.HELP32 294, 461
- SYSTEM.MENU32 294, 461
- SYSTEM.TEOS32.AC-
  - COUNT 208, 270, 513
- SYS-
  - TEM.TEOS32.DEVN
    - AMES 87, 143
  - analyze and parse 143
  - close 143
  - open 143
  - read record 143
- SYSTEM.TEOS32.KEY-
  - WORD 232, 292, 294, 376, 389, 630, 753
- SYSTEM.TEOS32.MES-
  - SAGE 164, 166, 231, 233, 294, 307, 452, 472, 498, 609–610, 632
- temporary 178
- temporary name 444, 628
- test end-of-file 162, 188
- test error status 189, 203, 233, 472
- test if console 369
- test if serial device 367
- unread byte 649
- unread character 649
- write byte 236–237
- write character 236–237
- write fields 264
- write integer 236, 501
- write string 236, 500, 731
- Float
  - convert from numeric string 85
  - convert to string 160, 184, 263, 266
- Floor 213
  - See also:* Ceiling
- Force
  - TWS to execute program 642
  - user to execute program 227
- Format disk functions 231–232
- Format specifications
  - printf 484
- Formatted
  - date and time
  - output to console 123
- Formatted input
  - reading 249, 540
    - format specifications 540
  - reading from string 592
- Formatted output
  - format specifications 484
  - printing 234, 484
  - printing to file 667
  - printing to stdout 667
  - printing to string 590, 667
- Free
  - allocated memory 244
  - allocated system memory 301
  - memory selector 245
  - system memory
    - shared 571
- Functions
  - \_absd 58**
  - \_absw 58**
  - \_addb 67**
  - \_addcsb 67**
  - \_addcsd 67**
  - \_addcsw 67**
  - \_addd 67**
  - \_addw 67**
  - \_alloca 70**
  - \_andb 71**
  - \_andcsb 71**
  - \_andcsd 71**

- \_andcsw **71**
- \_andd **71**
- \_andw **71**
- \_asm **77**
- \_atexit\_clear **83**
- \_atexit\_remove **83**
- \_attach **87, 767**
- \_baud2cd **92**
- \_cd2baud **92**
- \_cld **593**
- \_cli **350**
- \_con\_tran **120, 625**
- \_conrdy **119**
- \_decb **138**
- \_deccsb **138**
- \_deccsd **138**
- \_deccsw **138**
- \_decd **138**
- \_decnucb **138**
- \_decnucd **138**
- \_decnucw **138**
- \_decw **138**
- \_detach **87, 142, 767**
- \_devopen **148**
- \_df\_close **767**
- \_df\_fdb **767**
- \_df\_open **767**
- \_df\_read **767**
- \_df\_ucb **767**
- \_dir\_mount **153, 449**
- \_dirfdb **150–151, 285, 767**
- \_dirucb **150, 767**
- \_disable **350**
- \_disk\_capacity **154, 767**
- \_dup **159**
- \_enable **350**
- \_file\_alloc **199**
- \_filechange **200**
- \_find\_first **767**
- \_find\_next **767**
- \_fmemccpy **216**
- \_fmemchr **216**
- \_fmemcmp **216**
- \_fmemcpy **216, 303, 404, 438**
- \_fmemdcmp **216**
- \_fmemeq **216**
- \_fmemicmp **216**
- \_fmemieq **216**
- \_fmemmove **216**
- \_fmemrcpy **216**
- \_fmemset **216**
- \_force **227–228**
- \_FP\_OFF **443**
- \_FP\_SEG **443**
- \_free\_sel **245**
- \_fstreat **255**
- \_fstrchr **255**
- \_fstremp **255**
- \_fstrepy **255**
- \_fstrcspn **255**
- \_fstrdcmp **255**
- \_fstrend **255**
- \_fstreq **255**
- \_fstricmp **255**
- \_fstrieq **255**
- \_fstristr **255**
- \_fstrlen **255**
- \_fstrpbrk **255**
- \_fstrrchr **255**
- \_fstrset **255**
- \_fstrspn **255**
- \_fstrstr **255**
- \_fstrtok **255**
- \_get\_help\_filename **289, 461–462**
- \_get\_menu\_filename **289, 461–462**
- \_get\_sect **285, 315–316**
- \_getar **268, 423**
- \_getcs **268**
- \_getcsa **268**
- \_getds **268**
- \_geteax **268**
- \_getebp **268**
- \_getflag **268**
- \_getldt **268**
- \_getlimit **268, 438, 571**
- \_getline **296**
- \_getmem **244–245, 301, 303, 432**
- \_getmsw **268**
- \_getosver **202, 308, 326**
- \_getpid **268**
- \_getss **268**
- \_gettib **268**
- \_inb **340**
- \_incb **338**
- \_inccsb **338**
- \_inccsd **338**
- \_inccsw **338**
- \_incd **338**
- \_incnucb **338**
- \_incnucd **338**
- \_incnucw **338**
- \_incrmem **341, 432, 437**
- \_incw **338**
- \_ind **340**
- \_insb **340**
- \_insd **340**
- \_insw **340**
- \_inw **340**
- \_ioctl **351**
- \_ioctl\_ucb **351, 767**
- \_leapyear **384**
- \_lockres **59, 398, 438, 571**
- \_lockset **59, 398, 438, 571**
- \_lockshare **398, 571, 767**
- \_locktest **398, 571, 767**

- 
- `_lockwait` **398**, 571, 767
  - `_logon` **403–404**
  - `_lseek` 546
  - `_lubname` 272, 279, 367, 369, **417**
  - `_make_alias` 245, **420**
  - `_make_sel` 245, **423**, 648
  - `_makelib` 183, **421**
  - `_maxd` **429**
  - `_maxw` **429**
  - `_mem_grow` 432, **437–438**
  - `_memcpy` **433**
  - `_memset` **433**
  - `_mind` **429**
  - `_minw` **429**
  - `_MK_FP` 303, 438, **443**
  - `_mklib` 183, **421**
  - `_mount_fs` 153, **449**, 767
  - `_movsb` **450**
  - `_movsd` **450**
  - `_movsw` **450**
  - `_norm_fn` 286, 406, **453–454**
  - `_orb` **463**
  - `_orcsb` **463**
  - `_orcsd` **463**
  - `_orcsw` **463**
  - `_ord` **463**
  - `_orw` **463**
  - `_osmajor` **465**
  - `_osminor` **465**
  - `_osversion` **465**
  - `_outb` **466**
  - `_outd` **466**
  - `_outsb` **466**
  - `_outsd` **466**
  - `_outsw` **466**
  - `_outw` **466**
  - `_peekb` **470**, 553–554
  - `_peekcsb` **470**
  - `_peekcsd` **470**
  - `_peekcsp` **470**
  - `_peekcsw` **470**
  - `_peekd` **470**
  - `_peeknucb` **470**
  - `_peeknucd` **470**
  - `_peeknucp` **470**
  - `_peeknucw` **470**
  - `_peekp` **470**
  - `_peekscrb` **470**
  - `_peekscrd` **470**
  - `_peekscrp` **470**
  - `_peekscrw` **470**
  - `_peekw` **470**
  - `_phy_addr` **473**
  - `_PhyAddr` **473–474**
  - `_pokeb` **476**
  - `_pokecsb` **476**
  - `_pokecsd` **476**
  - `_pokecsp` **476**
  - `_pokecsw` **476**
  - `_poked` **476**
  - `_pokenucb` **476**
  - `_pokenucd` **476**
  - `_pokenucp` **476**
  - `_pokenucw` **476**
  - `_pokep` **476**
  - `_pokescrib` **476**
  - `_pokescrd` **476**
  - `_pokescrp` **476**
  - `_pokescrw` **476**
  - `_pokew` **476**
  - `_popf` **481**
  - `_pre_empty` **755**
  - `_pushf` **481**
  - `_putenv` **495**
  - `_putmem` 244–245, **301**, 303, 432
  - `_quitoff` **507**
  - `_quiton` **507**
  - `_remote` **530**
  - `_restoreints` **539**
  - `_rsvmem` 473, **538**
  - `_saveints` **539**
  - `_seek` **546**
  - `_set_slice` **566**
  - `_setprty` **566**
  - `_snu` **755**
  - `_std` **593**
  - `_sti` **350**
  - `_stosb` **594**
  - `_stosd` **594**
  - `_stosw` **594**
  - `_strcpy` **607**
  - `_strend` **607**
  - `_strlen` **607**
  - `_subb` **616**
  - `_subcsb` **616**
  - `_subcsd` **616**
  - `_subcsw` **616**
  - `_subd` **616**
  - `_subw` **616**
  - `_sync` **620**
  - `_tell` **627**
  - `_tmpnam_drive` **628**
  - `_tolower` **636**
  - `_toupper` **636**
  - `_uncache` **648**
  - `_unwait` **669**
  - `_wait` **669**
  - `_write` 90
  - `_xorb` **751**
  - `_xorcsb` **751**
  - `_xorcsd` **751**
  - `_xorcsw` **751**
  - `_xord` **751**
  - `_xorw` **751**
  - `_yesno` 389, **753–754**
  - `_yesnoall` **753**

