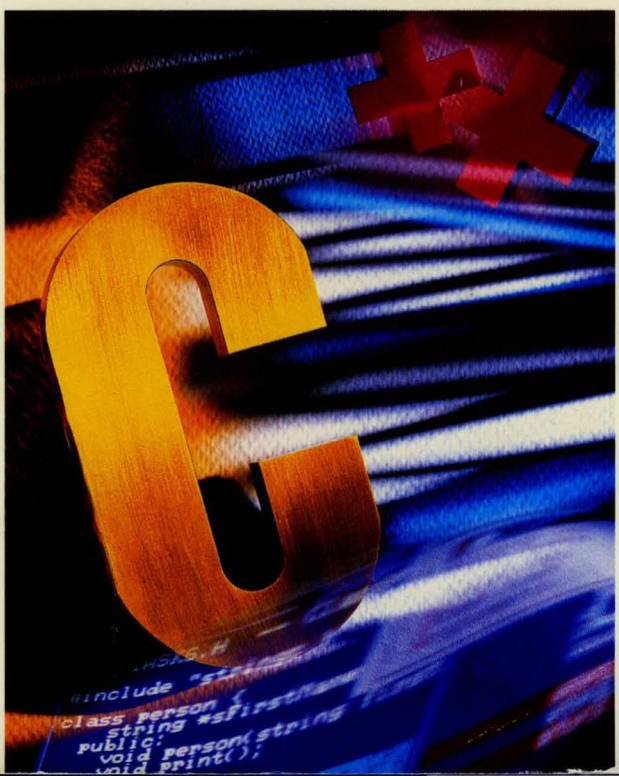


2.0

BORLAND[®] C++

LIBRARY
REFERENCE

B O R L A N D



Borland[®] C++
Version 2.0

Library Reference

Copyright © 1991 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders. Windows, as used in this manual, refers to Microsoft's implementation of a windows system.

C O N T E N T S

Introduction	1	bioscom	35
Class and member function		biosdisk	37
documentation	1	biosequip	40
Chapter 1 The main function	3	bioskey	41
Arguments to main	3	biosmemory	43
An example program	4	biosprint	44
Wildcard arguments	5	biostime	45
An example program	6	brk	46
Using -p (Pascal calling conventions)	7	bsearch	47
The value main returns	7	cabs	48
		calloc	49
Chapter 2 The run-time library	9	ceil	51
How to use function entries	10	cgets	51
abort	11	chdir	52
abs	11	_chmod	54
absread	12	chmod	55
abswrite	14	chsize	57
access	14	circle	58
acos	16	_clear87	59
allocmem	17	cleardevice	60
arc	18	clearerr	62
arg	19	clearviewport	63
asctime	20	clock	64
asin	21	_close	65
assert	22	close	66
atan	23	closegraph	67
atan2	24	creol	68
atexit	25	clrscr	69
atof	26	complex	69
atoi	27	conj	71
atol	28	_control87	71
bar	29	coreleft	72
bar3d	30	cos	73
bcd	32	cosh	74
bdos	33	country	75
bdosptr	34	cprintf	76

cputs	77	feof	132
_creat	78	ferror	133
creat	79	fflush	134
creatnew	81	fgetc	135
creattemp	82	fgetchar	136
cscanf	84	fgetpos	137
ctime	85	fgets	138
ctrlbrk	86	filelength	139
delay	87	fileno	140
delline	87	lillellipse	140
detectgraph	88	fillpoly	142
difftime	92	findfirst	143
disable	92	findnext	145
div	94	floodfill	147
dosexterr	95	floor	148
dostounix	96	flushall	149
drawpoly	97	_fmemccpy	150
dup	98	_fmemchr	150
dup2	100	_fmemcmp	150
ecvt	101	_fmemcpy	150
ellipse	102	_fmemicmp	150
__emit__	104	_fmemset	150
enable	105	fmod	151
eof	106	fnmerge	151
execl, execl, execlp, execlpe, execv, execve,		fnsplit	153
execvp, execvpe	107	fopen	155
_exit	113	FP_OFF	156
exit	114	_fpreset	157
exp	115	fprintf	158
fabs	116	FP_SEG	159
farcalloc	116	fputc	160
farcoreleft	118	fputchar	160
farfree	118	fputs	161
farheapcheck	119	fread	162
farheapcheckfree	120	free	163
farheapchecknode	122	freemem	164
farheapfillfree	123	freopen	165
farheapwalk	124	frexp	166
farmalloc	125	fscanf	167
farrealloc	127	fseek	168
fclose	128	fsetpos	169
fcloseall	129	fstat	170
fcvt	130	_fstrcat	172
fdopen	131	_fstrchr	172

_fstrcspn	173	getfillsettings	201
_fstrdup	173	getftime	204
_fstricmp	173	getgraphmode	205
_fstrlen	173	getimage	206
_fstrlwr	173	getlinesettings	208
_fstrncat	173	getmaxcolor	210
_fstrncmp	173	getmaxmode	212
_fstrncpy	174	getmaxx	213
_fstrnicmp	174	getmaxy	214
_fstrnset	174	getmodename	216
_fstrpbrk	174	getmoderange	217
_fstrchr	174	getpalette	218
_fstrrev	174	getpalettesize	220
_fstrset	174	getpass	221
_fstrspn	175	getpid	222
_fstrstr	175	getpixel	223
_fstrtok	175	getpsp	224
_fstrupr	175	gets	225
ftell	175	gettext	226
ftime	176	gettextinfo	227
fwrite	177	gettextsettings	228
gcvt	178	gettime	230
geninterrupt	179	getvect	231
getarccoords	180	getverify	232
getaspectratio	182	getviewsettings	233
getbkcolor	183	getw	235
getc	185	getx	236
getcbrk	185	gety	237
getch	186	gmtime	238
getchar	187	gotoxy	240
getche	187	graphdefaults	240
getcolor	188	grapherrormsg	242
getcurdir	190	_graphfreemem	243
getcwd	190	_graphgetmem	245
getdate	191	graphresult	247
getdefaultpalette	192	harderr	249
getdfree	193	hardresume	251
getdisk	194	hardretn	254
getdrivername	195	heapcheck	256
getdta	196	heapcheckfree	257
getenv	197	heapchecknode	258
getfat	198	heapfillfree	259
getfatd	199	heapwalk	261
getfillpattern	200	highvideo	262

hypot	262	lowvideo	309
imag	263	_lrotl	310
imagesize	264	_lrotr	311
initgraph	266	lsearch	311
inport	270	lseek	313
inportb	270	ltoa	314
inline	271	malloc	315
installuserdriver	272	matherr	316
installuserfont	274	max	319
int86	276	memccpy, _fmemccpy	319
int86x	277	memchr, _fmemchr	320
intdos	278	memcmp, _fmemcmp	321
intdosx	279	memcpy, _fmemcpy	322
intr	280	memicmp, _fmemicmp	323
ioctl	282	memmove	324
isalnum	283	memset, _fmemset	324
isalpha	284	min	325
isascii	285	mkdir	326
isatty	285	MK_FP	327
iscntrl	286	mktemp	327
isdigit	287	mktime	328
isgraph	288	modf	329
islower	288	movedata	330
isprint	289	moverel	331
ispunct	290	movetext	332
isspace	290	moveto	333
isupper	291	movmem	334
isxdigit	292	norm	335
itoa	292	normvideo	336
kbhit	293	nosound	337
keep	294	_open	337
labs	295	open	339
ldexp	296	outport	341
ldiv	297	outportb	341
lfind	298	outtext	342
line	299	outtextxy	344
linereel	300	_OvrInitEms	345
lineto	301	_OvrInitExt	346
localeconv	302	parsfnm	347
localtime	303	peek	348
lock	305	peekb	349
log	306	perror	350
log10	307	pieslice	351
longjmp	308	poke	353

pokeb	353	setbkcolor	419
polar	354	setblock	421
poly	355	setbuf	422
pow	356	setcbreak	424
pow10	357	setcolor	425
printf	357	_setcursortype	427
putc	366	setdate	428
putch	367	setdisk	429
putchar	368	setdta	429
putenv	369	setfillpattern	431
putimage	370	setfillstyle	432
putpixel	372	setftime	434
puts	374	setgraphbufsize	436
puttext	374	setgraphmode	437
putw	375	setjmp	439
qsort	376	setlinestyle	441
raise	377	setlocale	443
rand	378	setmem	444
randbrd	379	setmode	445
randbwr	381	setpalette	446
random	382	setrgbpalette	448
randomize	383	settextjustify	450
_read	384	settextstyle	452
read	385	settime	454
real	387	setusercharsize	455
realloc	388	setvbuf	457
rectangle	389	setvect	458
registerbgidriver	390	setverify	460
registerbgifont	392	setviewport	461
remove	393	setvisualpage	462
rename	394	setwritemode	464
restorecrtmode	395	signal	465
rewind	397	sin	469
rmdir	398	sinh	470
_rotl	399	sleep	471
_rotr	399	sopen	472
sbrk	400	sound	474
scanf	401	spawnl, spawnle, spawnlp, spawnlpe, spawnv, spawnve, spawnvp, and spawnvpe	475
searchpath	410	sprintf	478
sector	411	sqrt	479
segread	413	srand	480
setactivepage	414	sscanf	481
setallpalette	415		
setaspectratio	418		

stat	482	textheight	523
_status87	483	textmode	524
stime	484	textwidth	526
stpcpy	485	time	527
strcat, _fstrcat	486	tmpfile	528
strchr, _fstrchr	487	tmpnam	529
strcmp	488	toascii	530
strcmpi	489	_tolower	530
strcoll	490	tolower	531
strcpy	491	_toupper	532
strcspn, _fstrcspn	491	toupper	533
strdup, _fstrdup	492	tzset	533
_sterror	493	ultoa	535
strerror	494	ungetc	536
strftime	494	ungetch	537
stricmp, _fstricmp	496	unixtodos	538
strlen, _fstrlen	497	unlink	539
strlwr, _fstrlwr	497	unlock	540
strncat, _fstrncat	498	va_arg, va_end, va_start	541
strncmp, _fstrncmp	499	vfprintf	543
strncmpi	500	vfscanf	544
strncpy, _fstrncpy	501	vprintf	546
strnicmp, _fstrnicmp	501	vscanf	547
strnset, _fstrnset	502	vsprintf	548
strpbrk, _fstrpbrk	503	vsscanf	549
strrchr, _fstrrchr	504	wherex	550
strrev, _fstrrev	504	wherey	551
strset, _fstrset	505	window	551
strspn, _fstrspn	506	_write	552
strstr, _fstrstr	506	write	554
strtod	507	Chapter 3 Global variables	555
strtok, _fstrtok	508	_8087	555
strtol	509	_argc	556
strtol	511	_argv	556
strupr, _fstrupr	511	_ctype	556
strxfrm	512	daylight	556
swab	513	directvideo	557
system	514	environ	557
tan	515	errno, _doserrno, sys_errlist, sys_nerr	558
tanh	516	_fmode	560
tell	516	_heaplen	561
textattr	517	_new_handler	562
textbackground	520	_openfd	562
textcolor	521		

_osmajor, _osminor	563
_ovrbuffer	563
_psp	564
_stklen	564
timezone	565

tzname	565
_version	566
_wscroll	566

Index	567
--------------	-----

T A B L E S

2.1: detectgraph constants	89	2.4: Color palettes	267
2.2: Graphics drivers information	90	2.5: Graphics modes	268
2.3: Graphics drivers constants	267	2.6: Actual color table	416

This manual contains definitions of all the Borland C++ library routines, common variables, and common defined types, along with example program code to illustrate how to use most of these routines, variables, and types.

If you are new to C or C++ programming, or if you are looking for information on the contents of the Borland C++ manuals, see The introduction and Chapter 2, "Navigating the Borland C++ manuals," in *Getting Started*.

Here is a summary of the chapters in this manual:

Chapter 1: The main function discusses arguments to **main** (including wildcard arguments), provides some example programs, and gives some information on Pascal calling conventions and the value that **main** returns.

Chapter 2: The run-time library is an alphabetical reference of all Borland C++ library functions. Each entry gives syntax, include files, an operative description, return values, and portability information for the function, together with a reference list of related functions and examples of how the functions are used.

Chapter 3: Global variables defines and discusses Borland C++'s global variables. You can use these to save yourself a great deal of programming time on commonly needed variables (such as dates, time, error messages, stack size, and so on).

Class and member function documentation

Certain classes and class member functions are incorporated in Chapter 2. Here's a list of the classes and member functions and their page numbers.

The typefaces used in this manual are used as described in Getting Started.

Name	Type	Page number
abs	member function	11
acos	member function	16
arg	member function	19
asin	member function	21
atan	member function	23
bcd	class	32
complex	class	69
conj	member function	71
cos	member function	73
cosh	member function	74
exp	member function	115
imag	member function	263
log	member function	306
log10	member function	307
norm	member function	335
pow	member function	356
polar	member function	354
real	member function	387
sin	member function	469
sinh	member function	470
sqrt	member function	479
tan	member function	515
tanh	member function	516

The main function

Every C and C++ program must have a **main** function; where you place it is a matter of preference. Some programmers place **main** at the beginning of the file, others at the end. Regardless of its location, the following points about **main** always apply.

Arguments to main

Three parameters (arguments) are passed to **main** by the Borland C++ startup routine: *argc*, *argv*, and *env*.

- *argc*, an integer, is the number of command-line arguments passed to **main**.
- *argv* is an array of pointers to strings (**char *[]**).
 - Under 3.0 and higher versions of DOS, *argv[0]* is the full path name of the program being run.
 - Under versions of DOS before 3.0, *argv[0]* points to the null string ("").
 - *argv[1]* points to the first string typed on the DOS command line after the program name.
 - *argv[2]* points to the second string typed after the program name.
 - *argv[argc-1]* points to the last argument passed to **main**.
 - *argv[argc]* contains null.

- *env* is also an array of pointers to strings. Each element of *env*[] holds a string of the form `ENVVAR=value`.
 - `ENVVAR` is the name of an environment variable, such as `PATH` or `87`.
 - *value* is the value to which `ENVVAR` is set, such as `C:\DOS;C:\TOOLS;` (for `PATH`) or `YES` (for `87`).

If you declare any of these parameters, you *must* declare them exactly in the order given: *argc*, *argv*, *env*. For example, the following are all valid declarations of `main`'s arguments:

```
main()
main(int argc) /* legal but very unlikely */
main(int argc, char * argv[])
main(int argc, char * argv[], char * env[])
```

The declaration `main(int argc)` is legal, but it's very unlikely that you would use *argc* in your program without also using the elements of *argv*.

The argument *env* is also available through the global variable *environ*. Refer to the *environ* entry in Chapter 3 and the **putenv** and **getenv** lookup entries in Chapter 2 for more information.

argc and *argv* are also available via the global variables `_argc` and `_argv`.

An example program

Here is an example program, `ARGS.EXE`, that demonstrates a simple way of using these arguments passed to `main`.

```
/* Program ARGS.C */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[], char *env[])
{
    int i;

    printf("The value of argc is %d \n\n",argc);
    printf("These are the %d command-line arguments passed to
           main:\n\n",argc);

    for (i = 0; i < argc; i++)
        printf("  argv[%d]: %s\n", i, argv[i]);

    printf("\nThe environment string(s) on this system are:\n\n");
    for (i = 0; env[i] != NULL; i++)
```

```

        printf("  env[%d]: %s\n", i, env[i]);
    return 0;
}

```

Suppose you run ARG5.EXE at the DOS prompt with the following command line:

```
C:> args first_arg "arg with blanks" 3 4 "last but one" stop!
```

Note that you can pass arguments with embedded blanks by surrounding them with double quotes, as shown by "arg with blanks" and "last but one" in this example command line.

The output of ARG5.EXE (assuming that the environment variables are set as shown here) would then be like this:

The value of argc is 7

These are the 7 command-line arguments passed to main:

```

argv[0]: C:\BC\TESTARG5.EXE
argv[1]: first_arg
argv[2]: arg with blanks
argv[3]: 3
argv[4]: 4
argv[5]: last but one
argv[6]: stop!

```

The environment string(s) on this system are:

```

env[0]: COMSPEC=C:\COMMAND.COM
env[1]: PROMPT=$p $g
env[2]: PATH=C:\SPRINT;C:\DOS;C:\BC

```

The maximum combined length of the command-line arguments passed to **main** (including the space between adjacent arguments and the name of the program itself) is 128 characters; this is a DOS limit.

Wildcard arguments

Command-line arguments containing wildcard characters can be expanded to all the matching file names, much the same way DOS expands wildcards when used with commands like COPY. All you have to do to get wildcard expansion is to link your program with the WILDARG5.OBJ object file, which is included with Borland C++.

Once WILDARG5.OBJ is linked into your program code, you can send wildcard arguments of the type *.* to your **main** function. The argument will be expanded (in the *argv* array) to all files

matching the wildcard mask. The maximum size of the *argv* array varies, depending on the amount of memory available in your heap.

If no matching files are found, the argument is passed unchanged. (That is, a string consisting of the wildcard mask is passed to **main**.)

Arguments enclosed in quotes ("...") are not expanded.

An example program

The following commands will compile the file **ARGS.C** and link it with the wildcard expansion module **WILDARGS.OBJ**, then run the resulting executable file **ARGS.EXE**:

```
BCC ARGS WILDARGS.OBJ
ARGS C:\BORLANDC\INCLUDE\*.H "*.C"
```

When you run **ARGS.EXE**, the first argument is expanded to the names of all the ***.H** files in your Borland C++ **INCLUDE** directory. Note that the expanded argument strings include the entire path. The argument ***.C** is not expanded as it is enclosed in quotes.

In the IDE, simply specify a project file (from the project menu) that contains the following lines:

```
ARGS
WILDARGS.OBJ
```

Then use the **Run | Arguments** option to set the command-line parameters.



If you prefer the wildcard expansion to be the default, modify your standard **C?.LIB** library files to have **WILDARGS.OBJ** linked automatically. In order to accomplish that, remove **SETARGV** from the libraries and add **WILDARGS**. The following commands invoke the Turbo librarian (**TLIB**) to modify all the standard library files (assuming the current directory contains the standard **C** and **C++** libraries and **WILDARGS.OBJ**):

*For more on **TLIB**, see Chapter 7, "Utilities" in the User's Guide.*

```
tlib cs -setargv +wildargs
tlib cc -setargv +wildargs
tlib cm -setargv +wildargs
tlib cl -setargv +wildargs
tlib ch -setargv +wildargs
```

Using `-p` (Pascal calling conventions)

If you compile your program using Pascal calling conventions (described in detail in Chapter 9, “Interfacing with assembly language,” in the *Programmer’s Guide*), you *must* remember to explicitly declare `main` as a C type. Do this with the `cdecl` keyword, like this:

```
cdecl main(int argc, char * argv[], char * envp[])
```

The value `main` returns

The value returned by `main` is the status code of the program: an `int`. If, however, your program uses the routine `exit` (or `_exit`) to terminate, the value returned by `main` is the argument passed to the call to `exit` (or to `_exit`).

For example, if your program contains the call

```
exit(1)
```

the status is 1.

If you are using the IDE (as opposed to the command-line compiler) to run your program, you can display the return value from `main` by selecting **File | Get Info**.

The run-time library

All programming examples in this chapter are in the online help system. This means you can easily copy them from help and paste them into your files.

This chapter contains a detailed description of each of the functions in the Borland C++ library. A few of the routines are grouped by “family” (the **exec...** and **spawn...** functions that create, load, and run programs, for example) because they perform similar or related tasks. Otherwise, we have included an individual entry for every routine. For instance, if you want to look up information about the **free** routine, you would look under **free**; there you would find a listing for **free** that

- summarizes what **free** does
- gives the syntax for calling **free**
- tells you which header file(s) contains the prototype for **free**
- gives a detailed description of how **free** is implemented and how it relates to the other memory-allocation routines
- lists other language compilers that include similar functions
- refers you to related Borland C++ functions
- if appropriate, gives an example of how the function is used, or refers you to a function entry where there is an example

The following sample library lookup entry explains how to find out such details about the Borland C++ library functions.

How to use function entries

Function	Summary of what function does.
Syntax	<pre>#include <header.h></pre> <p>This part lists the header file(s) containing the prototype for function or definitions of constants, enumerated types, and so on used by the function.</p> <pre>function(modifier <i>parameter</i>[,...]);</pre> <p>This gives you the declaration syntax for function; parameter names are <i>italicized</i>. The <code>[, ...]</code> indicates that other parameters and their modifiers can follow.</p>
Prototype in	<pre>header.h</pre> <p>This lists the header file(s) containing the prototype for function. The prototype of some functions is contained in more than one header file; in such cases, each file is listed. You can use whichever header file best suits your needs.</p>
Remarks	This section describes what function does, the parameters it takes, and any details you need to use function and the related routines listed.
Return value	The value that function returns (if any) is given here. If function sets the global variable <i>errno</i> , that value is also listed. (See the DOS documentation for the interpretation of <i>errno</i> .)
Portability	The system(s) and language(s) that function is available for are listed here. These can include UNIX, IBM PCs and compatibles, C++, Windows, and ANSI C.
See also	Routines related to function that you might want to read about are listed here. If a routine name contains an <i>ellipsis</i> (funcname... , ...funcname , func...name), it indicates that you should refer to a family of functions (for example, exec... refers to the entire family of exe functions: execl , execle , execlp , execlpe , execv , execve , execvp , and execvpe).
Example	<pre>/*Here you'll find a small sample program showing the use of function (and possibly of related functions).*/</pre>

abort

Function	Abnormally terminates a program.
Syntax	<code>#include <stdlib.h></code> <code>void abort(void);</code>
Prototype in	stdlib.h, process.h
Remarks	abort writes a termination message (“Abnormal program termination”) on stderr, then aborts the program by a call to _exit with exit code 3.
Return value	abort returns the exit code 3 to the parent process or to DOS.
Portability	abort is available on UNIX systems and is defined in ANSI C.
See also	assert, atexit, exit, _exit, raise, signal, spawn...
Example	<pre>#include <stdio.h> #include <stdlib.h> int main(void) { printf("Calling abort()\n"); abort(); return 0; /* This is never reached */ }</pre>

abs

Function	Returns the absolute value of an integer.	
Syntax	<i>Real version:</i> <code>#include <math.h></code> <code>int abs(int x);</code>	<i>Complex version:</i> <code>#include <complex.h></code> <code>double abs (complex x);</code>
Prototype in	<i>Real version:</i> math.h, stdlib.h	<i>Complex version:</i> complex.h
Remarks	<p>abs returns the absolute value of the integer argument <i>x</i>. If abs is called when <code>stdlib.h</code> has been included, it's treated as a macro that expands to inline code.</p> <p>If you want to use the abs function instead of the macro, include <code>#undef abs</code> in your program, after the <code>#include <stdlib.h></code>.</p>	

abs

Return value The real version of **abs** returns an integer in the range of 0 to 32,767, with the exception that an argument of -32,768 is returned as -32,768. The complex version of **abs** returns a **double**.

Portability The real version of **abs** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

See also **cabs, complex, fabs, labs**

Example

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

int main(void)
{
    int number = -1234;
    printf("number: %d absolute value: %d\n", number, abs(number));
    return 0;
}
```

absread

Function Reads absolute disk sectors.

Syntax `#include <dos.h>`
`int absread(int drive, int nsects, int lsect, void *buffer);`

Prototype in dos.h

Remarks **absread** reads specific disk sectors. It ignores the logical structure of a disk and pays no attention to files, FATs, or directories.

absread uses DOS interrupt 0x25 to read specific disk sectors.

drive = drive number to read (0 = A, 1 = B, etc.)

nsects = number of sectors to read

lsect = beginning logical sector number

buffer = memory address where the data is to be read

The number of sectors to read is limited to 64K or the size of the buffer, whichever is smaller.

Return value If it is successful, **absread** returns 0.

On error, the routine returns -1 and sets the global variable *errno* to the value returned by the system call in the AX register.

Portability **absread** is unique to DOS. This function is not Windows compatible.

See also `abswrite`, `biosdisk`

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>

#define SECSIZE 512

unsigned char buf[SECSIZE];

int main(void)
{
    int i, j, sector, drive;
    char str[10];

    printf("Enter drive letter: ");
    gets(str);
    drive = toupper(str[0]) - 'A';

    printf("Enter sector number to read: ");
    gets(str);
    sector = atoi(str);

    if (absread(drive, 1, sector, &buf) != 0)
    {
        perror("Disk error");
        exit(1);
    }

    printf("\nDrive: %c  Sector: %d\n", 'A' + drive, sector);
    for (i = 0; i < SECSIZE; i += 16)
    {
        if ((i / 16) == 20)
        {
            printf("Press any key to continue...");
            getch();
            printf("\n");
        }


        printf("%03d: ", i);
        for (j = 0; j < 16; j++)
            printf("%02X ", buf[i+j]);
        printf("\t");

        for (j = 0; j < 16; j++)
            if (isprint(buf[i+j]))
                printf("%c", buf[i+j]);
            else
                printf(".");
    }
}
```


absread

```
        printf("\n");
    }
    return 0;
}
```

abswrite

Function	<code>#include <io.h></code> Writes absolute disk sectors.
Syntax	<code>#include <dos.h></code> <code>int abswrite(int <i>drive</i>, int <i>nsects</i>, int <i>lsect</i>, void *<i>buffer</i>);</code>
Prototype in	<code>dos.h</code>
Remarks	abswrite writes specific disk sectors. It ignores the logical structure of a disk and pays no attention to files, FATs, or directories.  If used improperly, abswrite can overwrite files, directories, and FATs. abswrite uses DOS interrupt 0x26 to write specific disk sectors. <i>drive</i> = drive number to write to (0 = A, 1 = B, etc.) <i>nsects</i> = number of sectors to write to <i>lsect</i> = beginning logical sector number <i>buffer</i> = memory address where the data is to be written The number of sectors to write to is limited to 64K or the size of the buffer, whichever is smaller.
Return value	If it is successful, abswrite returns 0. On error, the routine returns -1 and sets the global variable <i>errno</i> to the value of the AX register returned by the system call.
Portability	abswrite is unique to DOS. This function is not Windows compatible.
See also	absread , biosdisk

access

Function	Determines accessibility of a file.
Syntax	<code>#include <io.h></code> <code>int access(const char *<i>filename</i>, int <i>amode</i>);</code>
Prototype in	<code>io.h</code>

Remarks `access` checks the file named by *filename* to determine if it exists, and whether it can be read, written to, or executed.

The list of *amode* values is as follows:

- 06 Check for read and write permission
- 04 Check for read permission
- 02 Check for write permission
- 01 Execute (ignored)
- 00 Check for existence of file

➔ Under DOS, all existing files have read access (*amode* equals 04), so 00 and 04 give the same result. In the same vein, *amode* values of 06 and 02 are equivalent because under DOS write access implies read access.

If *filename* refers to a directory, `access` simply determines whether the directory exists.

Return value If the requested access is allowed, `access` returns 0; otherwise, it returns a value of -1, and the global variable *errno* is set to one of the following:

- ENOENT Path or file name not found
- EACCES Permission denied

Portability `access` is available on UNIX systems.

See also `chmod`, `fstat`, `stat`

Example

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

int file_exists(char *filename);

int main(void)
{
    printf("Does NOTEXIST.FIL exist: %s\n",
        file_exists("NOTEXIST.FIL") ? "YES" : "NO");
    return 0;
}

int file_exists(char *filename)
{
    return (access(filename, 0) == 0);
}
```

Program output

```
Does NOTEXIST.FIL exist? NO
```

ACOS

Function	Calculates the arc cosine.	
Syntax	<i>Real version:</i> <code>#include <math.h></code> <code>double acos(double x);</code>	<i>Complex version:</i> <code>#include <complex.h></code> <code>complex acos(complex x);</code>
Prototype in	<i>Real version:</i> <code>math.h</code>	<i>Complex version:</i> <code>complex.h</code>
Remarks	<p>acos returns the arc cosine of the input value. Real arguments to acos must be in the range -1 to 1, or else acos returns NAN and sets the global variable <i>errno</i> to</p> <p>EDOM Domain error</p> <p>The complex inverse cosine is defined by</p> $\text{acos}(z) = -i \log(z + i \sqrt{1 - z^2})$	
Return value	<p>acos of a real argument between -1 and $+1$ returns a value in the range 0 to π.</p> <p>Error handling for this routine can be modified through the function matherr.</p>	
Portability	<p>The real version of acos is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.</p>	
See also	<p>asin, atan, atan2, complex, cos, matherr, sin, tan</p>	
Example	<pre>#include <stdio.h> #include <math.h> int main(void) { double result; double x = 0.5; result = acos(x); printf("The arc cosine of %lf is %lf\n", x, result); return 0; }</pre>	

allocmem

Function Allocates DOS memory segment.

Syntax `#include <dos.h>`
`int allocmem(unsigned size, unsigned *segp);`

Prototype in dos.h

Remarks **allocmem** uses the DOS system call 0x48 to allocate a block of free memory and returns the segment address of the allocated block.

size is the desired size in paragraphs (a paragraph is 16 bytes). *segp* is a pointer to a word that will be assigned the segment address of the newly allocated block. No assignment is made to the word pointed to by *segp* if not enough room is available.

All allocated blocks are paragraph-aligned.



allocmem and **malloc** cannot coexist.

Return value **allocmem** returns `-1` on success. In the event of error, a number (the size in paragraphs of the largest available block) is returned.

An error return from **allocmem** sets the global variable `_doserrno` and sets the global variable `errno` to

`ENOMEM` Not enough memory

Portability **allocmem** is unique to DOS. This function is not Windows compatible.

See also **coreleft**, **freemem**, **malloc**, **setblock**

Example

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

int main(void)
{
    unsigned int size, segp;
    int stat;
    size = 64; /* (64 x 16) = 1024 bytes */
    stat = allocmem(size, &segp);
    if (stat == -1)
        printf("Allocated memory at segment: %x\n", segp);
    else
        printf("Failed: maximum number of paragraphs available is %u\n", stat);
}
```

```

        return 0;
    }

```

arc

- Function** Draws an arc.
- Syntax** `#include <graphics.h>`
`void far arc(int x, int y, int stangle, int endangle, int radius);`
- Prototype in** graphics.h
- Remarks** **arc** draws a circular arc in the current drawing color centered at (x,y) with a radius given by *radius*. The **arc** travels from *stangle* to *endangle*. If *stangle* equals 0 and *endangle* equals 360, the call to **arc** draws a complete circle.
- The angle for **arc** is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.
- Note** The *linestyle* parameter does not affect arcs, circles, ellipses, or pie slices. Only the *thickness* parameter is used.
- ➡ If you're using a CGA in high resolution mode or a monochrome graphics adapter, the examples in this book that show how to use graphics functions may not produce the expected results. If your system runs on a CGA or monochrome adapter, pass the value 1 to those functions that alter the fill or drawing color (**setcolor**, **setfillstyle**, and **setlinestyle**, for example), instead of a symbolic color constant (defined in graphics.h).
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **circle**, **ellipse**, **fillellipse**, **getarccoords**, **getaspectratio**, **graphresult**, **pieslice**, **sector**
- Example**
- ```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;

```

```

int stangle = 45, endangle = 135;
int radius = 100;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;
setcolor(getmaxcolor());

/* draw arc */
arc(midx, midy, stangle, endangle, radius);

/* clean up */
getch();
closegraph();
return 0;
}

```

## arg

---

**Function** Gives the angle of a number in the complex plane.

**Syntax** `#include <complex.h>`  
`double arg(complex x);`

**Prototype In** `complex.h`

**Remarks** `arg` gives the angle, in radians, of the number in the complex plane.

The positive real axis has angle 0, and the positive imaginary axis has angle  $\pi/2$ . If the argument passed to `arg` is complex 0 (zero), `arg` returns 0.

**Return value** `arg(x)` returns `atan2(imag(x), real(x))`.

**Portability** Complex functions require C++ and are not portable.

**See also** `complex`, `norm`, `polar`

```

Example #include <complex.h>

int main(void)
{
 double x = 3.1, y = 4.2;
 complex z = complex(x,y);
 cout << "z = " << z << "\n";
 cout << " has real part = " << real(z) << "\n";
 cout << " and imaginary part = " << imag(z) << "\n";
 cout << "z has complex conjugate = " << conj(z) << "\n";

 double mag = sqrt(norm(z));
 double ang = arg(z);

 cout << "The polar form of z is:\n";
 cout << " magnitude = " << mag << "\n";
 cout << " angle (in radians) = " << ang << "\n";
 cout << "Reconstructing z from its polar form gives:\n";
 cout << " z = " << polar(mag,ang) << "\n";
 return 0;
}

```

## asctime

---

|                     |                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Converts date and time to ASCII.                                                                                                                                                |
| <b>Syntax</b>       | #include <time.h><br>char *asctime(const struct tm *tblock);                                                                                                                    |
| <b>Prototype in</b> | time.h                                                                                                                                                                          |
| <b>Remarks</b>      | <b>asctime</b> converts a time stored as a structure in <i>tblock</i> to a 26-character string of the same form as the <b>ctime</b> string:<br><br>Sun Sep 16 01:03:52 1973\n\0 |
| <b>Return value</b> | <b>asctime</b> returns a pointer to the character string containing the date and time. This string is a static variable that is overwritten with each call to <b>asctime</b> .  |
| <b>Portability</b>  | <b>asctime</b> is available on UNIX systems and is defined in ANSI C.                                                                                                           |
| <b>See also</b>     | <b>ctime</b> , <b>difftime</b> , <b>ftime</b> , <b>gmtime</b> , <b>localtime</b> , <b>mktime</b> , <b>strftime</b> , <b>stime</b> , <b>time</b> , <b>tzset</b>                  |
| <b>Example</b>      | #include <stdio.h><br>#include <string.h><br>#include <time.h>                                                                                                                  |

```

int main(void)
{
 struct tm t;
 char str[80];

 /* sample loading of tm structure */
 t.tm_sec = 1; /* Seconds */
 t.tm_min = 30; /* Minutes */
 t.tm_hour = 9; /* Hour */
 t.tm_mday = 22; /* Day of the Month */
 t.tm_mon = 11; /* Month */
 t.tm_year = 56; /* Year - does not include century */
 t.tm_wday = 4; /* Day of the week */
 t.tm_yday = 0; /* Does not show in asctime */
 t.tm_isdst = 0; /* Is Daylight SavTime
 Does not show in asctime */

 /* converts structure to null terminated string */
 strcpy(str, asctime(&t));
 printf("%s\n", str);
 return 0;
}

```

## asin

---

**Function** Calculates the arc sine.

**Syntax** *Real version:*  
`#include <math.h>`  
`double asin(double x);`

*Complex version:*  
`#include <complex.h>`  
`complex asin(complex x);`

**Prototype in** *Real version:*  
`math.h`

*Complex version:*  
`complex.h`

**Remarks** **asin** of a real argument returns the arc sine of the input value. Real arguments to **asin** must be in the range  $-1$  to  $1$ , or else **asin** returns NAN and sets the global variable *errno* to

EDOM Domain error

The complex inverse sine is defined by

$$\text{asin}(z) = -i * \log(i * z + \text{sqrt}(1 - z^2))$$

**Return value** **asin** of a real argument returns a value in the range  $-\pi/2$  to  $\pi/2$ .

Error handling for this routine can be modified through the function **matherr**.



## asin

**Portability** The real version of **asin** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

**See also** **acos**, **atan**, **atan2**, **complex**, **cos**, **matherr**, **sin**, **tan**

**Example**

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

int main(void)
{
 double result;
 double x = 0.5;
 result = asin(x);
 printf("The arc sin of %lf is %lf\n", x, result);
 return(0);
}
```

## assert

---

**Function** Tests a condition and possibly aborts.

**Syntax** `#include <assert.h>`  
`void assert(int test);`

**Prototype in** `assert.h`

**Remarks** **assert** is a macro that expands to an **if** statement; if *test* evaluates to zero, **assert** prints a message on *stderr* and aborts the program (by calling **abort**).

**assert** prints this message:

```
Assertion failed: test, file filename, line linenum
```

The *filename* and *linenum* listed in the message are the source file name and line number where the **assert** macro appears.

If you place the `#define NDEBUG` directive ("no debugging") in the source code before the `#include <assert.h>` directive, the effect is to comment out the **assert** statement.

**Return value** None.

**Portability** **assert** is available on some UNIX systems, including Systems III and V, and is compatible with ANSI C.

**See also** **abort**

```

Example #include <assert.h>
#include <stdio.h>
#include <stdlib.h>

struct ITEM {
 int key;
 int value;
};

/* add item to list, make sure list is not null */
void additem(struct ITEM *itemptr) {
 assert(itemptr != NULL);
 /* add item to list */
}

int main(void)
{
 additem(NULL);
 return 0;
}

```

### Program output

```

Assertion failed: itemptr != NULL,
file C:\BC\ASSERT.C, line 12

```

## atan

---

**Function** Calculates the arc tangent.

**Syntax** *Real version:*

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

*Complex version:*

```
#include <complex.h>
complex atan(complex x);
```

**Prototype in** *Real version:*

math.h

*Complex version:*

complex.h

**Remarks** **atan** calculates the arc tangent of the input value.

The complex inverse tangent is defined by

$$\text{atan}(z) = -0.5 i \log((1 + i z)/(1 - i z))$$

**Return value** **atan** of a real argument returns a value in the range  $-pi/2$  to  $pi/2$ .

You can modify the error handling for this routine by using **matherr**.

**Portability** The real version of **atan** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

**See also** **acos, asin, atan2, complex, cos, matherr, sin, tan**

**Example**

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

int main(void)
{
 double result;
 double x = 0.5;
 result = atan(x);
 printf("The arc tangent of %lf is %lf\n", x, result);
 return(0);
}
```

## atan2

---

**Function** Calculates the arc tangent of  $y/x$ .

**Syntax** `#include <math.h>`  
`double atan2(double y, double x);`

**Prototype in** `math.h`

**Remarks** **atan2** returns the arc tangent of  $y/x$ ; it produces correct results even when the resulting angle is near  $\pi/2$  or  $-\pi/2$  ( $x$  near 0).

If both  $x$  and  $y$  are set to 0, the function sets the global variable *errno* to EDOM.

**Return value** **atan2** returns a value in the range  $-\pi$  to  $\pi$ .

You can modify the error handling for this routine by using the function **matherr**.