- `_yield` **755**
- `a64l` **55**, 88
- `abort` **56**, 78, 83–84
- `abs` **57**
- `acc_access` **59**, 767
- `acc_maint` **59**, 767
- `accept` **60**, 318, 385, 548, 582
- `access` **62–63**, 165, 167–168, 622
- `acos` **64**, 168
- `acot` **65**
- `acsc` **66**, 168
- `alarm` **69**, 140, 408, 451, 508, 552, 573, 577
- `asctime` **73–74**, 134
- `asec` **75**, 168
- `asin` **76**, 168
- `assert` **78–79**
- `at` **80–81**, 107, 125, 129, 131, 273, 323, 505–506
- `atan` **82**
- `atan2` **82**, 168
- `atexit` **56**, **83–84**, 171, 173, 712
- `atof` **85–86**, 175, 562
- `atoi` **85–86**, 168, 196
- `atol` **85–86**, 168
- `atoll` **85**
- `atstr` **80–81**
- `base64_decode` **88**, 90
- `base64_encode` **88–89**
- `Basic2C` **91**
- `BasicL2C` **91**
- `bcd2ieee` **93**
- `bcmp` **94**
- `bcopy` **94**
- `bell_oneshot` **457**
- `bind` **95**, 318, 568, 581
- `bininsert` **97**, 99
- `bsearch` 97–99, 505
- `bzero` **94**, 581, 583
- `C2Basic` **101**
- `C2BasicL` **101**
- `c3tol` **102**, 416
- `cabs` **103**
- `calloc` **104**, 244, 424, 428, 518–519
- `ceil` **105**
- `cgets` **106–107**, 125, 190–191
- `chdir` **108**, 168
- `clearerr` **109**, 168, 189
- `clock` **110**, 140, 577
- `close` **111**, 162, 512
- `closesocket` **112**, 568, 579, 582–583
- `ClrBypass` **560**
- `cmd_abbrev` **114**
- `conmask` 106–107, **115–116**, 125, 240, 242, 273, 494, 503, 716–717
- `connect` 95, **117**, 318, 579, 583
- `conrdy` **119**, 716–717
- `cos` **121**, 168
- `cosh` **121**
- `cot` **122**, 168
- `coth` **122**
- `cprintf` **123**, 129, 131, 193, 369, 494
- `cpu_time` **124**, 509–510
- `cputs` 81, 107, 119, **125**, 129, 236, 273, 494
- `crc16` **126**
- `crc32` **126**
- `creat` **127**
- `crt` 81, **128–129**, 131, 239–242, 441, 468, 505, 528, 713–714
- `crt_oneshot` **457**
- `crtcolor` 107, 125, **130–131**, 331
- `csc` **132**
- `csch` **132**
- `csi` **133**, 387, 623
- `ctime` **134**, 285, 647
- `CursorBlock` **135**
- `CursorHide` **135**
- `CursorLine` **135**
- `CursorShow` **135**
- `cuserid` **136**, 299, 324
- `delay` **140**, 577
- `deletem` **141**, 224, 523
- `devnames_close` **143**, 147
- `devnames_open` **143**, 146
- `devnames_parse` **143**, 146
- `devnames_read` **143**, 146
- `difftime` **149**
- `dirclose` **150**, 152, 210, 589, 767
- `directory search` 634
- `diropen` **150–151**, 168, 210, 424, 589, 767
- `dirread` **150–151**, 210, 589, 767
- `div` **156–157**
- `dup` **158**, 168
- `dup2` **158**
- `ecvt` **160–161**, 184
- `eof` **162**
- `erase` **163**, 165, 651
- `errmsg` 104, **166–167**, 519, 622
- `execl` **170**, 172, 387, 424, 588
- `execlp` **170**, 172, 387, 424, 588
- `execv` **170**, 172, 387, 424
- `execvp` **170**, 172, 387, 424
- `exit` 83, 90, 93, 99, 104, 108, 171, **173–175**, 177, 179, 181, 194, 196, 207, 215, 226, 230, 247, 251, 253, 262, 279, 283, 305, 308, 323, 328, 348, 367, 369, 377, 387, 389, 414, 425, 460, 468, 472, 480, 503, 512, 517, 537, 553, 559,



- 
- 581–584, 606, 621, 631,  
633, 683, 727, 734–735,  
754
  - exp **175**
  - fabs **176**
  - fbuf **177**–178, 424
  - fclose 93, 99, 109, 136, 168, **178**,  
181, 188, 193, 196, 203,  
206–207, 235, 238, 246,  
262, 299, 324, 328, 369,  
402, 462, 468, 480, 517,  
606, 633, 735
  - fcloseall 56, 83, 89–90, 168, 173,  
**178**–179, 226, 279, 376,  
452
  - fcntl **180**–181, 732
  - fcreate 168, **182**
  - fcvt 160, **184**–185
  - FD\_CLR **186**
  - FD\_ISSET **186**
  - FD\_SET **186**
  - FD\_ZERO **186**
  - fdopen **187**
  - feof **188**, 190–191, 262, 312,  
328, 468
  - ferror 109, **189**, 312
  - fflush **214**–215, 414, 547
  - fgetc 89, 168, **190**, 192, 312, 424
  - fgetl **190**, 193
  - fgetpos 168, **194**, 252–253
  - fgets 90, **190**, 193, 206, 262, 312,  
468, 480, 584, 606, 633
  - fgetsn **190**, 312
  - fgetw **190**, 193
  - fgrow **195**–196
  - fifo\_alloc **197**
  - fifo\_free **197**
  - fifo\_get **197**
  - fifo\_put **197**
  - fifo\_rdy **197**
  - file\_err **203**
  - filelock 191, **204**, 206, 234, 237,  
396, 511, 522, 667
  - fileno **207**, 475
  - find\_acc **208**–209, 271, 767
  - find\_first **210**–211, 634
  - find\_next **210**–211, 634
  - floor **213**
  - floppy\_oneshot **457**
  - flush **214**, 588
  - flushall **214**, 588
  - fmod **220**–221
  - fopen 89–90, 93, 99, 109, 136,  
168, 177, 179, 181–182,  
188, 192, 194, 196, 203,  
206–207, 215, **222**, 226,  
233, 235, 238, 246, 251,  
253, 262, 279, 295, 299,  
324, 328, 333, 346, 367,  
369, 402, 424, 461, 468,  
472, 517, 559, 606, 628,  
631, 633, 735
  - fork **229**–230, 305, 480, 553
  - forktask **229**
  - formask **231**
  - formclear **231**
  - formincr **231**
  - formparm **231**
  - FP\_OFF **443**
  - FP\_SEG **443**
  - ferror 63, 93, 99, 109, 127, 151,  
177, 179, 181–182, 194,  
196, 203, 207, 215, 222,  
226, **233**, 246–247, 251,  
253, 262, 279, 328, 333,  
367, 369, 414, 460, 462,  
468, 475, 478, 480, 512,  
535, 559, 606, 631, 633,  
638, 735
  - fprintf 108, 136, 203, **234**–235,  
299, 324, 333, 402, 581–  
584, 667, 734–735
  - fputc 109, 168, **236**, 238, 424
  - fputl **236**, 238
  - fputs 89, 159, **236**, 238, 468,  
480, 633, 683
  - fputsn **236**
  - fputsnl **236**
  - fputw **236**, 238
  - fread 188, **243**, 251, 511–512,  
735
  - free 90, 104, **244**, 275, 424, 518,  
720
  - free\_oneshot **457**
  - free\_sel **245**
  - freopen 159, **246**–247
  - frexp **248**
  - fscanf **249**
  - fseek 206, 223, **250**–251, 262,  
413
  - fsetpos **252**–253, 414, 547
  - fsign **254**
  - ftell 168, **261**–262, 627
  - ftoa **263**
  - fwrite **264**, 731, 735
  - gcd **265**
  - gcvl **266**–267
  - get\_acc\_name **270**–271, 503,  
767
  - get\_oneshot **457**
  - getbaud **272**
  - getc **190**
  - getch 106–107, 119, 125, 190,  
**273**, 467, 494, 650, 716
  - getchar 116, 153, **190**, 240, 449,  
720
  - getclass 146, **274**, 372
  - getcwd 168, **275**–276, 424

- getdate **277–278**
- getdev **279, 367, 369**
- getenv *155, 280–281, 295, 424, 496, 513, 612*
- getexvar **282–283**
- getftime 168, **285**
- getfiledate **285**
- getflat **286, 405–406**
- getfn **288, 633**
- gethostbyaddr **290**
- gethostbyname **290, 583**
- gethostname 290–**291**
- getkey **292–293, 376, 630**
- getlang **294**
- getlib **295**
- getll *146, 298, 632, 716, 735*
- getlogin 136, *146, 299, 310, 324, 402*
- getlub **300**
- getmpid **304–305**
- getmsec 278, **306**
- getmsg 168, **307, 452**
- getpeername **309**
- getpid **304–305, 582**
- getpl *81, 146, 298, 468, 632, 713–716*
- getppid **304–305**
- getpriv **310**
- getprotobyname **311**
- getprotobynumber **311**
- gets **63, 99, 114, 165, 167, 203, 281, 283, 312, 408, 499, 503, 505, 517, 528, 622, 637**
- GetScreenSize **313–314, 709**
- getservbyname **317**
- getservbyport **317**
- getsockname 96, **318**
- getsockopt **319, 569**
- gettime 278, 306, **321**
- GetUCB **322–323**
- getucb *153, 322–323, 449*
- getuid *136, 146, 299, 324, 402, 503*
- getver **325–326**
- getvol **327**
- getw **328**
- gmtime 73, 285, **329, 393, 446, 612**
- h\_errno **641**
- has **330–331**
- hascolor *131, 330–331*
- HasMouse **332**
- hasprt **333**
- htonl **334, 581**
- htons **334, 581, 583**
- hypot **335–336**
- i2tol **337**
- ieee2bcd **93**
- inet\_addr **342, 583**
- inet\_ntoa **342**
- InitFileClose **344, 349, 767**
- InitFileOpen **344, 348, 767**
- InitFileRead **344, 348, 767**
- InitFileReadInt **344, 767**
- InitFileReadNext **344, 767**
- InitFileWrite **344, 767**
- InitFileWriteInt **344, 767**
- in-line macro 363, 636
  - \_CTYPE\_NO\_MACRO 363, 636
- intoff **350, 539**
- inton **350, 539**
- ioctlsocket **360**
- ip *221, 361*
- is character 562, 564
- IsActive **366, 373, 696, 742**
- isalnum **362, 364**
- isalpha **362, 364, 564, 637**
- isascii **362, 364, 564**
- isatty **367, 369**
- IsBypass **368**
- isctrl *116, 362, 364*
- iscon *159, 369, 584*
- iscsym **362, 364**
- iscsymnum **362, 364**
- isdigit **362, 364**
- isdisp **362, 364, 564**
- isfnsym **362, 364**
- isgraph **362, 364, 494**
- ishex **362, 364**
- islower *241, 362, 364*
- isocal **362, 364**
- ispcterm **371**
- isprint *240, 362, 364*
- isprtnum **372**
- ispunct **362, 364**
- IsSession **373, 696**
- isspace **362, 364, 576**
- issymbol **362, 364**
- issymnum **362, 364**
- isupper *241, 362, 364*
- isvga **374**
- isxdigit **362, 364**
- itoa *146, 375*
- keyclose *292–293, 376, 630–631*
- killtask *230, 305, 377, 582*
- l3tol **378, 416**
- l64a 88, **379**
- labs **380**
- ldeletek *224, 381, 523*
- ldexp **383**
- ldiv **156–157**
- lfind **411–412**
- listen 61, 95, **385, 582**
- lltoa **415**
- load **387–388**
- load\_yn *215, 389, 499, 753–754*