**Portability** **atan2** is available on UNIX systems and is defined in ANSI C.

**See also** **acos, asin, atan, cos, matherr, sin, tan**

**Example**

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

int main(void)
{
 double result;
 double x = 90.0, y = 15.0;
```

```

 result = atan2(y, x);
 printf("The arc tangent ratio of %lf is %lf\n", (y / x), result);
 return 0;
}

```

## atexit

---

**Function** Registers termination function.

**Syntax** `#include <stdlib.h>`  
`int atexit(atexit_t func);`

**Prototype in** `stdlib.h`

**Remarks** `atexit` registers the function pointed to by *func* as an exit function. Upon normal termination of the program, `exit` calls `(*func)()` just before returning to the operating system.

Each call to `atexit` registers another exit function. Up to 32 functions can be registered. They are executed on a last-in, first-out basis (that is, the last function registered is the first to be executed).

**Return value** `atexit` returns 0 on success and nonzero on failure (no space left to register the function).

**Portability** `atexit` is compatible with ANSI C.

**See also** `abort`, `_exit`, `exit`, `spawn...`

**Example**

```

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

void exit_fn1(void)
{
 printf("Exit function #1 called\n");
}

void exit_fn2(void)
{
 printf("Exit function #2 called\n");
}

int main(void)
{
 /* post exit function #1 */
 atexit(exit_fn1);

 /* post exit function #2 */
 atexit(exit_fn2);
 printf("Done in main\n");
}

```

```

 return 0;
}

```

## atof

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Converts a string to a floating-point number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Syntax</b>       | <code>#include &lt;math.h&gt;</code><br><code>double atof(const char *s);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Prototype in</b> | math.h, stdlib.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Remarks</b>      | <p><b>atof</b> converts a string pointed to by <i>s</i> to <b>double</b>; this function recognizes the character representation of a floating-point number, made up of the following:</p> <ul style="list-style-type: none"> <li>■ an optional string of tabs and spaces.</li> <li>■ an optional sign.</li> <li>■ a string of digits and an optional decimal point (the digits can be on both sides of the decimal point).</li> <li>■ an optional <i>e</i> or <i>E</i> followed by an optional signed integer.</li> </ul> <p>The characters must match this generic format:</p> <pre>[whitespace] [sign] [ddd] [.] [ddd] [e E[sign]ddd]</pre> <p><b>atof</b> also recognizes +INF and -INF for plus and minus infinity, and +NAN and -NAN for Not-a-Number.</p> <p>In this function, the first unrecognized character ends the conversion.</p> <p><b>strtod</b> is similar to <b>atof</b>; it provides better error detection, and hence is preferred in some applications.</p> |
| <b>Return value</b> | <p><b>atof</b> returns the converted value of the input string.</p> <p>If there is an overflow, <b>atof</b> returns plus or minus HUGE_VAL, <i>errno</i> is set to ERANGE, and <b>matherr</b> is not called.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Portability</b>  | <b>atof</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>See also</b>     | <b>atoi, atol, ecvt, fcvt, gcvt, strtod</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>      | <pre> #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt;  int main(void) {     float f; </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

```

char *str = "12345.678";
f = atof(str);
printf("string = %s float = %5.3f\n", str, f);

return 0;
}

```

## atoi

---

**Function** Converts a string to an integer.

**Syntax** `#include <stdlib.h>`  
`int atoi(const char *s);`

**Prototype in** `stdlib.h`

**Remarks** **atoi** converts a string pointed to by *s* to **int**; **atoi** recognizes (in the following order)

- an optional string of tabs and spaces
- an optional sign
- a string of digits

The characters must match this generic format:

```
[ws] [sn] [ddd]
```

In this function, the first unrecognized character ends the conversion.

There are no provisions for overflow in **atoi** (results are undefined).

**Return value** **atoi** returns the converted value of the input string. If the string cannot be converted to a number of the corresponding type (**int**), the return value is 0.

**Portability** **atoi** is available on UNIX systems and is defined in ANSI C.

**See also** **atof**, **atol**, **ecvt**, **fcvt**, **gcvt**, **scanf**, **strtod**

**Example**

```

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

int main(void)
{
 int n;
 char *str = "12345";
 n = atoi(str);
 printf("string = %s integer = %d\n", str, n);
 return 0;
}

```

# atol

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Converts a string to a long.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Syntax</b>       | <pre>#include &lt;stdlib.h&gt; long atol(const char *s);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Prototype in</b> | stdlib.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Remarks</b>      | <p><b>atol</b> converts the string pointed to by <i>s</i> to <b>long</b>. <b>atol</b> recognizes (in the following order)</p> <ul style="list-style-type: none"><li>■ an optional string of tabs and spaces</li><li>■ an optional sign</li><li>■ a string of digits</li></ul> <p>The characters must match this generic format:</p> <pre>[ws] [sn] [ddd]</pre> <p>In this function, the first unrecognized character ends the conversion.</p> <p>There are no provisions for overflow in <b>atol</b> (results are undefined).</p> |
| <b>Return value</b> | <b>atol</b> returns the converted value of the input string. If the string cannot be converted to a number of the corresponding type ( <b>long</b> ), <b>atol</b> returns 0.                                                                                                                                                                                                                                                                                                                                                      |
| <b>Portability</b>  | <b>atol</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>See also</b>     | <b>atoi, atof, ecvt, fcvt, gcvt, scanf, strtod, strtol, strtoul</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Example</b>      | <pre>#include &lt;stdlib.h&gt; #include &lt;stdio.h&gt;  int main(void) {     long l;     char *lstr = "98765432";     l = atol(lstr);     printf("string = %s long = %ld\n", lstr, l);     return 0; }</pre>                                                                                                                                                                                                                                                                                                                     |

## bar

**Function** Draws a two-dimensional bar.

**Syntax** `#include <graphics.h>`  
`#include <conio.h>`  
`void far bar(int left, int top, int right, int bottom);`

**Prototype in** graphics.h

**Remarks** **bar** draws a filled-in, rectangular, two-dimensional bar. The bar is filled using the current fill pattern and fill color. **bar** does not outline the bar; to draw an outlined two-dimensional bar, use **bar3d** with *depth* equal to 0.

The upper left and lower right corners of the rectangle are given by (*left, top*) and (*right, bottom*), respectively. The coordinates refer to pixels.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **bar3d, rectangle, setcolor, setfillstyle, setlinestyle**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, i;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }
}
```



## bar

```
midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* loop through the fill patterns */
for (i=SOLID_FILL; i<USER_FILL; i++)
{
 /* set the fill style */
 setfillstyle(i, getmaxcolor());

 /* draw the bar */
 bar(midx-50, midy-50, midx+50, midy+50);
 getch();
}

/* clean up */
closegraph();
return 0;
}
```

## bar3d

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Draws a three-dimensional bar.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Syntax</b>       | <pre>#include &lt;graphics.h&gt; void far bar3d(int left, int top, int right, int bottom, int depth, int topflag);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Prototype in</b> | graphics.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>      | <p><b>bar3d</b> draws a three-dimensional rectangular bar, then fills it using the current fill pattern and fill color. The three-dimensional outline of the bar is drawn in the current line style and color. The bar's depth in pixels is given by <i>depth</i>. The <i>topflag</i> parameter governs whether a three-dimensional top is put on the bar. If <i>topflag</i> is nonzero, a top is put on; otherwise, no top is put on the bar (making it possible to stack several bars on top of one another).</p> <p>The upper left and lower right corners of the rectangle are given by (<i>left</i>, <i>top</i>) and (<i>right</i>, <i>bottom</i>), respectively.</p> <p>To calculate a typical depth for <b>bar3d</b>, take 25% of the width of the bar, like this:</p> <pre>bar3d(left,top,right,bottom, (right-left)/4,1);</pre> |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Portability</b>  | This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**See also** `bar`, `rectangle`, `setcolor`, `setfillstyle`, `setlinestyle`

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, i;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* loop through the fill patterns */
 for (i=EMPTY_FILL; i<USER_FILL; i++)
 {
 /* set the fill style */
 setfillstyle(i, getmaxcolor());

 /* draw the 3-d bar */
 bar3d(midx-50, midy-50, midx+50, midy+50, 10, 1);
 getch();
 }

 /* clean up */
 closegraph();
 return 0;
}
```

bcd

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Converts a number to binary coded decimal (BCD).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Syntax</b>       | <pre>#include &lt;bcd.h&gt; bcd bcd(int x); bcd bcd(double x); bcd bcd(double x, int <i>decimals</i>);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Prototype in</b> | bcd.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Remarks</b>      | <p>All of the usual arithmetic operators have been overloaded to work with BCD numbers.</p> <p>BCD numbers have about 17 decimal digits precision, and a range of about <math>1 \times 10^{-125}</math> to <math>1 \times 10^{125}</math>.</p> <p>Use the function <b>real</b> to convert a BCD number back to a <b>float</b>, <b>double</b>, or <b>long double</b>.</p> <p>The argument <i>decimals</i> is optional. You can use it to specify how many decimal digits after the decimal point are to be carried in the conversion.</p> <p>The number is rounded according to the rules of banker's rounding, which means round to nearest whole number, with ties being rounded to an even digit.</p> |
| <b>Return value</b> | The BCD equivalent of the given number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Portability</b>  | <b>bcd</b> is unique to Borland C++ and works only with C++.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>See also</b>     | <b>real</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Example</b>      | <pre>#include &lt;iostream.h&gt; #include &lt;bcd.h&gt;  double x = 10000.0;           // ten thousand dollars bcd a = bcd(x/3,2);          // a third, rounded to nearest penny  int main(void) {     cout &lt;&lt; "share of fortune = \$" &lt;&lt; a &lt;&lt; "\n";     return 0; }</pre>                                                                                                                                                                                                                                                                                                                                                                                                            |

## bdos

---

**Function** Accesses DOS system calls.

**Syntax** `#include <dos.h>`  
`int bdos(int dosfun, unsigned dosdx, unsigned dosal);`

**Prototype in** `dos.h`

**Remarks** **bdos** provides direct access to many of the DOS system calls. See your DOS reference manuals for details on each system call.

For system calls that require an integer argument, use **bdos**; if they require a pointer argument, use **bdosptr**. In the large data models (compact, large, and huge), it is important to use **bdosptr** instead of **bdos** for system calls that require a pointer as the call argument.

*dosfun* is defined in your DOS reference manuals.

*dosdx* is the value of register DX.

*dosal* is the value of register AL.

**Return value** The return value of **bdos** is the value of AX set by the system call.

**Portability** **bdos** is unique to DOS.

**See also** **bdosptr**, **geninterrupt**, **int86**, **int86x**, **intdos**, **intdosx**

**Example**

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

/* get current drive as 'A', 'B', ... */
char current_drive(void)
{
 char curdrive;

 /* get current disk as 0, 1, ... */
 curdrive = bdos(0x19, 0, 0);
 return('A' + curdrive);
}

int main(void)
{
 printf("The current drive is %c:\n", current_drive());
 return 0;
}
```

**Program output**

The current drive is C:

## bdosptr

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Accesses DOS system calls.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Syntax</b>       | <code>#include &lt;dos.h&gt;</code><br><code>int bdosptr(int <i>dosfun</i>, void *<i>argument</i>, unsigned <i>dosal</i>);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Prototype in</b> | dos.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Remarks</b>      | <p><b>bdosptr</b> provides direct access to many of the DOS system calls. See your DOS reference manuals for details of each system call.</p> <p>For system calls that require an integer argument, use <b>bdos</b>; if they require a pointer argument, use <b>bdosptr</b>. In the large data models (compact, large, and huge), it is important to use <b>bdosptr</b> for system calls that require a pointer as the call argument. In the small data models, the <i>argument</i> parameter to <b>bdosptr</b> specifies DX; in the large data models, it gives the DS:DX values to be used by the system call.</p> <p><i>dosfun</i> is defined in your DOS reference manuals.</p> <p><i>dosal</i> is the value of register AL.</p> |
| <b>Return value</b> | The return value of <b>bdosptr</b> is the value of AX on success or -1 on failure. On failure, the global variables <i>errno</i> and <i>_doserrno</i> are set.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Portability</b>  | <b>bdosptr</b> is unique to DOS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>See also</b>     | <b>bdos</b> , <b>geninterrupt</b> , <b>int86</b> , <b>int86x</b> , <b>intdos</b> , <b>intdosx</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;dos.h&gt;  /* get current drive as 'A', 'B', ... */ char current_drive(void) {     char curdrive;      /* get current disk as 0, 1, ... */     curdrive = bdos(0x19, 0, 0);     return('A' + curdrive); }  int main(void) {     printf("The current drive is %c:\n", current_drive()); }</pre>                                                                                                                                                                                                                                                                                                                                                                                            |

```
 return 0;
}
```

## bioscom

---

**Function** Performs serial I/O.

**Syntax** `#include <bios.h>`  
`int bioscom(int cmd, char abyte, int port);`

**Prototype in** bios.h

**Remarks** **bioscom** performs various RS-232 communications over the I/O port given in *port*.

A *port* value of 0 corresponds to COM1, 1 corresponds to COM2, and so forth.

The value of *cmd* can be one of the following:

- 0 Sets the communications parameters to the value in *abyte*.
- 1 Sends the character in *abyte* out over the communications line.
- 2 Receives a character from the communications line.
- 3 Returns the current status of the communications port.

*abyte* is a combination of the following bits (one value is selected from each of the groups):

|      |             |      |           |
|------|-------------|------|-----------|
| 0x02 | 7 data bits | 0x00 | 110 baud  |
| 0x03 | 8 data bits | 0x20 | 150 baud  |
|      |             | 0x40 | 300 baud  |
| 0x00 | 1 stop bit  | 0x60 | 600 baud  |
| 0x04 | 2 stop bits | 0x80 | 1200 baud |
| 0x00 | No parity   | 0xA0 | 2400 baud |
| 0x08 | Odd parity  | 0xC0 | 4800 baud |
| 0x18 | Even parity | 0xE0 | 9600 baud |

For example, a value of 0xEB (0xE0 | 0x08 | 0x00 | 0x03) for *abyte* sets the communications port to 9600 baud, odd parity, 1 stop bit, and 8 data bits. **bioscom** uses the BIOS 0x14 interrupt.

**Return value** For all values of *cmd*, **bioscom** returns a 16-bit integer of which the upper 8 bits are status bits and the lower 8 bits vary, depending on the value of *cmd*. The upper bits of the return value are defined as follows:

- Bit 15 Time out
- Bit 14 Transmit shift register empty

Bit 13 Transmit holding register empty  
 Bit 12 Break detect  
 Bit 11 Framing error  
 Bit 10 Parity error  
 Bit 9 Overrun error  
 Bit 8 Data ready

If the *abyte* value could not be sent, bit 15 is set to 1. Otherwise, the remaining upper and lower bits are appropriately set. For example, if a framing error has occurred, bit 11 is set to 1.

With a *cmd* value of 2, the byte read is in the lower bits of the return value if there is no error. If an error occurs, at least one of the upper bits is set to 1. If no upper bits are set to 1, the byte was received without error.

With a *cmd* value of 0 or 3, the return value has the upper bits set as defined, and the lower bits are defined as follows:

Bit 7 Received line signal detect  
 Bit 6 Ring indicator  
 Bit 5 Data set ready  
 Bit 4 Clear to send  
 Bit 3 Change in receive line signal detector  
 Bit 2 Trailing edge ring detector  
 Bit 1 Change in data set ready  
 Bit 0 Change in clear to send

**Portability** bioscom works with IBM PCs and compatibles only.

#### Example

```

#include <bios.h>
#include <conio.h>

#define COM1 0
#define DATA_READY 0x100
#define TRUE 1
#define FALSE 0
#define SETTINGS (0x80 | 0x02 | 0x00 | 0x00)

int main(void)
{
 int in, out, status, DONE = FALSE;
 bioscom(0, SETTINGS, COM1);
 printf("... BIOSCOM [ESC] to exit ...\n");
 while (!DONE)
 {
 status = bioscom(3, 0, COM1);
 if (status & DATA_READY)
 if ((out = bioscom(2, 0, COM1) & 0x7F) != 0)
 putchar(out);
 }
}

```

```

 if (kbhit())
 {
 if ((in = getch()) == '\x1B')
 DONE = TRUE;
 bioscom(1, in, COM1);
 }
 }
 return 0;
}

```

## biosdisk

---

**Function** Issues BIOS disk drive services.

**Syntax** `#include <bios.h>`  
`int biosdisk(int cmd, int drive, int head, int track, int sector, int nsects,`  
`void *buffer);`

**Prototype in** bios.h

**Remarks** **biosdisk** uses interrupt 0x13 to issue disk operations directly to the BIOS.

*drive* is a number that specifies which disk drive is to be used: 0 for the first floppy disk drive, 1 for the second floppy disk drive, 2 for the third, and so on. For hard disk drives, a *drive* value of 0x80 specifies the first drive, 0x81 specifies the second, 0x82 the third, and so forth.

For hard disks, the physical drive is specified, not the disk partition. If necessary, the application program must interpret the partition table information itself.

*cmd* indicates the operation to perform. Depending on the value of *cmd*, the other parameters may or may not be needed.

Here are the possible values for *cmd* for the IBM PC, XT, AT, or PS/2, or any compatible system:

- 0 Resets disk system, forcing the drive controller to do a hard reset. All other parameters are ignored.
- 1 Returns the status of the last disk operation. All other parameters are ignored.
- 2 Reads one or more disk sectors into memory. The starting sector to read is given by *head*, *track*, and *sector*. The number of sectors is given by *nsects*. The data is read, 512 bytes per sector, into *buffer*.



- 3 Writes one or more disk sectors from memory. The starting sector to write is given by *head*, *track*, and *sector*. The number of sectors is given by *nsects*. The data is written, 512 bytes per sector, from *buffer*.
- 4 Verifies one or more sectors. The starting sector is given by *head*, *track*, and *sector*. The number of sectors is given by *nsects*.
- 5 Formats a track. The track is specified by *head* and *track*. *buffer* points to a table of sector headers to be written on the named *track*. See the *Technical Reference Manual* for the IBM PC for a description of this table and the format operation.

The following *cmd* values are allowed only for the XT, AT, PS/2, and compatibles:

- 6 Formats a track and sets bad sector flags.
- 7 Formats the drive beginning at a specific track.
- 8 Returns the current drive parameters. The drive information is returned in *buffer* in the first 4 bytes.
- 9 Initializes drive-pair characteristics.
- 10 Does a long read, which reads 512 plus 4 extra bytes per sector.
- 11 Does a long write, which writes 512 plus 4 extra bytes per sector.
- 12 Does a disk seek.
- 13 Alternates disk reset.
- 14 Reads sector buffer.
- 15 Writes sector buffer.
- 16 Tests whether the named drive is ready.
- 17 Recalibrates the drive.
- 18 Controller RAM diagnostic.
- 19 Drive diagnostic.
- 20 Controller internal diagnostic.



**biosdisk** operates below the level of files on raw sectors. *It can destroy file contents and directories on a hard disk.*

**Return value** **biosdisk** returns a status byte composed of the following bits:

- 0x00 Operation successful.
- 0x01 Bad command.

|      |                                             |
|------|---------------------------------------------|
| 0x02 | Address mark not found.                     |
| 0x03 | Attempt to write to write-protected disk.   |
| 0x04 | Sector not found.                           |
| 0x05 | Reset failed (hard disk).                   |
| 0x06 | Disk changed since last operation.          |
| 0x07 | Drive parameter activity failed.            |
| 0x08 | Direct memory access (DMA) overrun.         |
| 0x09 | Attempt to perform DMA across 64K boundary. |
| 0x0A | Bad sector detected.                        |
| 0x0B | Bad track detected.                         |
| 0x0C | Unsupported track.                          |
| 0x10 | Bad CRC/ECC on disk read.                   |
| 0x11 | CRC/ECC corrected data error.               |
| 0x20 | Controller has failed.                      |
| 0x40 | Seek operation failed.                      |
| 0x80 | Attachment failed to respond.               |
| 0xAA | Drive not ready (hard disk only).           |
| 0xBB | Undefined error occurred (hard disk only).  |
| 0xCC | Write fault occurred.                       |
| 0xE0 | Status error.                               |
| 0xFF | Sense operation failed.                     |

0x11 is not an error because the data is correct. The value is returned to give the application an opportunity to decide for itself.

**Portability** **biosdisk** works with IBM PCs and compatibles only.

**See also** **absread**, **abswrite**

**Example**

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

int main(void)
{
 #define CMD 2 /* read sector command */
 #define DRIVE 0 /* drive number for A: */
 #define HEAD 0 /* disk head number */
 #define TRACK 1 /* track number */
 #define SECT 1 /* sector number */
 #define NSECT 1 /* sector count */

 int result;
 char buffer[512];
 printf("Attempting to read from drive A:\n");
 result = biosdisk(CMD, DRIVE, HEAD, TRACK, SECT, NSECT, buffer);
 if (result == 0)
 printf("Disk read from A: successful.\n");
 else
```

## biosdisk

```
 printf("Attempt to read from drive A: failed.\n");
 return 0;
}
```

## biosequip

---

**Function** Checks equipment.

**Syntax** #include <bios.h>  
int biosequip(void);

**Prototype in** bios.h

**Remarks** **biosequip** uses BIOS interrupt 0x11 to return an integer describing the equipment connected to the system.

**Return value** The return value is interpreted as a collection of bit-sized fields. The IBM PC values follow:

**Bits 14-15** Number of parallel printers installed

00 = 0 printers

01 = 1 printer

10 = 2 printers

11 = 3 printers

**Bit 13** Serial printer attached

**Bit 12** Game I/O attached

**Bits 9-11** Number of COM ports

000 = 0 ports

001 = 1 port

010 = 2 ports

011 = 3 ports

100 = 4 ports

101 = 5 ports

110 = 6 ports

111 = 7 ports

**Bit 8** Direct memory access (DMA)

0 = Machine has DMA

1 = Machine does not have DMA; for example, PC Jr.

**Bits 6-7** Number of disk drives

00 = 1 drive

01 = 2 drives

10 = 3 drives

*DOS only sees two ports but can be pushed to see four; the IBM PS/2 can see up to eight.*

11 = 4 drives, only if bit 0 is 1

**Bit 4-5** Initial video mode  
 00 = Unused  
 01 = 40x25 BW with color card  
 10 = 80x25 BW with color card  
 11 = 80x25 BW with mono card

**Bits 2-3** Motherboard RAM size  
 00 = 16K  
 01 = 32K  
 10 = 48K  
 11 = 64K

**Bit 1** Floating-point coprocessor

**Bit 0** Boot from disk

**Portability** biosequip works with IBM PCs and compatibles only.

**Example**

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

#define CO_PROCESSOR_MASK 0x0002

int main(void)
{
 int equip_check;

 /* get the current equipment configuration */
 equip_check = biosequip();

 /* check to see if there is a coprocessor installed */
 if (equip_check & CO_PROCESSOR_MASK)
 printf("There is a math coprocessor installed.\n");
 else
 printf("No math coprocessor installed.\n");
 return 0;
}
```

## bioskey

---

**Function** Keyboard interface, using BIOS services directly.

**Syntax** #include <bios.h>  
 int bioskey(int *cmd*);

**Prototype in** bios.h

## bioskey

**Remarks** **bioskey** performs various keyboard operations using BIOS interrupt 0x16. The parameter *cmd* determines the exact operation.

**Return value** The value returned by **bioskey** depends on the task it performs, determined by the value of *cmd*:

0 If the lower 8 bits are nonzero, **bioskey** returns the ASCII character for the next keystroke waiting in the queue or the next key pressed at the keyboard. If the lower 8 bits are zero, the upper 8 bits are the extended keyboard codes defined in the IBM PC *Technical Reference Manual*.

1 This tests whether a keystroke is available to be read. A return value of zero means no key is available. The return value is 0xFFFF (-1) if *Ctrl-Brk* has been pressed. Otherwise, the value of the next keystroke is returned. The keystroke itself is kept to be returned by the next call to **bioskey** that has a *cmd* value of zero.

2 Requests the current shift key status. The value is obtained by ORing the following values together:

|       |      |                        |
|-------|------|------------------------|
| Bit 7 | 0x80 | <i>Insert</i> on       |
| Bit 6 | 0x40 | <i>Caps</i> on         |
| Bit 5 | 0x20 | <i>Num Lock</i> on     |
| Bit 4 | 0x10 | <i>Scroll Lock</i> on  |
| Bit 3 | 0x08 | <i>Alt</i> pressed     |
| Bit 2 | 0x04 | <i>Ctrl</i> pressed    |
| Bit 1 | 0x02 | ← <i>Shift</i> pressed |
| Bit 0 | 0x01 | → <i>Shift</i> pressed |

**Portability** **bioskey** works with IBM PCs and compatibles only.

**Example**

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

#define RIGHT 0x01
#define LEFT 0x02
#define CTRL 0x04
#define ALT 0x08

int main(void)
{
 int key, modifiers;

 /* function 1 returns 0 until a key is pressed */
 while (bioskey(1) == 0);
```

```

/* function 0 returns the key that is waiting */
key = bioskey(0);

/* use function 2 to determine if shift keys are used */
modifiers = bioskey(2);
if (modifiers)
{
 printf("[");
 if (modifiers & RIGHT) printf("RIGHT");
 if (modifiers & LEFT) printf("LEFT");
 if (modifiers & CTRL) printf("CTRL");
 if (modifiers & ALT) printf("ALT");
 printf("]");
}

/* print out the character read */
if (isalnum(key & 0xFF))
 printf("%c\n", key);
else
 printf("%#02x\n", key);
return 0;
}

```

## biosmemory

---

|                     |                                                                                                                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Returns memory size.                                                                                                                                                                                               |
| <b>Syntax</b>       | <code>#include &lt;bios.h&gt;</code><br><code>int biosmemory(void);</code>                                                                                                                                         |
| <b>Prototype in</b> | bios.h                                                                                                                                                                                                             |
| <b>Remarks</b>      | <b>biosmemory</b> returns the size of RAM memory using BIOS interrupt 0x12. This does not include display adapter memory, extended memory, or expanded memory.                                                     |
| <b>Return value</b> | <b>biosmemory</b> returns the size of RAM memory in 1K blocks.                                                                                                                                                     |
| <b>Portability</b>  | <b>biosmemory</b> works with IBM PCs and compatibles only.                                                                                                                                                         |
| <b>Example</b>      | <pre> #include &lt;stdio.h&gt; #include &lt;bios.h&gt;  int main(void) {     int memory_size;     memory_size = biosmemory(); /* returns value up to 640K */     printf("RAM size = %dK\n", memory_size); } </pre> |

## biosprint

```
 return 0;
}
```

## biosprint

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|------|-----------------|-------|------|-----------|-------|------|----------|-------|------|--------------|-------|------|-------------|-------|------|----------|
| <b>Function</b>     | Printer I/O using BIOS services directly.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Syntax</b>       | <pre>#include &lt;bios.h&gt;<br/>int biosprint(int <i>cmd</i>, int <i>abyte</i>, int <i>port</i>);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                     |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Prototype In</b> | bios.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Remarks</b>      | <p><b>biosprint</b> performs various printer functions on the printer identified by the parameter <i>port</i> using BIOS interrupt 0x17.</p> <p>A <i>port</i> value of 0 corresponds to LPT1; a <i>port</i> value of 1 corresponds to LPT2; and so on.</p> <p>The value of <i>cmd</i> can be one of the following:</p> <ul style="list-style-type: none"><li>0 Prints the character in <i>abyte</i>.</li><li>1 Initializes the printer port.</li><li>2 Reads the printer status.</li></ul> <p>The value of <i>abyte</i> can be 0 to 255.</p> |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Return value</b> | <p>The value returned from any of these operations is the current printer status, which is obtained by ORing these bit values together:</p> <table><tr><td>Bit 0</td><td>0x01</td><td>Device time out</td></tr><tr><td>Bit 3</td><td>0x08</td><td>I/O error</td></tr><tr><td>Bit 4</td><td>0x10</td><td>Selected</td></tr><tr><td>Bit 5</td><td>0x20</td><td>Out of paper</td></tr><tr><td>Bit 6</td><td>0x40</td><td>Acknowledge</td></tr><tr><td>Bit 7</td><td>0x80</td><td>Not busy</td></tr></table>                                     | Bit 0           | 0x01 | Device time out | Bit 3 | 0x08 | I/O error | Bit 4 | 0x10 | Selected | Bit 5 | 0x20 | Out of paper | Bit 6 | 0x40 | Acknowledge | Bit 7 | 0x80 | Not busy |
| Bit 0               | 0x01                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Device time out |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| Bit 3               | 0x08                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | I/O error       |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| Bit 4               | 0x10                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Selected        |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| Bit 5               | 0x20                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Out of paper    |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| Bit 6               | 0x40                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Acknowledge     |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| Bit 7               | 0x80                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Not busy        |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Portability</b>  | <b>biosprint</b> works with IBM PCs and compatibles only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;<br/>#include &lt;conio.h&gt;<br/>#include &lt;bios.h&gt;<br/><br/>int main(void)<br/>{<br/>    #define STATUS 2 /* printer status command */<br/>    #define PORTNUM 0 /* port number for LPT1 */<br/><br/>    int status, abyte=0;<br/>    printf("Please turn off your printer. Press any key to continue\n");<br/>    getch();<br/>}</pre>                                                                                                                                                                  |                 |      |                 |       |      |           |       |      |          |       |      |              |       |      |             |       |      |          |

```

status = biosprint(STATUS, abyte, PORTNUM);
if (status & 0x01)
 printf("Device time out.\n");
if (status & 0x08)
 printf("I/O error.\n");
if (status & 0x10)
 printf("Selected.\n");
if (status & 0x20)
 printf("Out of paper.\n");
if (status & 0x40)
 printf("Acknowledge.\n");
if (status & 0x80)
 printf("Not busy.\n");
return 0;
}

```

## biostime

---

**Function** Reads or sets the BIOS timer.

**Syntax** `#include <bios.h>`  
`long biostime(int cmd, long newtime);`

**Prototype in** bios.h

**Remarks** **biostime** either reads or sets the BIOS timer. This is a timer counting ticks since midnight at a rate of roughly 18.2 ticks per second. **biostime** uses BIOS interrupt 0x1A.

If *cmd* equals 0, **biostime** returns the current value of the timer. If *cmd* equals 1, the timer is set to the **long** value in *newtime*.

**Return value** When **biostime** reads the BIOS timer (*cmd* = 0), it returns the timer's current value.

**Portability** **biostime** works with IBM PCs and compatibles only.

**Example**

```

#include <bios.h>
#include <time.h>
#include <conio.h>

int main(void)
{
 long bios_time;
 clrscr();
 cprintf("The number of clock ticks since midnight is:\r\n");
 cprintf("The number of seconds since midnight is:\r\n");
 cprintf("The number of minutes since midnight is:\r\n");
}

```



## biostime

```
printf("The number of hours since midnight is:\r\n");
printf("\r\nPress any key to quit:");
while(!kbhit())
{
 bios_time = biostime(0, 0L);
 gotoxy(50, 1);
 printf("%lu", bios_time);
 gotoxy(50, 2);
 printf("%.4f", bios_time / CLK_TCK);
 gotoxy(50, 3);
 printf("%.4f", bios_time / CLK_TCK / 60);
 gotoxy(50, 4);
 printf("%.4f", bios_time / CLK_TCK / 3600);
}
return 0;
}
```

## brk

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Changes data-segment space allocation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Syntax</b>       | <pre>#include &lt;alloc.h&gt; int brk(void *addr);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Prototype in</b> | alloc.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Remarks</b>      | <p><b>brk</b> dynamically changes the amount of space allocated to the calling program's heap. The change is made by resetting the program's <i>break value</i>, which is the address of the first location beyond the end of the data segment. The amount of allocated space increases as the break value increases.</p> <p><b>brk</b> sets the break value to <i>addr</i> and changes the allocated space accordingly.</p> <p>This function will fail without making any change in the allocated space if such a change would allocate more space than is allowable.</p> |
| <b>Return value</b> | Upon successful completion, <b>brk</b> returns a value of 0. On failure, this function returns a value of -1 and the global variable <i>errno</i> is set to                                                                                                                                                                                                                                                                                                                                                                                                                |
|                     | ENOMEM Not enough memory                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Portability</b>  | <b>brk</b> is available on UNIX systems. This function is not Windows compatible.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>See also</b>     | <b>coreleft, sbrk</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

```

Example #include <stdio.h>
 #include <alloc.h>

 int main(void)
 {
 char *ptr;
 printf("Changing allocation with brk()\n");
 ptr = malloc(1);
 printf("Before brk() call: %lu bytes free\n", coreleft());
 brk(ptr+1000);
 printf(" After brk() call: %lu bytes free\n", coreleft());
 return 0;
 }

```

## bsearch

---

**Function** Binary search of an array.

**Syntax** #include <stdlib.h>  
void \*bsearch(const void \*key, const void \*base, size\_t nelem, size\_t width,  
int (\*fcmp)(const void \*, const void \*));

**Prototype in** stdlib.h

**Remarks** **bsearch** searches a table (array) of *nelem* elements in memory, and returns the address of the first entry in the table that matches the search key. The array must be in order. If no match is found, **bsearch** returns 0. Note that since this is a binary search, the first matching entry is not necessarily the first entry in the table.

The type *size\_t* is defined as an unsigned integer.

- *nelem* gives the number of elements in the table.
- *width* specifies the number of bytes in each table entry.

The comparison routine *fcmp* is called with two arguments: *elem1* and *elem2*. Each argument points to an item to be compared. The comparison function compares each of the pointed-to items (*\*elem1* and *\*elem2*), and returns an integer based on the results of the comparison.

For **bsearch**, the *fcmp* return value is

```

< 0 if *elem1 < *elem2
== 0 if *elem1 == *elem2
> 0 if *elem1 > *elem2

```

## bsearch

**Return value** **bsearch** returns the address of the first entry in the table that matches the search key. If no match is found, **bsearch** returns 0.

**Portability** **bsearch** is available on UNIX systems and is defined in ANSI C.

**See also** **lfind, lsearch, qsort**

**Example**

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

#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))

int numarray[] = {123, 145, 512, 627, 800, 933};
int numeric(const int *p1, const int *p2)
{
 return(*p1 - *p2);
}

int lookup(int key)
{
 int *itemptr;

 /* the cast of (int*)(const void *,const void*) is needed to void
 a type mismatch error at compile time */
 itemptr = bsearch (&key, numarray, NELEMS(numarray), sizeof(int), int (*)(const
 void *,const void *))numeric);
 return (itemptr != NULL);
}

int main(void)
{
 if (lookup(512))
 printf("512 is in the table.\n");
 else
 printf("512 isn't in the table.\n");
 return 0;
}
```

## cabs

---

**Function** Calculates the absolute value of complex number.

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

**Prototype in** `math.h`

**Remarks** **cabs** is a macro that calculates the absolute value of *z*, a complex number. *z* is a structure with type *complex*; the structure is defined in `math.h` as

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

where  $x$  is the real part, and  $y$  is the imaginary part.

Calling **cabs** is equivalent to calling **sqrt** with the real and imaginary components of  $z$  as shown here:

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

If using C++, use the **complex** type defined in `complex.h`, and the function **abs**.

**Return value** **cabs** returns the absolute value of  $z$ , a **double**. On overflow, **cabs** returns `HUGE_VAL` and sets the global variable `errno` to

`ERANGE` Result out of range

Error handling for **cabs** can be modified through the function **matherr**.

**Portability** **cabs** is available on UNIX systems.

**See also** **abs**, **complex**, **fabs**, **labs**, **matherr**

**Example**

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

int main(void)
{
 struct complex z;
 double val;
 z.x = 2.0;
 z.y = 1.0;
 val = cabs(z);
 printf("The absolute value of %.21fi %.21fj is %.21f", z.x, z.y, val);
 return 0;
}
```

## calloc

---

**Function** Allocates main memory.

**Syntax** `#include <stdlib.h>`  
`void *calloc(size_t nitems, size_t size);`

**Prototype in** `stdlib.h`, `alloc.h`

**Remarks** **calloc** provides access to the C memory heap. The heap is available for dynamic allocation of variable-sized blocks of memory. Many data

structures, such as trees and lists, naturally employ heap memory allocation.

All the space between the end of the data segment and the top of the program stack is available for use in the small data models (tiny, small, and medium), except for a small margin immediately before the top of the stack. This margin is intended to allow some room for the application to grow on the stack, plus a small amount needed by DOS.

In the large data models (compact, large, and huge), all space beyond the program stack to the end of physical memory is available for the heap.

**calloc** allocates a block of size *nitems* × *size*. The block is cleared to 0. If you want to allocate a block larger than 64K, you must use **farcalloc**.

**Return value** **calloc** returns a pointer to the newly allocated block. If not enough space exists for the new block or *nitems* or *size* is 0, **calloc** returns null.

**Portability** **calloc** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**See also** **farcalloc**, **free**, **malloc**, **realloc**

**Example**

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

main()
{
 char *str = NULL;

 /* allocate memory for string */
 str = calloc(10, sizeof(char));
 if (str) {
 /* copy "Hello" into string */
 strcpy(str, "Hello");

 /* display string */
 printf("String is %s\n", str);

 /* free memory */
 free(str);
 }
 else
 {
 printf("Not enough memory!\n");
 }
 return 0;
}
```

## ceil

---

|                     |                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Rounds up.                                                                                                                                                                                                                                                                                                                                           |
| <b>Syntax</b>       | <code>#include &lt;math.h&gt;</code><br><code>double ceil(double x);</code>                                                                                                                                                                                                                                                                          |
| <b>Prototype in</b> | math.h                                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>      | <b>ceil</b> finds the smallest integer not less than <i>x</i> .                                                                                                                                                                                                                                                                                      |
| <b>Return value</b> | <b>ceil</b> returns the integer found (as a <b>double</b> ).                                                                                                                                                                                                                                                                                         |
| <b>Portability</b>  | <b>ceil</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                   |
| <b>See also</b>     | <b>floor, fmod</b>                                                                                                                                                                                                                                                                                                                                   |
| <b>Example</b>      | <pre>#include &lt;math.h&gt; #include &lt;stdio.h&gt;  int main(void) {     double number = 123.54;     double down, up;     down = floor(number);     up = ceil(number);     printf("original number    %5.2lf\n", number);     printf("number rounded down %5.2lf\n", down);     printf("number rounded up   %5.2lf\n", up);     return 0; }</pre> |

## cgets

---

|                     |                                                                                                                                                                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reads a string from the console.                                                                                                                                                                                                                                                                          |
| <b>Syntax</b>       | <code>#include &lt;conio.h&gt;</code><br><code>char *cgets(char *str);</code>                                                                                                                                                                                                                             |
| <b>Prototype in</b> | conio.h                                                                                                                                                                                                                                                                                                   |
| <b>Remarks</b>      | <p><b>cgets</b> reads a string of characters from the console, storing the string (and the string length) in the location pointed to by <i>str</i>.</p> <p><b>cgets</b> reads characters until it encounters a carriage-return/linefeed (CR/LF) combination, or until the maximum allowable number of</p> |

## cgets

characters have been read. If **cgets** reads a carriage return/linefeed combination, it replaces the combination with a `\0` (null terminator) before storing the string.

Before **cgets** is called, set `str[0]` to the maximum length of the string to be read. On return, `str[1]` is set to the number of characters actually read. The characters read start at `str[2]` and end with a null terminator. Thus, `str` must be at least `str[0]` plus 2 bytes long.

**Return value** On success, **cgets** returns a pointer to `str[2]`.

**Portability** It works only with IBM PCs and compatibles. This function is not Windows compatible.

**See also** **cputs**, **fgets**, **getch**, **getche**, **gets**

**Example**

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

main()
{
 char buffer[83];
 char *p;

 /* there's space for 80 characters plus the NULL terminator */
 buffer[0] = 81;
 printf("Input some chars:");
 p = cgets(buffer);
 printf("\ncgets read %d characters: \"%s\"\n", buffer[1], p);
 printf("The returned pointer is %p, buffer[0] is at %p\n", p, &buffer);

 /* leave room for 5 characters plus the NULL terminator */
 buffer[0] = 6;
 printf("Input some chars:");
 p = cgets(buffer);
 printf("\ncgets read %d characters: \"%s\"\n", buffer[1], p);
 printf("The returned pointer is %p, buffer[0] is at %p\n", p, &buffer);
 return 0;
}
```

## chdir

---

**Function** Changes current directory.

**Syntax** `#include <dir.h>`  
`int chdir(const char *path);`

**Prototype in** `dir.h`

**Remarks** **chdir** causes the directory specified by *path* to become the current working directory. *path* must specify an existing directory.

A drive can also be specified in the *path* argument, such as

```
chdir("a:\\BC")
```

but this changes only the current directory on that drive; it doesn't change the active drive.

**Return value** Upon successful completion, **chdir** returns a value of 0. Otherwise, it returns a value of -1, and the global variable *errno* is set to

ENOENT Path or file name not found

**Portability** **chdir** is available on UNIX systems.

**See also** **getcurdir**, **getcwd**, **getdisk**, **mkdir**, **rmdir**, **setdisk**, **system**

**Example**

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

char old_dir[MAXDIR];
char new_dir[MAXDIR];

int main(void)
{
 if (getcurdir(0, old_dir) {
 perror("getcurdir()");
 exit(1);
 }
 printf("Current directory is: \\%s\n", old_dir);
 if (chdir("\\\\")) {
 perror("chdir()");
 exit(1);
 }
 if (getcurdir(0, new_dir) {
 perror("getcurdir()");
 exit(1);
 }
 printf("Current directory is now: \\%s\n", new_dir);
 printf("\nChanging back to original directory: \\%s\n", old_dir);
 if (chdir(old_dir)) {
 perror("chdir()");
 exit(1);
 }
 return 0;
}
```



## `_chmod`

---

**Function** Changes file access mode.

**Syntax**

```
#include <dos.h>
#include <io.h>
int _chmod(const char *path, int func [, int attrib]);
```

**Prototype in** `io.h`

**Remarks** `_chmod` can either fetch or set the DOS file attributes. If *func* is 0, the function returns the current DOS attributes for the file. If *func* is 1, the attribute is set to *attrib*.

*attrib* can be one of the following symbolic constants (defined in `dos.h`):

|                        |                     |
|------------------------|---------------------|
| <code>FA_RDONLY</code> | Read-only attribute |
| <code>FA_HIDDEN</code> | Hidden file         |
| <code>FA_SYSTEM</code> | System file         |
| <code>FA_LABEL</code>  | Volume label        |
| <code>FA_DIREC</code>  | Directory           |
| <code>FA_ARCH</code>   | Archive             |

**Return value** Upon successful completion, `_chmod` returns the file attribute word; otherwise, it returns a value of -1.

In the event of an error, the global variable *errno* is set to one of the following:

|                     |                             |
|---------------------|-----------------------------|
| <code>ENOENT</code> | Path or file name not found |
| <code>EACCES</code> | Permission denied           |

**Portability** `_chmod` is unique to DOS.

**See also** `chmod`, `_creat`

**Example**

```
#include <errno.h>
#include <stdio.h>
#include <dos.h>
#include <io.h>

int get_file_attrib(char *filename);
int main(void)
{
 char filename[128];
 int attrib;
 printf("Enter a file name:");
 scanf("%s", filename);
 attrib = get_file_attrib(filename);
```

```

if (attrib == -1)
 switch(errno) {
 case ENOENT : printf("Path or file not found.\n");
 break;
 case EACCES : printf("Permission denied.\n");
 break;
 default: printf("Error number: %d", errno);
 break;
 }
else {
 if (attrib & FA_RDONLY)
 printf("%s is read-only.\n", filename);

 if (attrib & FA_HIDDEN)
 printf("%s is hidden.\n", filename);

 if (attrib & FA_SYSTEM)
 printf("%s is a system file.\n", filename);

 if (attrib & FA_LABEL)
 printf("%s is a volume label.\n", filename);

 if (attrib & FA_DIREC)
 printf("%s is a directory.\n", filename);

 if (attrib & FA_ARCH)
 printf("%s is an archive file.\n", filename);
}
return 0;
}

/* returns the attributes of a DOS file */
int get_file_attrib(char *filename) {
 return(_chmod(filename, 0));
}

```

## chmod

---

- Function** Changes file access mode.
- Syntax** `#include <sys/stat.h>`  
`int chmod(const char *path, int amode);`
- Prototype in** io.h
- Remarks** **chmod** sets the file-access permissions of the file given by *path* according to the mask given by *amode*. *path* points to a string; *\*path* is the first character of that string.

*amode* can contain one or both of the symbolic constants `S_IWRITE` and `S_IREAD` (defined in `sys\stat.h`).

| Value of <i>amode</i>         | Access permission            |
|-------------------------------|------------------------------|
| <code>S_IWRITE</code>         | Permission to write          |
| <code>S_IREAD</code>          | Permission to read           |
| <code>S_IREAD S_IWRITE</code> | Permission to read and write |

**Return value** Upon successfully changing the file access mode, **chmod** returns 0. Otherwise, **chmod** returns a value of -1.

In the event of an error, the global variable *errno* is set to one of the following:

`ENOENT` Path or file name not found  
`EACCES` Permission denied

**Portability** **chmod** is available on UNIX systems.

**See also** **access**, **\_chmod**, **fstat**, **open**, **sopen**, **stat**

**Example**

```
#include <errno.h>
#include <stdio.h>
#include <dos.h>
#include <io.h>

int get_file_attrib(char *filename);
int main(void)
{
 char filename[128];
 int attrib;
 printf("Enter a file name:");
 scanf("%s", filename);
 attrib = get_file_attrib(filename);
 if (attrib == -1)
 switch(errno) {
 case ENOENT : printf("Path or file not found.\n");
 break;
 case EACCES : printf("Permission denied.\n");
 break;
 default: printf("Error number: %d", errno);
 break;
 }
 else {
 if (attrib & FA_RDONLY)
 printf("%s is read-only.\n", filename);
 if (attrib & FA_HIDDEN)
 printf("%s is hidden.\n", filename);
 }
}
```

```

 if (attrib & FA_SYSTEM)
 printf("%s is a system file.\n", filename);
 if (attrib & FA_LABEL)
 printf("%s is a volume label.\n", filename);
 if (attrib & FA_DIREC)
 printf("%s is a directory.\n", filename);
 if (attrib & FA_ARCH)
 printf("%s is an archive file.\n", filename);
 }
 return 0;
}

/* returns the attributes of a DOS file */
int get_file_attrib(char *filename)
{
 return(_chmod(filename, 0));
}

```

## chsize

---

**Function** Changes the file size.

**Syntax** `#include <io.h>`  
`int chsize(int handle, long size);`

**Prototype in** `io.h`

**Remarks** **chsize** changes the size of the file associated with *handle*. It can truncate or extend the file, depending on the value of *size* compared to the file's original size.

The mode in which you open the file must allow writing.

If **chsize** extends the file, it will append null characters (\0). If it truncates the file, all data beyond the new end-of-file indicator is lost.

**Return value** On success, **chsize** returns 0. On failure, it returns -1 and the global variable *errno* is set to one of the following:

|         |                   |
|---------|-------------------|
| EACCESS | Permission denied |
| EBADF   | Bad file number   |
| ENOSPC  | UNIX—not DOS      |

**Portability** **chsize** is unique to DOS.

**See also** `close`, `_creat`, `creat`, `open`

## chsize

### Example

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

int main(void)
{
 int handle;
 char buf[11] = "0123456789";

 /* create a text file containing 10 bytes */
 handle = open("DUMMY.FIL", O_CREAT);
 write(handle, buf, strlen(buf));

 /* truncate the file to 5 bytes in size */
 chsize(handle, 5);

 /* close the file */
 close(handle);
 return 0;
}
```

## circle

---

- Function** Draws a circle of the given radius with its center at  $(x,y)$ .
- Syntax** `#include <graphics.h>`  
`void far circle(int x, int y, int radius);`
- Prototype in** `graphics.h`
- Remarks** **circle** draws a circle in the current drawing color with its center at  $(x,y)$  and the radius given by *radius*.
- ➡ The *linestyle* parameter does not affect arcs, circles, ellipses, or pie slices. Only the *thickness* parameter is used.
- If your circles are not perfectly round, adjust the aspect ratio.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **arc**, **ellipse**, **fillellipse**, **getaspectratio**, **sector**, **setaspectratio**
- Example** `#include <graphics.h>`  
`#include <stdlib.h>`  
`#include <stdio.h>`

```

#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;
 int radius = 100;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;
 setcolor(getmaxcolor());

 /* draw the circle */
 circle(midx, midy, radius);

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

## \_clear87

---

|                     |                                                                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Clears floating-point status word.                                                                                                                                   |
| <b>Syntax</b>       | #include <float.h><br>unsigned int _clear87 (void);                                                                                                                  |
| <b>Prototype in</b> | float.h                                                                                                                                                              |
| <b>Remarks</b>      | <b>_clear87</b> clears the floating-point status word, which is a combination of the 80x87 status word and other conditions detected by the 80x87 exception handler. |

## **\_clear87**

**Return value** The bits in the value returned indicate the floating-point status before it was cleared. For information on the status word, refer to the constants defined in `float.h`.

**Portability** `_clear87` is unique to DOS.

**See also** `_control87`, `_fpreset`, `_status87`

**Example**

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

int main(void)
{
 float x;
 double y = 1.5e-100;
 printf("\nStatus 87 before error: %X\n", _status87());
 x = y; /* create underflow and precision loss */
 printf("Status 87 after error: %X\n", _status87());
 _clear87();
 printf("Status 87 after clear: %X\n", _status87());
 y = x;
 return 0;
}
```

## cleardevice

---

**Function** Clears the graphics screen.

**Syntax** `#include <graphics.h>`  
`void far cleardevice(void);`

**Prototype in** `graphics.h`

**Remarks** `cleardevice` erases (that is, fills with the current background color) the entire graphics screen and moves the CP (current position) to home (0,0).

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** `clearviewport`

**Example**

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

```
int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;
 setcolor(getmaxcolor());

 /* for centering screen messages */
 setttextjustify(CENTER_TEXT, CENTER_TEXT);

 /* output a message to the screen */
 outtextxy(midx, midy, "Press any key to clear the screen:");

 /* wait for a key */
 getch();

 /* clear the screen */
 cleardevice();

 /* output another message */
 outtextxy(midx, midy, "Press any key to quit:");

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```



## clearerr

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Resets error indication.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Syntax</b>       | <pre>#include &lt;stdio.h&gt; void clearerr(FILE *stream);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Remarks</b>      | <p><b>clearerr</b> resets the named stream's error and end-of-file indicators to 0. Once the error indicator is set, stream operations continue to return error status until a call is made to <b>clearerr</b> or <b>rewind</b>.</p> <p>The end-of-file indicator is reset with each input operation.</p>                                                                                                                                                                                 |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Portability</b>  | <b>clearerr</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>See also</b>     | <b>eof, feof, ferror, perror, rewind</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;  int main(void) {     FILE *fp;     char ch;      /* open a file for writing */     fp = fopen("DUMMY.FIL", "w");      /* force an error condition by attempting to read */     ch = fgetc(fp);     printf("%c\n",ch);      if (ferror(fp)) {         /* display an error message */         printf("Error reading from DUMMY.FIL\n");          /* reset the error and EOF indicators */         clearerr(fp);     }      fclose(fp);     return 0; }</pre> |

# clearviewport

---

- Function** Clears the current viewport.
- Syntax** `#include <graphics.h>`  
`void far clearviewport(void);`
- Prototype in** `graphics.h`
- Remarks** **clearviewport** erases the viewport and moves the CP (current position) to home (0,0), relative to the viewport.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **cleardevice, getviewsettings, setviewport**

**Example**

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

#define CLIP_ON 1 /* activates clipping in viewport */

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int ht;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 setcolor(getmaxcolor());
 ht = textheight("W");
```

## clearviewport

```
/* message in default full-screen viewport */
outtextxy(0, 0, "* <-- (0, 0) in default viewport");

/* create a smaller viewport */
setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

/* display some messages */
outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");
outtextxy(0, 2*ht, "Press any key to clear viewport:");

/* wait for a key */
getch();

/* clear the viewport */
clearviewport();

/* output another message */
outtextxy(0, 0, "Press any key to quit:");

/* clean up */
getch();
closegraph();
return 0;
}
```

## clock

---

|                     |                                                                                                                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Determines processor time.                                                                                                                                                                                                         |
| <b>Syntax</b>       | <pre>#include &lt;time.h&gt; clock_t clock(void);</pre>                                                                                                                                                                            |
| <b>Prototype in</b> | time.h                                                                                                                                                                                                                             |
| <b>Remarks</b>      | <p><b>clock</b> can be used to determine the time interval between two events.</p> <p>To determine the time in seconds, the value returned by <b>clock</b> should be divided by the value of the macro <b>CLK_TCK</b>.</p>         |
| <b>Return value</b> | The <b>clock</b> function returns the processor time elapsed since the beginning of the program invocation. If the processor time is not available, or its value cannot be represented, the function returns the value <b>-1</b> . |
| <b>Portability</b>  | <b>clock</b> is compatible with ANSI C.                                                                                                                                                                                            |
| <b>See also</b>     | <b>time</b>                                                                                                                                                                                                                        |
| <b>Example</b>      | <pre>#include &lt;time.h&gt; #include &lt;stdio.h&gt; #include &lt;dos.h&gt;</pre>                                                                                                                                                 |

```

int main(void)
{
 clock_t start, end;
 start = clock();
 delay(2000);
 end = clock();
 printf("The time was: %f\n", (end - start) / CLK_TCK);
 return 0;
}

```

## \_close

---

**Function** Closes a file.

**Syntax** `#include <io.h>`  
`int _close(int handle);`

**Prototype in** `io.h`

**Remarks** `_close` closes the file associated with *handle*. *handle* is a file handle obtained from a `_creat`, `creat`, `creatnew`, `creattemp`, `dup`, `dup2`, `_open`, or `open` call.



This function does not write a *Ctrl-Z* character at the end of the file. If you want to terminate the file with a *Ctrl-Z*, you must explicitly output one.

**Return value** Upon successful completion, `_close` returns 0. Otherwise, it returns a value of -1.

`_close` fails if *handle* is not the handle of a valid, open file, and the global variable *errno* is set to

`EBADF` Bad file number

**Portability** `_close` is unique to DOS.

**See also** `close`, `_creat`, `open`, `read`, `write`

**Example**

```

#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
 int handle;
 char buf[11] = "0123456789";

 /* create a file containing 10 bytes */
 handle = open("DUMMY.FIL", O_CREAT);

```


## `_close`

```
write(handle, buf, strlen(buf));

/* close the file */
_close(handle);
return 0;
}
```

## close

---

|                                                                                   |                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>                                                                   | Closes a file.                                                                                                                                                                                                                                                                                                                                  |
| <b>Syntax</b>                                                                     | <code>#include &lt;io.h&gt;</code><br><code>int close(int <i>handle</i>);</code>                                                                                                                                                                                                                                                                |
| <b>Prototype in</b>                                                               | <code>io.h</code>                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>                                                                    | <b>close</b> closes the file associated with <i>handle</i> , a file handle obtained from a <b>_creat</b> , <b>creat</b> , <b>creatnew</b> , <b>creattemp</b> , <b>dup</b> , <b>dup2</b> , <b>_open</b> , or <b>open</b> call.                                                                                                                   |
|  | This function does not write a <i>Ctrl-Z</i> character at the end of the file. If you want to terminate the file with a <i>Ctrl-Z</i> , you must explicitly output one.                                                                                                                                                                         |
| <b>Return value</b>                                                               | Upon successful completion, <b>close</b> returns 0. Otherwise, a value of -1 is returned.                                                                                                                                                                                                                                                       |
|                                                                                   | <b>close</b> fails if <i>handle</i> is not the handle of a valid, open file, and the global variable <i>errno</i> is set to                                                                                                                                                                                                                     |
|                                                                                   | EBADF Bad file number                                                                                                                                                                                                                                                                                                                           |
| <b>Portability</b>                                                                | <b>close</b> is available on UNIX systems.                                                                                                                                                                                                                                                                                                      |
| <b>See also</b>                                                                   | <b>chsize</b> , <b>_close</b> , <b>creat</b> , <b>creatnew</b> , <b>dup</b> , <b>fclose</b> , <b>open</b> , <b>sopen</b>                                                                                                                                                                                                                        |
| <b>Example</b>                                                                    | <pre>#include &lt;string.h&gt; #include &lt;fcntl.h&gt; #include &lt;io.h&gt;  int main(void) {     int handle;     char buf[11] = "0123456789";      /* create a file containing 10 bytes */     handle = open("DUMMY.FIL", O_CREAT);     write(handle, buf, strlen(buf));      /* close the file */     _close(handle);     return 0; }</pre> |

# closegraph

C

**Function** Shuts down the graphics system.

**Syntax** `#include <graphics.h>`  
`void far closegraph(void);`

**Prototype in** `graphics.h`

**Remarks** **closegraph** deallocates all memory allocated by the graphics system, then restores the screen to the mode it was in before you called **initgraph**. (The graphics system deallocates memory, such as the drivers, fonts, and an internal buffer, through a call to **\_graphfreemem**.)

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **initgraph**, **setgraphbufsize**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int x, y;

 /* initialize graphics mode */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();

 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 x = getmaxx() / 2;
 y = getmaxy() / 2;
```

## closegraph

```
 /* output a message */
 settextjustify(CENTER_TEXT, CENTER_TEXT);
 outtextxy(x, y, "Press a key to close the graphics system:");

 /* wait for a key */
 getch();

 /* closes down the graphics system */
 closegraph();
 printf("We're now back in text mode.\n");
 printf("Press any key to halt:");
 getch();
 return 0;
}
```

## clreol

---

- Function** Clears to end of line in text window.
- Syntax** `#include <conio.h>`  
`void clreol(void);`
- Prototype in** conio.h
- Remarks** **clreol** clears all characters from the cursor position to the end of the line within the current text window, without moving the cursor.
- Return value** None.
- Portability** **clreol** works with IBM PCs and compatibles only. This function is not Windows compatible.
- See also** **clrscr, delline, window**
- Example**
- ```
#include <conio.h>

int main(void)
{
    clrscr();
    printf("The function CLREOL clears all characters from the\r\n");
    printf("cursor position to the end of the line within the\r\n");
    printf("current text window, without moving the cursor.\r\n");
    printf("Press any key to continue . . .");
    gotoxy(14, 4);
    getch();
    clreol();
    getch();
    return 0;
}
```

clrscr

C

Function	Clears the text-mode window.
Syntax	<code>#include <conio.h></code> <code>void clrscr(void);</code>
Prototype in	<code>conio.h</code>
Remarks	clrscr clears the current text window and places the cursor in the upper left-hand corner (at position 1,1).
Return value	None.
Portability	clrscr works with IBM PCs and compatibles only. This function is not Windows compatible.
See also	clreol, delline, window
Example	<pre>#include <conio.h> int main(void) { int i; clrscr(); for (i = 0; i < 20; i++) printf("%d\r\n", i); printf("\r\nPress any key to clear screen"); getch(); clrscr(); printf("The screen has been cleared!"); getch(); return 0; }</pre>

complex

Function	Creates complex numbers.
Syntax	<code>#include <complex.h></code> <code>complex complex(double <i>real</i>, double <i>imag</i>);</code>
Prototype in	<code>complex.h</code>
Remarks	Creates a complex number out of the given real and imaginary parts. The imaginary part is taken to be 0 if <i>imag</i> is omitted.

complex is the constructor for the C++ class **complex**, which is defined in `complex.h`. Other applicable functions (listed under **See also** below) are also defined in `complex.h`. Some of these are overloaded versions of C library functions declared in `math.h`. C++ is required for the complex versions.

If you don't want to program in C++, but instead want to program in C, the only constructs available to you are **struct complex** and **cabs**, which give the absolute value of a complex number. Both of these are defined in `math.h`.

`complex.h` also overloads the operators `+`, `-`, `*`, `/`, `+=`, `-=`, `*=`, `/=`, `=`, `==`, and `!=`. These operators give complex arithmetic in the usual sense. You can freely mix complex numbers in expressions with **ints**, **doubles**, and other numeric types. The operators `<<` and `>>` are overloaded for stream input and output of complex numbers, as they are for other data types in `iostream.h`.

- Return value** The complex number with the given real and imaginary parts.
- Portability** Complex functions require C++ and are not portable. Since we are following the AT&T C++ version 2.0, you *may* be able to port complex functions to C++ compilers compatible with that standard.
- See also** **abs**, **acos**, **arg**, **asin**, **atan**, **atan2**, **conj**, **cos**, **cosh**, **imag**, **log**, **log10**, **norm**, **polar**, **pow**, **real**, **sin**, **sinh**, **sqrt**, **tan**, **tanh**

Example

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
    double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
    cout << " has real part = " << real(z) << "\n";
    cout << " and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";
    return 0;
}
```

conj

- Function** Returns the complex conjugate of a complex number.
- Syntax** `#include <complex.h>`
`complex conj(complex x);`
- Prototype in** `complex.h`
- Remarks** `conj(z)` is the same as `complex(real(z), -imag(z))`.
- Return value** The complex conjugate of the complex number.
- Portability** Complex functions require C++ and are not portable.
- See also** **complex, imag, real**

Example

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
    double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
    cout << " has real part = " << real(z) << "\n";
    cout << " and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";
    return 0;
}
```

_control87

- Function** Manipulates the floating-point control word.
- Syntax** `#include <float.h>`
`unsigned int _control87(unsigned int newcw, unsigned int mask);`
- Prototype in** `float.h`
- Remarks** `_control87` retrieves or changes the floating-point control word.
- The floating-point control word is an **unsigned int** that, bit by bit, specifies certain modes in the floating-point package; namely, the precision, infinity, and rounding modes. Changing these modes allows you to mask or unmask floating-point exceptions.

`_control87`

`_control87` matches the bits in *mask* to the bits in *newcw*. If a *mask* bit equals 1, the corresponding bit in *newcw* contains the new value for the same bit in the floating-point control word, and `_control87` sets that bit in the control word to the new value.

Here's a simple illustration:

Original control word:	0100	0011	0110	0011
<i>mask</i>	1000	0001	0100	1111
<i>newcw</i>	1110	1001	0000	0101
Changing bits	1xxx	xxx1	x0xx	0101

If *mask* equals 0, `_control87` returns the floating-point control word without altering it.

`_control87` does not change the Denormal bit because Borland C++ uses denormal exceptions.

- Return value** The bits in the value returned reflect the new floating-point control word. For a complete definition of the bits returned by `_control87`, see the header file `float.h`.
- Portability** `_control87` is unique to DOS.
- See also** `_clear87`, `_fpreset`, `signal`, `_status87`
- Example** `Control.c`

`coreleft`

- Function** Returns a measure of unused RAM memory.
- Syntax** *In the tiny, small, and medium models:*
`#include <alloc.h>`
`unsigned coreleft(void);`
In the compact, large, and huge models:
`#include <alloc.h>`
`unsigned long coreleft(void);`
- Prototype in** `alloc.h`
- Remarks** `coreleft` returns a measure of RAM memory not in use. It gives a different measurement value, depending on whether the memory model is of the small data group or the large data group.

Return value In the small data models, **coreleft** returns the amount of unused memory between the top of the heap and the stack. In the large data models, **coreleft** returns the amount of memory between the highest allocated block and the end of available memory.

Portability **coreleft** is unique to DOS. This function is not Windows compatible.

See also **allocmem, brk, farcoreleft, malloc**

Example

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

int main(void)
{
    printf("The difference between the highest allocated block and\n");
    printf("the top of the heap is: %lu bytes\n", (unsigned long) coreleft());
    return 0;
}
```

COS

Function Calculates the cosine of a value.

Syntax

<i>Real version:</i>	<i>Complex version:</i>
<code>#include <math.h></code>	<code>#include <complex.h></code>
<code>double cos(double x);</code>	<code>complex cos(complex x);</code>

Prototype in

<i>Real version:</i>	<i>Complex version:</i>
<code>math.h</code>	<code>complex.h</code>

Remarks **cos** computes the cosine of the input value. The angle is specified in radians.

The complex cosine is defined by

$$\cos(z) = (\exp(i * z) + \exp(-i * z))/2$$

Return value **cos** of a real argument returns a value in the range -1 to 1.

Error handling for this routine can be modified through **matherr**.

Portability The real version of **cos** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and is not portable.

See also **acos, asin, atan, atan2, complex, matherr, sin, tan**

Example

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

```
int main(void)
{
    double result;
    double x = 0.5;
    result = cos(x);
    printf("The cosine of %lf is %lf\n", x, result);
    return 0;
}
```

cosh

Function	Calculates the hyperbolic cosine of a value.	
Syntax	<i>Real version:</i> #include <math.h> double cosh(double x);	<i>Complex version:</i> #include <complex.h> complex cosh(complex x);
Prototype in	<i>Real version:</i> math.h	<i>Complex version:</i> complex.h
Remarks	<p>cosh computes the hyperbolic cosine, $(e^x + e^{-x})/2$.</p> <p>The complex hyperbolic cosine is defined by</p> $\cosh(z) = (\exp(z) + \exp(-z))/2$	
Return value	<p>cosh returns the hyperbolic cosine of the argument.</p> <p>When the correct value would create an overflow, cosh returns the value HUGE_VAL with the appropriate sign, and the global variable <i>errno</i> is set to ERANGE.</p> <p>Error handling for cosh can be modified through matherr.</p>	
Portability	The real version of cosh is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.	
See also	acos, asin, atan, atan2, complex, cos, matherr, sin, sinh, tan, tanh	
Example	<pre>#include <stdio.h> #include <math.h> int main(void) { double result; double x = 0.5; result = cosh(x); printf("The hyperboic cosine of %lf is %lf\n", x, result); }</pre>	

```

    return 0;
}

```

country

Function Returns country-dependent information.

Syntax `#include <dos.h>`
`struct COUNTRY *country(int xcode, struct country *cp);`

Prototype in `dos.h`

Remarks **country** specifies how certain country-dependent data (such as dates, times, and currency) will be formatted. The values set by this function depend on the DOS version being used.

If *cp* has a value of `-1`, the current country is set to the value of *xcode*, which must be nonzero. Otherwise, the `COUNTRY` structure pointed to by *cp* is filled with the country-dependent information of the current country (if *xcode* is set to 0), or the country given by *xcode*.

The structure `COUNTRY` is defined as follows:

```

struct country {
    int co_date;           /* date format */
    char co_curr[5];      /* currency symbol */
    char co_thsep[2];     /* thousands separator */
    char co_dese[2];      /* decimal separator */
    char co_dtsep[2];    /* date separator */
    char co_tmsep[2];    /* time separator */
    char co_currstyle;    /* currency style */
    char co_digits;      /* significant digits in currency */
    char co_time;        /* time format */
    long co_case;        /* case map */
    char co_dasep[2];    /* data separator */
    char co_fill[10];    /* filler */
};

```

The date format in *co_date* is

- 0 for the U.S. style of month, day, year
- 1 for the European style of day, month, year
- 2 for the Japanese style of year, month, day

Currency display style is given by *co_currstyle* as follows:

- 0 Currency symbol precedes value with no spaces between the symbol and the number.

country

- 1 Currency symbol follows value with no spaces between the number and the symbol.
- 2 Currency symbol precedes value with a space after the symbol.
- 3 Currency symbol follows the number with a space before the symbol.

Return value On success, **country** returns the pointer argument *cp*. On error, it returns null.

Portability **country** is available only with DOS version 3.0 and greater.

Example

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

#define USA 0

int main(void)
{
    struct COUNTRY country_info;
    country(USA, &country_info);
    printf("The currency symbol for the USA is: %s\n", country_info.co_curr);
    return 0;
}
```

cprintf

Function Writes formatted output to the screen.

Syntax `#include <conio.h>`
`int cprintf(const char *format[, argument, ...]);`

Prototype in conio.h

Remarks **cprintf** accepts a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data directly to the current text window on the screen. There must be the same number of format specifiers as arguments.

See *printf* for details on format specifiers.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable *directvideo*.

Unlike **fprintf** and **printf**, **cprintf** does not translate linefeed characters (`\n`) into carriage-return/linefeed character pairs (`\r\n`).

Return value **cprintf** returns the number of characters output.

Portability **cprintf** works with IBM PCs and compatibles only. This function is not Windows compatible.

See also *directvideo* (global variable), **fprintf**, **printf**, **putch**, **sprintf**, **vprintf**

Example

```
#include <conio.h>

int main(void)
{
    /* clear the screen */
    clrscr();

    /* create a text window */
    window(10, 10, 80, 25);

    /* output some text in the window */
    cprintf("Hello world\r\n");

    /* wait for a key */
    getch();
    return 0;
}
```



cputs

Function Writes a string to the screen.

Syntax `#include <conio.h>`
`int cputs(const char *str);`

Prototype in conio.h

Remarks **cputs** writes the null-terminated string *str* to the current text window. It does not append a newline character.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable *directvideo*.

Unlike **puts**, **cputs** does not translate linefeed characters (`\n`) into carriage-return/linefeed character pairs (`\r\n`).

Return value **cputs** returns the last character printed.

Portability **cputs** works with IBM PCs and compatibles only. This function is not Windows compatible.

See also **cgets**, *directvideo* (global variable), **fputs**, **putch**, **puts**

Example

```
#include <conio.h>

int main(void)
{
```



```

    /* clear the screen */
    clrscr();

    /* create a text window */
    window(10, 10, 80, 25);

    /* output some text in the window */
    cputs("This is within the window\r\n");

    /* wait for a key */
    getch();
    return 0;
}

```

_creat

Function	Creates a new file or overwrites an existing one.
Syntax	<pre>#include <dos.h> int _creat(const char *path, int attrib);</pre>
Prototype in	io.h
Remarks	<p>_creat accepts <i>attrib</i>, a DOS attribute word. The file is always opened in binary mode. Upon successful file creation, the file pointer is set to the beginning of the file. The file is opened for both reading and writing.</p> <p>If the file already exists, its size is reset to 0. (This is essentially the same as deleting the file and creating a new file with the same name.)</p> <p>The <i>attrib</i> argument to _creat can be one of the following constants (defined in dos.h):</p> <pre> FA_RDONLY Read-only attribute FA_HIDDEN Hidden file FA_SYSTEM System file </pre>
Return value	<p>Upon successful completion, _creat returns the new file handle, a non-negative integer; otherwise, it returns -1.</p> <p>In the event of error, the global variable <i>errno</i> is set to one of the following:</p> <pre> ENOENT Path or file name not found EMFILE Too many open files EACCES Permission denied </pre>
Portability	_creat is unique to DOS.
See also	_chmod , chsize , _close , close , creat , creatnew , creattemp

**Example**

```

#include <sys\stat.h>
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <io.h>

int main(void)
{
    int handle;
    char buf[] = "0123456789";

    /* create a binary file for reading and writing */
    if ((handle = _creat("DUMMY.FIL", 0)) < 0) {
        switch(errno) {
            case ENOENT: printf("Error: Path or file not found.\n");
                        break;
            case EMFILE: printf("Error: Too many open files.\n");
                        break;
            case EACCES: printf("Error: Permission denied.\n");
                        break;
            default:     printf("Error creating file.\n");
                        break;
        }
        exit(1);
    }

    /* write a string and NULL terminator into the file */
    write(handle, buf, strlen(buf)+1);

    /* close the file */
    close(handle);
    return 0;
}

```

creat

Function Creates a new file or overwrites an existing one.

Syntax #include <sys\stat.h>
int creat(const char *path, int amode);

Prototype in io.h

Remarks **creat** creates a new file or prepares to rewrite an existing file given by *path*. *amode* applies only to newly created files.

creat

A file created with **creat** is always created in the translation mode specified by the global variable *_fmode* (O_TEXT or O_BINARY).

If the file exists and the write attribute is set, **creat** truncates the file to a length of 0 bytes, leaving the file attributes unchanged. If the existing file has the read-only attribute set, the **creat** call fails and the file remains unchanged.

The **creat** call examines only the S_IWRITE bit of the access-mode word *amode*. If that bit is 1, the file can be written to. If the bit is 0, the file is marked as read-only. All other DOS attributes are set to 0.

amode can be one of the following (defined in `sys\stat.h`):

Value of <i>amode</i>	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD S_IWRITE	Permission to read and write



In DOS, write permission implies read permission.

Return value

Upon successful completion, **creat** returns the new file handle, a non-negative integer; otherwise, it returns -1.

In the event of error, the global variable *errno* is set to one of the following:

ENOENT	Path or file name not found
EMFILE	Too many open files
EACCES	Permission denied

Portability **creat** is available on UNIX systems.

See also **chmod**, **chsize**, **close**, **_creat**, **creatnew**, **creattemp**, **dup**, **dup2**, *_fmode* (global variable), **fopen**, **open**, **sopen**, **write**

Example

```
#include <sys\stat.h>
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <io.h>

int main(void)
{
    int handle;
    char buf[] = "0123456789";

    /* create a binary file for reading and writing */
    if ((handle = _creat("DUMMY.FIL", 0)) < 0) {
```

```

switch(errno) {
    case ENOENT: printf("Error: Path or file not found.\n");
                break;
    case EMFILE: printf("Error: Too many open files.\n");
                break;
    case EACCES: printf("Error: Permission denied.\n");
                break;
    default:     printf("Error creating file.\n");
                break;
}
exit(1);
}

/* write a string and NULL terminator into the file */
write(handle, buf, strlen(buf)+1);

/* close the file */
close(handle);
return 0;
}

```



creatnew

Function Creates a new file.

Syntax `#include <dos.h>`
`int creatnew(const char *path, int mode);`

Prototype in io.h

Remarks **creatnew** is identical to **_creat** with one exception. If the file exists, **creatnew** returns an error and leaves the file untouched.

The *mode* argument to **creatnew** can be one of the following constants (defined in dos.h):

FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file

Return value Upon successful completion, **creat** returns the new file handle, a non-negative integer; otherwise, it returns -1.

In the event of error, the global variable *errno* is set to one of the following:

EEXIST	File already exists
ENOENT	Path or file name not found
EMFILE	Too many open files

EACCES Permission denied

Portability **creatnew** is unique to DOS 3.0; it doesn't work under earlier versions of DOS.

See also **close**, **_creat**, **creat**, **creattemp**, **dup**, **_fmode** (global variable), **open**

Example

```
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <dos.h>
#include <io.h>

int main(void)
{
    int handle;
    char buf[11] = "0123456789";

    /* attempt to create a file that doesn't already exist */
    handle = creatnew("DUMMY.FIL", 0);

    if (handle == -1)
        printf("DUMMY.FIL already exists.\n");
    else {
        printf("DUMMY.FIL successfully created.\n");
        write(handle, buf, strlen(buf));
        close(handle);
    }
    return 0;
}
```

creattemp

Function Creates a unique file in the directory associated with the path name.

Syntax `#include <dos.h>`
`int creattemp(char *path, int attrib);`

Prototype in io.h

Remarks A file created with **creattemp** is always created in the translation mode specified by the global variable **_fmode** (O_TEXT or O_BINARY).

path is a path name ending with a backslash (\). A unique file name is selected in the directory given by *path*. The newly created file name is stored in the *path* string supplied. *path* should be long enough to hold the resulting file name. The file is not automatically deleted when the program terminates.

createmp accepts *attrib*, a DOS attribute word. The file is always opened in binary mode. Upon successful file creation, the file pointer is set to the beginning of the file. The file is opened for both reading and writing.

The *attrib* argument to **createmp** can be one of the following constants (defined in `dos.h`):

FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file

Return value Upon successful completion, the new file handle, a nonnegative integer, is returned; otherwise, `-1` is returned.

In the event of error, the global variable *errno* is set to one of the following:

ENOENT	Path or file name not found
EMFILE	Too many open files
EACCES	Permission denied

Portability **createmp** is unique to DOS 3.0 and will not work on earlier versions.

See also `close`, `_creat`, `creat`, `creatnew`, `dup`, `_fmode` (global variable), `open`

Example

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

int main(void)
{
    int handle;
    char pathname[128];
    strcpy(pathname, "\\");

    /* create a unique file in the root directory */
    handle = createmp(pathname, 0);
    printf("%s was the unique file created.\n", pathname);
    close(handle);
    return 0;
}
```

cscanf

Function Scans and formats input from the console.

Syntax `#include <conio.h>`
`int cscanf(char *format[, address, ...]);`

Prototype in conio.h

Remarks **cscanf** scans a series of input fields one character at a time, reading directly from the console. Then each field is formatted according to a format specifier passed to **cscanf** in the format string pointed to by *format*. Finally, **cscanf** stores the formatted input at an address passed to it as an argument following *format*, and echoes the input directly to the screen. There must be the same number of format specifiers and addresses as there are input fields.

See **scanf** for details on format specifiers.

cscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely for a number of reasons. See **scanf** for a discussion of possible causes.

Return value **cscanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If **cscanf** attempts to read at end-of-file, the return value is EOF.

Portability **cscanf** is unique to DOS. This function is not Windows compatible.

See also **fscanf**, **getche**, **scanf**, **sscanf**

Example

```
#include <conio.h>

int main(void)
{
    char string[80];

    /* clear the screen */
    clrscr();

    /* prompt the user for input */
    cprintf("Enter a string:");

    /* read the input */
    cscanf("%s", string);

    /* display what was read */
    cprintf("\r\nThe string entered is: %s", string);
    return 0;
}
```

ctime

Function Converts date and time to a string.

Syntax `#include <time.h>`
`char *ctime(const time_t *time);`

Prototype in time.h

Remarks **ctime** converts a time value pointed to by *time* (the value returned by the function **time**) into a 26-character string in the following form, terminating with a newline character and a null character:

```
Mon Nov 21 11:31:54 1983\n\0
```

All the fields have constant width.

Set the global long variable *timezone* to the difference in seconds between GMT and local standard time (in PST, *timezone* is 8×60×60). The global variable *daylight* is nonzero *if and only if* the standard U.S. daylight saving time conversion should be applied.

Return value **ctime** returns a pointer to the character string containing the date and time. The return value points to static data that is overwritten with each call to **ctime**.

Portability **ctime** is available on UNIX systems and is defined in ANSI C.

See also **asctime**, *daylight* (global variable), **difftime**, **ftime**, **getdate**, **gmtime**, **localtime**, **settime**, **time**, *timezone* (global variable), **tzset**

Example

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

int main(void)
{
    time_t t;
    t = time(NULL);
    printf("Today's date and time: %s\n", ctime(&t));
    return 0;
}
```


ctrlbrk

Function Sets control-break handler.

Syntax `#include <dos.h>`
`void ctrlbrk(int (*handler)(void));`

Prototype in dos.h

Remarks **ctrlbrk** sets a new control-break handler function pointed to by *handler*. The interrupt vector 0x23 is modified to call the named function.

ctrlbrk establishes a DOS interrupt handler that calls the named function; the named function is not called directly.

The handler function can perform any number of operations and system calls. The handler does not have to return; it can use **longjmp** to return to an arbitrary point in the program. The handler function returns 0 to abort the current program; any other value causes the program to resume execution.

Return value **ctrlbrk** returns nothing.

Portability **ctrlbrk** is unique to DOS.

See also **getcbrk**, **signal**

Example

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

#define ABORT 0

int c_break(void)
{
    printf("Control-Break pressed.  Program aborting ...\n");
    return (ABORT);
}

main()
{
    ctrlbrk(c_break);
    for(;;)
    {
        printf("Looping...\n");
    }
}
```

delay

D

Function	Suspends execution for an interval (milliseconds).
Syntax	<code>#include <dos.h></code> <code>void delay(unsigned <i>milliseconds</i>);</code>
Prototype in	dos.h
Remarks	With a call to delay , the current program is suspended from execution for the number of milliseconds specified by the argument <i>milliseconds</i> . It is no longer necessary to make a calibration call to delay before using it. delay is accurate to a millisecond.
Return value	None.
Portability	It works only with IBM PCs and compatibles. This function is not Windows compatible.
See also	nosound, sleep, sound
Example	<pre>/* emits a 440-Hz tone for 500 milliseconds */ #include <dos.h> int main(void) { sound(440); delay(500); nosound(); return 0; }</pre>

delline

Function	Deletes line in text window.
Syntax	<code>#include <conio.h></code> <code>void delline(void);</code>
Prototype in	conio.h
Remarks	delline deletes the line containing the cursor and moves all lines below it one line up. delline operates within the currently active text window.
Return value	None.

delline

Portability It works only with IBM PCs and compatibles. This function is not Windows compatible.

See also `clreol`, `clrscr`, `inline`, `window`

Example

```
#include <conio.h>

int main(void)
{
    clrscr();
    printf("The function DELLINE deletes the line containing the\r\n");
    printf("cursor and moves all lines below it one line up.\r\n");
    printf("DELLINE operates within the currently active text\r\n");
    printf("window. Press any key to continue . . .");

    /* move cursor to the 2nd line, 1st column */
    gotoxy(1,2);
    getch();
    delline();
    getch();
    return 0;
}
```

detectgraph

Function Determines graphics driver and graphics mode to use by checking the hardware.

Syntax `#include <graphics.h>`
`void far detectgraph(int far *graphdriver, int far *graphmode);`

Prototype in `graphics.h`

Remarks **detectgraph** detects your system's graphics adapter and chooses the mode that provides the highest resolution for that adapter. If no graphics hardware is detected, *graphdriver* is set to `grNotDetected (-2)`, and **graphresult** returns `grNotDetected (-2)`.

graphdriver is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the *graphics_drivers* enumeration type, defined in `graphics.h` and listed in the following table.

Table 2.1
detectgraph
constants

<i>graphics_drivers</i> constant	Numeric value
DETECT	0 (requests autodetection)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

**graphmode* is an integer that specifies the initial graphics mode (unless **graphdriver* equals DETECT; in which case, **graphmode* is set to the highest resolution available for the detected driver). You can give **graphmode* a value using a constant of the *graphics_modes* enumeration type, defined in *graphics.h* and listed in the following table.



Table 2.2
Graphics drivers
information

Graphics driver	graphics_modes	Value	Column × row	Palette	Pages
CGA	CGAC0	0	320 × 200	C0	1
	CGAC1	1	320 × 200	C1	1
	CGAC2	2	320 × 200	C2	1
	CGAC3	3	320 × 200	C3	1
	CGAHI	4	640 × 200	2 color	1
MCGA	MCGAC0	0	320 × 200	C0	1
	MCGAC1	1	320 × 200	C1	1
	MCGAC2	2	320 × 200	C2	1
	MCGAC3	3	320 × 200	C3	1
	MCGAMED	4	640 × 200	2 color	1
EGA	MCGAHI	5	640 × 480	2 color	1
	EGALO	0	640 × 200	16 color	4
EGA64	EGAHI	1	640 × 350	16 color	2
	EGA64LO	0	640 × 200	16 color	1
EGA64	EGA64HI	1	640 × 350	4 color	1
	EGA-MONO	EGAMONOH1	3	640 × 350	2 color
EGAMONOH1		3	640 × 350	2 color	2**
HERC	HERCMONOH1	0	720 × 348	2 color	2
ATT400	ATT400C0	0	320 × 200	C0	1
	ATT400C1	1	320 × 200	C1	1
	ATT400C2	2	320 × 200	C2	1
	ATT400C3	3	320 × 200	C3	1
	ATT400MED	4	640 × 200	2 color	1
	ATT400HI	5	640 × 400	2 color	1
VGA	VGALO	0	640 × 200	16 color	2
	VGAMED	1	640 × 350	16 color	2
	VGAHI	2	640 × 480	16 color	1
PC3270	PC3270HI	0	720 × 350	2 color	1
IBM8514	IBM8514HI	0	640 × 480	256 color	
	IBM8514LO	0	1024 × 768	256 color	

* 64K on EGAMONO card

** 256K on EGAMONO card

Note: The main reason to call **detectgraph** directly is to override the graphics mode that **detectgraph** recommends to **initgraph**.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **graphresult, initgraph**

```

Example #include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the various cards supported */
char *dname[] = { "requests detection",
                 "a CGA",
                 "an MCGA",
                 "an EGA",
                 "a 64K EGA",
                 "a monochrome EGA",
                 "an IBM 8514",
                 "a Hercules monochrome",
                 "an AT&T 6300 PC",
                 "a VGA",
                 "an IBM 3270 PC"
               };

int main(void)
{
    /* used to return detected hardware info. */
    int gdriver, gmode, errorcode;

    /* detect the graphics hardware available */
    detectgraph(&gdriver, &gmode);

    /* read result of detectgraph call */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* display the information detected */
    clrscr();
    printf("You have %s video display card.\n", dname[gdriver]);
    printf("Press any key to halt:");
    getch();
    return 0;
}

```

difftime

Function	Computes the difference between two times.
Syntax	<pre>#include <time.h> double difftime(time_t <i>time2</i>, time_t <i>time1</i>);</pre>
Prototype in	time.h
Remarks	difftime calculates the elapsed time in seconds, from <i>time1</i> to <i>time2</i> .
Return value	difftime returns the result of its calculation as a double .
Portability	difftime is available on UNIX systems and is defined in ANSI C.
See also	asctime , ctime , <i>daylight</i> (global variable), gmtime , localtime , time , <i>timezone</i> (global variable)
Example	<pre>#include <time.h> #include <stdio.h> #include <dos.h> #include <conio.h> int main(void) { time_t first, second; clrscr(); first = time(NULL); /* gets system time */ delay(2000); /* waits 2000 millisecs or 2 secs */ second = time(NULL); /* gets system time again */ printf("The difference is: %f seconds\n", difftime(second, first)); getch(); return 0; }</pre>

disable

Function	Disables interrupts.
Syntax	<pre>#include <dos.h> void disable(void);</pre>
Prototype in	dos.h
Remarks	disable is designed to provide a programmer with flexible hardware interrupt control.

The **disable** macro disables interrupts. Only the NMI (non-maskable interrupt) is allowed from any external device.

Return value None.

Portability This macro is unique to the 80x86 architecture.

See also **enable, getvect**

Example

```

/* NOTE: This is an interrupt service routine. You canNOT compile this program
with Test Stack Overflow turned on and get an executable file that will
operate correctly. */
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0x1C /* the clock tick interrupt */

void interrupt (*oldhandler)(void);
int count=0;

void interrupt handler(void)
{
    /* disable interrupts during handling of interrupt */
    disable();

    /* increase the global counter */
    count++;

    /* re-enable interrupts at end of handler */
    enable();

    /* call the old routine */
    oldhandler();
}

int main(void)
{
    /* save the old interrupt vector */
    oldhandler = getvect(INTR);

    /* install the new interrupt handler */
    setvect(INTR, handler);

    /* loop until the counter exceeds 20 */
    while (count < 20)
        printf("count is %d\n",count);

    /* reset the old interrupt handler */
    setvect(INTR, oldhandler);
    return 0;
}

```


div

Function Divides two integers, returning quotient and remainder.

Syntax `#include <stdlib.h>`
`div_t div(int numer, int denom);`

Prototype in `stdlib.h`

Remarks `div` divides two integers and returns both the quotient and the remainder as a `div_t` type. *numer* and *denom* are the numerator and denominator, respectively. The `div_t` type is a structure of integers defined (with **typedef**) in `stdlib.h` as follows:

```
typedef struct {
    int quot;    /* quotient */
    int rem;     /* remainder */
} div_t;
```

Return value `div` returns a structure whose elements are *quot* (the quotient) and *rem* (the remainder).

Portability `div` is compatible with ANSI C.

See also `ldiv`

Example

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

div_t x;

int main(void)
{
    x = div(10,3);
    printf("10 div 3 = %d remainder %d\n", x.quot, x.rem);
    return 0;
}
```

Program output

```
10 div 3 = 3 remainder 1
```

doxterr

D

Function Gets extended DOS error information.

Syntax `#include <dos.h>`
`int doxterr(struct DOSEERROR *eblkp);`

Prototype in dos.h

Remarks This function fills in the **DOSEERROR** structure pointed to by *eblkp* with extended error information after a DOS call has failed. The structure is defined as follows:

```
struct DOSEERROR {
    int de_exterror;    /* extended error */
    char de_class;     /* error class */
    char de_action;    /* action */
    char de_locus;     /* error locus */
};
```

The values in this structure are obtained by way of DOS call 0x59. A *de_exterror* value of 0 indicates that the prior DOS call did not result in an error.

Return value **doxterr** returns the value *de_exterror*.

Portability **doxterr** is unique to DOS 3.0; it doesn't work under earlier versions of DOS.

Example

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

int main(void)
{
    FILE *fp;
    struct DOSEERROR info;
    fp = fopen("perror.dat", "r");
    if (!fp) perror("Unable to open file for reading");
    doxterr(&info);
    printf("Extended DOS error information:\n");
    printf("  Extended error:   %d\n", info.de_exterror);
    printf("                Class:   %x\n", info.de_class);
    printf("                Action:  %x\n", info.de_action);
    printf("                Error Locus: %x\n", info.de_locus);
    return 0;
}
```

dostounix

Function	Converts date and time to UNIX time format.
Syntax	<code>#include <dos.h></code> <code>long dostounix(struct date *d, struct dostime *t);</code>
Prototype In	dos.h
Remarks	dostounix converts a date and time as returned from getdate and gettime into UNIX time format. <i>d</i> points to a date structure, and <i>t</i> points to a dostime structure containing valid DOS date and time information.
Return value	UNIX version of current date and time parameters: number of seconds since 00:00:00 on January 1, 1970 (GMT).
Portability	dostounix is unique to DOS.
See also	unixtodos
Example	<pre>#include <time.h> #include <stddef.h> #include <dos.h> #include <stdio.h> int main(void) { time_t t; struct time d_time; struct date d_date; struct tm *local; getdate(&d_date); gettime(&d_time); t = dostounix(&d_date, &d_time); local = localtime(&t); printf("Time and Date: %s\n", asctime(local)); return 0; }</pre>

drawpoly

Function Draws the outline of a polygon.

Syntax `#include <graphics.h>`
`void far drawpoly(int numpoints, int far *polypoints);`

Prototype in `graphics.h`

Remarks **drawpoly** draws a polygon with *numpoints* points, using the current line style and color.

**polypoints* points to a sequence of (*numpoints* × 2) integers. Each pair of integers gives the *x* and *y* coordinates of a point on the polygon.



In order to draw a closed figure with *n* vertices, you must pass *n* + 1 coordinates to **drawpoly** where the *n*th coordinate is equal to the 0th.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **fillpoly**, **floodfill**, **graphresult**, **setwritemode**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;

    /* our polygon array */
    int poly[10];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
    }
}
```

drawpoly

```
    getch();
    exit(1);          /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();
poly[0] = 20;        /* first vertex */
poly[1] = maxy / 2;
poly[2] = maxx - 20; /* second vertex */
poly[3] = 20;
poly[4] = maxx - 50; /* third vertex */
poly[5] = maxy - 20;
poly[6] = maxx / 2;  /* fourth vertex */
poly[7] = maxy / 2;
poly[8] = poly[0];    /* drawpoly doesn't automatically close */
poly[9] = poly[1];    /* the polygon, so we close it */

/* draw the polygon */
drawpoly(5, poly);

/* clean up */
getch();
closegraph();
return 0;
}
```

dup

Function	Duplicates a file handle.
Syntax	<pre>#include <io.h> int dup(int <i>handle</i>);</pre>
Prototype in	io.h
Remarks	<p>dup creates a new file handle that has the following in common with the original file handle:</p> <ul style="list-style-type: none">■ same open file or device■ same file pointer (that is, changing the file pointer of one changes the other)■ same access mode (read, write, read/write) <p><i>handle</i> is a file handle obtained from a _creat, creat, _open, open, dup, or dup2 call.</p>
Return value	Upon successful completion, dup returns the new file handle, a non-negative integer; otherwise, dup returns -1 .