- localeconv **390–391**, 562
- localtime 73–74, 145, 285, **393–394**, 447, 612, 647
- locate 285, **395**
- lockf 168, 205, **396**
- locking 168, 205, **396**
- log 168, **400–401**
- log10 168, **400–401**
- log2 168, **400**
- logname **402**
- longfname **405–406**, 633
- longjmp **407–408**, 561
- lreadk 93, 224, **409**, 523
- lreadn 224, **409**, 523
- lsearch **411**
- lseek **413**, 512
- ltoa **415**
- ltoc3 102, 378, **415**
- ltol3 **415**
- ltol3 378, **415**
- lwritek 93, 224, **418**, 523
- make\_alias 245, **420**
- make\_sel 245, **423**, 648
- makelib 183, **421–422**
- malloc 90, 104, 244, 275, **424–425**, 428, 518, 597, 606, 640, 714, 720, 734
- matcharg 293, 376, **426**
- max **427**
- max\_alloc **428**
- mblen **430**
- mbstowcs **430**
- mbtowc **430**
- mem\_avail **432**
- memcpy **433**
- memchr **433**
- memcmp 94, **433**, 517
- memcpy 94, 99, 433
- memdcmp **433**
- memeq **433**
- memicmp **433**
- memieq **433**
- memmove **433**
- memrcpy **433**
- memset 94, 404, **433**, 747
- menu **439**, 441
- menu2 **439**, 441
- min **427**, 716
- MK\_FP **443**
- mkdir 168, 183, **442**
- mktemp **444**
- mktime 145, 329, 393–394, **446–447**, 647
- modf **448**
- msalarm 69, 140, **451**, 508, 552, 573, 577
- msgclose 307, **452**
- mvga\_oneshot **457**
- ntohl **455**
- ntohs **455**
- offsetof **456**
- oneshot **457**
- open 162, 168, 414, **459–460**, 512, 606
- openhlp 289, **461–462**
- openmenu 289, **461–462**
- pagewait 81, **467–468**, 632–633, 709, 714, 727, 735
- pause **469**
- perror 108, 276, 442, **472**, 537
- pipe 224, **475**
- popen 224, 475, **478**, 480
- pow 168, **482**
- prime **483**
- primel **483**
- printf 57, 63, 74, 79, 86, 99, 109–110, 114, 123–124, 134, 142, 146, 149, 152–153, 155, 157, 161–162, 165, 167, 171–172, 175–176, 181, 185, 188, 194, 196, 206–207, 209, 211, 215, 220, 230, 234, 239–241, 262, 267, 271–272, 274, 276, 278–279, 281, 283, 286, 288, 294–295, 298, 305, 308, 310, 316, 323, 326, 328, 331, 336, 348–349, 364, 367, 369, 372, 380, 384, 389, 391, 394, 401, 406, 408, 412, 447, 449, 454, 462, **484**, 489, 491, 503, 505, 510, 512, 514, 517, 528, 533, 535, 543, 554, 562, 565, 584, 590–591, 606, 609, 612, 622–623, 632, 637, 652, 656, 667–668, 675, 696, 709, 715–716, 727, 734, 742, 754
- putc **236**
- putch 107, 125, 128, 236, 273, 467, **493–494**
- putchar 84, 151, 203, 210, **236**, 239–242, 323, 715
- putenv **495–496**, 612
- putxvar **282–283**
- putfddate 168, **497**
- putmsg 164, 166, **498–499**
- puts 56, 84, 90, 99, 159, 174, 323, **500**
- putw **501**
- pwverify **502–503**, 767
- qsort 99, 424, **504–505**, 586
- raise **508**
- rand **509–510**
- read 162, 414, 424, **511–512**
- read\_acc 424, **513–514**, 767

- readk 224, **515**, 517, 523
- readn 224, **515**, 517, 523
- readp 224, **515**, 523
- realloc 104, 244, 424, **518–519**, 589
- reclock 191, 234, 237, **522**, 653, 667
- recunlock **522**
- recv 117, **520**, 548, 579–581, 584
- recvfrom **520**, 548, 579
- regcmp **524**, 528
- regex **524**, 528
- release\_dma **529**, 544, 767
- remove 163, 167–168, **532–533**, 622, 651
- rename 168, **534–535**
- reserve\_dma **529**, 544, 767
- rewind 146, 223, 262, 468, **536**
- rmdir 168, **537**
- rsema **550**
- rsemaname **550**, 553
- rsemares **550**
- rsemaset **550**, 554
- rsemawait **550**
- scanf 57, 176, 220, 249, 272, 336, 380, 384, 514, **540**, 543, 562, 591–592
- sch\_acc **513–514**, 767
- schedint **544**
- scsi\_oneshot **457**
- sec **545**
- sech **545**
- seek **546**
- select 117, 186, **548**
- sema **550**, 554
- semaphore 377, 480, **550**, 553–554
- semares 480, **550**
- semaset 480, **550**, 553
- semawait 377, 480, **550**, 553
- send 117, 548, **555**, 579–581, 584
- sendto 548, **555**, 579
- setbuf 177–178, **558**
- SetBypass **560**
- setjmp 407–408, **561**
- setlocale **562**, 636
- SetScreenSize **313–314**
- setsockopt 95, 319, 548, **567**
- setvbuf 177–178, 215, 424, **558–559**
- sgetl **570**
- shared 432, 438, 553–554, **571**
- signal 69, 140, 168, 407–408, 451, 507–508, 552, **573–574**, 577
- sin 168, **575**
- sinh **575**
- sizeof 99, 104, 190, 303, 412, **456**, 512, 519, 581–584, 589, 720, 735, 747
- skipsp **576**
- sleep 140, 206, **577**, 666, 742
- sleep\_msec **577**
- sleep\_sec **577**
- sleep\_until **577**
- socket 95, 385, **579**, 581, 583
- sort **586**, 589
- spawnl 168, 424, **587**
- spawnlp 168, 424, **587**
- spawnv 168, 424, **587**
- spawnvp 168, 424, **587**
- sprintf 146, 263, **590**, 667
- sputl **570**
- sqrt 103, 168, **591**
- srand **509–510**
- sscanf 422, **592**
- strcat 81, 146, 211, 326, 496, **596**, 612
- strchr **596**
- strcmp 99, 146, 172, 411, 504–505, 589, **596**, 615
- strcoll 562–563, **608**, 615
- strcpy 81, 100, 136, 146, 151, 155, 161, 185, 196, 211, 239–240, 242, 283, 288, 299, 307, 402, 452, 460, 462, 505, 514, 533, **596**, 609, 612, 631, 633, 652
- strcspn **596**
- strdcmp **596**
- strdup 424, 589, **596**
- strend **596**
- streq **596**, 630
- strerror 222, 246, 475, 478, **609**, 638
- strftime 73, 134, 146, 394, 447, 486, 562, 565, **610**, 612, 632, 647
- stricmp **596**
- stricq **596**
- stristr **596**
- strlen 99, 239–241, 517, 584, **596**, 604
- strlwr **596**
- strmake 209, 271, **596**
- strmatch **596**
- strncat **596**
- strncmp **596**
- strncpy 517, **596**
- strneq **596**
- strnicmp **596**
- strnset **596**
- strpbrk **596**
- strrchr **596**
- strset **596**
- strspn **596**
- strstr **596**

- strtod **613**
- strtok **596**, 606
- strtol **613**
- strtoul **613**
- strtrim 404, **596**
- strupr **596**
- strxfrm 562–563, **615**
- swab **618**
- swapbits **619**
- syserr 203, 244, 293, 300, 376, 422, 426, 432, 519, 606, **621–622**, 631, 633–634, 666, 748
- system 133, 229, 387, 561, **623**
- ta\_alloc **624**
- ta\_free **624**
- ta\_get **624**
- ta\_put **624**
- ta\_rdy **624**
- tan **626**
- tanh **626**
- tell 414, **627**
- tempnam 424, **628**
- testarg 426, **630–631**
- testhead **632–633**
- testwild **633–634**
- time 74, 134, 145, 149, 306, 394, 447, 612, **635**
- timer **550**, 554
- tmpfile 424, **628**
- tmpnam **628**
- toascii **636**
- tolower 218, 241, 435, 564, **636**
- topen 168, 424, **638**
- tot\_alloc **640**
- toupper 242, 564, **636–637**
- TSAGetLastError 60, 95, 112, 117, 290–291, 309, 311, 317–319, 360, 385, 521, 549, 556, 567, 579–584, **641**
- TSASetLastError **641**
- TSAStrError 581–584, **641**
- TWS\_disconnect **642**
- TWS\_disconnect\_now **642**
- TWS\_execute **642**
- TWS\_focus **642**
- TWS\_maximize **642**
- TWS\_minimize **642**
- TWS\_ontop **642**
- TWS\_receive **642**
- TWS\_restore **642**
- TWS\_send **642**
- TWS\_title **642**
- TWS\_user\_focus **642**
- tzset 329, 393, 446, 612, 635, **647**
- ub\_alloc **624**
- ub\_free **624**
- uncache **648**
- ungetc 250, 536, **649**
- ungetch 119, **650**
- unlink 168, **651–652**
- unload **387–388**
- unlock 205, 224, 523, **653**
- utoa **654**
- va\_alist **655**
- va\_arg **655–656**, 667
- va\_dcl **655**
- va\_end **655–656**, 667–668
- va\_start **655–656**, 667–668
- vdi **657**
- vdialpha **657**
- vdicarc **657**
- vdibar **657**
- vdicircle **657**
- vdiclear **657**
- vdiclose **657**, 666
- vdigraph **657**
- vdiline **657**
- vdimark **657**
- vdioopen **657**, 666
- vdipie **657**, 666
- vdisetfc **657**, 666
- vdisetfs **657**, 666
- vdisetlc **657**, 666
- vdisetlh **657**
- vdisetls **657**
- vdisetmc **657**
- vdisetmh **657**
- vdisetms **657**
- vdisetta **657**
- vdisettc **657**
- vdisetth **657**
- vdissetp **657**
- vdissets **657**
- vditext **657**
- vfprintf **667**
- vprintf **667–668**
- vsprintf **667**
- wChoice 424, **670**, 675
- wChoiceFuncKeys **670**
- wChoiceSelect **670**, 675
- wChoiceTimeout **670**, 675
- wClear **676**
- wClip 675, **677**, 727, 739
- wClose 675, **678**, 709, 717, 735
- wCloseAll **678**, 727, 735
- wColor 441, 675–676, **679–680**, 694, 703, 708–709, 716, 724, 734–735, 739
- wCopy **681**
- wCount 675, **683**, 721
- wcstombs **684**
- wcstrlen **684**
- wctomb **684**
- wEdit 135, 424, **685**
- wEnter **688**

wEnterFirst **688**  
 wFinish *441, 675, 678, 691*  
 wFrame *441, 675–676, 680, 692, 694, 703, 708–709, 716–717, 724, 734–735, 739*  
 wGetActive **695, 742**  
 wGetSess *366, 696, 742*  
 wGetStat **697**  
 wGetStr **698**  
 wGetTitle **699**  
 wGetWin **700, 708–709**  
 WhoAmI *145, 701*  
 wInvert *676, 680, 694, 702–703, 716, 724, 739*  
 wLocate **704**  
 wMenuBar *424, 705, 708*  
 wMouseDisable **710, 714, 717**  
 wMouseEnable *332, 710, 713, 716*  
 wMouseHit **710, 717**  
 wMouseRead **710, 714**  
 wMouseState **710, 716**  
 wMove **718, 739**  
 wOnKey **719–720**  
 wOpen *441, 675–676, 680, 694, 703, 708–709, 716, 721, 724, 727, 734, 747*  
 wOrder **725, 727**  
 wRefresh *676, 717, 728*  
 wRemove **729**  
 wRepaint *727, 730*  
 wRestore **733, 735**  
 write **731**  
 writek *224, 523, 732*  
 wSave **733, 735**  
 wSaveSize **733–734**  
 wScroll **736**  
 wSelect *441, 708–709, 716, 721, 726–727, 734–735, 737*  
 wStatus *735–736, 739*  
 wSwitch *373, 741–742*  
 wSwitchTo *366, 741–743*  
 wTake *681, 744*  
 wTitle *675, 716, 739, 746–747*  
 wVer **748**  
 xclose **749**  
 xcreat **749**  
 xdup **749**  
 xfdopen *187, 749*  
 xfileno *207, 749*  
 xflush **749**  
 xfopen *207, 749*  
 xlocking **749**  
 xlseek **749**  
 xopen *168, 749*  
 xread **749**  
 xtell **749**  
 xwrite **749**

yesno *167, 215, 389, 499, 622, 753–754*  
 yield **755**

---

**G**

Generate  
     checksum *126*  
 Get  
     account number *324*  
     active session number *695*  
     baud rate *272*  
     class code *274*  
     code segment alias selector *268*  
     code segment selector *268*  
     console line-length *298, 313*  
     console name *701*  
     console page-length *298, 313*  
     console UCB *701*  
     current date *277*  
     current date and time *134, 635*  
     current I/O position *194, 261, 627*  
     current time *124, 306, 321*  
     current window number *700*  
     data segment selector *268*  
     default library *295*  
     device name *417*  
     EAX register *268*  
     EBP register *268*  
     environment variable *280*  
     EXEC variable *282*  
     FDB *315*  
     file's change date *285*  
     file's device number *279*  
     file's full path specification *288*  
     flag register *268*  
     flat file description *286*  
     help file  
         full specification *289*  
     keyword text *292*  
     language code *294*  
     last mouse event *711*  
     LDT segment alias selector *268*  
     local host name *291*  
     locale settings *390, 562*  
     login name *299, 402*  
     machine-status-word *268*  
     main process id *304*  
     memory segment access rights *268*  
     memory segment limit *268*  
     menu file  
         full specification *289*  
     message text *307*  
     mouse status *711*  
     network address *701*  
     network client type *701*  
     network error code  
         Error status  
             network *641*

- network error message
  - Error status
    - network 641
- network host name 290
- network peer name 309
- network service name 317
- network socket name 318
- network socket options 319
- OS name 325
- OS serial 325
- OS version 308, 325
- parent process id 304
- printer line-length 298
- printer LUB 372
- printer page-length 298
- privilege level 310
- process id 268
- process number 701
- program version 325
- session count 701
- session manager version 748
- session number 696, 701
- stack segment selector 268
- status of network sockets 548
- string from window 698
- system memory 301
  - shared 571
- TIB segment alias selector 268
- UCB 300, 322
- version date 325
- window attributes 739
- window display order 725
- window status 739
- window title string 699
- Get item
  - fifo buffer 197
  - type-ahead buffer 624
- Go to
  - saved jump location 407
- Graphics
  - See:* VDI graphics
- Greatest common denominator 265
- Greenwich Mean Time 329
- Growth factor
  - set 195

---

**H**

- Handle
  - file 207
- Header files
  - acb.h 12
  - conio.h 16
  - crt.h 128, 330
  - ctype.h 15
  - direct.h 21
  - diskfind.h 21–22
  - disksize.h 22, 154
  - driver.h 13–14, 19
  - execvar.h 33
  - fcbl.h 180