In the event of error, the global variable *errno* is set to one of the following:

EMFILE	Too many open files
EBADF	Bad file number

Portability `dup` is available on all UNIX systems.

See also `_close`, `close`, `_creat`, `creat`, `creatnew`, `creattemp`, `dup2`, `fopen`, `_open`, `open`

Example

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

void flush(FILE *stream);

int main(void)
{
    FILE *fp;
    char msg[] = "This is a test";

    /* create a file */
    fp = fopen("DUMMY.FIL", "w");
    if (fp) {

        /* write some data to the file */
        fwrite(msg, strlen(msg), 1, fp);
        clrscr();
        printf("Press any key to flush DUMMY.FIL:");
        getch();

        /* flush the data to DUMMY.FIL without closing it */
        flush(fp);
        printf("\nFile was flushed, Press any key to quit:");
        getch();
    }
    else {
        printf("Error opening file!\n");
    }
    return 0;
}

void flush(FILE *stream)
{
    int duphandle;

    /* flush BC's internal buffer */
    fflush(stream);

    /* make a duplicate file handle */
    duphandle = dup(fileno(stream));
```

dup

```
    /* close duplicate handle to flush DOS buffer */  
    close(duphandle);  
}
```

dup2

Function	Duplicates a file handle (<i>oldhandle</i>) onto an existing file handle (<i>newhandle</i>).				
Syntax	<pre>#include <io.h> int dup2(int <i>oldhandle</i>, int <i>newhandle</i>);</pre>				
Prototype in	io.h				
Remarks	<p>dup2 creates a new file handle that has the following in common with the original file handle:</p> <ul style="list-style-type: none">■ same open file or device■ same file pointer (that is, changing the file pointer of one changes the other)■ same access mode (read, write, read/write) <p>dup2 creates a new handle with the value of <i>newhandle</i>. If the file associated with <i>newhandle</i> is open when dup2 is called, the file is closed. <i>newhandle</i> and <i>oldhandle</i> are file handles obtained from a creat, open, dup, or dup2 call.</p>				
Return value	<p>dup2 returns 0 on successful completion, -1 otherwise.</p> <p>In the event of error, the global variable <i>errno</i> is set to one of the following:</p> <table><tr><td>EMFILE</td><td>Too many open files</td></tr><tr><td>EBADF</td><td>Bad file number</td></tr></table>	EMFILE	Too many open files	EBADF	Bad file number
EMFILE	Too many open files				
EBADF	Bad file number				
Portability	dup2 is available on some UNIX systems, though not System III.				
See also	_close , close , _creat , creat , creatnew , creattemp , dup , fopen , _open , open				
Example	<pre>#include <sys\stat.h> #include <string.h> #include <fcntl.h> #include <io.h> #include <stdio.h> #define STDOUT 1 int main(void) { int fptr, oldstdout; char msg[] = "This is a test";</pre>				

```

/* create a file */
fptr = open("DUMMY.FIL", O_CREAT | O_RDWR, S_IRREAD | S_IWWRITE);
if (fptr) {
    /* create a duplicate handle for standard output */
    oldstdout = dup(STDOUT);

    /* redirect standard output to DUMMY.FIL by duplicating the */
    /* file handle onto the file handle for standard output */
    dup2(fptr, STDOUT);

    /* close the handle for DUMMY.FIL */
    close(fptr);

    /* this will be redirected into DUMMY.FIL */
    write(STDOUT, msg, strlen(msg));

    /* restore original standard output handle */
    dup2(oldstdout, STDOUT);

    /* close the duplicate handle for STDOUT */
    close(oldstdout);
}
else {
    printf("Error opening file!\n");
}
return 0;
}

```

ecvt

- Function** Converts a floating-point number to a string.
- Syntax** `#include <stdlib.h>`
`char *ecvt(double value, int ndig, int *dec, int *sign);`
- Prototype in** `stdlib.h`
- Remarks** **ecvt** converts *value* to a null-terminated string of *ndig* digits, starting with the leftmost significant digit, and returns a pointer to the string. The position of the decimal point relative to the beginning of the string is stored indirectly through *dec* (a negative value for *dec* means that the decimal lies to the left of the returned digits). There is no decimal point in the string itself. If the sign of *value* is negative, the word pointed to by *sign* is nonzero; otherwise, it's 0. The low-order digit is rounded.
- Return value** The return value of **ecvt** points to static data for the string of digits whose content is overwritten by each call to **ecvt**.

Portability **ecvt** is available on UNIX. It is not in ANSI C, and is not recommended for portable programs.

See also **fcvt**, **gcvt**, **sprintf**

Example

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

int main(void)
{
    char *string;
    double value;
    int dec, sign;
    int ndig = 10;
    value = 9.876;
    string = ecvt(value, ndig, &dec, &sign);
    printf("string = %s      dec = %d sign = %d\n", string, dec, sign);
    value = -123.45;
    ndig= 15;
    string = ecvt(value, ndig, &dec, &sign);
    printf("string = %s dec = %d sign = %d\n", string, dec, sign);
    value = 0.6789e5; /* scientific notation */
    ndig = 5;
    string = ecvt(value, ndig, &dec, &sign);
    printf("string = %s      dec = %d sign = %d\n", string, dec, sign);
    return 0;
}
```

ellipse

Function Draws an elliptical arc.

Syntax `#include <graphics.h>`
`void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);`

Prototype in graphics.h

Remarks **ellipse** draws an elliptical arc in the current drawing color with its center at (*x*,*y*) and the horizontal and vertical axes given by *xradius* and *yradius*, respectively. The ellipse travels from *stangle* to *endangle*. If *stangle* equals 0 and *endangle* equals 360, the call to **ellipse** draws a complete ellipse.

The angle for **ellipse** is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.



The *linestyle* parameter does not affect arcs, circles, ellipses, or pie slices. Only the *thickness* parameter is used.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **arc, circle, fillellipse, getaspectratio, sector, setaspectratio**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 0, endangle = 360;
    int xradius = 100, yradius = 50;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* draw ellipse */
    ellipse(midx, midy, stangle, endangle, xradius, yradius);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

__emit__

Function	Inserts literal values directly into code.
Syntax	<pre>#include <dos.h> void __emit__(argument, ...);</pre>
Prototype in	dos.h
Description	<p>__emit__ is an inline function that lets you insert literal values directly into object code as it is compiling. It is used to generate machine language instructions without using inline assembly language or an assembler.</p> <p>Generally the arguments of an __emit__ call are single-byte machine instructions. However, because of the capabilities of this function, more complex instructions, complete with references to C variables, can be constructed.</p> <p>➔ You should only use this function if you are familiar with the machine language of the 80x86 processor family. You can use this function to place arbitrary bytes in the instruction code of a function; if any of these bytes are incorrect, the program misbehaves and can easily crash your machine. Borland C++ does not attempt to analyze your calls for correctness in any way. If you encode instructions that change machine registers or memory, Borland C++ will not be aware of it and might not properly preserve registers, as it would in many cases with inline assembly language (for example, it recognizes the usage of SI and DI registers in inline instructions). You are completely on your own with this function.</p> <p>You must pass at least one argument to __emit__; any number can be given. The arguments to this function are not treated like any other function call arguments in the language. An argument passed to __emit__ will not be converted in any way.</p> <p>There are special restrictions on the form of the arguments to __emit__. They must be in the form of expressions that can be used to initialize a static object. This means that integer and floating-point constants and the addresses of static objects can be used. The values of such expressions are written to the object code at the point of the call, exactly as if they were being used to initialize data. The address of a parameter or auto variable, plus or minus a constant offset, may also be used. For these arguments, the offset of the variable from BP is stored.</p> <p>The number of bytes placed in the object code is determined from the type of the argument, except in the following cases:</p>



- If a signed integer constant (i.e. 0x90) appears that fits within the range of 0 to 255, it is treated as if it were a character.
- If the address of an auto or parameter variable is used, a byte is written if the offset of the variable from BP is between -128 and 127; otherwise, a word is written.

Simple bytes are written as follows:

```
__emit__(0x90);
```

If you want a word written, but the value you are passing is under 255, simply cast it to unsigned as follows:

```
__emit__(0xB8, (unsigned)17);
```

or

```
__emit__(0xB8, 17u);
```

Two- or four-byte address values can be forced by casting an address to **void near *** or **void far ***, respectively.

Return value None.

Portability __emit__ is unique to Intel 80x86 processors.

Example

```
#include <dos.h>

int main(void)
{
    /* emit code that generates a print screen via int 5 */
    __emit__(0xcd,0x05); /* INT 05h */
    return 0;
}
```

enable

Function Enables hardware interrupts.

Syntax #include <dos.h>
void enable(void);

Prototype in dos.h

Remarks **enable** is designed to provide a programmer with flexible hardware interrupt control.

The **enable** macro enables interrupts, allowing any device interrupts to occur.

enable

Return value None.

Portability **enable** is unique to the 80x86 architecture.

See also **disable**, **getvect**

Example

```
#include <dos.h>

void interrupt (*oldhandler)(void);
int count=0;
void interrupt handler(void)
{
    disable();                /* disable interrupts */
    count++;
    enable();                 /* enable interrupts */
}

int main(void)
{
    /* save the old user timer interrupt vector */
    oldhandler = getvect(0x1c);

    /* set the user timer interrupt vector to our handler */
    setvect(0x1c, handler);
    while (count < 20);      /* call handler 20 times */
    setvect(0x1c, oldhandler); /* restore original vector */
    return 0;
}
```

eof

Function Checks for end-of-file.

Syntax `#include <io.h>`
`int eof(int handle);`

Prototype in io.h

Remarks **eof** determines whether the file associated with *handle* has reached end-of-file.

Return value If the current position is end-of-file, **eof** returns the value 1; otherwise, it returns 0. A return value of -1 indicates an error; the global variable *errno* is set to

EBADF Bad file number

Portability **eof** is unique to DOS.

See also `clearerr`, `feof`, `ferror`, `perror`

Example

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <io.h>

int main(void)
{
    FILE *temp_file;
    int handle;
    char msg[] = "This is a test";
    char ch;

    /* create a unique temporary file */
    if ((temp_file = tmpfile()) == NULL) {
        perror("OPENING FILE:");
        exit(1);
    }

    /* get handle associated with file */
    handle = fileno(temp_file);

    /* write some data to the file */
    write(handle, msg, strlen(msg));

    /* seek to the beginning of the file */
    lseek(handle, 0L, SEEK_SET);

    /* reads chars from the file until EOF is hit */
    do {
        read(handle, &ch, 1);
        printf("%c", ch);
    } while (!eof(handle));

    /* close and remove the temporary file */
    fclose(temp_file);
    return 0;
}
```

E

`execl`, `execle`, `execlp`, `execlepe`, `execv`, `execve`,
`execvp`, `execvpe`

Function Loads and runs other programs.

Syntax `#include <process.h>`
`int execl(char *path, char *arg0 *arg1, ..., *argn, NULL);`
`int execle(char *path, char *arg0, *arg1, ..., *argn, NULL, char **env);`

execl, execlp, execlpe, execv, execve, execvp, execvpe

```
int execlp(char *path, char *arg0, *arg1, ..., *argn, NULL);
int execlpe(char *path, char *arg0, *arg1, ..., *argn, NULL, char **env);

int execv(char *path, char *argv[]);
int execve(char *path, char *argv[], char **env);

int execvp(char *path, char *argv[]);
int execvpe(char *path, char *argv[], char **env);
```

Prototype in process.h

Remarks The functions in the **exec...** family load and run (execute) other programs, known as *child processes*. When an **exec...** call succeeds, the child process overlays the *parent process*. There must be sufficient memory available for loading and executing the child process.

path is the file name of the called child process. The **exec...** functions search for *path* using the standard DOS search algorithm:

- If no explicit extension is given, the functions search for the file as given. If the file is not found, they add .COM and search again. If that search is not successful, they add .EXE and search one last time.
- If an explicit extension or a period is given, the functions search for the file exactly as given.

The suffixes *l*, *v*, *p*, and *e* added to the **exec...** “family name” specify that the named function operate with certain capabilities.

- p** The function searches for the file in those directories specified by the DOS PATH environment variable (without the *p* suffix, the function searches only the current working directory). If the *path* parameter does not contain an explicit directory, the function searches first the current directory, then the directories set with the DOS PATH environment variable.
- l** The argument pointers (*arg0*, *arg1*, ..., *argn*) are passed as separate arguments. Typically, the *l* suffix is used when you know in advance the number of arguments to be passed.
- v** The argument pointers (*argv[0]* ..., *argv[n]*) are passed as an array of pointers. Typically, the *v* suffix is used when a variable number of arguments is to be passed.
- e** The argument *env* can be passed to the child process, letting you alter the environment for the child process. Without the *e* suffix, child processes inherit the environment of the parent process.

execl, execlp, execl, execlp, execv, execve, execvp, execvpe

Each function in the **exec...** family *must* have one of the two argument-specifying suffixes (either *l* or *v*). The *path search* and *environment inheritance* suffixes (*p* and *e*) are optional.

For example,

- **execl** is an **exec...** function that takes separate arguments, searches only the root or current directory for the child, and passes on the parent's environment to the child.
- **execvpe** is an **exec...** function that takes an array of argument pointers, incorporates PATH in its search for the child process, and accepts the *env* argument for altering the child's environment.

The **exec...** functions must pass at least one argument to the child process (*arg0* or *argv[0]*); this argument is, by convention, a copy of *path*. (Using a different value for this 0th argument won't produce an error.)

Under DOS 3.x, *path* is available for the child process; under earlier versions of DOS, the child process cannot use the passed value of the 0th argument (*arg0* or *argv[0]*).

When the *l* suffix is used, *arg0* usually points to *path*, and *arg1*, ..., *argn* point to character strings that form the new list of arguments. A mandatory null following *argn* marks the end of the list.

When the *e* suffix is used, you pass a list of new environment settings through the argument *env*. This environment argument is an array of character pointers. Each element points to a null-terminated character string of the form

envvar = value

where *envvar* is the name of an environment variable, and *value* is the string value to which *envvar* is set. The last element in *env* is null. When *env* is null, the child inherits the parents' environment settings.

The combined length of *arg0* + *arg1* + ... + *argn* (or of *argv[0]* + *argv[1]* + ... + *argv[n]*), including space characters that separate the arguments, must be less than 128 bytes. Null terminators are not counted.

When an **exec...** function call is made, any open files remain open in the child process.

Return value If successful, the **exec...** functions do not return. On error, the **exec...** functions return -1, and the global variable *errno* is set to one of the following:

E2BIG	Arg list too long
EACCES	Permission denied

E

execl, execl_e, execl_p, execl_p_e, execv, execv_e, execv_p, execv_p_e

EMFILE	Too many open files
ENOENT	Path or file name not found
ENOEXEC	Exec format error
ENOMEM	Not enough core

Portability `exec...` is unique to DOS.

See also `abort`, `atexit`, `_exit`, `exit`, `_fpreset`, `searchpath`, `spawn...`, `system`

Examples

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

int main(int argc, char *argv[])
{
    int i;
    printf("Child running ...\n");
    printf("%s\n", getenv("PATH"));
    for(i = 0; i < argc; i++)
        printf("argv[%d]: %s\n", i, argv[i]);
    return 0;
}
```

Each function has its own example program.

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

int main(int argc, char *argv[])
{
    int loop;
    printf("%s running...\n\n", argv[0]);
    if (argc == 1) /* check for only one command-line parameter */
    {
        printf("%s calling itself again...\n", argv[0]);
        execl(argv[0], argv[0], "ONE", "TWO", "THREE", NULL);
        perror("EXEC:");
        exit(1);
    }
    printf("%s called with arguments:\n", argv[0]);
    for (loop = 1; loop <= argc; loop++)
        puts(argv[loop]); /* display all command-line parameters */
    return 0;
}
```

```
#include <process.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <dos.h>
```

```
int main(int argc, char *argv[], char *env[])
{
    int loop;
    char *new_env[] = { "TESTING", NULL };
    printf("%s running...\n\n", argv[0]);
    if (argc == 1) /* check for only one command-line parameter */
    {
        printf("%s calling itself again...\n", argv[0]);
        execl(argv[0], argv[0], "ONE", "TWO", "THREE", NULL, new_env);
        perror("EXEC:");
        exit(1);
    }
    printf("%s called with arguments:\n", argv[0]);
    for (loop = 1; loop <= argc; loop++)
        puts(argv[loop]); /* display all command-line parameters */

    /* display the first environment parameter */
    printf("value of env[0]: %s\n", env[0]);
    return 0;
}
```

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

int main(int argc, char *argv[], char **envp)
{
    int i;
    printf("Command line arguments:\n");
    for (i=0; i < argc; ++i)
        printf("[%2d] %s\n", i, argv[i]);
    printf("About to exec child with arg1 arg2 ...\n");
    execlpe("CHILD.EXE", "CHILD.EXE", "arg1", "arg2", NULL, envp);
    perror("exec error");
    exit(1);
    return 0;
}
```

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

int main(int argc, char *argv[], char **envp)
{
    int i;
    printf("Command line arguments:\n");
    for (i=0; i < argc; ++i)
        printf("[%2d] %s\n", i, argv[i]);
}
```

execl, execl_e, execl_p, execlpe, execv, execve, execvp, execvpe

```
    printf("About to exec child with arg1 arg2 ...\\n");
    execlpe("CHILD.EXE", "CHILD.EXE", "arg1", "arg2", NULL, envp);
    perror("exec error");
    exit(1);
    return 0;
}

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
    int i;
    printf("Command line arguments:\\n");
    for (i=0; i<argc; i++)
        printf("[%2d] : %s\\n", i, argv[i]);
    printf("About to exec child with arg1 arg2 ...\\n");
    execv("CHILD.EXE", argv);
    perror("exec error");
    exit(1);
}

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[], char **envp)
{
    int i;
    printf("Command line arguments:\\n");
    for (i=0; i<argc; ++i)
        printf("[%2d] : %s\\n", i, argv[i]);
    printf("About to exec child with arg1 arg2 ...\\n");
    execve("CHILD.EXE", argv, envp);
    perror("exec error");
    exit(1);
}

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
    int i;
    printf("Command line arguments:\\n");
    for (i=0; i<argc; ++i)
```

```

        printf("[%2d] : %s\n", i, argv[i]);
    printf("About to exec child with arg1 arg2 ...\n");
    execvp("CHILD.EXE", argv);
    perror("exec error");
    exit(1);
}

#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[], char **envp)
{
    int i;
    printf("Command line arguments:\n");
    for (i=0; i<argc; ++i)
        printf("[%2d] : %s\n", i, argv[i]);
    printf("About to exec child with arg1 arg2 ...\n");
    execvp("CHILD.EXE", argv, envp);
    perror("exec error");
    exit(1);
}

```

_exit

Function	Terminates program.
Syntax#	#include <stdlib.h> void _exit(int <i>status</i>);
Prototype in	process.h, stdlib.h
Remarks	<p>_exit terminates execution without closing any files, flushing any output, or calling any exit functions.</p> <p>The calling process uses <i>status</i> as the exit status of the process. Typically a value of 0 is used to indicate a normal exit, and a nonzero value indicates some error.</p>
Return value	None.
Portability	_exit is available on UNIX systems.
See also	abort, atexit, exec..., exit, spawn...
Example	#include <stdlib.h> #include <stdio.h>

```
void done(void);  
int main(void)  
{  
    atexit(done);  
    _exit(0);  
    return 0;  
}  
  
void done()  
{  
    printf("hello\n");  
}
```

exit

Function	Terminates program.				
Syntax	<code>#include <stdlib.h></code> <code>void exit(int <i>status</i>);</code>				
Prototype in	<code>process.h</code> , <code>stdlib.h</code>				
Remarks	<p>exit terminates the calling process. Before termination, all files are closed, buffered output (waiting to be output) is written, and any registered "exit functions" (posted with atexit) are called.</p> <p><i>status</i> is provided for the calling process as the exit status of the process. Typically a value of 0 is used to indicate a normal exit, and a nonzero value indicates some error. It is set with one of the following</p> <table><tr><td><code>EXIT_SUCCESS</code></td><td>Normal program termination.</td></tr><tr><td><code>EXIT_FAILURE</code></td><td>Abnormal program termination; signal to operating system that program has terminated with an error.</td></tr></table>	<code>EXIT_SUCCESS</code>	Normal program termination.	<code>EXIT_FAILURE</code>	Abnormal program termination; signal to operating system that program has terminated with an error.
<code>EXIT_SUCCESS</code>	Normal program termination.				
<code>EXIT_FAILURE</code>	Abnormal program termination; signal to operating system that program has terminated with an error.				
Return value	None.				
Portability	exit is available on UNIX systems and is defined in ANSI C.				
See also	abort , atexit , exec... , _exit , keep , signal , spawn...				
Example	<pre>#include <stdlib.h> #include <conio.h> #include <stdio.h> int main(void) { int status;</pre>				

```

printf("Enter either 1 or 2\n");
status = getch();
exit(status - '0'); /* sets DOS error level */
return 0;          /* NOTE: This line is never reached */
}

```

exp

Function Calculates the exponential e to the x .

Syntax *Real version:* `#include <math.h>`
`double exp(double x);` *Complex version:* `#include <complex.h>`
`complex exp(complex x);`

Prototype in *Real version:* `math.h` *Complex version:* `complex.h`

Remarks **exp** calculates the exponential function e^x .

The complex exponential function is defined by

$$\exp(x + y i) = \exp(x) (\cos(y) + i \sin(y))$$

Return value **exp** returns e^x .

Sometimes the arguments passed to **exp** produce results that overflow or are incalculable. When the correct value overflows, **exp** returns the value `HUGE_VAL`. Results of excessively large magnitude will cause the global variable *errno* to be set to

`ERANGE` Result out of range

On underflow, **exp** returns 0.0, and the global variable *errno* is not changed.

Error handling for **exp** can be modified through **matherr**.

Portability **exp** is available on UNIX systems and is defined in ANSI C.

See also **frexp**, **ldexp**, **log**, **log10**, **matherr**, **pow**, **pow10**, **sqrt**

Example

```

#include <stdio.h>
#include <math.h>

int main(void)
{
    double result;
    double x = 4.0;
    result = exp(x);
    printf("'e' raised to the power of %lf (e ^ %lf) = %lf\n", x, x, result);
}

```

```

    return 0;
}

```

fabs

Function	Returns the absolute value of a floating-point number.
Syntax	<code>#include <math.h></code> <code>double fabs(double <i>x</i>);</code>
Prototype in	math.h
Remarks	fabs calculates the absolute value of <i>x</i> , a double .
Return value	fabs returns the absolute value of <i>x</i> .
Portability	fabs is available on UNIX systems and is defined in ANSI C.
See also	abs , cabs , labs
Example	<pre> #include <stdio.h> #include <math.h> int main(void) { float number = -1234.0; printf("number: %f absolute value: %f\n", number, fabs(number)); return 0; } </pre>

farcalloc

Function	Allocates memory from the far heap.
Syntax	<code>#include <alloc.h></code> <code>void far *farcalloc(unsigned long <i>nunits</i>, unsigned long <i>unitsz</i>);</code>
Prototype in	alloc.h
Remarks	<p>farcalloc allocates memory from the far heap for an array containing <i>nunits</i> elements, each <i>unitsz</i> bytes long.</p> <p>For allocating from the far heap, note that</p> <ul style="list-style-type: none"> ■ all available RAM can be allocated. ■ blocks larger than 64K can be allocated.

- far pointers (or huge pointers if blocks are larger than 64K) are used to access the allocated blocks.

In the compact, large, and huge memory models, **farcalloc** is similar, though not identical, to **calloc**. It takes **unsigned long** parameters, while **calloc** takes **unsigned** parameters.

A tiny model program cannot make use of **farcalloc**.

Return value **farcalloc** returns a pointer to the newly allocated block, or null if not enough space exists for the new block.

Portability **farcalloc** is unique to DOS.

See also **calloc**, **farcoreleft**, **farfree**, **farmalloc**, **malloc**

Example

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
    char far *fptr;
    char *str = "Hello";

    /* allocate memory for the far pointer */
    fptr = farcalloc(10, sizeof(char));
    if (fptr) {

        /* copy "Hello" into the allocated memory */
        /* NOTE: movedata is used because we might be in a small data
           model; in which case, a normal string copy routine cannot be
           used since it assumes the pointer size is near. */
        movedata(FP_SEG(str), FP_OFF(str), FP_SEG(fptr), FP_OFF(fptr),
                strlen(str));

        /* display string (note the F modifier) */
        printf("Far string is: %Fs\n", fptr);

        /* free the memory */
        farfree(fptr);
    }
    return 0;
}
```

F

farcoreleft

Function	Returns a measure of unused memory in far heap.
Syntax	<pre>#include <alloc.h> unsigned long farcoreleft(void);</pre>
Prototype in	alloc.h
Remarks	farcoreleft returns a measure of the amount of unused memory in the far heap beyond the highest allocated block. A tiny model program cannot make use of farcoreleft .
Return value	farcoreleft returns the total amount of space left in the far heap, between the highest allocated block and the end of available memory.
Portability	farcoreleft is unique to DOS. This function is not Windows compatible.
See also	coreleft , farcalloc , farmalloc
Example	<pre>#include <stdio.h> #include <alloc.h> int main(void) { printf("The difference between the highest allocated block in the far\n"); printf("heap and the top of the far heap is: %lu bytes\n", farcoreleft()); return 0; }</pre>

farfree

Function	Frees a block from far heap.
Syntax	<pre>#include <alloc.h> void farfree(void far * <i>block</i>);</pre>
Prototype in	alloc.h
Remarks	farfree releases a block of memory previously allocated from the far heap. A tiny model program cannot make use of farfree . In the small and medium memory models, blocks allocated by farmalloc cannot be freed with normal free , and blocks allocated with malloc cannot be freed with farfree . In these models, the two heaps are completely distinct.

Return value None.

Portability **farfree** is unique to DOS.

See also **farcalloc**, **farmalloc**

Example

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
    char far *fptr;
    char *str = "Hello";

    /* allocate memory for the far pointer */
    fptr = farmalloc(10);

    /* copy "Hello" into the allocated memory */
    /* NOTE: movedata is used in case you are in a small data model.
       If you are, a normal string copy routine can't be used because
       it assumes the pointer size is near. */
    movedata(FP_SEG(str), FP_OFF(str),
             FP_SEG(fptr), FP_OFF(fptr), strlen(str));

    /* display string (note the F modifier) */
    printf("Far string is: %Fs\n", fptr);

    /* free the memory */
    farfree(fptr);
    return 0;
}
```

F

farheapcheck

Function Checks and verifies the far heap.

Syntax #include <alloc.h>
int farheapcheck(void);

Prototype in alloc.h

Remarks **farheapcheck** walks through the far heap and examines each block, checking its pointers, size, and other critical attributes.

Return value The return value is less than zero for an error and greater than zero for success.

`_HEAPEMPTY` is returned if there is no heap (value 1).

farheapcheck

`_HEAPOK` is returned if the heap is verified (value 2).

`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).

Portability `farheapcheck` is unique to DOS. This function is not Windows compatible.

See also `heapcheck`

Example

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

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
    char far *array[NUM_PTRS];
    int i;
    for(i = 0; i < NUM_PTRS; i++)
        array[i] = farmalloc(NUM_BYTES);
    for(i = 0; i < NUM_PTRS; i += 2)
        farfree(array[i]);
    if(farheapcheck() == _HEAPCORRUPT)
        printf("Heap is corrupted.\n");
    else
        printf("Heap is OK.\n");
    return 0;
}
```

farheapcheckfree

Function Checks the free blocks on the far heap for a constant value.

Syntax `#include <alloc.h>`
`int farheapcheckfree(unsigned int fillvalue);`

Prototype in `alloc.h`

Return value The return value is less than zero for an error and greater than zero for success.

`_HEAPEMPTY` is returned if there is no heap (value 1).

`_HEAPOK` is returned if the heap is accurate (value 2).

`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).

`_BADVALUE` is returned if a value other than the fill value was found (value -3).

Portability `farheapcheckfree` is unique to DOS. This function is not Windows compatible.

See also `farheapfillfree`, `heapcheckfree`

Example

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

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
    char far *array(NUM_PTRS);
    int i;
    int j;
    int res;
    for (i = 0; i < NUM_PTRS; i++)
        if ((array[i] = farmalloc(NUM_BYTES)) == NULL) {
            printf("No memory for allocation\n");
            return 1;
        }
    for (i = 0; i < NUM_PTRS; i += 2)
        farfree(array[i]);
    if (farheapfillfree(1) < 0) {
        printf("Heap corrupted.\n");
        return 1;
    }
    for (i = 1; i < NUM_PTRS; i += 2)
        for (j = 0; j < NUM_BYTES; j++)
            array[i][j] = 0;
    res = farheapcheckfree(1);
    if (res < 0)
        switch(res) {
            case _HEAPCORRUPT:
                printf("Heap corrupted.\n");
                return 1;
            case _BADVALUE:
                printf("Bad value in free space.\n");
                return 1;
            default:
                printf("Unknown error.\n");
                return 1;
        }
    printf("Test successful.\n");
    return 0;
}
```

farheapchecknode

Function	Checks and verifies a single node on the far heap.
Syntax	<code>#include <alloc.h></code> <code>int farheapchecknode(void *node);</code>
Prototype in	<code>alloc.h</code>
Remarks	If a node has been freed and farheapchecknode is called with a pointer to the freed block, farheapchecknode can return <code>_BADNODE</code> rather than the expected <code>_FREEENTRY</code> . This is because adjacent free blocks on the heap are merged, and the block in question no longer exists.
Return value	The return value is less than zero for an error and greater than zero for success. <code>_HEAPEMPTY</code> is returned if there is no heap (value 1). <code>_HEAPCORRUPT</code> is returned if the heap has been corrupted (value -1). <code>_BADNODE</code> is returned if the node could not be found (value -2). <code>_FREEENTRY</code> is returned if the node is a free block (value 3). <code>_USEDENTRY</code> is returned if the node is a used block (value 4).
Portability	farheapchecknode is unique to DOS. This function is not Windows compatible.
See also	heapchecknode
Example	<pre>#include <stdio.h> #include <alloc.h> #define NUM_PTRS 10 #define NUM_BYTES 16 int main(void) { char far *array[NUM_PTRS]; int i; for(i=0; i < NUM_PTRS; i++) array[i] = farmalloc(NUM_BYTES); for(i=0; i < NUM_PTRS; i+=2) farfree(array[i]); for(i=0; i < NUM_PTRS; i++) { printf("Node %2d ", i); switch(farheapchecknode(array[i])) { case _HEAPEMPTY: printf("No heap.\n"); break; case _HEAPCORRUPT:</pre>

```

        printf("Heap corrupt.\n");
        break;
    case _BADNODE:
        printf("Bad node.\n");
        break;
    case _FREEENTRY:
        printf("Free entry.\n");
        break;
    case _USEDENTRY:
        printf("Used entry.\n");
        break;
    default:
        printf("Unknown return code.\n");
        break;
    }
}
return 0;
}

```

farheapfillfree

Function Fills the free blocks on the far heap with a constant value.

Syntax `#include <alloc.h>`
`int farheapfillfree(unsigned int fillvalue);`

Prototype in alloc.h

Return value The return value is less than zero for an error and greater than zero for success.

`_HEAPEMPTY` is returned if there is no heap (value 1).

`_HEAPOK` is returned if the heap is accurate (value 2).

`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).

Portability `farheapfillfree` is unique to DOS. This function is not Windows compatible.

See also `farheapcheckfree`, `heapfillfree`

Example

```

#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{

```

farheapfillfree

```
char far *array[NUM_PTRS];
int i,j;
int res;
for(i=0; i < NUM_PTRS; i++)
    array[i] = farmalloc(NUM_BYTES);
for(i=0; i < NUM_PTRS; i+= 2)
    farfree(array[i]);
if(farheapfillfree(1) < 0) {
    printf("Heap corrupted.\n");
    return 1;
}
for(i=1; i < NUM_PTRS; i+=2)
    for(j=0; j < NUM_BYTES; j++)
        array[i][j] = 0;
res = farheapcheckfree(1);
if (res < 0)
switch(res) {
    case _HEAPCORRUPT:
        printf("Heap corrupted.\n");
        return 1;
    case _BADVALUE:
        printf("Bad value in free space.\n");
        return 1;
    default:
        printf("Unknown error.\n");
        return 1;
}
printf("Test successful.\n");
return 0;
}
```

farheapwalk

Function **farheapwalk** is used to “walk” through the far heap node by node.

Syntax #include <alloc.h>
int farheapwalk(struct farheapinfo *hi);

Prototype in alloc.h

Remarks **farheapwalk** assumes the heap is correct. Use **farheapcheck** to verify the heap before using **farheapwalk**. **_HEAPOK** is returned with the last block on the heap. **_HEAPEND** will be returned on the next call to **farheapwalk**.

farheapwalk receives a pointer to a structure of type *heapinfo* (defined in alloc.h). For the first call to **farheapwalk**, set the *hi.ptr* field to null.

farheapwalk returns with *hi.ptr* containing the address of the first block.

`hi.size` holds the size of the block in bytes. `hi.in_use` is a flag that's set if the block is currently in use.

Return value `_HEAPEMPTY` is returned if there is no heap (value 1).
`_HEAPOK` is returned if the `heapinfo` block contains valid data (value 2).
`_HEAPEND` is returned if the end of the heap has been reached (value 5).

Portability `farheapwalk` is unique to DOS. This function is not Windows compatible.

See also `heapwalk`

Example

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

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
    struct farheapinfo hi;
    char far *array[NUM_PTRS];
    int i;
    for(i=0; i < NUM_PTRS; i++)
        array[i] = farmalloc(NUM_BYTES);
    for(i=0; i < NUM_PTRS; i+=2)
        farfree(array[i]);
    hi.ptr = NULL;
    printf("  Size  Status\n");
    printf(" ---- -\n");
    while(farheapwalk(&hi) == _HEAPOK)
        if (hi.in_use)
            printf("%7u  used\n", hi.size);
        else
            printf("%7u  free\n", hi.size);
    return 0;
}
```

farmalloc

Function Allocates from far heap.

Syntax `#include <alloc.h>`
`void far *farmalloc(unsigned long nbytes);`

Prototype in `alloc.h`

Remarks `farmalloc` allocates a block of memory *nbytes* bytes long from the far heap. For allocating from the far heap, note that



- all available RAM can be allocated.
- blocks larger than 64K can be allocated.
- far pointers are used to access the allocated blocks.

In the compact, large, and huge memory models, **farmalloc** is similar though not identical to **malloc**. It takes **unsigned long** parameters, while **malloc** takes **unsigned** parameters.

A tiny model program cannot make use of **farmalloc**.

Return value **farmalloc** returns a pointer to the newly allocated block, or null if not enough space exists for the new block.

Portability **farmalloc** is unique to DOS.

See also **farcalloc**, **farcoreleft**, **farfree**, **farrealloc**, **malloc**

Example

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
    char far *fptr;
    char *str = "Hello";

    /* allocate memory for the far pointer */
    fptr = farmalloc(10);

    /* copy "Hello" into the allocated memory */
    /* NOTE: movedata is used because we might be in a small data
       model; in which case, a normal string copy routine cannot be
       used since it assumes the pointer size is near. */
    movedata(FP_SEG(str), FP_OFF(str), FP_SEG(fptr), FP_OFF(fptr), strlen(str));

    /* display string (note the F modifier) */
    printf("Far string is: %Fs\n", fptr);

    /* free the memory */
    farfree(fptr);
    return 0;
}
```

farrealloc

Function Adjusts allocated block in far heap.

Syntax `#include <alloc.h>`
`void far *farrealloc(void far *oldblock, unsigned long nbytes);`

Prototype in alloc.h

Remarks **farrealloc** adjusts the size of the allocated block to *nbytes*, copying the contents to a new location, if necessary.

For allocating from the far heap, note that

- all available RAM can be allocated.
- blocks larger than 64K can be allocated.
- far pointers are used to access the allocated blocks.

A tiny model program cannot make use of **farrealloc**.

Return value **farrealloc** returns the address of the reallocated block, which might be different than the address of the original block. If the block cannot be reallocated, **farrealloc** returns null.

Portability **farrealloc** is unique to DOS.

See also **farmalloc**, **realloc**

Example

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

int main(void)
{
    char far *fptr;
    fptr = farmalloc(10);
    printf("First address: %Fp\n", fptr);
    fptr = farrealloc(fptr,20);
    printf("New address : %Fp\n", fptr);
    farfree(fptr);
    return 0;
}
```

fclose

Function	Closes a stream.
Syntax	<pre>#include <stdio.h> int fclose(FILE *stream);</pre>
Prototype in	stdio.h
Remarks	fclose closes the named stream. All buffers associated with the stream are flushed before closing. System-allocated buffers are freed upon closing. Buffers assigned with setbuf or setvbuf are not automatically freed. (But if setvbuf is passed null for the buffer pointer, it <i>will</i> free it upon close.)
Return value	fclose returns 0 on success. It returns EOF if any errors were detected.
Portability	fclose is available on UNIX systems and is defined in ANSI C.
See also	close , fcloseall , fdopen , fflush , flushall , fopen , freopen
Example	<pre>#include <string.h> #include <stdio.h> int main(void) { FILE *fp; char buf[11] = "0123456789"; /* create a file containing 10 bytes */ fp = fopen("DUMMY.FIL", "w"); if (fp) { fwrite(&buf, strlen(buf), 1, fp); /* close the file */ fclose(fp); } else { printf("Unable to open file!\n"); } return 0; }</pre>

fcloseall

- Function** Closes open streams.
- Syntax** `#include <stdio.h>`
`int fcloseall(void);`
- Prototype in** `stdio.h`
- Remarks** **fcloseall** closes all open streams except `stdin`, `stdout`, `stderr`, and `stdaux`.
- Return value** **fcloseall** returns the total number of streams it closed. It returns EOF if any errors were detected.
- Portability** **fcloseall** is available on UNIX systems.
- See also** **fclose**, **fdopen**, **flushall**, **fopen**, **freopen**

Example

```
#include <stdio.h>

int main(void)
{
    int streams_closed;

    /* open two streams */
    fopen("DUMMY.ONE", "w");
    fopen("DUMMY.TWO", "w");

    /* close the open streams */
    streams_closed = fcloseall();
    if (streams_closed == EOF)
        perror("Error");          /* issue an error message */
    else
        /* print result of fcloseall() function */
        printf("%d streams were closed.\n", streams_closed);
    return 0;
}
```

fcvt

Function	Converts a floating-point number to a string.
Syntax	<pre>#include <stdlib.h> char *fcvt(double <i>value</i>, int <i>ndig</i>, int *<i>dec</i>, int *<i>sign</i>);</pre>
Prototype in	stdlib.h
Remarks	<p>fcvt converts <i>value</i> to a null-terminated string of <i>ndig</i> digits, starting with the leftmost significant digit, and returns a pointer to the string. The position of the decimal point relative to the beginning of the string is stored indirectly through <i>dec</i> (a negative value for <i>dec</i> means to the left of the returned digits). There is no decimal point in the string itself. If the sign of <i>value</i> is negative, the word pointed to by <i>sign</i> is nonzero; otherwise, it is 0.</p> <p>The correct digit has been rounded for the number of digits specified by <i>ndig</i>.</p>
Return value	The return value of fcvt points to static data whose content is overwritten by each call to fcvt .
Portability	fcvt is available on UNIX. It is not in ANSI C, and is not recommended for portable programs.
See also	ecvt , gcvt , sprintf
Example	<pre>#include <stdlib.h> #include <stdio.h> int main(void) { char *str; double num; int dec, sign; int ndig = 5; /* a regular number */ num = 9.876; str = fcvt(num, ndig, &dec, &sign); printf("string = %10s decimal place = %d sign = %d\n", str, dec, sign); /* a negative number */ num = -123.45; str = fcvt(num, ndig, &dec, &sign); printf("string = %10s decimal place = %d sign = %d\n", str, dec, sign); /* scientific notation */ num = 0.678e5; }</pre>

```

    str = fcvt(num, ndig, &dec, &sign);
    printf("string = %10s decimal place= %d sign = %d\n", str, dec, sign);
    return 0;
}

```

fdopen

F

Function Associates a stream with a file handle.

Syntax `#include <stdio.h>`
`FILE *fdopen(int handle, char *type);`

Prototype in `stdio.h`

Remarks **fdopen** associates a stream with a file handle obtained from **creat**, **dup**, **dup2**, or **open**. The type of stream must match the mode of the open *handle*.

The *type* string used in a call to **fdopen** is one of the following values:

- r* Open for reading only.
- w* Create for writing.
- a* Append; open for writing at end-of-file or create for writing if the file does not exist.
- r+* Open an existing file for update (reading and writing).
- w+* Create a new file for update.
- a+* Open for append; open (or create if the file does not exist) for update at the end of the file.

To specify that a given file is being opened or created in text mode, append a *t* to the value of the *type* string (*rt*, *w+t*, and so on); similarly, to specify binary mode, append a *b* to the *type* string (*wb*, *a+b*, and so on).

If a *t* or *b* is not given in the *type* string, the mode is governed by the global variable `_fmode`. If `_fmode` is set to `O_BINARY`, files will be opened in binary mode. If `_fmode` is set to `O_TEXT`, they will be opened in text mode. These `O_...` constants are defined in `fcntl.h`.

When a file is opened for update, both input and output can be done on the resulting stream. However, output cannot be directly followed by input without an intervening **fseek** or **rewind**, and input cannot be directly followed by output without an intervening **fseek**, **rewind**, or an input that encounters end-of-file.

fdopen

Return value On successful completion, **fdopen** returns a pointer to the newly opened stream. In the event of error, it returns null.

Portability **fdopen** is available on UNIX systems.

See also **fclose**, **fopen**, **freopen**, **open**

Example

```
#include <sys\stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
    int handle;
    FILE *stream;

    /* open a file */
    handle = open("DUMMY.FIL", O_CREAT, S_IREAD | S_IWRITE);

    /* now turn the handle into a stream */
    stream = fdopen(handle, "w");
    if (stream == NULL)
        printf("fdopen failed\n");
    else {
        fprintf(stream, "Hello world\n");
        fclose(stream);
    }
    return 0;
}
```

feof

Function Detects end-of-file on a stream.

Syntax `#include <stdio.h>`
`int feof(FILE *stream);`

Prototype in `stdio.h`

Remarks **feof** is a macro that tests the given stream for an end-of-file indicator. Once the indicator is set, read operations on the file return the indicator until **rewind** is called, or the file is closed.

The end-of-file indicator is reset with each input operation.

Return value

feof returns nonzero if an end-of-file indicator was detected on the last input operation on the named stream, and 0 if end-of-file has not been reached.

Portability **feof** is available on UNIX systems and is defined in ANSI C.