- fifo.h 13
- format.h 22
- getmem.h 36
- handle.h 29
- initfile.h 31
- io.h 24, 651
- ioctl.h 351
- locale.h 33, 390
- locking.h 29
- lub.h 274, 300, 333
- malloc.h 36
- math.h 34–35, 49
- memory.h 14
- peek.h 39
- process.h 37, 40
- sc.h 19, 21, 23–24, 37, 40, 47
- search.h 41
- signal.h 47, 508, 573
- socket.h 38, 580
- stdarg.h 50
- stdio.h 16, 21, 25, 42, 651
- stdlib.h 17, 19, 23–24, 32, 36
- string.h 43
- time.h 46
- timer.h 46–47
- twsh.h 48
- unistd.h 29
- vdi.h 30
- wmapi.h 51, 689, 739
- Help file
  - full path name 289
  - open 461
- Help file functions 32
- Hide
  - cursor 135
  - window 737
- Host-byte order
  - convert to network-byte order 334
- Hot-keys
  - choice list 670–671
  - during window edit 685, 689
  - during window menu bar 705
  - See also:* On-Key
- Hyperbolic cosecant 132
- Hyperbolic cosine 121
- Hyperbolic cotangent 122
- Hyperbolic secant 545
- Hyperbolic sine 575
- Hyperbolic tangent 626
- Hypotenuse of right triangle 335

---

**I**

- IEEE
  - convert to BCD 93
- Increment memory functions 338
- Initialization
  - memory 104
- Initialization file
  - close 344

- open 344
  - read integer item 345
  - read item 344
  - read next item 345
  - write integer item 345
  - write item 345
  - Initialize
    - yesno function 389
  - Input
    - from console 115, 273, 753
    - unget 650
  - Input port functions 340
  - Integer
    - convert from numeric string 85
    - convert to string 375, 379, 415
    - convert to 3-byte integer 415
    - convert to 2-byte 415
  - Integer portion 361
  - Interrupts
    - assign ISR 544
    - awaken ISR 669
    - disable/enable 350
    - save/restore status 539
    - software 69, 451, 508, 573
    - wait for interrupt 669
- 
- J**    Jump location
  - return to 407
  - set 561
- 
- K**    Keyword
  - abbreviation 292, 426, 630
  - test 630
- Keyword file functions 32
- Keywords
  - all 389
  - go 389
  - no 389
  - yes 389
- Keywords file
  - close 376
  - get keyword 292
  - open 292
- 
- L**    Language code
  - get 294
- Larger value 427
  - series of integers 429
- LDT segment
  - get alias selector 268
- Leap year
  - test 384
- Library
  - create 421
  - get default 295
- Line-drawing codes 128
- Line-length
  - console 298, 313
- Listen
  - incoming network connection 385
- Load
  - file or program 387
- Local time
  - convert from UTC 393
- Locale
  - "is" character types 562
  - change 562
  - collating sequence 562
  - date and time format 610
  - date format 562
  - decimal-point character 562
  - get current settings 390, 562
- Locate
  - character in far memory buffer 216
  - character in memory buffer 433
  - file 395
- Lock
  - memory location 398
    - release 398
  - memory location, shared 398
  - test memory location 398
  - wait for memory location 398
- Locking
  - record
    - automatic 224
    - region of file 204
- Logarithm, base-2 400
- Logarithm, common 400
- Logarithm, natural 175, 400
- Login name
  - get 136, 299, 324, 402
- Logoff command 59
- Logon command 59
- Long integer
  - convert to machine independent 570
  - convert to 3-byte integer 415
- Lowercase
  - convert to 636
  - string 597
  - test for 363
- Lub
  - get printer 372
- 
- M**    Machine independent
  - convert to long integer 570
- Macros, in-line 363, 636
- Mantissa 248, 383
- Math constants 35
- Math functions 34
  - \_maxd 429
  - \_maxw 429
  - \_mind 429
  - \_minw 429
  - abs 57
  - acos 64



- 
- acot 65
  - acsc 66
  - asec 75
  - asin 76
  - atan 82
  - atan2 82
  - cabs 103
  - ceil 105
  - cos 121
  - cosh 121
  - cot 122
  - coth 122
  - csc 132
  - csch 132
  - div 156
  - exp 175
  - fabs 176
  - floor 213
  - fmod 220
  - frexp 248
  - fsign 254
  - gcd 265
  - hypot 335
  - ip 361
  - labs 380
  - ldexp 383
  - ldiv 156
  - log 400
  - log10 400
  - log2 400
  - max 427
  - min 427
  - modf 448
  - pow 482
  - prime 483
  - primel 483
  - rand 509
  - sec 545
  - sech 545
  - sin 575
  - sinh 575
  - sqrt 591
  - srand 509
  - tan 626
  - tanh 626
  - Maximum value 427, 429
  - Memory
    - buffers 14
    - lock location 398
    - lock location, shared 398
    - test lock on location 398
    - unlock location 398
    - wait lock on location 398
  - Memory allocation 70, 104, 301, 424
    - changing size of 437, 518
    - currently allocated 640
    - maximum 428, 432
    - release 244, 301
    - shared 571
    - freeing 571
  - Memory compaction 473
    - reserve from 538
  - Memory management functions 36
  - Memory segment
    - offset 473
    - selector 473
  - Memory selector
    - create 423
    - create alias 420
    - free 245
  - Menu
    - display 439
    - selection 439
    - window bar 705
    - window choice 670
      - hot-keys 671
      - input timeout 671
  - Menu file
    - full path name 289
    - open 461
  - Menu file functions 32
  - Message
    - display system-defined 472
    - display system-defined text 609, 621
    - display user-defined 498
    - get system-defined text 307
  - Message file
    - close 452
    - get message 307
    - open 307
  - Message file functions 32
  - MIME 88
  - Minimum value 427, 429
  - Modulo function 220, 448
  - Month name 610
  - Mouse
    - buttons 711
    - disable 710
    - during window edit 687
    - enable 710
    - get last event 711
    - last event 710
    - status 711
    - test if last event in specified window 710
    - test if present 332
  - Mouse functions 51
  - Move
    - window position 718
  - Move functions 450
  - Multi-byte character
    - convert to wide-character 430
  - Multi-byte string
    - convert to wide-character string 430
    - length of 430
  - Multipurpose internet mail exten-

sions  
  *See: MIME*  
Multitask  
  get main process id 304  
  get parent process id 304  
  semaphore  
    catalogue 550–551  
    clear 550–551  
    set 550–551  
    testing 550–551  
    timer 551  
    wait 550–551  
  starting subtask 587  
  stopping all 133, 623  
  stopping subtask 377  
Multitasking functions 37

---

**N** Name  
  account 12  
Named pipe 224  
Network address  
  get 701  
Network client type  
  get 701  
Network-byte order  
  convert to host-byte order 455  
Networking  
  *See: Sockets*  
Normalize  
  file name 453  
Numeric string  
  convert to double 613  
  convert to long integer 613  
  convert to unsigned long integer  
    613

---

**O** Offset of  
  member within structure 456  
On-Key  
  disable trapping 719  
  enable trapping 719  
On-Key functions 51  
Open  
  device names file 143  
  file 182, 187, 222, 459  
  help file 461  
  initialization file 344  
  menu file 461  
  pipe 478  
  tape file 638  
  temporary file 628  
  VDI graphics 660  
  window 721  
Operating system  
  get serial 325  
  get version 308, 325  
  version variables 465  
Or functions 463  
Output port functions 466

---

**P** Page heading  
  test 632  
Page size  
  console 298, 313  
Page wait 467  
  suppressing 115  
Parent task  
  *See: Multitask*  
Password  
  verify 502  
Path  
  full specification 405  
  help file 289  
  menu file 289  
  get full specification 288  
  normalize 453  
PC term  
  test 371  
Peek and poke memory functions 39  
Peek memory functions 470  
Physical address 473  
  compute 473  
pid  
  *See: Process id*  
Pipe  
  open 475, 478  
  opening 224  
Poke memory functions 476  
Pop flag register 481  
Power function 482  
Prime number, compute 483  
Print  
  to console 123, 125  
Printer  
  class code  
    Class code  
      get 274  
  line-length  
    get 298  
  output  
    page heading 632  
    page-length  
      get 298  
    test attribute 333  
Printer names 372  
printf  
  format specifications 484  
Printing  
  formatted output 234, 484  
    format specifications 484  
  formatted output to file 667  
  formatted output to stdout 667  
  formatted output to string 590,  
    667  
Priority  
  default 566  
  set 566  
Privilege level

- 
- get 310
    - Process control functions 40
    - Process id
      - get 268
    - Process number
      - get 701
    - Processor time used 110
    - Program
      - load 387
      - unload 387
    - Program execution
      - started as subtask 587
    - Program version
      - get 325
    - Program version date
      - get 325
    - Prompt
      - yes/no 753
    - Prompt text
      - choice-list 670
      - during window menu bar 705
    - Pseudorandom number 509
      - seed 509
    - Push flag register 481
    - Put
      - environment variable 495
      - EXEC variable 282
- 
- Q**
- Quick sort
    - array of strings 504
  - Quotient function 156
- 
- R**
- Random number 509
    - seed 509
  - Read
    - byte 190–191
    - character 190–191
    - device names file record 143
    - direct-access record 409
    - fields 243
    - formatted input 249, 540
      - format specifications 540
    - formatted input from string 592
    - incoming network data 520
    - indexed-access record 515
    - initialization file integer item
      - 345
    - initialization file item 344
    - initialization file next item 345
    - integer 190–191
    - keyed-access record 515
    - line 190, 312, 511
    - next direct-access record 409
    - next indexed-access record 515
    - next keyed-access record 515
    - previous keyed-access record
      - 515
    - string 190, 312, 511
  - Record
    - direct-access
      - delete 381
      - read 409
      - read next 409
    - indexed-access
      - delete 141
      - read 515
      - read next 515
      - write 732
    - keyed-access
      - delete 141
      - read 515
      - read next 515
      - read previous 515
      - write 732
    - locking 396, 522
    - unlocking 522, 653
  - Refresh
    - all windows 730
    - window display 728, 737
    - window sequence 726
  - Regular expressions 524–527
    - anchors 525
    - character sets 526
    - control characters 525
    - literal characters 524
    - metacharacters 525
    - repeat counts 527
    - searching with 524
    - wild cards 526
  - Remainder function 156, 220
  - Remote function 530
  - Rename file 534
  - Reserving memory 473
  - Return code 56, 173, 621
  - Rewind
    - to start of file 536
- 
- S**
- Scancodes 371
  - Screen-editing codes 128
  - Scroll bar
    - during window edit 687
    - update 736
  - Search
    - array of sorted strings 97–98
    - array of unsorted strings 411
    - character in string 255–256, 596, 599
    - characters in string 255–256, 597, 599
    - characters not in string 256, 599
    - delimited substrings 257, 599
    - disk directory 150, 210
    - file 395
    - string with regular expression
      - 524
    - substring 257, 599
      - case-insensitive 256, 597
  - Searching and sorting functions 41

- Secant 545
- Seek
  - relative to offset 413, 546
- Select
  - window number 737
- Select next user 755
- Semaphore
  - catalogue 550–551
  - clear 550–551
  - set 550–551
  - test 550–551
  - timer 551
  - wait 550–551
- Session
  - enable/disable switching 741
  - get active number 695
  - get count 701
  - get number 701
  - get session number 696
  - get version 748
  - switch to specific 741
- Session management functions 51
- Sessions
  - clear bypass mode 560
  - set bypass mode 560
  - test if bypass mode 368
  - test if program session active 366
  - test if session-switching capable 373
- Session-switching
  - disable 560, 741
  - enable 560, 741
  - test if capable 373
- Set
  - active session number 741
  - color 130
  - console line-length 313
  - console page-length 313
  - current I/O position 250, 252, 413, 536, 546
  - jump location 561
  - network error code
    - Error status
      - network 641
  - network socket options 567
  - priority 566
  - semaphore 550–551
  - time slice 566
  - video output 128
  - window display order 725
  - window invert status 702
- Show
  - cursor 135
- Sign
  - of floating-point value 254
- Signal
  - alarm 69, 451, 508
  - processing 573
  - raise 508
  - timer 69, 451
- Sine 575
- Size of data item 456
- Skip
  - white-space characters 576
- Smaller value 427
  - series of integers 429
- Sockets
  - accept new connection 60
  - add socket to array 186
  - bind name to socket 95
  - clear blocking mode 360
  - close socket 112
  - connect to socket 117
  - convert network address 334, 342, 455
  - create new 60
  - create new socket 579
  - get error code 641
  - get error message 641
  - get host name 290
  - get local host name 291
  - get peer name 309
  - get service name 317
  - get socket name 318
  - get socket options 319
  - get status of sockets 548
  - listen for incoming connection 385
  - read incoming data 520
  - remove all sockets 186
  - remove socket 186
  - send data 555
  - set blocking mode 360
  - set error code 641
  - set socket options 567
  - test socket existence in array 186
- Sort
  - array of pointers to strings 586
  - array of strings 504
- Spawning
  - subtask 587
- Square root 591
- Stack 70, 104, 244, 424, 428, 640
  - get segment selector 268
- Standard file access functions 25
- Standard I/O functions 42
- Starting
  - subtask
    - stderr 56, 78, 166, 171, 233, 246, 472, 482, 498, 621
    - stdin 190–191, 246, 312, 369, 536, 540–542, 650
    - stdout 234, 236–237, 246, 369, 484, 486–487, 490, 500, 632, 667
- Stopping
  - all subtasks 623

---

- subtask 377
- subtasks, all 133
- Store memory functions 594
- String
  - write 236
- String and memory manipulation functions 43
- Strings
  - add to array of strings 411
  - collating sequence 562
  - compare
    - using locale 608
  - compare to mask 598
  - compare two 597–599
    - case-insensitive 597, 599
  - compare two for equality 597
    - case-insensitive 597
  - compare two, with numbers 597
  - concatenate 596, 598
  - convert to lowercase 597
  - convert to uppercase 600
  - copy 597–598, 607
  - copy to end 596, 598
  - display and edit 688
  - display in window 685
  - duplicate copy 597
  - edit in window 685
  - find end of 597, 607
  - formatted date and time 610
  - get current date 277
  - get current time 306, 321
  - initialize to character 599
  - length of 597, 607
  - printing formatted output to 667
  - printing to 590
  - read formatted input from 592
  - search array of sorted 97–98
  - search array of unsorted 411
  - search for character 596, 599
  - search for characters in 597, 599
  - search for characters not in 599
  - search for delimited substrings 599
  - search for substring 599
    - case-insensitive 597
  - search with regular expression 524
  - skip white-space characters 576
  - sort array of 504, 586
  - transform 615
  - trim white-space characters 600
  - write 236, 500, 731
  - See also:* Far strings
- Structures
  - WHOAMI 701
- Subtract memory functions 616
- Suspend
  - execution 140, 469, 577
- System memory