See also **clearerr**, **eof**, **ferror**, **perror**

Example

```
#include <stdio.h>

int main(void)
{
    FILE *stream;

    /* open a file for reading */
    stream = fopen("DUMMY.FIL", "r");

    /* read a character from the file */
    fgetc(stream);

    /* check for EOF */
    if (feof(stream))
        printf("We have reached end-of-file\n");

    /* close the file */
    fclose(stream);
    return 0;
}
```

F

ferror

Function Detects errors on stream.

Syntax `#include <stdio.h>`
`int ferror(FILE *stream);`

Prototype in `stdio.h`

Remarks **ferror** is a macro that tests the given stream for a read or write error. If the stream's error indicator has been set, it remains set until **clearerr** or **rewind** is called, or until the stream is closed.

Return value **ferror** returns nonzero if an error was detected on the named stream.

Portability **ferror** is available on UNIX systems and is defined in ANSI C.

See also **clearerr**, **eof**, **feof**, **fopen**, **gets**, **perror**

Example

```
#include <stdio.h>

int main(void)
```


feof

```
{
    FILE *stream;

    /* open a file for writing */
    stream = fopen("DUMMY.FIL", "w");

    /* force an error condition by attempting to read */
    (void) getc(stream);
    if (feof(stream)) /* test for error on the stream */
    {
        /* display an error message */
        printf("Error reading from DUMMY.FIL\n");

        /* reset the error and EOF indicators */
        clearerr(stream);
    }
    fclose(stream);
    return 0;
}
```

fflush

- Function** Flushes a stream.
- Syntax** `#include <stdio.h>`
`int fflush(FILE *stream);`
- Prototype in** `stdio.h`
- Remarks** If the given stream has buffered output, **fflush** writes the output for *stream* to the associated file.
- The stream remains open after **fflush** has executed. **fflush** has no effect on an unbuffered stream.
- Return value** **fflush** returns 0 on success. It returns EOF if any errors were detected.
- Portability** **fflush** is available on UNIX systems and is defined in ANSI C.
- See also** **fclose**, **flushall**, **setbuf**, **setvbuf**
- Example**
- ```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
 FILE *stream;
```

```

char msg[] = "This is a test";

/* create a file */
stream = fopen("DUMMY.FIL", "w");

/* write some data to the file */
fwrite(msg, strlen(msg), 1, stream);
clrscr();
printf("Press any key to flush DUMMY.FIL:");
getch();

/* flush the data to DUMMY.FIL without closing it */
flush(stream);
printf("\nFile was flushed, Press any key to quit:");
getch();
return 0;
}

void flush(FILE *stream) {
 int duphandle;

 /* flush the stream's internal buffer */
 fflush(stream);

 /* make a duplicate file handle */
 duphandle = dup(fileno(stream));

 /* close the duplicate handle to flush the DOS buffer */
 close(duphandle);
}

```

F

## fgetc

---

|                     |                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets character from stream.                                                                                                                                |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int fgetc(FILE *stream);</code>                                                                             |
| <b>Prototype in</b> | stdio.h                                                                                                                                                    |
| <b>Remarks</b>      | <b>fgetc</b> returns the next character on the named input stream.                                                                                         |
| <b>Return value</b> | On success, <b>fgetc</b> returns the character read, after converting it to an <b>int</b> without sign extension. On end-of-file or error, it returns EOF. |
| <b>Portability</b>  | <b>fgetc</b> is available on UNIX systems and is defined in ANSI C.                                                                                        |
| <b>See also</b>     | <b>fgetchar</b> , <b>fputc</b> , <b>getc</b> , <b>getch</b> , <b>getchar</b> , <b>getche</b> , <b>ungetc</b> , <b>ungetch</b>                              |
| <b>Example</b>      | <code>#include &lt;string.h&gt;</code><br><code>#include &lt;stdio.h&gt;</code>                                                                            |

```

#include <conio.h>

int main(void)
{
 FILE *stream;
 char string[] = "This is a test";
 char ch;

 /* open a file for update */
 stream = fopen("DUMMY.FIL", "w+");

 /* write a string into the file */
 fwrite(string, strlen(string), 1, stream);

 /* seek to the beginning of the file */
 fseek(stream, 0, SEEK_SET);
 do {
 ch = fgetc(stream); /* read a char from the file */
 putchar(ch); /* display the character */
 }
 while (ch != EOF);
 fclose(stream);
 return 0;
}

```

## fgetchar

---

|                     |                                                                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets character from stdin.                                                                                                                                             |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int fgetchar(void);</code>                                                                                              |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                |
| <b>Remarks</b>      | <b>fgetchar</b> returns the next character from stdin. It is defined as <b>fgetc(stdin)</b> .                                                                          |
| <b>Return value</b> | On success, <b>fgetchar</b> returns the character read, after converting it to an <b>int</b> without sign extension. On end-of-file or error, it returns EOF.          |
| <b>Portability</b>  | <b>fgetchar</b> is available on UNIX systems.                                                                                                                          |
| <b>See also</b>     | <b>fgetc</b> , <b>fputchar</b> , <b>getchar</b>                                                                                                                        |
| <b>Example</b>      | <pre> #include &lt;stdio.h&gt;  int main(void) {     char ch;      /* prompt the user for input */     printf("Enter a character followed by &lt;Enter&gt;: "); </pre> |

```

/* read the character from stdin */
ch = fgetchar();

/* display what was read */
printf("The character read is: '%c'\n", ch);
return 0;
}

```

## fgetpos

F

**Function** Gets the current file pointer.

**Syntax** `#include <stdio.h>`  
`int fgetpos(FILE *stream, fpos_t *pos);`

**Prototype in** `stdio.h`

**Remarks** **fgetpos** stores the position of the file pointer associated with the given stream in the location pointed to by *pos*. The exact value is a magic cookie; in other words, it is irrelevant to your purposes.

The type *fpos\_t* is defined in `stdio.h` as `typedef long fpos_t;`

**Return value** On success, **fgetpos** returns 0. On failure, it returns a nonzero value and sets the global variable *errno* to EBADF or EINVAL.

**Portability** **fgetpos** is compatible with ANSI C.

**See also** **fseek, fsetpos, ftell, tell**

**Example**

```

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

int main(void)
{
 FILE *stream;
 char string[] = "This is a test";
 fpos_t filepos;

 /* open a file for update */
 stream = fopen("DUMMY.FIL", "w+");

 /* write a string into the file */
 fwrite(string, strlen(string), 1, stream);

 /* report the file pointer position */
 fgetpos(stream, &filepos);
 printf("The file pointer is at byte %ld\n", filepos);
 fclose(stream);
}

```

```

 return 0;
}

```

## fgets

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets a string from a stream.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>char *fgets(char *s, int n, FILE *stream);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Remarks</b>      | <b>fgets</b> reads characters from <i>stream</i> into the string <i>s</i> . The function stops reading when it reads either <i>n</i> – 1 characters or a newline character, whichever comes first. <b>fgets</b> retains the newline character at the end of <i>s</i> . A null byte is appended to <i>s</i> to mark the end of the string.                                                                                                                                                                                                                      |
| <b>Return value</b> | On success, <b>fgets</b> returns the string pointed to by <i>s</i> ; it returns null on end-of-file or error.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Portability</b>  | <b>fgets</b> is available on UNIX systems and is defined in ANSI C. It is also defined in Kernighan and Ritchie.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>See also</b>     | <b>cgets, fputs, gets</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Example</b>      | <pre> #include &lt;string.h&gt; #include &lt;stdio.h&gt;  int main(void) {     FILE *stream;     char string[] = "This is a test";     char msg[20];      /* open a file for update */     stream = fopen("DUMMY.FIL", "w+");      /* write a string into the file */     fwrite(string, strlen(string), 1, stream);      /* seek to the start of the file */     fseek(stream, 0, SEEK_SET);      /* read a string from the file */     fgets(msg, strlen(string)+1, stream);      /* display the string */     printf("%s", msg);     fclose(stream); </pre> |

```

 return 0;
}

```

## filelength

---

F

**Function** Gets file size in bytes.

**Syntax** `#include <io.h>`  
`long filelength(int handle);`

**Prototype in** `io.h`

**Remarks** **filelength** returns the length (in bytes) of the file associated with *handle*.

**Return value** On success, **filelength** returns a **long** value, the file length in bytes. On error, it returns `-1` and the global variable *errno* is set to

`EBADF` Bad file number

**Portability** **filelength** is unique to DOS.

**See also** **fopen**, **lseek**, **open**

**Example**

```

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <string.h>

int main(void)
{
 int handle;
 char buf[11] = "0123456789";

 /* create a file containing 10 bytes */
 handle = open("DUMMY.FIL", O_RDWR|O_CREAT|O_TRUNC, S_IREAD|S_IWRITE);
 write(handle, buf, strlen(buf));

 /* display the size of the file */
 printf("file length in bytes: %ld\n", filelength(handle));

 /* close the file */
 close(handle);
 return 0;
}

```

## fileno

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets file handle.                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int fileno(FILE *stream);</code>                                                                                                                                                                                                                                                                                                                |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Remarks</b>      | <b>fileno</b> is a macro that returns the file handle for the given stream. If <i>stream</i> has more than one handle, <b>fileno</b> returns the handle assigned to the stream when it was first opened.                                                                                                                                                                                       |
| <b>Return value</b> | <b>fileno</b> returns the integer file handle associated with <i>stream</i> .                                                                                                                                                                                                                                                                                                                  |
| <b>Portability</b>  | <b>fileno</b> is available on UNIX systems.                                                                                                                                                                                                                                                                                                                                                    |
| <b>See also</b>     | <b>fdopen, fopen, freopen</b>                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;  int main(void) {     FILE *stream;     int handle;      /* create a file */     stream = fopen("DUMMY.FIL", "w");      /* obtain the file handle associated with the stream */     handle = fileno(stream);      /* display the handle number */     printf("handle number: %d\n", handle);      /* close the file */     fclose(stream);     return 0; }</pre> |

## fileellipse

---

|                     |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Draws and fills an ellipse.                                                                                            |
| <b>Syntax</b>       | <code>#include &lt;graphics.h&gt;</code><br><code>void far fileellipse(int x, int y, int xradius, int yradius);</code> |
| <b>Prototype in</b> | graphics.h                                                                                                             |

**Remarks** Draws an ellipse using  $(x,y)$  as a center point and  $xradius$  and  $yradius$  as the horizontal and vertical axes, and fills it with the current fill color and fill pattern.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **arc, circle, ellipse, getaspectratio, pieslice, setaspectratio**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, i;
 int xradius = 100, yradius = 50;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* loop through the fill patterns */
 for (i = EMPTY_FILL; i < USER_FILL; i++) {
 /* set fill pattern */
 setfillstyle(i, getmaxcolor());

 /* draw a filled ellipse */
 fillellipse(midx, midy, xradius, yradius);
 getch();
 }

 /* clean up */
 closegraph();
}
```



```

 return 0;
 }

```

## fillpoly

---

- Function** Draws and fills a polygon.
- Syntax** `#include <graphics.h>`  
`void far fillpoly(int numpoints, int far *polypoints);`
- Prototype in** graphics.h
- Remarks** **fillpoly** draws the outline of a polygon with *numpoints* points in the current line style and color (just as **drawpoly** does), then fills the polygon using the current fill pattern and fill color.
- polypoints* points to a sequence of (*numpoints* × 2) integers. Each pair of integers gives the *x* and *y* coordinates of a point on the polygon.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **drawpoly, floodfill, graphresult, setfillstyle**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int i, maxx, maxy;

 /* our polygon array */
 int poly[8];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 }
}

```

```

 getch();
 exit(1); /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();

poly[0] = 20; /* first vertex */
poly[1] = maxy / 2;
poly[2] = maxx - 20; /* second vertex */
poly[3] = 20;
poly[4] = maxx - 50; /* third vertex */
poly[5] = maxy - 20;
poly[6] = maxx / 2; /* fourth, fillpoly automatically */
poly[7] = maxy / 2; /* closes the polygon */

/* loop through the fill patterns */
for (i=EMPTY_FILL; i<USER_FILL; i++) {
 /* set fill pattern */
 setfillstyle(i, getmaxcolor());

 /* draw a filled polygon */
 fillpoly(4, poly);
 getch();
}

/* clean up */
closegraph();
return 0;
}

```

## findfirst

---

**Function** Searches a disk directory.

**Syntax** #include <dir.h>  
 #include <dos.h>  
 int findfirst(const char \*pathname, struct ffblk \*ffblk, int attrib);

**Prototype in** dir.h

**Remarks** **findfirst** begins a search of a disk directory by using the DOS system call 0x4E.

*pathname* is a string with an optional drive specifier, path, and file name of the file to be found. The file name portion can contain wildcard match characters (such as ? or \*). If a matching file is found, the **ffblk** structure is filled with the file-directory information.

The format of the structure **ffblk** is as follows:

```
struct ffblk {
 char ff_reserved[21]; /* reserved by DOS */
 char ff_attrib; /* attribute found */
 int ff_ftime; /* file time */
 int ff_fdate; /* file date */
 long ff_fsize; /* file size */
 char ff_name[13]; /* found file name */
};
```

*attrib* is a DOS file-attribute byte used in selecting eligible files for the search. *attrib* can be one of the following constants defined in `dos.h`:

|           |                     |
|-----------|---------------------|
| FA_RDONLY | Read-only attribute |
| FA_HIDDEN | Hidden file         |
| FA_SYSTEM | System file         |
| FA_LABEL  | Volume label        |
| FA_DIREC  | Directory           |
| FA_ARCH   | Archive             |

For more detailed information about these attributes, refer to your DOS reference manuals.

Note that *ff\_time* and *ff\_fdate* contain bit fields for referring to the current date and time. The structure of these fields was established by MS-DOS. Both are 16-bit structures divided into three fields.

***ff\_time*:**

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| bits 0 to 4   | The result of seconds divided by 2 (e.g., 10 here means 20 seconds) |
| bits 5 to 10  | Minutes                                                             |
| bits 11 to 15 | Hours                                                               |

***ff\_fdate*:**

|           |                                            |
|-----------|--------------------------------------------|
| bits 0-4  | Day                                        |
| bits 5-8  | Month                                      |
| bits 9-15 | Years since 1980 (e.g., 9 here means 1989) |

The structure **ftime** declared in `io.h` uses time and date bit fields similar in structure to *ff\_time*, and *ff\_fdate*. See **gettime** or **settime** for examples.

**Return value** **findfirst** returns 0 on successfully finding a file matching the search *pathname*. When no more files can be found, or if there is some error in the file name, -1 is returned, and the global variable *errno* is set to

|        |                             |
|--------|-----------------------------|
| ENOENT | Path or file name not found |
|--------|-----------------------------|

and *doserrno* is set to one of the following:

ENOENT Path or file name not found  
 ENMFILE No more files

**Portability** **findfirst** is unique to DOS.

**See also** **findnext**

**Example**

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

int main(void)
{
 struct ffblk ffblk;
 int done;
 printf("Directory listing of *.*\n");
 done = findfirst("*.*", &ffblk, 0);
 while (!done) {
 printf(" %s\n", ffblk.ff_name);
 done = findnext(&ffblk);
 }
 return 0;
}
```

**Program output**

```
Directory listing of *.*
 FINDFRST.C
 FINDFRST.OBJ
 FINDFRST.MAP
 FINDFRST.EXE
```

## findnext

---

**Function** Continues **findfirst** search.

**Syntax** #include <dir.h>  
 int findnext(struct ffblk \*ffblk);

**Prototype in** dir.h

**Remarks** **findnext** is used to fetch subsequent files that match the *pathname* given in **findfirst**. *ffblk* is the same block filled in by the **findfirst** call. This block contains necessary information for continuing the search. One file name for each call to **findnext** will be returned until no more files are found in the directory matching the *pathname*.

## findnext

**Return value** **findnext** returns 0 on successfully finding a file matching the search *pathname*. When no more files can be found, or if there is some error in the file name, -1 is returned, and the global variable *errno* is set to

ENOENT Path or file name not found

and *doserro* is set to one of the following:

ENOENT Path or file...

ENMFILE No more files

**Portability** **findnext** is unique to DOS.

**See also** **findfirst**

**Example**

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

int main(void)
{
 struct fblk fblk;
 int done;
 printf("Directory listing of *.*\n");
 done = findfirst("*.*", &fblk, 0);
 while (!done) {
 printf(" %s\n", fblk.ff_name);
 done = findnext(&fblk);
 }
 return 0;
}
```

### Program output

```
Directory listing of *.*
FINDFRST.C
FINDFRST.OBJ
FINDFRST.MAP
FINDFRST.EXE
```

## floodfill

**Function** Flood-fills a bounded region.

**Syntax** `#include <graphics.h>`  
`void far floodfill(int x, int y, int border);`

**Prototype in** graphics.h

**Remarks** **floodfill** fills an enclosed area on bitmap devices.  $(x,y)$  is a “seed point” within the enclosed area to be filled. The area bounded by the color *border* is flooded with the current fill pattern and fill color. If the seed point is within an enclosed area, the inside will be filled. If the seed is outside the enclosed area, the exterior will be filled.

Use **fillpoly** instead of **floodfill** whenever possible so that you can maintain code compatibility with future versions.



**floodfill** does not work with the IBM-8514 driver.

**Return value** If an error occurs while flooding a region, **graphresult** returns a value of `-7`.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **drawpoly**, **fillpoly**, **graphresult**, **setcolor**, **setfillstyle**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int maxx, maxy;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
```



## floodfill

```
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 maxx = getmaxx();
 maxy = getmaxy();

 /* select drawing color */
 setcolor(getmaxcolor());

 /* select fill color */
 setfillstyle(SOLID_FILL, getmaxcolor());

 /* draw a border around the screen */
 rectangle(0, 0, maxx, maxy);

 /* draw some circles */
 circle(maxx / 3, maxy / 2, 50);
 circle(maxx / 2, 20, 100);
 circle(maxx-20, maxy-50, 75);
 circle(20, maxy-20, 25);

 /* wait for a key */
 getch();

 /* fill in bounded region */
 floodfill(2, 2, getmaxcolor());

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

## floor

---

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <b>Function</b>     | Rounds down.                                                        |
| <b>Syntax</b>       | #include <math.h><br>double floor(double x);                        |
| <b>Prototype in</b> | math.h                                                              |
| <b>Remarks</b>      | <b>floor</b> finds the largest integer not greater than <i>x</i> .  |
| <b>Return value</b> | <b>floor</b> returns the integer found (as a <b>double</b> ).       |
| <b>Portability</b>  | <b>floor</b> is available on UNIX systems and is defined in ANSI C. |

**See also** `ceil`, `fmod`

**Example**

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

int main(void)
{
 double number = 123.54;
 double down, up;
 down = floor(number);
 up = ceil(number);
 printf("original number %10.2lf\n", number);
 printf("number rounded down %10.2lf\n", down);
 printf("number rounded up %10.2lf\n", up);
 return 0;
}
```

F

## flushall

---

**Function** Flushes all streams.

**Syntax** `#include <stdio.h>`  
`int flushall(void);`

**Prototype in** `stdio.h`

**Remarks** `flushall` clears all buffers associated with open input streams, and writes all buffers associated with open output streams to their respective files. Any read operation following `flushall` reads new data into the buffers from the input files.

Streams stay open after `flushall` executes.

**Return value** `flushall` returns an integer, the number of open input and output streams.

**Portability** `flushall` is available on UNIX systems.

**See also** `fclose`, `fcloseall`, `fflush`

**Example**

```
#include <stdio.h>

int main(void)
{
 FILE *stream;

 /* create a file */
 stream = fopen("DUMMY.FIL", "w");
```



## flushall

```
/* flush all open streams */
printf("%d streams were flushed.\n", flushall());

/* close the file */
fclose(stream);
return 0;
}
```

## \_fmemccpy

---

See **memccpy**.

## \_fmemchr

---

See **memchr**.

## \_fmemcmp

---

See **memcmp**.

## \_fmemcpy

---

See **memcpy**.

## \_fmemicmp

---

See **memicmp**.

## \_fmemset

---

See **memset**.

## fmod

---

**Function** Calculates  $x$  modulo  $y$ , the remainder of  $x/y$ .

**Syntax** `#include <math.h>`  
`double fmod(double  $x$ , double  $y$ );`

**Prototype in** `math.h`

**Remarks** **fmod** calculates  $x$  modulo  $y$  (the remainder  $f$ , where  $x = ay + f$  for some integer  $a$  and  $0 \leq f < y$ ).

**Return value** **fmod** returns the remainder  $f$ , where  $x = ay + f$  (as described). Where  $y = 0$ , **fmod** returns 0.

**Portability** **fmod** is compatible with ANSI C.

**See also** **ceil**, **floor**, **modf**

**Example**

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

int main(void)
{
 double x = 5.0, y = 2.0;
 double result;
 result = fmod(x,y);
 printf("The remainder of (%lf / %lf) is %lf\n", x, y, result);
 return 0;
}
```

### Program output

The remainder of 5.0 / 2.0 is 1.0.

## fnmerge

---

**Function** Builds a path from component parts.

**Syntax** `#include <dir.h>`  
`void fnmerge(char * $path$ , const char * $drive$ , const char * $dir$ ,  
const char * $name$ , const char * $ext$ );`

**Prototype in** `dir.h`

**Remarks** **fnmerge** makes a path name from its components. The new path name is

```
X:\DIR\SUBDIR\NAME.EXT
```

where

```
drive = X:
dir = \DIR\SUBDIR\
name = NAME
ext = .EXT
```

**fnmerge** assumes there is enough space in *path* for the constructed path name. The maximum constructed length is MAXPATH. MAXPATH is defined in dir.h.

**fnmerge** and **fnsplit** are invertible; if you split a given *path* with **fnsplit**, then merge the resultant components with **fnmerge**, you end up with *path*.

**Return value** None.

**Portability** **fnmerge** is available on DOS systems only.

**See also** **fnsplit**

**Example**

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

int main(void)
{
 char s[MAXPATH];
 char drive[MAXDRIVE];
 char dir[MAXDIR];
 char file[MAXFILE];
 char ext[MAXEXT];
 getcwd(s,MAXPATH); /* get current working directory */
 strcat(s,"\\"); /* append a trailing \ character */
 fnsplit(s,drive,dir,file,ext); /* split the string to separate elems */
 strcpy(file,"DATA");
 strcpy(ext,".TXT");
 fnmerge(s,drive,dir,file,ext); /* merge everything into one string */
 puts(s); /* display resulting string */
 return 0;
}
```

## fnsplit

**Function** Splits a full path name into its components.

**Syntax** #include <dir.h>  
int fnsplit(const char \*path, char \*drive, char \*dir, char \*name, char \*ext);

**Prototype in** dir.h

**Remarks** **fnsplit** takes a file's full path name (*path*) as a string in the form

*X:\DIR\SUBDIR\NAME.EXT*

and splits *path* into its four components. It then stores those components in the strings pointed to by *drive*, *dir*, *name*, and *ext*. (All five components must be passed, but any of them can be a null, which means the corresponding component will be parsed but not stored.)

The maximum sizes for these strings are given by the constants MAXDRIVE, MAXDIR, MAXPATH, MAXFILE, and MAXEXT (defined in dir.h), and each size includes space for the null-terminator.

| Constant | Max  | String                                                     |
|----------|------|------------------------------------------------------------|
| MAXPATH  | (80) | <i>path</i>                                                |
| MAXDRIVE | (3)  | <i>drive</i> ; includes colon (:)                          |
| MAXDIR   | (66) | <i>dir</i> ; includes leading and trailing backslashes (\) |
| MAXFILE  | (9)  | <i>name</i>                                                |
| MAXEXT   | (5)  | <i>ext</i> ; includes leading dot (.)                      |

**fnsplit** assumes that there is enough space to store each non-null component.

When **fnsplit** splits *path*, it treats the punctuation as follows:

- *drive* includes the colon (C:, A:, and so on).
- *dir* includes the leading and trailing backslashes (\BC\include\, \source\, and so on).
- *name* includes the file name.
- *ext* includes the dot preceding the extension (.C, .EXE, and so on).

**fnmerge** and **fnsplit** are invertible; if you split a given *path* with **fnsplit**, then merge the resultant components with **fnmerge**, you end up with *path*.

**Return value** **fnsplit** returns an integer (composed of five flags, defined in `dir.h`) indicating which of the full path name components were present in *path*; these flags and the components they represent are

|           |                                                 |
|-----------|-------------------------------------------------|
| EXTENSION | An extension                                    |
| FILENAME  | A file name                                     |
| DIRECTORY | A directory (and possibly subdirectories)       |
| DRIVE     | A drive specification (see <code>dir.h</code> ) |
| WILDCARDS | Wildcards (* or ?)                              |

**Portability** **fnsplit** is available on DOS systems only.

**See also** **fnmerge**

**Example**

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

int main(void)
{
 char *s;
 char drive[MAXDRIVE];
 char dir[MAXDIR];
 char file[MAXFILE];
 char ext[MAXEXT];
 int flags;

 /* get comspec environment parameter */
 s = getenv("COMSPEC");
 flags = fnsplit(s,drive,dir,file,ext);
 printf("Command processor info:\n");
 if(flags & DRIVE)
 printf("\tdrive: %s\n",drive);
 if(flags & DIRECTORY)
 printf("\tdirectory: %s\n",dir);
 if(flags & FILENAME)
 printf("\tfile: %s\n",file);
 if(flags & EXTENSION)
 printf("\textension: %s\n",ext);
 return 0;
}
```

## fopen

|                     |                                                                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Opens a stream.                                                                                                                                                                  |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>FILE *fopen(const char *filename, const char *mode);</code>                                                                       |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                          |
| <b>Remarks</b>      | <b>fopen</b> opens the file named by <i>filename</i> and associates a stream with it. <b>fopen</b> returns a pointer to be used to identify the stream in subsequent operations. |

The *mode* string used in calls to **fopen** is one of the following values:

| Mode      | Description                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------|
| <i>r</i>  | Open for reading only.                                                                                             |
| <i>w</i>  | Create for writing. If a file by that name already exists, it will be overwritten.                                 |
| <i>a</i>  | Append; open for writing at end of file, or create for writing if the file does not exist.                         |
| <i>r+</i> | Open an existing file for update (reading and writing).                                                            |
| <i>w+</i> | Create a new file for update (reading and writing). If a file by that name already exists, it will be overwritten. |
| <i>a+</i> | Open for append; open for update at the end of the file, or create if the file does not exist.                     |

To specify that a given file is being opened or created in text mode, append a *t* to the *mode* string (*rt*, *w+t*, and so on). Similarly, to specify binary mode, append a *b* to the *mode* string (*wb*, *a+b*, and so on). **fopen** also allows the *t* or *b* to be inserted between the letter and the + character in the mode string; for example, *rt+* is equivalent to *r+t*.

If a *t* or *b* is not given in the *mode* string, the mode is governed by the global variable `_fmode`. If `_fmode` is set to `O_BINARY`, files are opened in binary mode. If `_fmode` is set to `O_TEXT`, they are opened in text mode. These `O_...` constants are defined in `fcntl.h`.

When a file is opened for update, both input and output can be done on the resulting stream. However, output cannot be followed directly by input without an intervening **fseek** or **rewind**, and input cannot be directly followed by output without an intervening **fseek**, **rewind**, or an input that encounters end-of-file.

## fopen

- Return value** On successful completion, **fopen** returns a pointer to the newly opened stream. In the event of error, it returns null.
- Portability** **fopen** is available on UNIX systems and is defined in ANSI C. It is defined by Kernighan and Ritchie.
- See also** **creat, dup, fclose, fdopen, ferror, \_fmode** (global variable), **fread, freopen, fseek, fwrite, open, rewind, setbuf, setmode**

**Example**

```
/* program to create backup of the AUTOEXEC.BAT file */
#include <stdio.h>

int main(void)
{
 FILE *in, *out;
 if ((in = fopen("\\\\AUTOEXEC.BAT", "rt")) == NULL) {
 fprintf(stderr, "Cannot open input file.\n");
 return 1;
 }
 if ((out = fopen("\\\\AUTOEXEC.BAK", "wt")) == NULL) {
 fprintf(stderr, "Cannot open output file.\n");
 return 1;
 }
 while (!feof(in))
 fputc(fgetc(in), out);
 fclose(in);
 fclose(out);
 return 0;
}
```

## FP\_OFF

---

- Function** Gets a far address offset.
- Syntax** `#include <dos.h>`  
`unsigned FP_OFF(void far *p);`
- Prototype in** `dos.h`
- Remarks** The **FP\_OFF** macro can be used to get or set the offset of the far pointer *\*p*.
- Return value** **FP\_OFF** returns an **unsigned** integer value representing an offset value.
- Portability** **FP\_OFF** is unique to DOS.
- See also** **FP\_SEG, MK\_FP, movedata, segread**
- Example** `#include <dos.h>`

```

#include <stdio.h>

int main(void)
{
 char *str = "fpoff.c";
 printf("The offset of this string in memory is: %X\n", FP_OFF(str));
 return 0;
}

```

### Program output

The offset of this string in memory is: FF02

## \_fpreset

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reinitializes floating-point math package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>       | <code>#include &lt;float.h&gt;</code><br><code>void _fpreset(void);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Prototype in</b> | float.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Remarks</b>      | <p><b>_fpreset</b> reinitializes the floating-point math package. This function is usually used in conjunction with <b>system</b> or the <b>exec...</b> or <b>spawn...</b> functions.</p> <p>➔ Under DOS, if an 80x87 coprocessor is used in a program, a child process (executed by <b>system</b> or by an <b>exec...</b> or <b>spawn...</b> function) might alter the parent process's floating-point state.</p> <p>If you use an 80x87, take the following precautions:</p> <ul style="list-style-type: none"> <li>■ Do not call <b>system</b> or an <b>exec...</b> or <b>spawn...</b> function while a floating-point expression is being evaluated.</li> <li>■ Call <b>_fpreset</b> to reset the floating-point state after using <b>system</b>, <b>exec...</b>, or <b>spawn...</b> if there is <i>any</i> chance that the child process performed a floating-point operation with the 80x87.</li> </ul> |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Portability</b>  | <b>_fpreset</b> is unique to DOS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>See also</b>     | <b>_clear87</b> , <b>_control87</b> , <b>exec...</b> , <b>spawn...</b> , <b>_status87</b> , <b>system</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Example</b>      | <pre> #include &lt;stdio.h&gt; #include &lt;float.h&gt; #include &lt;setjmp.h&gt; #include &lt;signal.h&gt; </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |



## `_fpreset`

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

jmp_buf reenter;

/* define a handler for trapping floating-point errors */
void float_trap(int sig)
{
 printf("Trapping floating point error, signal is %d\n",sig);
 printf("Press a key to continue\n");
 getch();

 /* reset the 8087 chip or emulator to clear any extraneous garbage */
 _fpreset();

 /* return to the problem spot */
 longjmp(reenter, -1);
}

int main(void)
{
 float one = 3.14, two = 0.0;

 /* install signal handler for floating-point exception */
 if (signal(SIGFPE, float_trap) == SIG_ERR) {
 printf("error installing signal handler\n");
 exit(3);
 }
 printf("Generating a math error, press a key\n");
 getch();
 if (setjmp(reenter) == 0)
 one /= two;
 printf("Returned from signal trap\n");

 return 0;
}
```

## `fprintf`

---

**Function** Writes formatted output to a stream.

**Syntax** `#include <stdio.h>`  
`int fprintf(FILE *stream, const char *format[, argument, ...]);`

**Prototype in** `stdio.h`

**Remarks** `fprintf` accepts a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a stream. There must be the same number of format specifiers as arguments.

See `printf` for details on format specifiers.

**Return value** **fprintf** returns the number of bytes output. In the event of error, it returns EOF.

**Portability** **fprintf** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**See also** **cprintf, fscanf, printf, putc, sprintf**

**Example**

```
#include <stdio.h>

int main(void)
{
 FILE *stream;
 int i = 100;
 char c = 'C';
 float f = 1.234;

 /* open a file for update */
 stream = fopen("DUMMY.FIL", "w+");

 /* write some data to the file */
 fprintf(stream, "%d %c %f", i, c, f);

 /* close the file */
 fclose(stream);
 return 0;
}
```

F

## FP\_SEG

---

**Function** Gets far address segment.

**Syntax** `#include <dos.h>`  
`unsigned FP_SEG(void far *p);`

**Prototype in** dos.h

**Remarks** **FP\_SEG** is a macro that gets or sets the segment value of the far pointer *\*p*.

**Return value** **FP\_SEG** returns an **unsigned** integer representing a segment value.

**Portability** **FP\_SEG** is unique to DOS.

**See also** **FP\_OFF, MK\_FP, movedata, segread**

**Example**

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

int main(void)
```

```

{
 char *filename = "fpseg.c";
 printf("The segment of this string in memory is: %X\n", FP_SEG(filename));
 return 0;
}

```

## fputc

---

|                     |                                                                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Puts a character on a stream.                                                                                                                                                                 |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int fputc(int <i>c</i>, FILE *<i>stream</i>);</code>                                                                                           |
| <b>Prototype in</b> | <code>stdio.h</code>                                                                                                                                                                          |
| <b>Remarks</b>      | <b>fputc</b> outputs character <i>c</i> to the named stream.                                                                                                                                  |
| <b>Return value</b> | On success, <b>fputc</b> returns the character <i>c</i> . On error, it returns EOF.                                                                                                           |
| <b>Portability</b>  | <b>fputc</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                           |
| <b>See also</b>     | <b>fgetc</b> , <b>putc</b>                                                                                                                                                                    |
| <b>Example</b>      | <pre> #include &lt;stdio.h&gt;  int main(void) {     char msg[] = "Hello world";     int i = 0;     while (msg[i]) {         fputc(msg[i], stdout);         i++;     }     return 0; } </pre> |

## fputchar

---

|                     |                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Outputs a character on stdout.                                                                                                  |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int fputchar(int <i>c</i>);</code>                                               |
| <b>Prototype in</b> | <code>stdio.h</code>                                                                                                            |
| <b>Remarks</b>      | <b>fputchar</b> outputs character <i>c</i> to stdout. <b>fputchar(<i>c</i>)</b> is the same as <b>fputc(<i>c</i>, stdout)</b> . |

**Return value** On success, **fputc** returns the character *c*. On error, it returns EOF.

**Portability** **fputc** is available on UNIX systems.

**See also** **fgetchar**, **putchar**

**Example**

```
#include <stdio.h>

int main(void)
{
 char msg[] = "This is a test\n";
 int i = 0;
 while (msg[i]) {
 fputc(msg[i]);
 i++;
 }
 return 0;
}
```



## fputs

---

**Function** Outputs a string on a stream.

**Syntax** `#include <stdio.h>`  
`int fputs(const char *s, FILE *stream);`

**Prototype in** `stdio.h`

**Remarks** **fputs** copies the null-terminated string *s* to the given output stream; it does not append a newline character, and the terminating null character is not copied.

**Return value** On successful completion, **fputs** returns the last character written. Otherwise, it returns a value of EOF.

**Portability** **fputs** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**See also** **fgets**, **gets**, **puts**

**Example**

```
#include <stdio.h>

int main(void)
{
 /* write a string to standard output */
 fputs("Hello world\n", stdout);
 return 0;
}
```

## fread

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reads data from a stream.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>size_t fread(void *ptr, size_t size, size_t n, FILE *stream);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Remarks</b>      | <b>fread</b> reads <i>n</i> items of data, each of length <i>size</i> bytes, from the given input stream into a block pointed to by <i>ptr</i> .<br><br>The total number of bytes read is ( <i>n</i> × <i>size</i> ).                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Return value</b> | On successful completion, <b>fread</b> returns the number of items (not bytes) actually read. It returns a short count (possibly 0) on end-of-file or error.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Portability</b>  | <b>fread</b> is available on all UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>See also</b>     | <b>fopen, fwrite, printf, read</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>      | <pre>#include &lt;string.h&gt; #include &lt;stdio.h&gt;  int main(void) {     FILE *stream;     char msg[] = "this is a test";     char buf[20];     if ((stream = fopen("DUMMY.FIL", "w+")) == NULL) {         fprintf(stderr, "Cannot open output file.\n");         return 1;     }     /* write some data to the file */     fwrite(msg, strlen(msg)+1, 1, stream);      /* seek to the beginning of the file */     fseek(stream, SEEK_SET, 0);      /* read the data and display it */     fread(buf, strlen(msg)+1, 1, stream);     printf("%s\n", buf);     fclose(stream);     return 0; }</pre> |

## free

---

- Function** Frees allocated block.
- Syntax** `#include <alloc.h>`  
`void free(void *block);`
- Prototype in** `stdlib.h`, `alloc.h`
- Remarks** **free** deallocates a memory block allocated by a previous call to **calloc**, **malloc**, or **realloc**.
- Return value** None.
- Portability** **free** is available on UNIX systems and is defined in ANSI C.
- See also** **calloc**, **freemem**, **malloc**, **realloc**, **strdup**

**Example**

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

int main(void)
{
 char *str;

 /* allocate memory for string */
 str = malloc(10);

 /* copy "Hello" to string */
 strcpy(str, "Hello");

 /* display string */
 printf("String is %s\n", str);

 /* free memory */
 free(str);
 return 0;
}
```

## freemem

---

- Function** Frees a previously allocated DOS memory block.
- Syntax** `#include <dos.h>`  
`int freemem(unsigned segx);`
- Prototype in** `dos.h`
- Remarks** **freemem** frees a memory block allocated by a previous call to **allocmem**. *segx* is the segment address of that block.
- Return value** **freemem** returns 0 on success. In the event of error, it returns -1 and the global variable *errno* is set to
- ENOMEM Insufficient memory
- Portability** **freemem** is unique to DOS. This function is not Windows compatible.
- See also** **allocmem**, **free**
- Example**
- ```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>

int main(void)
{
    unsigned int size, segp;
    int stat;
    size = 64;          /* allocmem requests blocks in 16 byte chunks,
                        64 of these is 1024 bytes of memory */
    stat = allocmem(size, &segp);
    if (stat == -1)
        printf("Allocated memory at segment: %x\n", segp);
    else
        printf("Failed: maximum number of paragraphs available is %u\n", stat);
    freemem(segp);
    return 0;
}
```

freopen

Function	Associates a new file with an open stream.
Syntax	<code>#include <stdio.h></code> <code>FILE *freopen(const char *filename, const char *mode, FILE *stream);</code>
Prototype in	stdio.h
Remarks	<p>freopen substitutes the named file in place of the open stream. It closes <i>stream</i>, regardless of whether the open succeeds. freopen is useful for changing the file attached to stdin, stdout, or stderr.</p> <p>The <i>mode</i> string used in calls to fopen is one of the following values:</p> <ul style="list-style-type: none"> <i>r</i> Open for reading only. <i>w</i> Create for writing. <i>a</i> Append; open for writing at end-of-file, or create for writing if the file does not exist. <i>r+</i> Open an existing file for update (reading and writing). <i>w+</i> Create a new file for update. <i>a+</i> Open for append; open (or create if the file does not exist) for update at the end of the file. <p>To specify that a given file is being opened or created in text mode, append a <i>t</i> to the <i>mode</i> string (<i>rt</i>, <i>w+t</i>, and so on); similarly, to specify binary mode, append a <i>b</i> to the <i>mode</i> string (<i>wb</i>, <i>a+b</i>, and so on).</p> <p>If a <i>t</i> or <i>b</i> is not given in the <i>mode</i> string, the mode is governed by the global variable <code>_fmode</code>. If <code>_fmode</code> is set to <code>O_BINARY</code>, files are opened in binary mode. If <code>_fmode</code> is set to <code>O_TEXT</code>, they are opened in text mode. These <code>O_...</code> constants are defined in <code>fcntl.h</code>.</p> <p>When a file is opened for update, both input and output can be done on the resulting stream. However, output cannot be directly followed by input without an intervening fseek or rewind, and input cannot be directly followed by output without an intervening fseek, rewind, or an input that encounters end-of-file.</p>
Return value	On successful completion, freopen returns the argument <i>stream</i> . In the event of error, it returns null.
Portability	freopen is available on UNIX systems and is defined in ANSI C.



freopen

See also `fclose`, `fdopen`, `fopen`, `open`, `setmode`

Example

```
#include <stdio.h>

int main(void)
{
    /* redirect standard output to a file */
    if (freopen("OUTPUT.FIL", "w", stdout) == NULL)
        fprintf(stderr, "error redirecting stdout\n");

    /* this output will go to a file */
    printf("This will go into a file.");

    /* close the standard output stream */
    fclose(stdout);
    return 0;
}
```

frexp

Function Splits a **double** number into mantissa and exponent.

Syntax `#include <math.h>`
`double frexp(double x, int *exponent);`

Prototype in `math.h`

Remarks `frexp` calculates the mantissa *m* (a **double** greater than or equal to 0.5 and less than 1) and the integer value *n*, such that *x* (the original **double** value) equals $m \times 2^n$. `frexp` stores *n* in the integer that *exponent* points to.

Return value `frexp` returns the mantissa *m*.

You can use the function `matherr` to modify error handling for `frexp`.

Portability `frexp` is available on UNIX systems and is defined in ANSI C.

See also `exp`, `ldexp`

Example

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

int main(void)
{
    double mantissa, number;
    int exponent;
    number = 8.0;
    mantissa = frexp(number, &exponent);
    printf("The number %lf is %lf times two to the power of %d\n", number,
           mantissa, exponent);
}
```

```

    return 0;
}

```

fscanf

F

Function Scans and formats input from a stream.

Syntax `#include <stdio.h>`
`int fscanf(FILE *stream, const char *format[, address, ...]);`

Prototype in `stdio.h`

Remarks `fscanf` scans a series of input fields, one character at a time, reading from a stream. Then each field is formatted according to a format specifier passed to `fscanf` in the format string pointed to by *format*. Finally, `fscanf` stores the formatted input at an address passed to it as an argument following *format*. The number of format specifiers and addresses must be the same as the number of input fields.

See *scanf* for details on format specifiers.

`fscanf` can stop scanning a particular field before it reaches the normal end-of-field character (whitespace), or it can terminate entirely for a number of reasons. See `scanf` for a discussion of possible causes.

Return value `fscanf` returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored.

If `fscanf` attempts to read at end-of-file, the return value is EOF. If no fields were stored, the return value is 0.

Portability `fscanf` is available on UNIX systems and is defined in Kernighan and Ritchie. It is compatible with ANSI C.

See also `atoi`, `cscanf`, `fprintf`, `printf`, `scanf`, `sscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example

```

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

int main(void)
{
    int i;
    printf("Input an integer: ");

    /* read an integer from the standard input stream */
    if (fscanf(stdin, "%d", &i))
        printf("The integer read was: %i\n", i);
    else {
        fprintf(stderr, "Error reading an integer from stdin.\n");
    }
}

```

```

        exit(1);
    }
    return 0;
}

```

fseek

Function Repositions a file pointer on a stream.

Syntax `#include <stdio.h>`
`int fseek(FILE *stream, long offset, int whence);`

Prototype in `stdio.h`

Remarks **fseek** sets the file pointer associated with *stream* to a new position that is *offset* bytes from the file location given by *whence*. For text mode streams, *offset* should be 0 or a value returned by **ftell**.

whence must be one of the values 0, 1, or 2, which represent three symbolic constants (defined in `stdio.h`) as follows:

<i>whence</i>		File location
SEEK_SET	(0)	File beginning
SEEK_CUR	(1)	Current file pointer position
SEEK_END	(2)	End-of-file

fseek discards any character pushed back using **ungetc**.

fseek is used with stream I/O; for file handle I/O, use **lseek**.

After **fseek**, the next operation on an update file can be either input or output.

Return value **fseek** returns 0 if the pointer is successfully moved and a nonzero on failure.



fseek can return a zero, indicating that the pointer has been moved successfully, when in fact it has not been. This is because DOS, which actually resets the pointer, does not verify the setting. **fseek** returns an error code only on an unopened file or device.

Portability **fseek** is available on all UNIX systems and is defined in ANSI C.

See also **fgetpos**, **fopen**, **fsetpos**, **ftell**, **lseek**, **rewind**, **setbuf**, **tell**

Example `#include <stdio.h>`
`long filesize(FILE *stream);`

```

int main(void)
{
    FILE *stream;
    stream = fopen("MYFILE.TXT", "w+");
    fprintf(stream, "This is a test");
    printf("Filesize of MYFILE.TXT is %ld bytes\n", filesize(stream));
    fclose(stream);
    return 0;
}

long filesize(FILE *stream) {
    long curpos, length;

    /* save the current location in the file */
    curpos = ftell(stream);

    /* seek to the end of the file */
    fseek(stream, 0L, SEEK_END);

    /* get the current offset into the file */
    length = ftell(stream);

    /* restore saved cursor position */
    fseek(stream, curpos, SEEK_SET);
    return length;
}

```



fsetpos

Function	Positions the file pointer of a stream.
Syntax	<code>#include <stdio.h></code> <code>int fsetpos(FILE *stream, const fpos_t *pos);</code>
Prototype in	stdio.h
Remarks	fsetpos sets the file pointer associated with <i>stream</i> to a new position. The new position is the value obtained by a previous call to fgetpos on that stream. It also clears the end-of-file indicator on the file that <i>stream</i> points to and undoes any effects of ungetc on that file. After a call to fsetpos , the next operation on the file can be input or output.
Return value	On success, fsetpos returns 0. On failure, it returns a nonzero value and also sets the global variable <i>errno</i> to a nonzero value.
Portability	fsetpos is compatible with ANSI C.
See also	fgetpos , fseek , ftell
Example	<code>#include <stdlib.h></code>

fsetpos

```
#include <stdio.h>

void showpos(FILE *stream);

int main(void)
{
    FILE *stream;
    fpos_t filepos;

    /* open a file for update */
    stream = fopen("DUMMY.FIL", "w+");

    /* save the file pointer position */
    fgetpos(stream, &filepos);

    /* write some data to the file */
    fprintf(stream, "This is a test");

    /* show the current file position */
    showpos(stream);

    /* set a new file position and display it */
    if (fsetpos(stream, &filepos) == 0)
        showpos(stream);
    else {
        fprintf(stderr, "Error setting file pointer.\n");
        exit(1);
    }

    /* close the file */
    fclose(stream);
    return 0;
}

void showpos(FILE *stream) {
    fpos_t pos;

    /* display the current file pointer position of a stream */
    fgetpos(stream, &pos);
    printf("File position: %ld\n", pos);
}
```

fstat

Function	Gets open file information.
Syntax	<code>#include <sys\stat.h></code> <code>int fstat(int <i>handle</i>, struct stat *<i>statbuf</i>);</code>
Prototype In	sys\stat.h

Remarks **fstat** stores information in the **stat** structure about the open file or directory associated with *handle*.

statbuf points to the **stat** structure (defined in `sys\stat.h`). That structure contains the following fields:

<i>st_mode</i>	Bit mask giving information about the open file's mode
<i>st_dev</i>	Drive number of disk containing the file, or file handle if the file is on a device
<i>st_rdev</i>	Same as <i>st_dev</i>
<i>st_nlink</i>	Set to the integer constant 1
<i>st_size</i>	Size of the open file in bytes
<i>st_atime</i>	Most recent time the open file was modified
<i>st_mtime</i>	Same as <i>st_atime</i>
<i>st_ctime</i>	Same as <i>st_atime</i>

The **stat** structure contains three more fields not mentioned here. They contain values that are not meaningful under DOS.

The bit mask that gives information about the mode of the open file includes the following bits:

One of the following bits will be set

<code>S_IFCHR</code>	If <i>handle</i> refers to a device.
<code>S_IFREG</code>	If an ordinary file is referred to by <i>handle</i> .

One or both of the following bits will be set

<code>S_IWRITE</code>	If user has permission to write to file.
<code>S_IREAD</code>	If user has permission to read to file.

The bit mask also includes the read/write bits; these are set according to the file's permission mode.

Return value **fstat** returns 0 if it has successfully retrieved the information about the open file. On error (failure to get the information), it returns -1 and sets the global variable *errno* to

<code>EBADF</code>	Bad file handle
--------------------	-----------------

Portability **fstat** is available on UNIX systems and is defined in ANSI C.

See also **access**, **chmod**, **stat**

Example

```
#include <sys\stat.h>
#include <stdio.h>
```

fstat

```
#include <time.h>

int main(void)
{
    struct stat statbuf;
    FILE *stream;

    /* open a file for update */
    if ((stream = fopen("DUMMY.FIL", "w+")) == NULL) {
        fprintf(stderr, "Cannot open output file.\n");
        return(1);
    }
    fprintf(stream, "This is a test");
    fflush(stream);

    /* get information about the file */
    fstat(fileno(stream), &statbuf);
    fclose(stream);

    /* display the information returned */
    if (statbuf.st_mode & S_IFCHR)
        printf("Handle refers to a device.\n");
    if (statbuf.st_mode & S_IFREG)
        printf("Handle refers to an ordinary file.\n");
    if (statbuf.st_mode & S_IRREAD)
        printf("User has read permission on file.\n");
    if (statbuf.st_mode & S_IWRITE)
        printf("User has write permission on file.\n");
    printf("Drive letter of file: %c\n", 'A'+statbuf.st_dev);
    printf("Size of file in bytes: %ld\n", statbuf.st_size);
    printf("Time file last opened: %s\n", ctime(&statbuf.st_ctime));
    return 0;
}
```

_fstrcat

See **strcat**.

_fstrchr

See **strchr**.

_fstrcspn

See **strcspn**.

_fstrdup

See **strdup**.

_fstricmp

See **stricmp**.

_fstrlen

See **strlen**.

_fstrlwr

See **strlwr**.

_fstrncat

See **strncat**.

_fstrncmp

See **strncmp**.

`_fstrncpy`

`_fstrncpy`

See **strncpy**.

`_fstrnicmp`

See **strnicmp**.

`_fstrnset`

See **strnset**.

`_fstrpbrk`

See **strpbrk**.

`_fstrchr`

See **strchr**.

`_fstrrev`

See **strrev**.

`_fstrset`

See **strset**.

_fstrspn

See **strspn**.

_fstrstr

See **strstr**.

_fstrtok

See **strtok**.

_fstrupr

See **strupr**.

ftell

Function	Returns the current file pointer.
Syntax	<code>#include <stdio.h></code> <code>long int ftell(FILE *stream);</code>
Prototype in	stdio.h
Remarks	ftell returns the current file pointer for <i>stream</i> . The offset is measured in bytes from the beginning of the file (if the file is binary). The value returned by ftell can be used in a subsequent call to fseek .
Return value	ftell returns the current file pointer position on success. It returns <code>-1L</code> on error and sets the global variable <i>errno</i> to a positive value.
Portability	ftell is available on all UNIX systems and is defined in ANSI C.
See also	fgetpos , fseek , fsetpos , lseek , rewind , tell
Example	<code>#include <stdio.h></code>

```
int main(void)
{
    FILE *stream;
    stream = fopen("MYFILE.TXT", "w+");
    fprintf(stream, "This is a test");
    printf("The file pointer is at byte %ld\n", ftell(stream));
    fclose(stream);
    return 0;
}
```

ftime

Function	Stores current time in timeb structure.
Syntax	<code>#include <sys\timeb.h></code> <code>void ftime(struct timeb *buf)</code>
Prototype in	<code>sys\timeb.h</code>
Remarks	<p>ftime determines the current time and fills in the fields in the timeb structure pointed to by <i>buf</i>. The timeb structure contains four fields: <i>time</i>, <i>millitm</i>, <i>timezone</i>, and <i>dstflag</i>:</p> <pre>struct timeb { long time ; short millitm ; short timezone ; short dstflag ; };</pre> <ul style="list-style-type: none"> ■ <i>time</i> provides the time in seconds since 00:00:00 Greenwich mean time (GMT), January 1, 1970. ■ <i>millitm</i> is the fractional part of a second in milliseconds. ■ <i>timezone</i> is the difference in minutes between GMT and the local time. This value is computed going west from GMT. ftime gets this field from the global variable <i>timezone</i>, which is set by tzset. ■ <i>dstflag</i> is used to indicate whether daylight saving time will be taken into account during time calculations. <p>➡ ftime calls tzset. Therefore, it isn't necessary to call tzset explicitly when you use ftime.</p>
Return value	None.
Portability	ftime is available on UNIX System V systems.
See also	asctime , ctime , gmtime , localtime , stime , time , tzset

```

Example #include <stdio.h>
           #include <stdlib.h>
           #include <time.h>
           #include <sys\timeb.h>

           /* pacific standard & daylight savings */
           char *tzstr = "TZ=PST8PDT";

           int main(void)
           {
               struct timeb t;
               putenv(tzstr);
               tzset();
               ftime(&t);
               printf("Seconds since 1/1/1970 GMT: %ld\n", t.time);
               printf("Thousandths of a second: %d\n", t.millitm);
               printf("Difference between local time and GMT: %d\n", t.timezone);
               printf("Daylight savings in effect (1) not (0): %d\n", t.dstflag);
               return 0;
           }

```

fwrite

Function Writes to a stream.

Syntax #include <stdio.h>
 size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);

Prototype in stdio.h

Remarks **fwrite** appends *n* items of data, each of length *size* bytes, to the given output file. The data written begins at *ptr*.

The total number of bytes written is ($n \times size$).

ptr in the declarations is a pointer to any object.

Return value On successful completion, **fwrite** returns the number of items (not bytes) actually written. It returns a short count on error.

Portability **fwrite** is available on all UNIX systems and is defined in ANSI C.

See also **fopen**, **fread**

Example #include <stdio.h>

```

struct mystruct
{
    int i;

```

fwrite

```
    char ch;
};

int main(void)
{
    FILE *stream;
    struct mystruct s;

    /* open file TEST.$$$ */
    if ((stream = fopen("TEST.$$$", "wb")) == NULL) {
        fprintf(stderr, "Cannot open output file.\n");
        return 1;
    }
    s.i = 0;
    s.ch = 'A';
    fwrite(&s, sizeof(s), 1, stream); /* write struct s to file */
    fclose(stream);                  /* close file */
    return 0;
}
```

gcvt

- Function** Converts floating-point number to a string.
- Syntax** `#include <stdlib.h>`
`char *gcvt(double value, int ndec, char *buf);`
- Prototype in** `stdlib.h`
- Remarks** **gcvt** converts *value* to a null-terminated ASCII string and stores the string in *buf*. It produces *ndec* significant digits in FORTRAN F format, if possible; otherwise, it returns the value in the **printf** E format (ready for printing). It might suppress trailing zeros.
- Return value** **gcvt** returns the address of the string pointed to by *buf*.
- Portability** **gcvt** is available on UNIX. It is not in ANSI C, and is not recommended for portable programs.
- See also** **ecvt**, **fcvt**, **sprintf**
- Example**
- ```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
 char str[25];
 double num;
 int sig = 5; /* significant digits */
```

```

 /* a regular number */
 num = 9.876;
 gcvt(num, sig, str);
 printf("string = %s\n", str);

 /* a negative number */
 num = -123.4567;
 gcvt(num, sig, str);
 printf("string = %s\n", str);

 /* scientific notation */
 num = 0.678e5;
 gcvt(num, sig, str);
 printf("string = %s\n", str);
 return(0);
}

```

## geninterrupt

---

**Function** Generates a software interrupt.

**Syntax** `#include <dos.h>`  
`void geninterrupt(int intr_num);`

**Prototype in** dos.h

**Remarks** The **geninterrupt** macro triggers a software trap for the interrupt given by *intr\_num*. The state of all registers after the call depends on the interrupt called.

➡ Interrupts can leave registers used by C in unpredictable states.

**Return value** None.

**Portability** **geninterrupt** is unique to the 8086 architecture.

**See also** **bdos**, **bdosptr**, **disable**, **enable**, **getvect**, **int86**, **int86x**, **intdos**, **intdosx**, **intr**

**Example**

```

#include <conio.h>
#include <dos.h>

void writechar(char ch); /* function prototype */

int main(void)
{
 clrscr();
 gotoxy(80,25);
 writechar('*');
 getch();
 return 0;
}

```

```

}

/* outputs a character at the current cursor position */
/* using the video BIOS to avoid scrolling of the screen */
/* when writing to location (80,25) */

void writechar(char ch) {
 struct text_info ti;
 gettextinfo(&ti); /* grab current text settings */
 _AH = 9; /* interrupt 0x10 sub-function 9 */
 _AL = ch; /* character to be output */
 _BH = 0; /* video page */
 _BL = ti.attribute; /* video attribute */
 _CX = 1; /* repetition factor */
 geninterrupt(0x10); /* output the char */
}

```

## ← getarccoords

---

**Function** Gets coordinates of the last call to **arc**.

**Syntax** `#include <graphics.h>`  
`void far getarccoords(struct arccoordstype far *arccoords);`

**Prototype in** graphics.h

**Remarks** **getarccoords** fills in the **arccoordstype** structure pointed to by *arccoords* with information about the last call to **arc**. The **arccoordstype** structure is defined in graphics.h as follows:

```

struct arccoordstype {
 int x, y;
 int xstart, ystart, xend, yend;
};

```

The members of this structure are used to specify the center point (*x,y*), the starting position (*xstart, ystart*), and the ending position (*xend, yend*) of the arc. These values are useful if you need to make a line meet at the end of an arc.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **arc, fillellipse, sector**

**Example** `#include <graphics.h>`

```

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 struct arccoordstype arcinfo;
 int midx, midy;
 int stangle = 45, endangle = 270;
 char sstr[80], estr[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* draw arc and get coordinates */
 setcolor(getmaxcolor());
 arc(midx, midy, stangle, endangle, 100);
 getarccoords(&arcinfo);

 /* convert arc information into strings */
 sprintf(ssstr, "%d %d", arcinfo.xstart, arcinfo.ystart);
 sprintf(estr, "%d %d", arcinfo.xend, arcinfo.yend);

 /* output the arc information */
 outtextxy(arcinfo.xstart, arcinfo.ystart, sstr);
 outtextxy(arcinfo.xend, arcinfo.yend, estr);

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```



## getaspectratio

---

- Function** Retrieves the current graphics mode's aspect ratio.
- Syntax** `#include <graphics.h>`  
`void far getaspectratio(int far *xasp, int far *yasp);`
- Prototype in** `graphics.h`
- Remarks** The *y* aspect factor, *\*yasp*, is normalized to 10,000. On all graphics adapters except the VGA, *\*xasp* (the *x* aspect factor) is less than *\*yasp* because the pixels are taller than they are wide. On the VGA, which has "square" pixels, *\*xasp* equals *\*yasp*. In general, the relationship between *\*yasp* and *\*xasp* can be stated as
- $$\begin{aligned} *yasp &= 10,000 \\ *xasp &\leq 10,000 \end{aligned}$$
- getaspectratio** gets the values in *\*xasp* and *\*yasp*.
- Return value** None.
- Portability** A similar routine exists in Turbo Pascal. This function is not Windows compatible.
- See also** **arc, circle, ellipse, fillellipse, pieslice, sector, setaspectratio**

**Example**

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

main()
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int xasp, yasp, midx, midy;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }
}
```

```

midx = getmaxx() / 2;
midy = getmaxy() / 2;
setcolor(getmaxcolor());

/* get current aspect ratio settings */
getaspectratio(&xasp, &yasp);

/* draw normal circle */
circle(midx, midy, 100);
getch();

/* draw wide circle */
cleardevice();
setaspectratio(xasp/2, yasp);
circle(midx, midy, 100);
getch();

/* draw narrow circle */
cleardevice();
setaspectratio(xasp, yasp/2);
circle(midx, midy, 100);

/* clean up */
getch();
closegraph();
return 0;
}

```

## getbkcolor

---

**Function** Returns the current background color.

**Syntax** #include <graphics.h>  
int far getbkcolor(void);

**Prototype in** graphics.h

**Remarks** **getbkcolor** returns the current background color. (See the table under **setbkcolor** for details.)

**Return value** **getbkcolor** returns the current background color.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **getcolor**, **getmaxcolor**, **getpalette**, **setbkcolor**

**Example** #include <graphics.h>  
#include <stdlib.h>

## getbkcolor

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int bkcolor, midx, midy;
 char bkname[35];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;
 setcolor(getmaxcolor());

 /* for centering text on the display */
 settextjustify(CENTER_TEXT, CENTER_TEXT);

 /* get the current background color */
 bkcolor = getbkcolor();

 /* convert color value into a string */
 itoa(bkcolor, bkname, 10);
 strcat(bkname, " is the current background color.");

 /* display a message */
 outtextxy(midx, midy, bkname);

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

## getc

---

|                     |                                                                                                                                                                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets character from stream.                                                                                                                                                                                                                              |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int getc(FILE *stream);</code>                                                                                                                                                                            |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                  |
| <b>Remarks</b>      | <b>getc</b> is a macro that returns the next character on the given input stream and increments the stream's file pointer to point to the next character.                                                                                                |
| <b>Return value</b> | On success, <b>getc</b> returns the character read, after converting it to an <b>int</b> without sign extension. On end-of-file or error, it returns EOF.                                                                                                |
| <b>Portability</b>  | <b>getc</b> is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.                                                                                                                                          |
| <b>See also</b>     | <b>fgetc</b> , <b>getch</b> , <b>getchar</b> , <b>getche</b> , <b>gets</b> , <b>putc</b> , <b>putchar</b> , <b>ungetc</b>                                                                                                                                |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;  int main(void) {     char ch;     printf("Input a character:");      /* read a character from the standard input stream */     ch = getc(stdin);     printf("The character input was: '%c'\n", ch);     return 0; }</pre> |

G

## getcbrk

---

|                     |                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets control-break setting.                                                                           |
| <b>Syntax</b>       | <code>#include &lt;dos.h&gt;</code><br><code>int getcbrk(void);</code>                                |
| <b>Prototype in</b> | dos.h                                                                                                 |
| <b>Remarks</b>      | <b>getcbrk</b> uses the DOS system call 0x33 to return the current setting of control-break checking. |
| <b>Return value</b> | <b>getcbrk</b> returns 0 if control-break checking is off, or 1 if checking is on.                    |

## getcbrk

**Portability** `getcbrk` is unique to DOS.

**See also** `ctrlbrk`, `setcbrk`

**Example**

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

int main(void)
{
 if (getcbrk())
 printf("Ctrl-brk flag is on\n");
 else
 printf("Ctrl-brk flag is off\n");
 return 0;
}
```

## getch

---

**Function** Gets character from keyboard, does not echo to screen.

**Syntax** `#include <conio.h>`  
`int getch(void);`

**Prototype in** `conio.h`

**Remarks** `getch` reads a single character directly from the keyboard, without echoing to the screen.

**Return value** `getch` returns the character read from the keyboard.

**Portability** `getch` is unique to DOS. This function is not Windows compatible.

**See also** `cgets`, `cscanf`, `fgetc`, `getc`, `getchar`, `getche`, `getpass`, `kbhit`, `putch`, `ungetch`

**Example**

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

int main(void)
{
 int c;
 int extended = 0;
 c = getch();
 if (!c)
 extended = getch();
 if (extended)
 printf("The character is extended\n");
 else
 printf("The character isn't extended\n");
}
```

```

 return 0;
}

```

## getchar

---

**Function** Gets character from stdin.

**Syntax** `#include <stdio.h>`  
`int getchar(void);`

**Prototype in** `stdio.h`

**Remarks** **getchar** is a macro that returns the next character on the named input stream `stdin`. It is defined to be **getc(stdin)**.

**Return value** On success, **getchar** returns the character read, after converting it to an **int** without sign extension. On end-of-file or error, it returns EOF.

**Portability** **getchar** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie. This function is not Windows compatible.

**See also** **fgetc**, **fgetchar**, **getc**, **getch**, **getche**, **gets**, **putc**, **putchar**, **scanf**, **ungetc**

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

```

int main(void)
{
 int c;

 /* Note that getchar reads from stdin and is line buffered; */
 /* this means it will not return until you press <ENTER> */
 while ((c = getchar()) != '\n')
 printf("%c", c);
 return 0;
}

```

## getche

---

**Function** Gets character from the keyboard, echoes to screen.

**Syntax** `#include <conio.h>`  
`int getche(void);`

**Prototype in** `conio.h`

## getche

|                     |                                                                                                                                                                                                          |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Remarks</b>      | <b>getche</b> reads a single character from the keyboard and echoes it to the current text window, using direct video or BIOS.                                                                           |
| <b>Return value</b> | <b>getche</b> returns the character read from the keyboard.                                                                                                                                              |
| <b>Portability</b>  | <b>getche</b> is unique to DOS. This function is not Windows compatible.                                                                                                                                 |
| <b>See also</b>     | <b>cgets, cscanf, fgetc, getc, getch, getchar, kbhit, putch, ungetch</b>                                                                                                                                 |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;conio.h&gt;  int main(void) {     char ch;     printf("Input a character:");     ch = getche();     printf("\nYou input a '%c'\n", ch);     return 0; }</pre> |

## getcolor

---

|                     |                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Returns the current drawing color.                                                                                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>       | <pre>#include &lt;graphics.h&gt; int far getcolor(void);</pre>                                                                                                                                                                                                                                                                                                        |
| <b>Prototype in</b> | graphics.h                                                                                                                                                                                                                                                                                                                                                            |
| <b>Remarks</b>      | <p><b>getcolor</b> returns the current drawing color.</p> <p>The drawing color is the value to which pixels are set when lines and so on are drawn. For example, in CGAC0 mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, if <b>getcolor</b> returns 1, the current drawing color is light green.</p> |
| <b>Return value</b> | <b>getcolor</b> returns the current drawing color.                                                                                                                                                                                                                                                                                                                    |
| <b>Portability</b>  | This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.                                                                                                                                                                                        |
| <b>See also</b>     | <b>getbkcolor, getmaxcolor, getpalette, setcolor</b>                                                                                                                                                                                                                                                                                                                  |
| <b>Example</b>      | <pre>#include &lt;graphics.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #include &lt;stdio.h&gt;</pre>                                                                                                                                                                                                                                                   |

```
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int color, midx, midy;
 char colname[35];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;
 setcolor(getmaxcolor());

 /* for centering text on the display */
 settextjustify(CENTER_TEXT, CENTER_TEXT);

 /* get the current drawing color */
 color = getcolor();

 /* convert color value into a string */
 itoa(color, colname, 10);
 strcat(colname, " is the current drawing color.");

 /* display a message */
 outtextxy(midx, midy, colname);

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```



## getcurdir

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets current directory for specified drive.                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Syntax</b>       | <pre>#include &lt;dir.h&gt; int getcurdir(int <i>drive</i>, char *<i>directory</i>);</pre>                                                                                                                                                                                                                                                                                                                             |
| <b>Prototype in</b> | dir.h                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Remarks</b>      | <p><b>getcurdir</b> gets the name of the current working directory for the drive indicated by <i>drive</i>.</p> <p><i>drive</i> specifies a drive number (0 for default, 1 for A, and so on).</p> <p><i>directory</i> points to an area of memory of length MAXDIR where the null-terminated directory name will be placed. The name does not contain the drive specification and does not begin with a backslash.</p> |
| <b>Return value</b> | <b>getcurdir</b> returns 0 on success or -1 in the event of error.                                                                                                                                                                                                                                                                                                                                                     |
| <b>Portability</b>  | <b>getcurdir</b> is unique to DOS.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>See also</b>     | <b>chdir</b> , <b>getcwd</b> , <b>getdisk</b> , <b>mkdir</b> , <b>rmdir</b>                                                                                                                                                                                                                                                                                                                                            |

## getcwd

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets current working directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Syntax</b>       | <pre>#include &lt;dir.h&gt; char *getcwd(char *<i>buf</i>, int <i>buflen</i>);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Prototype in</b> | dir.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Remarks</b>      | <p><b>getcwd</b> gets the full path name (including the drive) of the current working directory, up to <i>buflen</i> bytes long and stores it in <i>buf</i>. If the full path name length (including the null terminator) is longer than <i>buflen</i> bytes, an error occurs.</p> <p>If <i>buf</i> is null, a buffer <i>buflen</i> bytes long is allocated for you with <b>malloc</b>. You can later free the allocated buffer by passing the return value of <b>getcwd</b> to the function <b>free</b>.</p> |
| <b>Return value</b> | <p><b>getcwd</b> returns the following values:</p> <ul style="list-style-type: none"> <li>■ If <i>buf</i> is not null on input, <b>getcwd</b> returns <i>buf</i> on success, null on error.</li> <li>■ If <i>buf</i> is null on input, <b>getcwd</b> returns a pointer to the allocated buffer.</li> </ul>                                                                                                                                                                                                    |

In the event of an error return, the global variable *errno* is set to one of the following:

|        |                     |
|--------|---------------------|
| ENODEV | No such device      |
| ENOMEM | Not enough core     |
| ERANGE | Result out of range |

**Portability** **getcwd** is unique to DOS.

**See also** **chdir**, **getcurdir**, **getdisk**, **mkdir**, **rmdir**

**Example**

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

int main(void)
{
 char buffer[MAXPATH];
 getcwd(buffer, MAXPATH);
 printf("The current directory is: %s\n", buffer);
 return 0;
}
```

G

## getdate

---

**Function** Gets system date.

**Syntax** `#include <dos.h>`  
`void getdate(struct date *datep);`

**Prototype in** dos.h

**Remarks** **getdate** fills in the **date** structure (pointed to by *datep*) with the system's current date.

The **date** structure is defined as follows:

```
struct date {
 int da_year; /* current year */
 char da_day; /* day of the month */
 char da_mon; /* month (1 = Jan) */
};
```

**Return value** None.

**Portability** **getdate** is unique to DOS.

**See also** **ctime**, **gettime**, **setdate**, **settime**

**Example**

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

## getdate

```
int main(void)
{
 struct date d;
 getdate(&d);
 printf("The current year is: %d\n", d.da_year);
 printf("The current day is: %d\n", d.da_day);
 printf("The current month is: %d\n", d.da_mon);
 return 0;
}
```

## getdefaultpalette

---

- Function** Returns the palette definition structure.
- Syntax** `#include <graphics.h>`  
`struct palettetype *far getdefaultpalette(void);`
- Prototype in** `graphics.h`
- Remarks** **getdefaultpalette** finds the **palettetype** structure that contains the palette initialized by the driver during **initgraph**.
- Return value** **getdefaultpalette** returns a pointer to the default palette set up by the current driver when that driver was initialized.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getpalette, initgraph**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* far pointer to palette structure */
    struct palettetype far *pal = NULL;
    int i;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
```

```

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

/* return a pointer to the default palette */
pal = getdefaultpalette();
for (i=0; i<pal->size; i++) {
    printf("colors[%d] = %d\n", i, pal->colors[i]);
    getch();
}

/* clean up */
getch();
closegraph();
return 0;
}

```

getdfree

Function Gets disk free space.

Syntax #include <dos.h>
void getdfree(unsigned char *drive*, struct dfree **dtable*);

Prototype in dos.h

Remarks **getdfree** accepts a drive specifier in *drive* (0 for default, 1 for A, and so on) and fills in the **dfree** structure pointed to by *dtable* with disk characteristics.

The **dfree** structure is defined as follows:

```

struct dfree {
    unsigned df_avail; /* available clusters */
    unsigned df_total; /* total clusters */
    unsigned df_bsec; /* bytes per sector */
    unsigned df_sclus; /* sectors per cluster */
};

```

Return value **getdfree** returns no value. In the event of an error, *df_sclus* in the **dfree** structure is set to 0xFFFF.

Portability **getdfree** is unique to DOS.

getdfree

See also [getfat](#), [getfatd](#)

Example

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

int main(void)
{
    struct dfree free;
    long avail;
    int drive;
    drive = getdisk();
    getdfree(drive+1, &free);
    if (free.df_sclus == 0xFFFF) {
        printf("Error in getdfree() call\n");
        exit(1);
    }
    avail = (long) free.df_avail * (long) free.df_bsec * (long) free.df_sclus;
    printf("Drive %c: has %ld bytes available\n", 'A' + drive, avail);
    return 0;
}
```

getdisk

Function Gets current drive number.

Syntax `#include <dir.h>`
`int getdisk(void);`

Prototype in `dir.h`

Remarks **getdisk** gets the current drive number. It returns an integer: 0 for A, 1 for B, 2 for C, and so on (equivalent to DOS function 0x19).

Return value **getdisk** returns the current drive number.

Portability **getdisk** is unique to DOS.

See also [getcurdir](#), [getcwd](#), [setdisk](#)

Example

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

int main(void)
{
    int disk;
    disk = getdisk() + 'A';
}
```

```

    printf("The current drive is: %c\n", disk);
    return 0;
}

```

getdrivername

G

Function Returns a pointer to a string containing the name of the current graphics driver.

Syntax `#include <graphics.h>`
`char *far getdrivername(void);`

Prototype in `graphics.h`

Remarks After a call to **initgraph**, **getdrivername** returns the name of the driver that is currently loaded.

Return value **getdrivername** returns a pointer to a string with the name of the currently loaded graphics driver.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **initgraph**

Example

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* stores the device driver name */
    char *drivername;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
    }
}

```

getdrivename

```
        exit(1);          /* terminate with an error code */
    }
    setcolor(getmaxcolor());

    /* get the name of the device driver in use */
    drivename = getdrivename();

    /* for centering text onscreen */
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    /* output the name of the driver */
    outtextxy(getmaxx() / 2, getmaxy() / 2, drivename);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getdta

- Function** Gets disk transfer address.
- Syntax** `#include <dos.h>`
`char far *getdta(void);`
- Prototype in** dos.h
- Remarks** **getdta** returns the current setting of the disk transfer address (DTA).
In the small and medium memory models, it's assumed the segment is the current data segment. If you use C exclusively, this will be the case, but assembly routines can set the DTA to any hardware address.
In the compact, large, or huge memory models, the address returned by **getdta** is the correct hardware address and can be located outside the program.
- Return value** **getdta** returns a far pointer to the current DTA.
- Portability** **getdta** is unique to DOS.
- See also** **fc** (structure), **setdta**
- Example**
- ```
#include <dos.h>
#include <stdio.h>

int main(void)
{
 char far *dta;
 dta = getdta();
}
```

```

 printf("The current disk transfer address is: %Fp\n", dta);
 return 0;
}

```

## getenv

---

**Function** Gets a string from environment.

**Syntax** `#include <stdlib.h>`  
`char *getenv(const char *name);`

**Prototype in** `stdlib.h`

**Remarks** `getenv` returns the value of a specified variable. The variable name can be either uppercase or lowercase, but it must not include the equal sign (=). If the specified environment variable does not exist, `getenv` returns a NULL pointer.

**Return value** On success, `getenv` returns the value associated with *name*. If the specified *name* is not defined in the environment, `getenv` returns a NULL pointer.



Environment entries must not be changed directly. If you want to change an environment value, you must use `putenv`.

**Portability** `getenv` is available on UNIX systems and is defined in ANSI C.

**See also** `environ` (global variable), `getpsp`, `putenv`

**Example**

```

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

int main(void)
{
 char *s;

 /* get the comspec environment parameter */
 s=getenv("COMSPEC");

 /* display comspec parameter */
 printf("Command processor: %s\n",s);
 return 0;
}

```



# getfat

---

**Function** Gets file allocation table information for given drive.

**Syntax** `#include <dos.h>`  
`void getfat(unsigned char drive, struct fatinfo *dtable);`

**Prototype in** dos.h

**Remarks** **getfat** gets information from the file allocation table (FAT) for the drive specified by *drive* (0 for default, 1 for A, 2 for B, and so on). *dtable* points to the **fatinfo** structure to be filled in. The **fatinfo** structure filled in by **getfat** is defined as follows:

```
struct fatinfo {
 char fi_sclus; /* sectors per cluster */
 char fi_fatid; /* the FAT id byte */
 int fi_nclus; /* number of clusters */
 int fi_bysec; /* bytes per sector */
};
```

**Return value** None.

**Portability** **getfat** is unique to DOS.

**See also** **getdfree**, **getfatd**

**Example**

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

int main()
{
 struct fatinfo diskinfo;
 int flag = 0;
 printf("Please insert a diskette in drive 'A'\n");
 getch();
 getfat(1, &diskinfo); /* get drive information */
 printf("\nDrive A: is ");
 switch((unsigned char) diskinfo.fi_fatid) {
 case 0xFD: printf("a 360K low density\n");
 break;
 case 0xF9: printf("a 1.2 Meg 5-1/4" or 720 K 3-1/2"\n");
 break;
 case 0xF0: printf("1.44 Meg 3-1/2"\n");
 break;
 default: printf("unformatted\n");
 flag = 1;
 }
}
```

```

if (!flag) {
 printf("sectors per cluster: %5d\n", diskinfo.fi_sclus);
 printf("number of clusters: %5d\n", diskinfo.fi_nclus);
 printf("bytes per sector: %5d\n", diskinfo.fi_bysec);
}
return 0;
}

```

## getfatd

## G

**Function** Gets file allocation table information.

**Syntax** `#include <dos.h>`  
`void getfatd(struct fatinfo *dtable);`

**Prototype in** dos.h

**Remarks** **getfatd** gets information from the file allocation table (FAT) of the default drive. *dtable* points to the **fatinfo** structure to be filled in.

The **fatinfo** structure filled in by **getfatd** is defined as follows:

```

struct fatinfo {
 char fi_sclus; /* sectors per cluster */
 char fi_fatid; /* the FAT id byte */
 int fi_nclus; /* number of clusters */
 int fi_bysec; /* bytes per sector */
};

```

**Return value** None.

**Portability** **getfatd** is unique to DOS.

**See also** **getdfree**, **getfat**

**Example**

```

#include <stdio.h>
#include <dos.h>

int main()
{
 struct fatinfo diskinfo;

 /* get default drive information */
 getfatd(&diskinfo);
 printf("\nDefault Drive:\n");
 printf("sectors per cluster: %5d\n",diskinfo.fi_sclus);
 printf("FAT ID byte: %5X\n",diskinfo.fi_fatid & 0xFF);
 printf("number of clusters %5d\n",diskinfo.fi_nclus);
 printf("bytes per sector %5d\n",diskinfo.fi_bysec);
}

```

```

 return 0;
 }

```

## getfillpattern

---

**Function** Copies a user-defined fill pattern into memory.

**Syntax** `#include <graphics.h>`  
`void far getfillpattern(char far *pattern);`

**Prototype in** graphics.h

**Remarks** **getfillpattern** copies the user-defined fill pattern, as set by **setfillpattern**, into the 8-byte area pointed to by *pattern*.

*pattern* is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel will be plotted. For example, the following user-defined fill pattern represents a checkerboard:

```

char checkboard[8] = {
 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55
};

```

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **getfillsettings**, **setfillpattern**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int maxx, maxy;
 char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x25, 0x27, 0x04, 0x04};

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();

```

```

if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();
setcolor(getmaxcolor());

/* select a user-defined fill pattern */
setfillpattern(pattern, getmaxcolor());

/* fill the screen with the pattern */
bar(0, 0, maxx, maxy);
getch();

/* get the current user-defined fill pattern */
getfillpattern(pattern);

/* alter the pattern we grabbed */
pattern[4] -= 1;
pattern[5] -= 3;
pattern[6] += 3;
pattern[7] -= 4;

/* select our new pattern */
setfillpattern(pattern, getmaxcolor());

/* fill the screen with the new pattern */
bar(0, 0, maxx, maxy);

/* clean up */
getch();
closegraph();
return 0;
}

```

## getfillsettings

---

**Function** Gets information about current fill pattern and color.

**Syntax** #include <graphics.h>  
void far getfillsettings(struct fillsettingstype far \*fillinfo);

**Prototype in** graphics.h

**Remarks** `getfillsettings` fills in the `fillsettingstype` structure pointed to by `fillinfo` with information about the current fill pattern and fill color. The `fillsettingstype` structure is defined in `graphics.h` as follows:

```
struct fillsettingstype {
 int pattern; /* current fill pattern */
 int color; /* current fill color */
};
```

The functions `bar`, `bar3d`, `fillpoly`, `floodfill`, and `pieslice` all fill an area with the current fill pattern in the current fill color. There are 11 predefined fill pattern styles (such as solid, crosshatch, dotted, and so on). Symbolic names for the predefined patterns are provided by the enumerated type `fill_patterns` in `graphics.h` (see the following table). In addition, you can define your own fill pattern.

If `pattern` equals 12 (`USER_FILL`), then a user-defined fill pattern is being used; otherwise, `pattern` gives the number of a predefined pattern.

The enumerated type `fill_patterns`, defined in `graphics.h`, gives names for the predefined fill patterns, plus an indicator for a user-defined pattern.

| Name                         | Value | Description                |
|------------------------------|-------|----------------------------|
| <code>EMPTY_FILL</code>      | 0     | Fill with background color |
| <code>SOLID_FILL</code>      | 1     | Solid fill                 |
| <code>LINE_FILL</code>       | 2     | Fill with —                |
| <code>LTSLASH_FILL</code>    | 3     | Fill with ///              |
| <code>SLASH_FILL</code>      | 4     | Fill with ///, thick lines |
| <code>BKSLASH_FILL</code>    | 5     | Fill with \\, thick lines  |
| <code>LTBKSLASH_FILL</code>  | 6     | Fill with \\               |
| <code>HATCH_FILL</code>      | 7     | Light hatch fill           |
| <code>XHATCH_FILL</code>     | 8     | Heavy crosshatch fill      |
| <code>INTERLEAVE_FILL</code> | 9     | Interleaving line fill     |
| <code>WIDE_DOT_FILL</code>   | 10    | Widely spaced dot fill     |
| <code>CLOSE_DOT_FILL</code>  | 11    | Closely spaced dot fill    |
| <code>USER_FILL</code>       | 12    | User-defined fill pattern  |

All but `EMPTY_FILL` fill with the current fill color; `EMPTY_FILL` uses the current background color.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** `getfillpattern`, `setfillpattern`, `setfillstyle`

**Example** `#include <graphics.h>`

```

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

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL", "LTSLASH_FILL",
 "SLASH_FILL", "BKSLASH_FILL", "LTBKSLASH_FILL", "HATCH_FILL",
 "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL",
 "CLOSE_DOT_FILL", "USER_FILL" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 struct fillsettingstype fillinfo;
 int midx, midy;
 char patstr[40], colstr[40];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* get info about current fill pattern and color */
 getfillsettings(&fillinfo);

 /* convert fill information into strings */
 sprintf(patstr, "%s is the fill style.", fname[fillinfo.pattern]);
 sprintf(colstr, "%d is the fill color.", fillinfo.color);

 /* display the information */
 setttextjustify(CENTER_TEXT, CENTER_TEXT);
 outtextxy(midx, midy, patstr);
 outtextxy(midx, midy+2*textheight("W"), colstr);

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

# getftime

---

**Function** Gets file date and time.

**Syntax** `#include <io.h>`  
`int getftime(int handle, struct ftime *ftimep);`

**Prototype in** `io.h`

**Remarks** **getftime** retrieves the file time and date for the disk file associated with the open *handle*. The **ftime** structure pointed to by *ftimep* is filled in with the file's time and date.