- allocating from 301, 437
- freeing allocated memory 301
- maximum available 432
- shared
  - allocating from 571
  - freeing 571
- System time
  - get 124, 635

---

**T**

- Tangent 626
- Tape files
  - open 638
- Task, multiple
  - See:* Multitask
- TCP/IP
  - See:* Sockets
- TCP/IP networking functions 38
- Temporary
  - file name 444, 628
- Temporary files 178
- Terminate execution 56, 173, 621
- Test
  - alphabetic character 362
  - alphanumeric character 362
  - ASCII character 362
  - character ready (console) 119
  - console capabilities 330
  - console PC Term 371
  - console VGA 374
  - control character 362
  - decimal digit character 362
  - displayable character 362–363
  - file name character 362
  - for serial device 367
  - hexadecimal digit character 363
  - if active session 366
  - if mouse present 332
  - if window open 697
  - last mouse event 710
  - leap year 384
  - lock on memory location 398
  - lowercase character 363
  - octal digit character 363
  - output page heading 632
  - punctuation character 363
  - semaphore 550–551
  - session bypass mode 368
  - session-switching capable 373
  - symbol character 362–363
  - uppercase character 363
  - white-space character 363
  - wild card 634
- Text
  - input 106, 273, 296, 753
    - unset 650
  - output 493
- Text-clipping
  - in window 677
- THEOS WorkStation

*See*: TWS  
3-byte  
    convert to long integer 378  
3-byte integer  
    convert to long 102  
TIB segment  
    get alias selector 268  
Time  
    get system 124, 635  
    processor time used 110  
    since midnight 124  
Time and date  
    *See*: Date and time  
Time and date functions 46  
Time slice  
    default 566  
    release 755  
    set 566  
Time structure  
    convert to string 73  
    convert to time value 446  
    daylight savings time 446  
    normalize 446  
Time value  
    convert to string 134  
    convert to time structure 329,  
        393  
Timer  
    semaphore 551  
Trigonometric functions 49  
    acos 64  
    acot 65  
    acsc 66  
    asec 75  
    asin 76  
    atan 82  
    atan2 82  
    cos 121  
    cosh 121  
    cot 122  
    coth 122  
    csc 132  
    csch 132  
    hypot 335  
    log 400  
    log10 400  
    log2 400  
    sec 545  
    sech 545  
    sin 575  
    sinh 575  
    tan 626  
    tanh 626  
2-byte integer  
    convert to long 337  
TWS  
    activates focus of program's win-  
        dow 643  
    disable user disconnect 642

    disable user window switching  
        645  
    disable window on-top feature  
        643  
    disconnect workstation 642  
    enable user disconnect 642  
    enable user window switching  
        645  
    enable window on-top feature  
        643  
    force execution of program 642  
    maximizes program's window  
        643  
    minimize program's window 643  
    receive file from workstation  
        643  
    restores program's window 645  
    send file to workstation 645  
    set window title 645  
TWS control functions 48  
Type-ahead  
    buffers 13  
Type-ahead buffers 624  
    add item 624  
    allocate 624  
    check for item 624  
    deallocate 624  
    get item 624  
Type-ahead buffers, user  
    allocate 624  
    deallocate 624  
TZ environment variable 647

---

## U

UCB  
    console 701  
    get 300  
    get file 150  
    get pointer 322  
Unique  
    file name 444, 628  
Universal Time Coordinate 329  
Unix-style file access functions 29  
Unload  
    file or program 387  
Unread  
    byte 649  
    character 649  
Unsigned long  
    convert to string 654  
Uppercase  
    convert to 636  
    string 600  
    test for 363  
UTC time  
    convert to local 393

---

## V

Variable argument management  
    655  
Variable-argument management

---

functions 50  
 Variables  
   \_errarg 109, 150, 163, **168**, 222,  
     233, 475, 609, 638  
   \_errbot **164**  
   \_errnum 109, 150, 163, **168**,  
     178, 182, 222, 233, 243,  
     264, 346, 460–461, 475,  
     478, 534, 638  
   \_osmajor 465  
   \_osminor 465  
   \_osversion 465  
   daylight 647  
   errno 62, 64, 76, 85, 87, 108,  
     111, 121, 127, 163, **168**,  
     175, 194, 207, 222, 233,  
     237, 243, 246, 250, 252,  
     261, 264, 275, 285, 335,  
     346, 383, 397, 400, 413,  
     442, 460, 472, 475, 478,  
     482, 497, 511, 532, 534,  
     537, 573, 588, 591, 638,  
     651–652  
   EXEC  
     get 282  
     put 282  
   get environment 280  
   put environment 495  
   timezone 647  
   tzname 647  
 VDI graphics  
   clear page 659  
   close 659  
   draw  
     arc 658  
     bar 659  
     circle 659  
     line 659  
     mark 660  
     pie slice 660  
     text 663  
   eject page 659  
   general interface  
   open 660  
   set  
     fill color 660  
     fill style 660  
     graphics mode 659  
     line color 661  
     line height 661  
     line style 661  
     marker color 661  
     marker size 661  
     marker style 661  
     text angle 661  
     text color 662  
     text mode 658  
     text path angle 662  
     text size 662

    text style 663  
 VDI graphics functions 30  
 Version  
   operating system 308  
   session manager 748  
 VGA  
   test 374  
 Video attribute  
   setting 128  
 Virtual screen 722  
 Volume label 327

---

## W

Wait  
   for elapsed time 140, 469, 577  
   for interrupt 669  
   for semaphore 550–551  
   for time of day 577  
   lock on memory location 398  
 Weekday name 610  
 White-space  
   remove leading and trailing 600  
   skip 576  
 WhoAmI structure 701  
 Wide-character  
   compute length required 684  
   convert to multi-byte 684  
 Wide-character string  
   convert to multi-byte string 684  
 Wild card  
   test 634  
 Window  
   active 737  
   attributes 739  
   choice 670, 705  
     hot-keys 671  
     input timeout 671  
   clear to character 676  
   close 678  
   close all 691  
   color 679  
     title 746  
   compute save size 733  
   copy from string 733  
   copy region 681, 744  
   copy to string 733  
   current cursor position 704  
   display order 725  
   edit string array 685  
   editing string 688  
     first character action 688  
   frame style 692  
   get current number 700  
   get string 698  
   get title text 699  
   hidden 737  
   menu bar 705  
   move 718  
   next available number 683  
   number in use 683

- on top 737
- open 721
- position 721
- refresh 728, 737
- refresh all 730
- refresh sequence 726
- remove 729
- remove region 744
- scroll bar 687, 736
- set invert status 702
- shadow style 692
- size 298, 721
- size limits 722
- status 739
- test if mouse event in 710
- test if open 697
- text-clipping 677
- title text 746
- virtual screen 722

Window management functions 51

Working directory

- change 108
- get current 275

Write

- byte 236–237
- character 236–237
- fields 264
- indexed-access record 732
- initialization file integer item  
345
- initialization file item 345
- integer 236, 501
- keyed-access record 732
- network data 555
- new-line 236
- string 236, 500, 731

---

**X**    Xor memory functions 751

---

**Y**    Yes/no  
      prompt 753