The **ftime** structure is defined as follows:

```
struct ftime {
 unsigned ft_tsec: 5; /* two seconds */
 unsigned ft_min: 6; /* minutes */
 unsigned ft_hour: 5; /* hours */
 unsigned ft_day: 5; /* days */
 unsigned ft_month: 4; /* months */
 unsigned ft_year: 7; /* year - 1980*/
};
```

**Return value** **getftime** returns 0 on success.

In the event of an error return, `-1` is returned and the global variable *errno* is set to one of the following:

|         |                         |
|---------|-------------------------|
| EINVFNC | Invalid function number |
| EBADF   | Bad file number         |

**Portability** **getftime** is unique to DOS.

**See also** **open**, **setftime**

**Example**

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

int main()
{
 FILE *stream;
 struct ftime ft;
 if ((stream = fopen("TEST.$$$", "wt")) == NULL)
 \{
 fprintf(stderr, "Cannot open output file.\n");
 return 1;
 }
 getftime(fileno(stream), &ft);
 printf("File time: %02u:%02u:%02u\n", ft.ft_hour, ft.ft_min, ft.ft_tsec * 2);
```

```

printf("File date: %02u/%02u/%04u\n", ft.ft_month, ft.ft_day,
 ft.ft_year+1980);
fclose(stream);
return 0;
}

```

## getgraphmode

G

**Function** Returns the current graphics mode.

**Syntax** `#include <graphics.h>`  
`int far getgraphmode(void);`

**Prototype in** `graphics.h`

**Remarks** Your program must make a successful call to **initgraph** before calling **getgraphmode**.

The enumeration *graphics\_mode*, defined in `graphics.h`, gives names for the predefined graphics modes. For a table listing these enumeration values, refer to the description for **initgraph**.

**Return value** **getgraphmode** returns the graphics mode set by **initgraph** or **setgraphmode**.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **getmoderange**, **restorecrtmode**, **setgraphmode**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, mode;
 char numname[80], modename[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */

```



## getgraphmode

```
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = .getmaxy() / 2;

/* get mode number and name strings */
mode = getgraphmode();
sprintf(numname, "%d is the current mode number.", mode);
sprintf(modename, "%s is the current graphics mode.", getmodename(mode));

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, numname);
outtextxy(midx, midy+2*textheight("W"), modename);

/* clean up */
getch();
closegraph();
return 0;
}
```

## getimage

---

- Function** Saves a bit image of the specified region into memory.
- Syntax** `#include <graphics.h>`  
`void far getimage(int left, int top, int right, int bottom, void far *bitmap);`
- Prototype in** `graphics.h`
- Remarks** **getimage** copies an image from the screen to memory.  
*left*, *top*, *right*, and *bottom* define the screen area to which the rectangle is copied. *bitmap* points to the area in memory where the bit image is stored. The first two words of this area are used for the width and height of the rectangle; the remainder holds the image itself.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **imagesize**, **putimage**, **putpixel**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

void save_screen(void far *buf[4]);
void restore_screen(void far *buf[4]);

int maxx, maxy;
int main(void)
{
 int gdriver=DETECT, gmode, errorcode;
 void far *ptr[4];

 /* autodetect the graphics driver and mode */
 initgraph(&gdriver, &gmode, "");
 errorcode = graphresult(); /* check for any errors */
 if (errorcode != grOk) {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1);
 }

 maxx = getmaxx();
 maxy = getmaxy();

 /* draw an image on the screen */
 rectangle(0, 0, maxx, maxy);
 line(0, 0, maxx, maxy);
 line(0, maxy, maxx, 0);
 save_screen(ptr); /* save the current screen */
 getch(); /* pause screen */
 cleardevice(); /* clear screen */
 restore_screen(ptr); /* restore the screen */
 getch(); /* pause screen */
 closegraph();
 return 0;
}

void save_screen(void far *buf[4])
{
 unsigned size;
 int ystart=0, yend, yincr, block;
 yincr = (maxy+1) / 4;
 yend = yincr;

 /* get byte size of image */
 size = imagesize(0, ystart, maxx, yend);
 for (block=0; block<=3; block++) {

```

## getimage

```
 if ((buf[block] = farmalloc(size)) == NULL) {
 closegraph();
 printf("Error: not enough heap space in save_screen().\n");
 exit(1);
 }
 getimage(0, ystart, maxx, yend, buf[block]);
 ystart = yend + 1;
 yend += yincr + 1;
 }
}

void restore_screen(void far *buf[4])
{
 int ystart=0, yend, yincr, block;
 yincr = (maxy+1) / 4;
 yend = yincr;
 for (block=0; block<=3; block++) {
 putimage(0, ystart, buf[block], COPY_PUT);
 farfree(buf[block]);
 ystart = yend + 1;

 yend += yincr + 1;
 }
}
```

## getlinesettings

---

- Function** Gets the current line style, pattern, and thickness.
- Syntax** `#include <graphics.h>`  
`void far getlinesettings(struct linesettingstype far *lineinfo);`
- Prototype in** `graphics.h`
- Remarks** **getlinesettings** fills a **linesettingstype** structure pointed to by *lineinfo* with information about the current line style, pattern, and thickness.

The **linesettingstype** structure is defined in `graphics.h` as follows:

```
struct linesettingstype {
 int linestyle;
 unsigned upattern;
 int thickness;
};
```

*linestyle* specifies in which style subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration *line\_styles*, defined in `graphics.h`, gives names to these operators:

| Name         | Value | Description             |
|--------------|-------|-------------------------|
| SOLID_LINE   | 0     | Solid line              |
| DOTTED_LINE  | 1     | Dotted line             |
| CENTER_LINE  | 2     | Centered line           |
| DASHED_LINE  | 3     | Dashed line             |
| USERBIT_LINE | 4     | User-defined line style |

*thickness* specifies whether the width of subsequent lines drawn will be normal or thick.

| Name        | Value | Description   |
|-------------|-------|---------------|
| NORM_WIDTH  | 1     | 1 pixel wide  |
| THICK_WIDTH | 3     | 3 pixels wide |

*upattern* is a 16-bit pattern that applies only if *linestyle* is USERBIT\_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. For example, a solid line corresponds to a *upattern* of 0xFFFF (all pixels drawn), while a dashed line can correspond to a *upattern* of 0x3333 or 0x0F0F. If the *linestyle* parameter to **setlinestyle** is not USERBIT\_LINE (!=4), the *upattern* parameter must still be supplied but is ignored.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **setlinestyle**

**Example**

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

/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE", "DASHED_LINE",
 "USERBIT_LINE" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 struct linesettingstype lineinfo;
 int midx, midy;
 char lstyle[80], lpattern[80], lwidth[80];
```

## getlinesettings

```
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get information about current line settings */
getlinesettings(&lineinfo);

/* convert line information into strings */
sprintf(lstyle, "%s is the line style.", lname[lineinfo.linestyle]);
sprintf(lpattern, "0x%X is the user-defined line pattern.",
 lineinfo.upattern);
sprintf(lwidth, "%d is the line thickness.", lineinfo.thickness);

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, lstyle);
outtextxy(midx, midy+2*textheight("W"), lpattern);
outtextxy(midx, midy+4*textheight("W"), lwidth);

/* clean up */
getch();
closegraph();
return 0;
}
```

## getmaxcolor

---

- Function** Returns maximum color value that can be passed to the **setcolor** function.
- Syntax** `#include <graphics.h>`  
`int far getmaxcolor(void);`
- Prototype in** `graphics.h`
- Remarks** **getmaxcolor** returns the highest valid color value for the current graphics driver and mode that can be passed to **setcolor**.

For example, on a 256K EGA, **getmaxcolor** always returns 15, which means that any call to **setcolor** with a value from 0 to 15 is valid. On a CGA in high-resolution mode or on a Hercules monochrome adapter, **getmaxcolor** returns a value of 1.

**Return value** **getmaxcolor** returns the highest available color value.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **getbkcolor**, **getcolor**, **getpalette**, **getpalettesize**, **setcolor**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;
 char colstr[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* grab the color info. and convert it to a string */
 sprintf(colstr, "This mode supports colors 0..%d", getmaxcolor());

 /* display the information */
 settextjustify(CENTER_TEXT, CENTER_TEXT);
 outtextxy(midx, midy, colstr);

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

}

## getmaxmode

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Returns the maximum mode number for the current driver.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Syntax</b>       | <code>#include &lt;graphics.h&gt;</code><br><code>int far getmaxmode(void);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Prototype in</b> | <code>graphics.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Remarks</b>      | <b>getmaxmode</b> lets you find out the maximum mode number for the currently loaded driver, directly from the driver. This gives it an advantage over <b>getmoderange</b> , which works for Borland drivers only. The minimum mode is 0.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Return value</b> | <b>getmaxmode</b> returns the maximum mode number for the current driver.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Portability</b>  | This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>See also</b>     | <b>getmodename</b> , <b>getmoderange</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Example</b>      | <pre>#include &lt;graphics.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt;  int main(void) {     /* request autodetection */     int gdriver = DETECT, gmode, errorcode;     int midx, midy;     char modestr[80];      /* initialize graphics and local variables */     initgraph(&amp;gdriver, &amp;gmode, "");      /* read result of initialization */     errorcode = graphresult();     if (errorcode != grOk) /* an error occurred */     {         printf("Graphics error: %s\n", grapherrormsg(errorcode));         printf("Press any key to halt:");         getch();         exit(1); /* terminate with an error code */     } }</pre> |

```

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* grab the mode info. and convert it to a string */
sprintf(modestr, "This driver supports modes 0..%d", getmaxmode());

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, modestr);

/* clean up */
getch();
closegraph();
return 0;
}

```

## getmaxx

---

- Function** Returns maximum  $x$  screen coordinate.
- Syntax** `#include <graphics.h>`  
`int far getmaxx(void);`
- Prototype in** `graphics.h`
- Remarks** **getmaxx** returns the maximum (screen-relative)  $x$  value for the current graphics driver and mode.
- For example, on a CGA in 320×200 mode, **getmaxx** returns 319. **getmaxx** is invaluable for centering, determining the boundaries of a region onscreen, and so on.
- Return value** **getmaxx** returns the maximum  $x$  screen coordinate.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getmaxy**, **getx**
- Example**
- ```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;

```


getmaxx

```
char xrange[80], yrange[80];

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* convert max resolution values to strings */
sprintf(xrange, "X values range from 0..%d", getmaxx());
sprintf(yrange, "Y values range from 0..%d", getmaxy());

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, xrange);
outtextxy(midx, midy + textheight("W"), yrange);

/* clean up */
getch();
closegraph();
return 0;
}
```

getmaxy

- Function** Returns maximum *y* screen coordinate.
- Syntax** `#include <graphics.h>`
`int far getmaxy(void);`
- Prototype in** `graphics.h`
- Remarks** **getmaxy** returns the maximum (screen-relative) *y* value for the current graphics driver and mode.
- For example, on a CGA in 320×200 mode, **getmaxy** returns 199. **getmaxy** is invaluable for centering, determining the boundaries of a region onscreen, and so on.

Return value `getmaxy` returns the maximum *y* screen coordinate.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also `getmaxx`, `getx`, `gety`

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char xrange[80], yrange[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* convert max resolution values into strings */
    sprintf(xrange, "X values range from 0..%d", getmaxx());
    sprintf(yrange, "Y values range from 0..%d", getmaxy());

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, xrange);
    outtextxy(midx, midy+textheight("W"), yrange);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getmodename

- Function** Returns a pointer to a string containing the name of a specified graphics mode.
- Syntax** `#include <graphics.h>`
`char *far getmodename(int mode_number);`
- Prototype in** graphics.h
- Remarks** **getmodename** accepts a graphics mode number as input and returns a string containing the name of the corresponding graphics mode. The mode names are embedded in each driver. The return values ("320x200 CGA P1," "640x200 CGA", and so on) are useful for building menus or displaying status.
- Return value** **getmodename** returns a pointer to a string with the name of the graphics mode.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getmaxmode, getmoderange**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, mode;
    char numname[80], modename[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }
}
```

```

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get mode number and name strings */
mode = getgraphmode();
sprintf(numname, "%d is the current mode number.", mode);
sprintf(modename, "%s is the current graphics mode.", getmodename(mode));

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, numname);
outtextxy(midx, midy+2*textheight("W"), modename);

/* clean up */
getch();
closegraph();
return 0;
}

```

getmoderange

Function Gets the range of modes for a given graphics driver.

Syntax `#include <graphics.h>`
`void far getmoderange(int graphdriver, int far lomode, int far himode);`

Prototype in `graphics.h`

Remarks **getmoderange** gets the range of valid graphics modes for the given graphics driver, *graphdriver*. The lowest permissible mode value is returned in *lomode*, and the highest permissible value is *himode*. If *graphdriver* specifies an invalid graphics driver, both *lomode* and *himode* are set to -1. If the value of *graphdriver* is -1, the currently loaded driver modes are given.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **getgraphmode**, **getmaxmode**, **getmodename**, **initgraph**, **setgraphmode**

Example

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)

```

getmoderange

```
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int low, high;
    char mrange[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* get the mode range for this driver */
    getmoderange(gdriver, &low, &high);

    /* convert mode range info. into strings */
    sprintf(mrange, "This driver supports modes %d..%d", low, high);

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, mrange);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getpalette

Function	Gets information about the current palette.
Syntax	<pre>#include <graphics.h> void far getpalette(struct palettetype far *palette);</pre>
Prototype in	graphics.h
Remarks	getpalette fills the palettetype structure pointed to by <i>palette</i> with information about the current palette's size and colors.

The `MAXCOLORS` constant and the `palettetype` structure used by `getpalette` are defined in `graphics.h` as follows:

```
#define MAXCOLORS 15

struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS + 1];
};
```

`size` gives the number of colors in the palette for the current graphics driver in the current mode.

`colors` is an array of `size` bytes containing the actual raw color numbers for each entry in the palette.



`getpalette` cannot be used with the IBM-8514 driver.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also `getbkcolor`, `getcolor`, `getdefaultpalette`, `getmaxcolor`, `setallpalette`, `setpalette`

Example

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

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct palettetype pal;
    char psize[80], pval[20];
    int i, ht;
    int y = 10;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
    }
}
```

getpalette

```
        exit(1);                /* terminate with an error code */
    }

    /* grab a copy of the palette */
    getpalette(&pal);

    /* convert palette info into strings */
    sprintf(psize, "The palette has %d modifiable entries.", pal.size);

    /* display the information */
    outtextxy(0, y, psize);
    if (pal.size != 0) {
        ht = textheight("W");
        y += 2*ht;
        outtextxy(0, y, "Here are the current values:");
        y += 2*ht;
        for (i=0; i<pal.size; i++, y+=ht) {
            sprintf(pval, "palette[%02d]: 0x%02X", i, pal.colors[i]);
            outtextxy(0, y, pval);
        }
    }

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getpalettesize

- Function** Returns size of palette color lookup table.
- Syntax** `#include <graphics.h>`
`int far getpalettesize(void);`
- Prototype in** `graphics.h`
- Remarks** **getpalettesize** is used to determine how many palette entries can be set for the current graphics mode. For example, the EGA in color mode returns 16.
- Return value** **getpalettesize** returns the number of palette entries in the current palette.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **setpalette**, **setallpalette**
- Example** `#include <graphics.h>`

```

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

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char psize[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* convert palette size info into string */
    sprintf(psize, "The palette has %d modifiable entries.", getpalettesize());

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, psize);

    /* clean up */
    getch();
    closegraph();
    return 0;
}

```

getpass

Function Reads a password.

Syntax #include <conio.h>
char *getpass(const char *prompt);

Prototype in conio.h

getpass

Remarks	getpass reads a password from the system console, after prompting with the null-terminated string <i>prompt</i> and disabling the echo. A pointer is returned to a null-terminated string of up to eight characters (not counting the null-terminator).
Return value	The return value is a pointer to a static string, which is overwritten with each call.
Portability	getpass is available on UNIX systems. This function is not Windows compatible.
See also	getch
Example	<pre>#include <conio.h> int main() { char *password; password = getpass("Input a password:"); printf("The password is: %s\r\n", password); return 0; }</pre>

getpid

Function	Gets the process ID of a program.
Syntax	<pre>#include <process.h> unsigned getpid(void)</pre>
Prototype in	process.h
Remarks	A process ID uniquely identifies a program. The concept is borrowed from multitasking operating systems like UNIX, where each process is associated with a unique process number.
Return value	getpid returns the segment value of a program's PSP.
Portability	getpid is available on UNIX systems.
See also	getpsp , <i>_psp</i> (global variable)
Example	<pre>#include <stdio.h> #include <process.h> int main() { printf("This program's process identification number (PID) " "number is %X\n", getpid()); }</pre>

```

    printf("Note: under DOS it is the PSP segment\n");
    return 0;
}

```

getpixel

G

- Function** Gets the color of a specified pixel.
- Syntax** `#include <graphics.h>`
`unsigned far getpixel(int x, int y);`
- Prototype in** `graphics.h`
- Remarks** `getpixel` gets the color of the pixel located at (x,y) .
- Return value** `getpixel` returns the color of the given pixel.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** `getimage`, `putpixel`

Example

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME 100 /* in milliseconds */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int i, x, y, color, maxx, maxy, maxcolor, seed;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }
}

```

getpixel

```
maxx = getmaxx() + 1;
maxy = getmaxy() + 1;
maxcolor = getmaxcolor() + 1;
while (!kbhit()) {
    seed = random(32767); /* seed the random number generator */
    srand(seed);
    for (i=0; i<PIXEL_COUNT; i++) {
        x = random(maxx);
        y = random(maxy);
        color = random(maxcolor);
        putpixel(x, y, color);
    }
    delay(DELAY_TIME);
    srand(seed);
    for (i=0; i<PIXEL_COUNT; i++) {
        x = random(maxx);
        y = random(maxy);
        color = random(maxcolor);
        if (color == getpixel(x, y))
            putpixel(x, y, 0);
    }
}

/* clean up */
getch();
closegraph();
return 0;
}
```

getpsp

Function	Gets the program segment prefix.
Syntax	<pre>#include <dos.h> unsigned getpsp(void);</pre>
Prototype in	dos.h
Remarks	<p>getpsp gets the segment address of the program segment prefix (PSP) using DOS call 0x62.</p> <p>This call exists only in DOS 3.x. For versions of DOS 2.x and 3.x, the global variable <code>_psp</code> set by the start-up code can be used instead.</p>
Return value	getpsp returns the segment address of the PSP.
Portability	getpsp is unique to DOS 3.x and is not available under earlier versions of DOS.

See also `getenv`, `_psp` (global variable)

Example

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

int main(void)
{
    static char command[128];
    char far *cp;
    int len, i;
    printf("The program segment prefix is: %x\n", getpsp());

    /* _psp is preset to the segment of the Program Segment Prefix (PSP).
       The remainder of the command line is located at offset 0x80
       from the start of PSP. Try passing this program arguments. */
    cp = MK_FP(_psp, 0x80);
    len = *cp;
    for (i = 0; i < len; i++)
        command[i] = cp[i+1];
    printf("Command line: %s\n", command);
    return 0;
}
```

G

gets

Function Gets a string from `stdin`.

Syntax `#include <stdio.h>`
`char *gets(char *s);`

Prototype in `stdio.h`

Remarks `gets` collects a string of characters terminated by a new line from the standard input stream `stdin` and puts it into `s`. The new line is replaced by a null character (`\0`) in `s`.

`gets` allows input strings to contain certain whitespace characters (spaces, tabs). `gets` returns when it encounters a new line; everything up to the new line is copied into `s`.

Return value On success, `gets` returns the string argument `s`; it returns null on end-of-file or error.

Portability `gets` is available on UNIX systems and is defined in ANSI C. This function is not Windows compatible.

See also `cgets`, `ferror`, `fgets`, `fopen`, `fputs`, `fread`, `getc`, `puts`, `scanf`

gets

Example

```
#include <stdio.h>

int main(void)
{
    char string[80];
    printf("Input a string:");
    gets(string);
    printf("The string input was: %s\n", string);
    return 0;
}
```

gettext

- Function** Copies text from text mode screen to memory.
- Syntax** `#include <conio.h>`
`int gettext(int left, int top, int right, int bottom, void *destin);`
- Prototype in** `conio.h`
- Remarks** **gettext** stores the contents of an onscreen text rectangle defined by *left*, *top*, *right*, and *bottom* into the area of memory pointed to by *destin*.
- All coordinates are absolute screen coordinates, not window-relative. The upper left corner is (1,1).
- gettext** reads the contents of the rectangle into memory sequentially from left to right and top to bottom.
- Each position onscreen takes 2 bytes of memory: The first byte is the character in the cell, and the second is the cell's video attribute. The space required for a rectangle *w* columns wide by *h* rows high is defined as
- $$\text{bytes} = (h \text{ rows}) \times (w \text{ columns}) \times 2$$
- Return value** **gettext** returns 1 if the operation succeeds. It returns 0 if it fails (for example, if you gave coordinates outside the range of the current screen mode).
- Portability** **gettext** works only on IBM PCs and BIOS-compatible systems. This function is not Windows compatible.
- See also** **movetext**, **puttext**
- Example**
- ```
#include <conio.h>

char buffer[4096];

int main(void)
```

```

{
 int i;
 clrscr();
 for (i = 0; i <= 20; i++)
 printf("Line #%d\r\n", i);
 gettext(1, 1, 80, 25, buffer);
 gotoxy(1, 25);
 printf("Press any key to clear screen...");
 getch();
 clrscr();
 gotoxy(1, 25);
 printf("Press any key to restore screen...");
 getch();
 puttext(1, 1, 80, 25, buffer);
 gotoxy(1, 25);
 printf("Press any key to quit...");
 getch();
 return 0;
}

```

## getttextinfo

---

**Function** Gets text mode video information.

**Syntax** `#include <conio.h>`  
`void gettextinfo(struct text_info *r);`

**Prototype in** conio.h

**Remarks** `gettextinfo` fills in the `text_info` structure pointed to by `r` with the current text video information.

The `text_info` structure is defined in `conio.h` as follows:

```

struct text_info {
 unsigned char winleft; /* left window coordinate */
 unsigned char wintop; /* top window coordinate */
 unsigned char winright; /* right window coordinate */
 unsigned char winbottom; /* bottom window coordinate */
 unsigned char attribute; /* text attribute */
 unsigned char normattr; /* normal attribute */
 unsigned char currmode; /* BW40, BW80, C40, C80, or C4350 */
 unsigned char screenheight; /* text screen's height */
 unsigned char screenwidth; /* text screen's width */
 unsigned char curx; /* x-coordinate in current window */
 unsigned char cury; /* y-coordinate in current window */
};

```

## gettextinfo

- Return value** **gettextinfo** returns nothing; the results are returned in the structure pointed to by *r*.
- Portability** **gettextinfo** works only with IBM PCs and compatibles. This function is not Windows compatible.
- See also** **textattr**, **textbackground**, **textcolor**, **textmode**, **wherex**, **wherey**, **window**
- Example**

```
#include <conio.h>

int main(void)
{
 struct text_info ti;
 gettextinfo(&ti);
 printf("window left %2d\r\n",ti.winleft);
 printf("window top %2d\r\n",ti.wintop);
 printf("window right %2d\r\n",ti.winright);
 printf("window bottom %2d\r\n",ti.winbottom);
 printf("attribute %2d\r\n",ti.attribute);
 printf("normal attribute %2d\r\n",ti.normattr);
 printf("current mode %2d\r\n",ti.currmode);
 printf("screen height %2d\r\n",ti.screenheight);
 printf("screen width %2d\r\n",ti.screenwidth);
 printf("current x %2d\r\n",ti.curx);
 printf("current y %2d\r\n",ti.cury);
 return 0;
}
```

## gettextsettings

---

- Function** Gets information about the current graphics text font.
- Syntax** `#include <graphics.h>`  
`void far gettextsettings(struct textsettingstype far *texttypeinfo);`
- Prototype in** graphics.h
- Remarks** **gettextsettings** fills the **textsettingstype** structure pointed to by *textinfo* with information about the current text font, direction, size, and justification.
- The **textsettingstype** structure used by **gettextsettings** is defined in graphics.h as follows:

```
struct textsettingstype {
 int font;
 int direction;
```

```

 int charsize;
 int horiz;
 int vert;
};

```

See **settextstyle** for a description of these fields.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **outtext**, **outtextxy**, **registerbgifont**, **settextjustify**, **settextstyle**, **setusercharsize**, **textheight**, **textwidth**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the supported fonts */
char *font[] = { "DEFAULT_FONT", "TRIPLEX_FONT", "SMALL_FONT", "SANS_SERIF_FONT",
 "GOTHIC_FONT" };

/* the names of the text directions supported */
char *dir[] = { "HORIZ_DIR", "VERT_DIR" };

/* horizontal text justifications supported */
char *hjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };

/* vertical text justifications supported */
char *vjust[] = { "BOTTOM_TEXT", "CENTER_TEXT", "TOP_TEXT" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 struct textsettingstype textinfo;
 int midx, midy, ht;
 char fontstr[80], dirstr[80], sizestr[80];
 char hjuststr[80], vjuststr[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 }
}

```



## gettextsettings

```
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get information about current text settings */
gettextsettings(&textinfo);

/* convert text information into strings */
sprintf(fontstr, "%s is the text style.", font[textinfo.font]);
sprintf(dirstr, "%s is the text direction.", dir[textinfo.direction]);
sprintf(sizestr, "%d is the text size.", textinfo.charsize);
sprintf(hjuststr, "%s is the horizontal justification.",
 hjust[textinfo.horiz]);
sprintf(vjuststr, "%s is the vertical justification.", vjust[textinfo.vert]);

/* display the information */
ht = textheight("W");
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, fontstr);
outtextxy(midx, midy+2*ht, dirstr);
outtextxy(midx, midy+4*ht, sizestr);
outtextxy(midx, midy+6*ht, hjuststr);
outtextxy(midx, midy+8*ht, vjuststr);

/* clean up */
getch();
closegraph();
return 0;
}
```

## gettime

---

- Function** Gets system time.
- Syntax** `#include <dos.h>`  
`void gettime(struct time *timep);`
- Prototype in** dos.h
- Remarks** **gettime** fills in the **time** structure pointed to by *timep* with the system's current time.
- The **time** structure is defined as follows:

```
struct time {
 unsigned char ti_min; /* minutes */
 unsigned char ti_hour; /* hours */
};
```

```

 unsigned char ti_hund; /* hundredths of seconds */
 unsigned char ti_sec; /* seconds */
 };

```

**Return value** None.

**Portability** `gettime` is unique to DOS.

**See also** `getdate`, `setdate`, `settime`, `stime`, `time`

**Example**

```

#include <stdio.h>
#include <dos.h>

int main(void)
{
 struct time t;
 gettime(&t);
 printf("The current time is: %2d:%02d:%02d.%02d\n", t.ti_hour, t.ti_min,
 t.ti_sec, t.ti_hund);
 return 0;
}

```

G

## getvect

---

**Function** Gets interrupt vector.

**Syntax** `#include <dos.h>`  
`void interrupt(*getvect(int interruptno)) ();`

**Prototype in** `dos.h`

**Remarks** Every processor of the 8086 family includes a set of interrupt vectors, numbered 0 to 255. The 4-byte value in each vector is actually an address, which is the location of an interrupt function.

`getvect` reads the value of the interrupt vector given by *interruptno* and returns that value as a (**far**) pointer to an interrupt function. The value of *interruptno* can be from 0 to 255.

**Return value** `getvect` returns the current 4-byte value stored in the interrupt vector named by *interruptno*.

**Portability** `getvect` is unique to DOS.

**See also** `disable`, `enable`, `geninterrupt`, `setvect`

**Example**

```

#include <stdio.h>
#include <dos.h>

void interrupt get_out(); /* interrupt prototype */

```

## getvect

```
void interrupt (*oldfunc)(); /* interrupt function pointer */
int looping = 1;
int main(void)
{
 puts("Press <Shift><PrtSc> to terminate");
 /* save the old interrupt */
 oldfunc = getvect(5);
 /* install interrupt handler */
 setvect(5,get_out);
 /* do nothing */
 while (looping);
 /* restore to original interrupt routine */
 setvect(5,oldfunc);
 puts("Success");
 return 0;
}

void interrupt get_out() {
 looping = 0; /* change global var to get out of loop */
}
```

## getverify

---

- Function** Returns the state of the DOS verify flag.
- Syntax** #include <dos.h>  
int getverify(void);
- Prototype in** dos.h
- Remarks** **getverify** gets the current state of the verify flag.  
The verify flag controls output to the disk. When verify is off, writes are not verified; when verify is on, all disk writes are verified to ensure proper writing of the data.
- Return value** **getverify** returns the current state of the verify flag, either 0 or 1.
- A return of 0 = verify flag off.
  - A return of 1 = verify flag on.
- Portability** **getverify** is unique to DOS.
- See also** **setverify**

```

Example #include <stdio.h>
 #include <dos.h>

 int main(void)
 {
 if (getverify())
 printf("DOS verify flag is on\n");
 else
 printf("DOS verify flag is off\n");
 return 0;
 }

```

## getviewsettings

---

**Function** Gets information about the current viewport.

**Syntax** #include <graphics.h>  
 void far getviewsettings(struct viewporttype far \*viewport);

**Prototype in** graphics.h

**Remarks** **getviewsettings** fills the **viewporttype** structure pointed to by *viewport* with information about the current viewport.

The **viewporttype** structure used by **getviewport** is defined in graphics.h as follows:

```

 struct viewporttype {
 int left, top, right, bottom;
 int clip;
 };

```

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **clearviewport, getx, gety, setviewport**

```

Example #include <graphics.h>
 #include <stdlib.h>
 #include <stdio.h>
 #include <conio.h>

 char *clip[] = { "OFF", "ON" };

 int main(void)
 {

```

```
/* request autodetection */
int gdriver = DETECT, gmode, errorcode;
struct viewporttype viewinfo;
int midx, midy, ht;
char topstr[80], botstr[80], clipstr[80];

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get information about current viewport */
getviewsettings(&viewinfo);

/* convert text information into strings */
sprintf(topstr, "(%d, %d) is the upper left viewport corner.", viewinfo.left,
 viewinfo.top);
sprintf(botstr, "(%d, %d) is the lower right viewport corner.",
 viewinfo.right, viewinfo.bottom);
sprintf(clipstr, "Clipping is turned %s.", clip[viewinfo.clip]);

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
ht = textheight("W");
outtextxy(midx, midy, topstr);
outtextxy(midx, midy+2*ht, botstr);
outtextxy(midx, midy+4*ht, clipstr);

/* clean up */
getch();
closegraph();
return 0;
}
```

## getw

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets integer from stream.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int getw(FILE *stream);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Remarks</b>      | <b>getw</b> returns the next integer in the named input stream. It assumes no special alignment in the file.<br><br><b>getw</b> should not be used when the stream is opened in text mode.                                                                                                                                                                                                                                                                                                                                         |
| <b>Return value</b> | <b>getw</b> returns the next integer on the input stream. On end-of-file or error, <b>getw</b> returns EOF. Because EOF is a legitimate value for <b>getw</b> to return, <b>feof</b> or <b>ferror</b> should be used to detect end-of-file or error.                                                                                                                                                                                                                                                                               |
| <b>Portability</b>  | <b>getw</b> is available on UNIX systems.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>See also</b>     | <b>putw</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  #define FNAME "test.\$\$\$"  int main(void) {     FILE *fp;     int word;      /* place the word in a file */     fp = fopen(FNAME, "wb");     if (fp == NULL) {         printf("Error opening file %s\n", FNAME);         exit(1);     }      word = 94;     putw(word, fp);     if (ferror(fp))         printf("Error writing to file\n");     else         printf("Successful write\n");     fclose(fp);      /* reopen the file */     fp = fopen(FNAME, "rb");</pre> |

## getw

```
if (fp == NULL) {
 printf("Error opening file %s\n", FNAME);
 exit(1);
}

/* extract the word */
word = getw(fp);
if (ferror(fp))
 printf("Error reading file\n");
else
 printf("Successful read: word = %d\n", word);

/* clean up */
fclose(fp);
unlink(FNAME);
return 0;
}
```

## getx

---

- Function** Returns the current graphics position's x-coordinate.
- Syntax** `#include <graphics.h>`  
`int far getx(void);`
- Prototype in** `graphics.h`
- Remarks** **getx** finds the current graphics position's x-coordinate. The value is viewport-relative.
- Return value** **getx** returns the x-coordinate of the current position.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getmaxx**, **getmaxy**, **getviewsettings**, **gety**, **moveto**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];
```

```

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

/* move to the screen center point */
moveto(getmaxx() / 2, getmaxy() / 2);

/* create a message string */
sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());

/* display the message */
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}

```

gety

- Function** Returns the current graphics position's y-coordinate.
- Syntax** `#include <graphics.h>`
`int far gety(void);`
- Prototype in** `graphics.h`
- Remarks** **gety** returns the current graphics position's y-coordinate. The value is viewport-relative.
- Return value** **gety** returns the y-coordinate of the current position.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getmaxx**, **getmaxy**, **getviewsettings**, **getx**, **moveto**
- Example** `#include <graphics.h>`
`#include <stdlib.h>`

gety

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* move to the screen center point */
    moveto(getmaxx() / 2, getmaxy() / 2);

    /* create a message string */
    sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());

    /* display the message */
    outtext(msg);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

gmtime

Function	Converts date and time to Greenwich mean time (GMT).
Syntax	<pre>#include <time.h> struct tm *gmtime(const time_t *timer);</pre>
Prototype in	time.h
Remarks	gmtime accepts the address of a value returned by time and returns a pointer to the structure of type tm containing the broken-down time. gmtime converts directly to GMT.

The global long variable *timezone* should be set to the difference in seconds between GMT and local standard time (in PST, *timezone* is $8 \times 60 \times 60$). The global variable *daylight* should be set to nonzero *only* if the standard U.S. daylight saving time conversion should be applied.

The **tm** structure declaration from the `time.h` include file is

```
struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1 to 31), month (0 to 11), weekday (Sunday equals 0), year – 1900, day of year (0 to 365), and a flag that is nonzero if daylight saving time is in effect.

Return value **gmtime** returns a pointer to the structure containing the broken-down time. This structure is a static that is overwritten with each call.

Portability **gmtime** is available on UNIX systems and is defined in ANSI C.

See also **asctime**, **ctime**, **ftime**, **localtime**, **stime**, **time**, **tzset**

Example

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

/* pacific standard & daylight savings time */
char *tzstr = "TZ=PST8PDT";

int main(void)
{
    time_t t;
    struct tm *gmt, *area;
    putenv(tzstr);
    tzset();
    t = time(NULL);
    area = localtime(&t);
    printf("Local time is: %s", asctime(area));
    gmt = gmtime(&t);
    printf("GMT is:          %s", asctime(gmt));
}
```

gotoxy

```
        return 0;  
    }
```

gotoxy

Function	Positions cursor in text window.
Syntax	<pre>#include <conio.h> void gotoxy(int x, int y);</pre>
Prototype in	conio.h
Remarks	gotoxy moves the cursor to the given position in the current text window. If the coordinates are in any way invalid, the call to gotoxy is ignored. An example of this is a call to gotoxy (40,30), when (35,25) is the bottom right position in the window.
Return value	None.
Portability	gotoxy works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.
See also	wherex, wherey, window
Example	<pre>#include <conio.h> int main(void) { clrscr(); gotoxy(35, 12); cprintf("Hello world"); getch(); return 0; }</pre>

graphdefaults

Function	Resets all graphics settings to their defaults.
Syntax	<pre>#include <graphics.h> void far graphdefaults(void);</pre>
Prototype in	graphics.h
Remarks	graphdefaults resets all graphics settings to their defaults: <ul style="list-style-type: none">■ sets the viewport to the entire screen.

- moves the current position to (0,0).
- sets the default palette colors, background color, and drawing color.
- sets the default fill style and pattern.
- sets the default text font and justification.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also `initgraph`, `setgraphmode`

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    maxx = getmaxx();
    maxy = getmaxy();

    /* output line with nondefault settings */
    setlinestyle(DOTTED_LINE, 0, 3);
    line(0, 0, maxx, maxy);
    outtextxy(maxx/2, maxy/3, "Before default values are restored.");
    getch();

    /* restore default values for everything */
    graphdefaults();

    /* clear the screen */
    cleardevice();
}
```

graphdefaults

```
/* output line with default settings */
line(0, 0, maxx, maxy);
outtextxy(maxx/2, maxy/3, "After restoring default values.");

/* clean up */
getch();
closegraph();
return 0;
}
```

grapherrormsg

- Function** Returns a pointer to an error message string.
- Syntax** `#include <graphics.h>`
`char * far grapherrormsg(int errorcode);`
- Prototype in** `graphics.h`
- Remarks** **grapherrormsg** returns a pointer to the error message string associated with *errorcode*, the value returned by **graphresult**.
Refer to the entry for *errno* in Chapter 3 ("Global variables") for a list of error messages and mnemonics.
- Return value** **grapherrormsg** returns a pointer to an error message string.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **graphresult**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define NONSENSE -50

int main(void)
{
 /* force an error to occur */
 int gdriver = NONSENSE, gmode, errorcode;

 /* initialize graphics mode */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
```

```

/* if an error occurred, then output descriptive error message */
if (errorcode != grOk) {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

/* draw a line */
line(0, 0, getmaxx(), getmaxy());

/* clean up */
getch();
closegraph();
return 0;
}

```

## \_graphfreemem

---

**Function** User hook into graphics memory deallocation.

**Syntax** `#include <graphics.h>`  
`void far _graphfreemem(void far *ptr, unsigned size);`

**Prototype in** graphics.h

**Remarks** The graphics library calls **\_graphfreemem** to release memory previously allocated through **\_graphgetmem**. You can choose to control the graphics library memory management by simply defining your own version of **\_graphfreemem** (you must declare it exactly as shown in the declaration). The default version of this routine merely calls **free**.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **\_graphgetmem**, **setgraphbufsize**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{

```

```
/* request autodetection */
int gdriver = DETECT, gmode, errorcode;
int midx, midy;

/* clear the text screen */
clrscr();
printf("Press any key to initialize graphics mode:");
getch();
clrscr();

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* display a message */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, "Press any key to exit graphics mode:");

/* clean up */
getch();
closegraph();
return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size) {
 printf("_graphgetmem called to allocate %d bytes.\n", size);
 printf("hit any key:");
 getch();
 printf("\n");

 /* allocate memory from far heap */
 return farmalloc(size);
}

/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size) {
 printf("_graphfreemem called to free %d bytes.\n", size);
 printf("hit any key:");
 getch();
}
```

```
printf("\n");
/* free ptr from far heap */
farfree(ptr);
}
```

## `_graphgetmem`

---



**Function** User hook into graphics memory allocation.

**Syntax** `#include <graphics.h>`  
`void far * far _graphgetmem(unsigned size);`

**Prototype in** `graphics.h`

**Remarks** Routines in the graphics library (not the user program) normally call `_graphgetmem` to allocate memory for internal buffers, graphics drivers, and character sets. You can choose to control the memory management of the graphics library by defining your own version of `_graphgetmem` (you must declare it exactly as shown in the declaration). The default version of this routine merely calls `malloc`.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** `_graphfreemem`, `initgraph`, `setgraphbufsize`

**Example**

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;

 /* clear the text screen */
 clrscr();
 printf("Press any key to initialize graphics mode:");
 getch();
 clrscr();
```



```
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* display a message */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, "Press any key to exit graphics mode:");

/* clean up */
getch();
closegraph();
return 0;
}

/* called by the graphics kernel to allocate memory */
void far * far _graphgetmem(unsigned size) {
 printf("_graphgetmem called to allocate %d bytes.\n", size);
 printf("hit any key:");
 getch();
 printf("\n");

 /* allocate memory from far heap */
 return farmalloc(size);
}

/* called by the graphics kernel to free memory */
void far _graphfreemem(void far *ptr, unsigned size) {
 printf("_graphfreemem called to free %d bytes.\n", size);
 printf("hit any key:");
 getch();
 printf("\n");

 /* free ptr from far heap */
 farfree(ptr);
}
```

## graphresult

**Function** Returns an error code for the last unsuccessful graphics operation.

**Syntax**

```
#include <graphics.h>
int far graphresult(void);
```

**Prototype in** graphics.h

**Remarks** **graphresult** returns the error code for the last graphics operation that reported an error and resets the error level to grOk.

The following table lists the error codes returned by **graphresult**. The enumerated type *graph\_errors* defines the errors in this table. *graph\_errors* is declared in graphics.h.

| Error code | <i>graphics_errors</i> constant | Corresponding error message string                   |
|------------|---------------------------------|------------------------------------------------------|
| 0          | grOk                            | No error                                             |
| -1         | grNoInitGraph                   | (BGI) graphics not installed (use <b>initgraph</b> ) |
| -2         | grNotDetected                   | Graphics hardware not detected                       |
| -3         | grFileNotFound                  | Device driver file not found                         |
| -4         | grInvalidDriver                 | Invalid device driver file                           |
| -5         | grNoLoadMem                     | Not enough memory to load driver                     |
| -6         | grNoScanMem                     | Out of memory in scan fill                           |
| -7         | grNoFloodMem                    | Out of memory in flood fill                          |
| -8         | grFontNotFound                  | Font file not found                                  |
| -9         | grNoFontMem                     | Not enough memory to load font                       |
| -10        | grInvalidMode                   | Invalid graphics mode for selected driver            |
| -11        | grError                         | Graphics error                                       |
| -12        | grIOerror                       | Graphics I/O error                                   |
| -13        | grInvalidFont                   | Invalid font file                                    |
| -14        | grInvalidFontNum                | Invalid font number                                  |
| -15        | grInvalidDeviceNum              | Invalid device number                                |
| -18        | grInvalidVersion                | Invalid version number                               |

Note that the variable maintained by **graphresult** is reset to 0 after **graphresult** has been called. Therefore, you should store the value of **graphresult** into a temporary variable and then test it.

**Return value** **graphresult** returns the current graphics error number, an integer in the range -15 to 0; **grapherrormsg** returns a pointer to a string associated with the value returned by **graphresult**.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **detectgraph**, **drawpoly**, **fillpoly**, **floodfill**, **grapherrormsg**, **initgraph**, **pieslice**, **registerbgdriver**, **registerbgifont**, **setallpalette**, **setcolor**, **setfillstyle**, **setgraphmode**, **setlinestyle**, **setpalette**, **settextjustify**, **settextstyle**, **setusercharsize**, **setviewport**, **setvisualpage**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();

 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 /* draw a line */
 line(0, 0, getmaxx(), getmaxy());

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

# harderr

---

**Function** Establishes a hardware error handler.

**Syntax** `#include <dos.h>`  
`void harderr(int (*handler)());`

**Prototype in** dos.h

**Remarks** **harderr** establishes a hardware error handler for the current program. This error handler is invoked whenever an interrupt 0x24 occurs. (See your DOS reference manuals for a discussion of the interrupt.)

The function pointed to by *handler* is called when such an interrupt occurs. The handler function is called with the following arguments:

```
handler(int errval, int ax, int bp, int si);
```

*errval* is the error code set in the DI register by DOS. *ax*, *bp*, and *si* are the values DOS sets for the AX, BP, and SI registers, respectively.

- *ax* indicates whether a disk error or other device error was encountered. If *ax* is nonnegative, a disk error was encountered; otherwise, the error was a device error. For a disk error, *ax* ANDed with 0x00FF give the failing drive number (0 equals A, 1 equals B, and so on).
- *bp* and *si* together point to the device driver header of the failing driver. *bp* contains the segment address, and *si* the offset.

The function pointed to by *handler* is not called directly. **harderr** establishes a DOS interrupt handler that calls the function.

The handler can issue DOS calls 1 through 0xC; any other DOS call corrupts DOS. In particular, any of the C standard I/O or UNIX-emulation I/O calls *cannot* be used.

The handler must return 0 for ignore, 1 for retry, and 2 for abort.

**Return value** None.

**Portability** **harderr** is unique to DOS. This function is not Windows compatible.

**See also** **hardresume**, **hardretn**, **peek**, **poke**

**Example**

```
/* This program traps disk errors and prompts the user for action. */
/* Try running it with no disk in drive A to invoke its functions. */

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

```

#include <dos.h>

#define IGNORE 0
#define RETRY 1
#define ABORT 2

int buf[500];

/* define the error messages for trapping disk problems */
static char *err_msg[] = { "write protect", "unknown unit", "drive not ready",
 "unknown command", "data error (CRC)", "bad request",
 "seek error", "unknown media type",
 "sector not found", "printer out of paper",
 "write fault", "read fault", "general failure",
 "reserved", "reserved", "invalid disk change" };

error_win(char *msg)
{
 int retval;
 cputs(msg);

 /* prompt for user to press a key to abort, retry, ignore */
 while(1) {
 retval= getch();
 if (retval == 'a' || retval == 'A') {
 retval = ABORT;
 break;
 }
 if (retval == 'r' || retval == 'R') {
 retval = RETRY;
 break;
 }
 if (retval == 'i' || retval == 'I') {
 retval = IGNORE;
 break;
 }
 }
 return(retval);
}

/* pragma warn -par reduces warnings which occur due to the */
/* nonuse of the parameters errval, bp, and si to the handler */
#pragma warn -par

int handler(int errval,int ax,int bp,int si)
{
 static char msg[80];
 unsigned di;
 int drive;
 int errorno;
 di= _DI;

```

```

/* if this not disk error then another device having trouble */
if (ax < 0) {
 error_win("Device error"); /* report the error */

 /* return to the program directly requesting abort */
 hardretn(ABORT);
}
drive = ax & 0x00FF; /* otherwise it was disk error */
errorno = di & 0x00FF;

/* report which error it was */
sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
 err_msg[errorno], 'A' + drive);

/* return to program via dos interrupt 0x23 with abort, retry */
/* or ignore as input by the user */
hardresume(error_win(msg));
return ABORT;
}

#pragma warn +par
int main(void)
{
 /* install our handler on the hardware problem interrupt */
 harderr(handler);
 clrscr();
 printf("Make sure there is no disk in drive A:\n");
 printf("Press any key\n");
 getch();
 printf("Trying to access drive A:\n");
 printf("fopen returned %p\n", fopen("A:temp.dat", "w"));
 return 0;
}

```

## hardresume

---

- Function** Hardware error handler.
- Syntax** #include <dos.h>  
void hardresume(int *axret*);
- Prototype in** dos.h
- Remarks** The error handler established by **harderr** can call **hardresume** to return to DOS. The return value of the *rescode* (result code) of **hardresume** contains an abort (2), retry (1), or ignore (0) indicator. The abort is accomplished by invoking DOS interrupt 0x23, the control-break interrupt.

The handler must return 0 for ignore, 1 for retry, and 2 for abort.

**Return value** None.

**Portability** **hardresume** is unique to DOS. This function is not Windows compatible.

**See also** **harderr**, **hardretn**

**Example**

```
/* This program traps disk errors and prompts the user for action. */
/* Try running it with no disk in drive A to invoke its functions. */

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

#define IGNORE 0
#define RETRY 1
#define ABORT 2

int buf[500];

/* define the error messages for trapping disk problems */
static char *err_msg[] = { "write protect", "unknown unit", "drive not ready",
 "unknown command", "data error (CRC)", "bad request",
 "seek error", "unknown media type",
 "sector not found", "printer out of paper",
 "write fault", "read fault", "general failure",
 "reserved", "reserved", "invalid disk change" };

error_win(char *msg)
{
 int retval;
 cputs(msg);

 /* prompt for user to press a key to abort, retry, ignore */
 while(1) {
 retval= getch();
 if (retval == 'a' || retval == 'A')
 {
 retval = ABORT;
 break;
 }
 if (retval == 'r' || retval == 'R') {
 retval = RETRY;
 break;
 }
 if (retval == 'i' || retval == 'I') {
 retval = IGNORE;
 break;
 }
 }
 return(retval);
}
```

```

}

/* pragma warn -par reduces warnings which occur due to the */
/* nonuse of the parameters errval, bp, and si to the handler */
#pragma warn -par

int handler(int errval,int ax,int bp,int si)
{
 static char msg[80];
 unsigned di;
 int drive;
 int errorno;
 di= _DI;

 /* if this not disk error then another device having trouble */
 if (ax < 0) {
 error_win("Device error"); /* report the error */

 /* return to the program directly requesting abort */
 hardretn(ABORT);
 }
 drive = ax & 0x00FF; /* otherwise it was disk error */
 errorno = di & 0x00FF;

 /* report which error it was */
 sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
 err_msg[errorno], 'A' + drive);

 /* return to program via dos interrupt 0x23 with abort, retry */
 /* or ignore as input by the user */
 hardresume(error_win(msg));
 return ABORT;
}

#pragma warn +par

int main(void)
{
 /* install our handler on the hardware problem interrupt */
 harderr(handler);
 clrscr();
 printf("Make sure there is no disk in drive A:\n");
 printf("Press any key\n");
 getch();
 printf("Trying to access drive A:\n");
 printf("fopen returned %p\n",fopen("A:temp.dat", "w"));
 return 0;
}

```



# hardretn

---

- Function** Hardware error handler.
- Syntax** `#include <dos.h>`  
`void hardretn(int retn);`
- Prototype in** dos.h
- Remarks** The error handler established by **harderr** can return directly to the application program by calling **hardretn**. The returned value is whatever value you passed to **hardretn**.
- Return value** None.
- Portability** **hardretn** is unique to DOS. This function is not Windows compatible.
- See also** **harderr**, **hardresume**

**Example**

```

/* This program traps disk errors and prompts the user for action. */
/* Try running it with no disk in drive A to invoke its functions. */

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

#define IGNORE 0
#define RETRY 1
#define ABORT 2

int buf[500];

/* define the error messages for trapping disk problems */
static char *err_msg[] = { "write protect", "unknown unit", "drive not ready",
 "unknown command", "data error (CRC)", "bad request",
 "seek error", "unknown media type", "sector not
 found", "printer out of paper", "write fault", "read
 fault", "general failure", "reserved", "reserved",
 "invalid disk change" };

error_win(char *msg)
{
 int retval;
 cputs(msg);

 /* prompt for user to press a key to abort, retry, ignore */
 while(1) {
 retval= getch();
 if (retval == 'a' || retval == 'A') {
 retval = ABORT;
 break;
 }
 }
}

```

```

 if (retval == 'r' || retval == 'R') {
 retval = RETRY;
 break;
 }
 if (retval == 'i' || retval == 'I') {
 retval = IGNORE;
 break;
 }
 }
 return(retval);
}

/* pragma warn -par reduces warnings which occur due to the */
/* nonuse of the parameters errval, bp, and si to the handler */
#pragma warn -par

int handler(int errval,int ax,int bp,int si)
{
 static char msg[80];
 unsigned di;
 int drive;
 int errorno;
 di= _DI;

 /* if this not disk error then another device having trouble */
 if (ax < 0) {
 error_win("Device error"); /* report the error */

 /* return to the program directly requesting abort */
 hardretn(ABORT);
 }
 drive = ax & 0x00FF; /* otherwise it was disk error */
 errorno = di & 0x00FF;

 /* report which error it was */
 sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
 err_msg[errorno], 'A' + drive);

 /* return to program via DOS interrupt 0x23 with abort, retry, */
 /* or ignore as input by the user */
 hardresume(error_win(msg));
 return ABORT;
}

#pragma warn +par

int main(void)
{
 /* install our handler on the hardware problem interrupt */
 harderr(handler);
 clrscr();
 printf("Make sure there is no disk in drive A:\n");
}

```

```

printf("Press any key\n");
getch();
printf("Trying to access drive A:\n");
printf("fopen returned %p\n", fopen("A:temp.dat", "w"));
return 0;
}

```

## heapcheck

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Checks and verifies the heap.                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Syntax</b>       | <code>#include &lt;alloc.h&gt;</code><br><code>int heapcheck(void);</code>                                                                                                                                                                                                                                                                                                                                                          |
| <b>Prototype in</b> | alloc.h                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Remarks</b>      | <b>heapcheck</b> walks through the heap and examines each block checking its pointers, size, and other critical attributes. In the large data models, <b>heapcheck</b> maps to <b>farheapcheck</b> .                                                                                                                                                                                                                                |
| <b>Return value</b> | The return value is less than zero for an error and greater than zero for success.<br><br>_ <b>HEAPEMPTY</b> is returned if there is no heap (value 1).<br>_ <b>HEAPOK</b> is returned if the heap is verified (value 2).<br>_ <b>HEAPCORRUPT</b> is returned if the heap has been corrupted (value -1).                                                                                                                            |
| <b>Portability</b>  | <b>heapcheck</b> is unique to DOS. This function is not Windows compatible.                                                                                                                                                                                                                                                                                                                                                         |
| <b>See also</b>     | <b>farheapcheck</b>                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Example</b>      | <pre> #include &lt;stdio.h&gt; #include &lt;alloc.h&gt;  #define NUM_PTRS 10 #define NUM_BYTES 16  int main(void) {     char *array[NUM_PTRS];     int i;     for(i = 0; i &lt; NUM_PTRS; i++)         array[i] = malloc(NUM_BYTES);     for(i = 0; i &lt; NUM_PTRS; i += 2)         free(array[i]);     if(heapcheck() == _HEAPCORRUPT)         printf("Heap is corrupted.\n");     else         printf("Heap is OK.\n"); } </pre> |

```

 return 0;
}

```

## heapcheckfree

---

**Function** Checks the free blocks on the heap for a constant value.

**Syntax** `#include <alloc.h>`  
`int heapcheckfree(unsigned int fillvalue);`

**Prototype in** `alloc.h`

**Return value** The return value is less than zero for an error and greater than zero for success.

`_HEAPEMPTY` is returned if there is no heap (value 1).

`_HEAPOK` is returned if the heap is accurate (value 2).

`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).

`_BADVALUE` is returned if a value other than the fill value was found (value -3).

**Portability** `heapcheckfree` is unique to DOS. This function is not Windows compatible.

**See also** `farheapcheckfree`

**Example**

```

#include <mem.h>
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
 char *array[NUM_PTRS];
 int i;
 int res;
 for(i = 0; i < NUM_PTRS; i++)
 array[i] = malloc(NUM_BYTES);
 for(i = 0; i < NUM_PTRS; i += 2)
 free(array[i]);
 if(heapfillfree(1) < 0) {
 printf("Heap corrupted.\n");
 return 1;
 }
 for(i = 1; i < NUM_PTRS; i += 2)
 memset(array[i], 0, NUM_BYTES);
}

```

## heapcheckfree

```
res = heapcheckfree(1);
if(res < 0)
 switch(res) {
 case _HEAPCORRUPT:
 printf("Heap corrupted.\n");
 return 1;
 case _BADVALUE:
 printf("Bad value in free space.\n");
 return 1;
 default:
 printf("Unknown error.\n");
 return 1;
 }
printf("Test successful.\n");
return 0;
}
```

## heapchecknode

---

- Function** Checks and verifies a single node on the heap.
- Syntax** `#include <alloc.h>`  
`int heapchecknode(void *node);`
- Prototype in** `alloc.h`
- Remarks** If a node has been freed and **heapchecknode** is called with a pointer to the freed block, **heapchecknode** can return `_BADNODE` rather than the expected `_FREEENTRY`. This is because adjacent free blocks on the heap are merged, and the block in question no longer exists.
- Return value** The return value is less than zero for an error and greater than zero for success.
- `_HEAPEMPTY` is returned if there is no heap (value 1).  
`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).  
`_BADNODE` is returned if the node could not be found (value -2).  
`_FREEENTRY` is returned if the node is a free block (value 3).  
`_USEDENTRY` is returned if the node is a used block (value 4).
- Portability** **heapchecknode** is unique to DOS. This function is not Windows compatible.
- See also** **farheapchecknode**
- Example** `#include <stdio.h>`  
`#include <alloc.h>`

```

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
 char *array[NUM_PTRS];
 int i;
 for(i = 0; i < NUM_PTRS; i++)
 array[i] = malloc(NUM_BYTES);
 for(i = 0; i < NUM_PTRS; i += 2)
 free(array[i]);
 for(i = 0; i < NUM_PTRS; i++) {
 printf("Node %2d ", i);
 switch(heapchecknode(array[i])) {
 case _HEAPEMPTY:
 printf("No heap.\n");
 break;
 case _HEAPCORRUPT:
 printf("Heap corrupt.\n");
 break;
 case _BADNODE:
 printf("Bad node.\n");
 break;
 case _FREEENTRY:
 printf("Free entry.\n");
 break;
 case _USEDENTRY:
 printf("Used entry.\n");
 break;
 default:
 printf("Unknown return code.\n");
 break;
 }
 }
 return 0;
}

```

## heapfillfree

---

**Function** Fills the free blocks on the heap with a constant value.

**Syntax** `#include <alloc.h>`  
`int heapfillfree(unsigned int fillvalue);`

**Prototype in** `alloc.h`

## heapfillfree

**Return value** The return value is less than zero for an error and greater than zero for success.

`_HEAPEMPTY` is returned if there is no heap (value 1).

`_HEAPOK` is returned if the heap is accurate (value 2).

`_HEAPCORRUPT` is returned if the heap has been corrupted (value -1).

**Portability** `heapfillfree` is unique to DOS. This function is not Windows compatible.

**See also** `farheapfillfree`

**Example**

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

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
 char *array[NUM_PTRS];
 int i;
 int res;
 for(i = 0; i < NUM_PTRS; i++)
 array[i] = malloc(NUM_BYTES);
 for(i = 0; i < NUM_PTRS; i += 2)
 free(array[i]);
 if(heapfillfree(1) < 0) {
 printf("Heap corrupted.\n");
 return 1;
 }
 for(i = 1; i < NUM_PTRS; i += 2)
 memset(array[i], 0, NUM_BYTES);
 res = heapcheckfree(1);
 if(res < 0)
 switch(res) {
 case _HEAPCORRUPT:
 printf("Heap corrupted.\n");
 return 1;
 case _BADVALUE:
 printf("Bad value in free space.\n");
 return 1;
 default:
 printf("Unknown error.\n");
 return 1;
 }
 printf("Test successful.\n");
 return 0;
}
```

# heapwalk

---

**Function** **heapwalk** is used to “walk” through the heap node by node.

**Syntax** `#include <alloc.h>`  
`int heapwalk(struct heapinfo *hi);`

**Prototype in** `alloc.h`

**Remarks** **heapwalk** assumes the heap is correct. Use **heapcheck** to verify the heap before using **heapwalk**. `_HEAPOK` is returned with the last block on the heap. `_HEAPEND` will be returned on the next call to **heapwalk**.

**heapwalk** receives a pointer to a structure of type *heapinfo* (declared in `alloc.h`). For the first call to **heapwalk**, set the `hi.ptr` field to null. **heapwalk** returns with `hi.ptr` containing the address of the first block. `hi.size` holds the size of the block in bytes. `hi.in_use` is a flag that’s set if the block is currently in use.

**Return value** `_HEAPEMPTY` is returned if there is no heap (value 1).  
`_HEAPOK` is returned if the *heapinfo* block contains valid data (value 2).  
`_HEAPEND` is returned if the end of the heap has been reached (value 5).

**Portability** **heapwalk** is unique to DOS. This function is not Windows compatible.

**See also** **farheapwalk**

**Example**

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

#define NUM_PTRS 10
#define NUM_BYTES 16

int main(void)
{
 struct heapinfo hi;
 char *array[NUM_PTRS];
 int i;
 for(i = 0; i < NUM_PTRS; i++)
 array[i] = malloc(NUM_BYTES);
 for(i = 0; i < NUM_PTRS; i += 2)
 free(array[i]);
 hi.ptr = NULL;
 printf(" Size Status\n");
 printf(" ---- -\n");
 while(heapwalk(&hi) == _HEAPOK)
 printf("%7u %s\n", hi.size, hi.in_use ? "used" : "free");
 return 0;
}
```



## highvideo

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Selects high-intensity characters.                                                                                                                                                                                                                                                                                                                                              |
| <b>Syntax</b>       | <code>#include &lt;conio.h&gt;</code><br><code>void highvideo(void);</code>                                                                                                                                                                                                                                                                                                     |
| <b>Prototype in</b> | conio.h                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Remarks</b>      | <p><b>highvideo</b> selects high-intensity characters by setting the high-intensity bit of the currently selected foreground color.</p> <p>This function does not affect any characters currently on the screen, but does affect those displayed by functions (such as <b>cprintf</b>) that perform direct video, text mode output <i>after</i> <b>highvideo</b> is called.</p> |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Portability</b>  | <b>highvideo</b> works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.                                                                                                                                                                                                                             |
| <b>See also</b>     | <b>cprintf, cputs, gettextinfo, lowvideo, normvideo, textattr, textcolor</b>                                                                                                                                                                                                                                                                                                    |
| <b>Example</b>      | <pre>#include &lt;conio.h&gt;  int main(void) {     clrscr();     lowvideo();     cprintf("Low Intensity text\r\n");     highvideo();     gotoxy(1,2);     cprintf("High Intensity Text\r\n");     return 0; }</pre>                                                                                                                                                            |

## hypot

---

|                     |                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Calculates hypotenuse of a right triangle.                                                           |
| <b>Syntax</b>       | <code>#include &lt;math.h&gt;</code><br><code>double hypot(double <i>x</i>, double <i>y</i>);</code> |
| <b>Prototype in</b> | math.h                                                                                               |
| <b>Remarks</b>      | <p><b>hypot</b> calculates the value <i>z</i> where</p> $z^2 = x^2 + y^2$                            |

and

$$z \geq 0$$

This is equivalent to the length of the hypotenuse of a right triangle, if the lengths of the two sides are  $x$  and  $y$ .

**Return value** On success, **hypot** returns  $z$ , a **double**. On error (such as an overflow), **hypot** sets the global variable *errno* to

ERANGE Result out of range

and returns the value HUGE\_VAL.

Error handling for **hypot** can be modified through the function **matherr**.

**Portability** **hypot** is available on UNIX systems.

**Example**

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

int main(void)
{
 double result;
 double x = 3.0;
 double y = 4.0;

 result = hypot(x, y);
 printf("The hypotenuse is: %lf\n", result);

 return 0;
}
```

## imag

---

**Function** Returns the imaginary part of a complex number.

**Syntax** `#include <complex.h>`  
`double imag(complex x);`

**Prototype in** `complex.h`

**Remarks** The data associated to a complex number consists of two floating-point (double) numbers. **imag** returns the one considered to be the imaginary part.

**Return value** The imaginary part of the complex number.

**Portability** Complex functions require C++ and are not portable.

**See also** **complex, conj, real**

**Example**

```
#include <complex.h>

int main(void)
{
 double x = 3.1, y = 4.2;
 complex z = complex(x,y);
 cout << "z = " << z << "\n";
 cout << " has real part = " << real(z) << "\n";
 cout << " and imaginary real part = " << imag(z) << "\n";
 cout << "z has complex conjugate = " << conj(z) << "\n";
 return 0;
}
```

## imagesize

---

- Function** Returns the number of bytes required to store a bit image.
- Syntax** `#include <graphics.h>`  
`unsigned far imagesize(int left, int top, int right, int bottom);`
- Prototype in** `graphics.h`
- Remarks** **imagesize** determines the size of the memory area required to store a bit image. If the size required for the selected image is greater than or equal to 64K – 1 bytes, **imagesize** returns 0xFFFF (-1).
- Return value** **imagesize** returns the size of the required memory area in bytes.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getimage, putimage**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    void *arrow;
    int x, y, maxx;
```

```

unsigned int size;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

maxx = getmaxx();
x = 0;
y = getmaxy() / 2;

/* draw the image to be grabbed */
draw_arrow(x, y);

/* calculate the size of the image */
size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);

/* allocate memory to hold the image */
arrow = malloc(size);

/* grab the image */
getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

/* repeat until a key is pressed */
while (!kbhit()) {
    /* erase old image */
    putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
    x += ARROW_SIZE;
    if (x >= maxx)
        x = 0;

    /* plot new image */
    putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
}

/* clean up */
free(arrow);
closegraph();
return 0;
}

void draw_arrow(int x, int y)
{
    /* draw an arrow on the screen */
    moveto(x, y);

```

```

    linerel(4*ARROW_SIZE, 0);
    linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}

```

initgraph

Function Initializes the graphics system.

Syntax `#include <graphics.h>`
`void far initgraph(int far *graphdriver, int far *graphmode,`
`char far *pathtodriver);`

Prototype in graphics.h

Remarks **initgraph** initializes the graphics system by loading a graphics driver from disk (or validating a registered driver), and putting the system into graphics mode.

To start the graphics system, first call the **initgraph** function. **initgraph** loads the graphics driver and puts the system into graphics mode. You can tell **initgraph** to use a particular graphics driver and mode, or to autodetect the attached video adapter at run time and pick the corresponding driver.

If you tell **initgraph** to autodetect, it calls **detectgraph** to select a graphics driver and mode. **initgraph** also resets all graphics settings to their defaults (current position, palette, color, viewport, and so on) and resets **graphresult** to 0.

Normally, **initgraph** loads a graphics driver by allocating memory for the driver (through **_graphgetmem**), then loading the appropriate .BGI file from disk. As an alternative to this dynamic loading scheme, you can link a graphics driver file (or several of them) directly into your executable program file. See UTIL.DOC (included with your distribution disks) for more information on BGI OBJ.

pathtodriver specifies the directory path where **initgraph** looks for graphics drivers. **initgraph** first looks in the path specified in *pathtodriver*, then (if they're not there) in the current directory. Accordingly, if *pathtodriver* is null, the driver files (*.BGI) must be in the current directory. This is also the path **settextstyle** searches for the stroked character font files (*.CHR).

**graphdriver* is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the *graphics_drivers* enumeration type, defined in *graphics.h* and listed in Table 2.3.

Table 2.3
Graphics drivers
constants

<i>graphics_drivers</i> constant	Numeric value
DETECT	0 (requests autodetection)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

graphmode* is an integer that specifies the initial graphics mode (unless **graphdriver* equals DETECT; in which case, **graphmode* is set by **initgraph to the highest resolution available for the detected driver). You can give **graphmode* a value using a constant of the *graphics_modes* enumeration type, defined in *graphics.h* and listed in Table 2.5.



graphdriver and *graphmode* must be set to valid values from tables 2.3 and 2.5, or you'll get unpredictable results. The exception is *graphdriver* = DETECT.

In Table 2.5, the **Palette** listings C0, C1, C2, and C3 refer to the four predefined four-color palettes available on CGA (and compatible) systems. You can select the background color (entry #0) in each of these palettes, but the other colors are fixed. These palettes are described in greater detail in Chapter 8, "Video functions" in the *Programmer's Guide* (in the section titled "Color control," toward the end of the chapter) and summarized in Table 2.4.

Table 2.4
Color palettes

Palette number	Color assigned to pixel value		
	1	2	3
0	LIGHTGREEN	LIGHTRED	YELLOW
1	LIGHTCYAN	LIGHTMAGENTA	WHITE
2	GREEN	RED	BROWN
3	CYAN	MAGENTA	LIGHTGRAY

After a call to **initgraph**, **graphdriver* is set to the current graphics driver, and **graphmode* is set to the current graphics mode.

Table 2.5
Graphics modes

Graphics driver	<i>graphics_modes</i>	Value	Column ×row	Palette	Pages
CGA	CGAC0	0	320×200	C0	1
	CGAC1	1	320×200	C1	1
	CGAC2	2	320×200	C2	1
	CGAC3	3	320×200	C3	1
	CGAHI	4	640×200	2 color	1
MCGA	MCGAC0	0	320×200	C0	1
	MCGAC1	1	320×200	C1	1
	MCGAC2	2	320×200	C2	1
	MCGAC3	3	320×200	C3	1
	MCGAMED	4	640×200	2 color	1
	MCGAHI	5	640×480	2 color	1
EGA	EGALO	0	640×200	16 color	4
	EGAHI	1	640×350	16 color	2
EGA64	EGA64LO	0	640×200	16 color	1
	EGA64HI	1	640×350	4 color	1
EGA-MONO	EGAMONOH1	3	640×350	2 color	1*
	EGAMONOH1	3	640×350	2 color	2**
HERC ATT400	HERCMONOH1	0	720×348	2 color	2
	ATT400C0	0	320×200	C0	1
	ATT400C1	1	320×200	C1	1
	ATT400C2	2	320×200	C2	1
	ATT400C3	3	320×200	C3	1
	ATT400MED	4	640×200	2 color	1
	ATT400HI	5	640×400	2 color	1
VGA	VGALO	0	640×200	16 color	2
	VGAMED	1	640×350	16 color	2
	VGAHI	2	640×480	16 color	1
PC3270	PC3270HI	0	720×350	2 color	1
IBM8514	IBM8514HI	1	1024×768	256 color	
	IBM8514LO	0	640×480	256 color	

* 64K on EGAMONO card
** 256K on EGAMONO card

Return value **initgraph** always sets the internal error code; on success, it sets the code to 0. If an error occurred, **graphdriver* is set to -2, -3, -4, or -5, and **graphresult** returns the same value as listed here:

- grNotDetected -2 Cannot detect a graphics card
- grFileNotFound -3 Cannot find driver file
- grInvalidDriver -4 Invalid driver
- grNoLoadMem -5 Insufficient memory to load driver

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also `closegraph`, `detectgraph`, `getdefaultpalette`, `getdrivername`, `getgraphmode`, `getmoderange`, `graphdefaults`, `_graphgetmem`, `graphresult`, `installuserdriver`, `registerbgidriver`, `registerbgifont`, `restorecrtmode`, `setgraphbufsize`, `setgraphmode`

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* return with error code */
    }

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```


inport

Function	Reads a word from a hardware port.
Syntax	<code>#include <dos.h></code> <code>int inport(int <i>portid</i>);</code>
Prototype in	dos.h
Remarks	inport works just like the 80x86 instruction IN . It reads the low byte of a word from the input port specified by <i>portid</i> ; it reads the high byte from <i>portid</i> + 1.
Return value	inport returns the value read.
Portability	inport is unique to the 8086 family.
See also	inportb , outport , outportb
Example	<pre>#include <stdio.h> #include <dos.h> int main(void) { int result; int port = 0; /* serial port 0 */ result = inport(port); printf("Word read from port %d = 0x%X\n", port, result); return 0; }</pre>

inportb

Function	Reads a byte from a hardware port.
Syntax	<code>#include <dos.h></code> <code>unsigned char inportb(int <i>portid</i>);</code>
Prototype in	dos.h
Remarks	inportb is a macro that reads a byte from the input port specified by <i>portid</i> . If inportb is called when dos.h has been included, it will be treated as a macro that expands to inline code. If you don't include dos.h, or if you do include dos.h and #undef the macro inportb , you get the inportb function.
Return value	inportb returns the value read.
Portability	inportb is unique to the 8086 family.

See also `inport`, `outport`, `outportb`

Example

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

int main(void)
{
    unsigned char result;
    int port = 0;          /* serial port 1 */
    result = inportb(port);
    printf("Byte read from port %d = 0x%X\n", port, result);
    return 0;
}
```

inline

I-J

Function Inserts a blank line in the text window.

Syntax `#include <conio.h>`
`void inline(void);`

Prototype in `conio.h`

Remarks `inline` inserts an empty line in the text window at the cursor position using the current text background color. All lines below the empty one move down one line, and the bottom line scrolls off the bottom of the window.

`inline` is used in text mode.

Return value None.

Portability `inline` works with IBM PCs and compatibles only; a corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also `clreol`, `delline`, `window`

Example

```
#include <conio.h>

int main(void)
{
    clrscr();
    printf("INLINE inserts an empty line in the text window\r\n");
    printf("at the cursor position using the current text\r\n");
    printf("background color. All lines below the empty one\r\n");
    printf("move down one line and the bottom line scrolls\r\n");
    printf("off the bottom of the window.\r\n");
    printf("\r\nPress any key to continue:");
}
```

```

gotoxy(1, 3);
getch();
inline();
getch();
return 0;
}

```

installuserdriver

Function Installs a vendor-added device driver to the BGI device driver table.

Syntax `#include <graphics.h>`
`int far installuserdriver(char far *name, int huge (*detect)(void));`

Prototype in graphics.h

Remarks **installuserdriver** allows you to add a vendor-added device driver to the BGI internal table. The *name* parameter is the name of the new device driver file (.BGI), and the *detect* parameter is a pointer to an optional autodetect function that can accompany the new driver. This autodetect function takes no parameters and returns an integer value.

There are two ways to use this vendor-supplied driver. Let's assume you have a new video card called the Spiffy Graphics Array (SGA) and that the SGA manufacturer provided you with a BGI device driver (SGA.BGI). The easiest way to use this driver is to install it by calling **installuserdriver** and then passing the return value (the assigned driver number) directly to **initgraph**.

The other, more general way to use this driver is to link in an autodetect function that will be called by **initgraph** as part of its hardware-detection logic (presumably, the manufacturer of the SGA gave you this autodetect function). When you install the driver (by calling **installuserdriver**), you pass the address of this function, along with the device driver's file name.

After you install the device driver file name and the SGA autodetect function, call **initgraph** and let it go through its normal autodetection process. Before **initgraph** calls its built-in autodetection function (**detectgraph**), it first calls the SGA autodetect function. If the SGA autodetect function doesn't find the SGA hardware, it returns a value of -11 (`grError`), and **initgraph** proceeds with its normal hardware detection logic (which can include calling any other vendor-supplied autodetection functions in the order in which they were "installed"). If, however, the autodetect function determines that an SGA is present, it returns a non-negative mode number; then **initgraph** locates and loads SGA.BGI, puts

the hardware into the default graphics mode recommended by the autodetect function, and finally returns control to your program.

You can install up to ten drivers at one time.

Return value The value returned by **installuserdriver** is the driver number parameter you would pass to **initgraph** in order to select the newly installed driver manually.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **initgraph**, **registerbgidriver**

Example

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

/* function prototypes */
int huge detectEGA(void);
void checkerrors(void);
int main(void)
{
    int gdriver, gmode;

    /* install a user written device driver */
    gdriver = installuserdriver("EGA", detectEGA);

    /* must force use of detection routine */
    gdriver = DETECT;

    /* check for any installation errors */
    checkerrors();

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* check for any initialization errors */
    checkerrors();

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}

/* detects EGA or VGA cards */
int huge detectEGA(void)
```

installuserdriver

```
{
    int driver, mode, sugmode = 0;
    detectgraph(&driver, &mode);
    if ((driver == EGA) || (driver == VGA))
        return sugmode;      /* return suggested video mode number */
    else
        return grError;      /* return an error code */
}

/* check for and report any graphics errors */
void checkerrors(void)
{
    int errorcode;

    /* read result of last graphics operation */
    errorcode = graphresult();
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
}
```

installuserfont

Function	Loads a font file (.CHR) that is not built into the BGI system.
Syntax	<pre>#include <graphics.h> int far installuserfont(char far *name);</pre>
Prototype in	graphics.h
Remarks	<i>name</i> is a path name to a font file containing a stroked font. Up to twenty fonts can be installed at one time.
Return value	installuserfont returns a font ID number that can then be passed to settextstyle to select the corresponding font. If the internal font table is full, a value of -11 (grError) is returned.
Portability	This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
See also	settextstyle
Example	<pre>#include <graphics.h> #include <stdlib.h></pre>

```

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

/* function prototype */
void checkerrors(void);
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode;
    int userfont;
    int midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* check for any initialization errors */
    checkerrors();

    /* install a user-defined font file */
    userfont = installuserfont("USER.CHR");

    /* check for any installation errors */
    checkerrors();

    /* select the user font */
    settextstyle(userfont, HORIZ_DIR, 4);

    /* output some text */
    outtextxy(midx, midy, "Testing!");

    /* clean up */
    getch();
    closegraph();
    return 0;
}

/* check for and report any graphics errors */
void checkerrors(void)
{
    int errorcode;

    /* read result of last graphics operation */
    errorcode = graphresult();
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
}

```

int86

Function	General 8086 software interrupt.
Syntax	<code>#include <dos.h></code> <code>int int86(int <i>intno</i>, union REGS *<i>inregs</i>, union REGS *<i>outregs</i>);</code>
Prototype in	dos.h
Remarks	<p>int86 executes an 8086 software interrupt specified by the argument <i>intno</i>. Before executing the software interrupt, it copies register values from <i>inregs</i> into the registers.</p> <p>After the software interrupt returns, int86 copies the current register values to <i>outregs</i>, copies the status of the carry flag to the <i>x.cflag</i> field in <i>outregs</i>, and copies the value of the 8086 flags register to the <i>x.flags</i> field in <i>outregs</i>. If the carry flag is set, it usually indicates that an error has occurred.</p> <p>Note that <i>inregs</i> can point to the same structure that <i>outregs</i> points to.</p>
Return value	int86 returns the value of AX after completion of the software interrupt. If the carry flag is set (<code>outregs -> x.cflag != 0</code>), indicating an error, this function sets the global variable <code>_doserrno</code> to the error code.
Portability	int86 is unique to the 8086 family of processors.
See also	bdos , bdosptr , geninterrupt , int86x , intdos , intdosx , intr
Example	<pre>#include <stdio.h> #include <conio.h> #include <dos.h> #define VIDEO 0x10 void movetoxy(int x, int y) { union REGS regs; regs.h.ah = 2; /* set cursor position */ regs.h.dh = y; regs.h.dl = x; regs.h.bh = 0; /* video page 0 */ int86(VIDEO, &regs, &regs); } int main(void) { clrscr(); movetoxy(35, 10); printf("Hello\n"); }</pre>

```

    return 0;
}

```

int86x

Function General 8086 software interrupt interface.

Syntax `#include <dos.h>`
`int int86x(int intno, union REGS *inregs, union REGS *outregs,
 struct SREGS *segregs);`

Prototype in `dos.h`

Remarks **int86x** executes an 8086 software interrupt specified by the argument *intno*. Before executing the software interrupt, it copies register values from *inregs* into the registers.

In addition, **int86x** copies the *segregs ->ds* and *segregs ->es* values into the corresponding registers before executing the software interrupt. This feature allows programs that use far pointers or a large data memory model to specify which segment is to be used for the software interrupt.

After the software interrupt returns, **int86x** copies the current register values to *outregs*, the status of the carry flag to the *x.cflag* field in *outregs*, and the value of the 8086 flags register to the *x.flags* field in *outregs*. In addition, **int86x** restores DS and sets the *segregs ->es* and *segregs ->ds* fields to the values of the corresponding segment registers. If the carry flag is set, it usually indicates that an error has occurred.

int86x lets you invoke an 8086 software interrupt that takes a value of DS different from the default data segment, and/or takes an argument in ES.

Note that *inregs* can point to the same structure that *outregs* points to.

Return value **int86x** returns the value of AX after completion of the software interrupt. If the carry flag is set (`outregs -> x.cflag != 0`), indicating an error, this function sets the global variable `_doserrno` to the error code.

Portability **int86x** is unique to the 8086 family of processors.

See also `bdos`, `bdosptr`, `geninterrupt`, `intdos`, `intdosx`, `int86`, `intr`, `segread`

Example

```

#include <dos.h>
#include <process.h>
#include <stdio.h>

int main(void)
{

```

I-J


```

char filename[80];
union REGS inregs, outregs;
struct SREGS segregs;
printf("Enter file name: ");
gets(filename);
inregs.h.ah = 0x43;
inregs.h.al = 0;
inregs.x.dx = FP_OFF(filename);
segregs.ds = FP_SEG(filename);
int86x(0x21, &inregs, &outregs, &segregs);
printf("File attribute: %X\n", outregs.x.cx);
return 0;
}

```

intdos

Function	General DOS interrupt interface.
Syntax	<code>#include <dos.h></code> <code>int intdos(union REGS *inregs, union REGS *outregs);</code>
Prototype in	dos.h
Remarks	<p>intdos executes DOS interrupt 0x21 to invoke a specified DOS function. The value of <i>inregs</i> -> <i>h.ah</i> specifies the DOS function to be invoked.</p> <p>After the interrupt 0x21 returns, intdos copies the current register values to <i>outregs</i>, copies the status of the carry flag to the <i>x.cflag</i> field in <i>outregs</i>, and copies the value of the 8086 flags register to the <i>x.flags</i> field in <i>outregs</i>. If the carry flag is set, it indicates that an error has occurred.</p> <p>Note that <i>inregs</i> can point to the same structure that <i>outregs</i> points to.</p>
Return value	intdos returns the value of AX after completion of the DOS function call. If the carry flag is set (<i>outregs</i> -> <i>x.cflag</i> != 0), indicating an error, it sets the global variable <i>_doserrno</i> to the error code.
Portability	intdos is unique to DOS.
See also	bdos, bdosptr, geninterrupt, int86, int86x, intdosx, intr
Example	<pre> #include <stdio.h> #include <dos.h> /* deletes file name; returns 0 on success, nonzero on failure */ int delete_file(char near *filename) { union REGS regs; </pre>

```

int ret;
regs.h.ah = 0x41;                               /* delete file */
regs.x.dx = (unsigned) filename;
ret = intdos(&regs, &regs);

/* if carry flag is set, there was an error */
return(regs.x.cflag ? ret : 0);
}

int main(void)
{
    int err;
    err = delete_file("NOTEXIST.$$$");
    if (!err)
        printf("Able to delete NOTEXIST.$$$\n");
    else
        printf("Not able to delete NOTEXIST.$$$\n");
    return 0;
}

```

I-J

intdosx

Function General DOS interrupt interface.

Syntax #include <dos.h>
int intdosx(union REGS *inregs, union REGS *outregs,
struct SREGS *segregs);

Prototype in dos.h

Remarks **intdosx** executes DOS interrupt 0x21 to invoke a specified DOS function. The value of *inregs* -> *h.ah* specifies the DOS function to be invoked.

In addition, **intdosx** copies the *segregs* -> *ds* and *segregs* -> *es* values into the corresponding registers before invoking the DOS function. This feature allows programs that use far pointers or a large data memory model to specify which segment is to be used for the function execution.

After the interrupt 0x21 returns, **intdosx** copies the current register values to *outregs*, copies the status of the carry flag to the *x.cflag* field in *outregs*, and copies the value of the 8086 flags register to the *x.flags* field in *outregs*. In addition, **intdosx** sets the *segregs* -> *es* and *segregs* -> *ds* fields to the values of the corresponding segment registers and then restores DS. If the carry flag is set, it indicates that an error occurred.

intdosx lets you invoke a DOS function that takes a value of DS different from the default data segment and/or takes an argument in ES.

intdosx

Note that *inregs* can point to the same structure that *outregs* points to.

Return value **intdosx** returns the value of AX after completion of the DOS function call. If the carry flag is set (*outregs* -> *x.cflag* != 0), indicating an error, it sets the global variable *_doserrno* to the error code.

Portability **intdosx** is unique to DOS.

See also **bdos**, **bdosptr**, **geninterrupt**, **int86**, **int86x**, **intdos**, **intr**, **segread**

Example

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

/* deletes file name; returns 0 on success, nonzero on failure */
int delete_file(char far *filename)
{
    union REGS regs; struct SREGS sregs;
    int ret;
    regs.h.ah = 0x41;                               /* delete file */
    regs.x.dx = FP_OFF(filename);
    sregs.ds = FP_SEG(filename);
    ret = intdosx(&regs, &regs, &sregs);

    /* if carry flag is set, there was an error */
    return(regs.x.cflag ? ret : 0);
}

int main(void)
{
    int err;
    err = delete_file("NOTEXIST.$$$");
    if (!err)
        printf("Able to delete NOTEXIST.$$$\n");
    else
        printf("Not Able to delete NOTEXIST.$$$\n");
    return 0;
}
```

intr

Function Alternate 8086 software interrupt interface.

Syntax `#include <dos.h>`
`void intr(int intno, struct REGPACK *preg);`

Prototype in dos.h

Remarks The **intr** function is an alternate interface for executing software interrupts. It generates an 8086 software interrupt specified by the argument *intno*.

intr copies register values from the **REGPACK** structure **preg* into the registers before executing the software interrupt. After the software interrupt completes, **intr** copies the current register values into **preg*, including the flags.

The arguments passed to **intr** are as follows:

intno Interrupt number to be executed

preg Address of a structure containing

 (a) the input registers before the interrupt call

 (b) the value of the registers after the interrupt call

The **REGPACK** structure (defined in `dos.h`) has the following format:

```
struct REGPACK {
    unsigned r_ax, r_bx, r_cx, r_dx;
    unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

Return value No value is returned. The **REGPACK** structure **preg* contains the value of the registers after the interrupt call.

Portability **intr** is unique to the 8086 family of processors.

See also **geninterrupt**, **int86**, **int86x**, **intdos**, **intdosx**

Example

```
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <dos.h>
#define CF 1 /* Carry flag */

int main(void)
{
    char directory[80];
    struct REGPACK reg;
    printf("Enter directory to change to: ");
    gets(directory);
    reg.r_ax = 0x3B << 8;          /* shift 3Bh into AH */
    reg.r_dx = FP_OFF(directory);
    reg.r_ds = FP_SEG(directory);
    intr(0x21, &reg);
    if (reg.r_flags & CF)
        printf("Directory change failed\n");
    getcwd(directory, 80);
}
```

```

printf("The current directory is: %s\n", directory);
return 0;
}

```

ioctl

Function Controls I/O device.

Syntax #include <io.h>
int ioctl(int *handle*, int *func* [, void **argdx*, int *argcx*]);

Prototype in io.h

Remarks This is a direct interface to the DOS call 0x44 (IOCTL).

The exact function depends on the value of *func* as follows:

- 0 Get device information.
- 1 Set device information (in *argdx*).
- 2 Read *argcx* bytes into the address pointed to by *argdx*.
- 3 Write *argcx* bytes from the address pointed to by *argdx*.
- 4 Same as 2 except *handle* is treated as a drive number (0 equals default, 1 equals A, and so on).
- 5 Same as 3 except *handle* is a drive number (0 equals default, 1 equals A, and so on).
- 6 Get input status.
- 7 Get output status.
- 8 Test removability; DOS 3.0 only.
- 11 Set sharing conflict retry count; DOS 3.0 only.

ioctl can be used to get information about device channels. Regular files can also be used, but only *func* values 0, 6, and 7 are defined for them. All other calls return an EINVAL error for files.

See the documentation for system call 0x44 in your DOS reference manuals for detailed information on argument or return values.

The arguments *argdx* and *argcx* are optional.

ioctl provides a direct interface to DOS device drivers for special functions. As a result, the exact behavior of this function varies across different vendors' hardware and in different devices. Also, several vendors do not follow the interfaces described here. Refer to the vendor BIOS documentation for exact use of **ioctl**.

Return value For *func* 0 or 1, the return value is the device information (DX of the IOCTL call).

For *func* values of 2 through 5, the return value is the number of bytes actually transferred.

For *func* values of 6 or 7, the return value is the device status.

In any event, if an error is detected, a value of -1 is returned, and the global variable *errno* is set to one of the following:

EINVAL	Invalid argument
EBADF	Bad file number
EINVDAT	Invalid data

Portability **ioctl** is available on UNIX systems, but not with these parameters or functionality. UNIX version 7 and System III differ from each other in their use of **ioctl**. **ioctl** calls are not portable to UNIX and are rarely portable across DOS machines.

DOS 3.0 extends **ioctl** with *func* values of 8 and 11.

Example

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

int main(void)
{
    int stat;

    /* use func 8 to determine if the default drive is removable */
    stat = ioctl(0, 8, 0, 0);
    if (!stat)
        printf("Drive %c is removable.\n", getdisk() + 'A');
    else
        printf("Drive %c is not removable.\n", getdisk() + 'A');
    return 0;
}
```

isalnum

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isalnum(int c);`

Prototype in `ctype.h`

Remarks **isalnum** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii**(*c*) is true or *c* is EOF.

isalnum

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isalnum** returns nonzero if *c* is a letter (*A* to *Z* or *a* to *z*) or a digit (0 to 9).

Portability **isalnum** is available on UNIX machines.

Example

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

int main(void)
{
    char c = 'C';
    if (isalnum(c))
        printf("%c is alphanumeric\n",c);
    else
        printf("%c isn't alphanumeric\n",c);
    return 0;
}
```

isalpha

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isalpha(int c);`

Prototype in `ctype.h`

Remarks **isalpha** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii**(*c*) is true or *c* is EOF.

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isalpha** returns nonzero if *c* is a letter (*A* to *Z* or *a* to *z*).

Portability **isalpha** is available on UNIX machines and is compatible with ANSI C. It is compatible with Kernighan and Ritchie.

Example

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

int main(void)
{
    char c = 'C';
    if (isalpha(c))
        printf("%c is alphabetic\n",c);
    else
        printf("%c isn't alphabetic\n",c);
}
```

```
    return 0;
}
```

isascii

Function	Character classification macro.
Syntax	<code>#include <ctype.h></code> <code>int isascii(int c);</code>
Prototype in	ctype.h
Remarks	isascii is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. isascii is defined on all integer values.
Return value	isascii returns nonzero if the low order byte of <i>c</i> is in the range 0 to 127 (0x00-0x7F).
Portability	isascii is available on UNIX machines.
Example	<pre>#include <ctype.h> #include <stdio.h> int main(void) { char c = 'C'; if (isascii(c)) printf("%c is ascii\n",c); else printf("%c isn't ascii\n",c); return 0; }</pre>



isatty

Function	Checks for device type.
Syntax	<code>#include <io.h></code> <code>int isatty(int <i>handle</i>);</code>
Prototype in	io.h
Remarks	isatty determines whether <i>handle</i> is associated with any one of the following character devices:

isatty

- a terminal
- a console
- a printer
- a serial port

Return value If the device is a character device, **isatty** returns a nonzero integer. If it is not such a device, **isatty** returns 0.

Portability **isatty** is unique to DOS.

Example

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

int main(void)
{
    int handle;
    handle = fileno(stdprn);
    if (isatty(handle))
        printf("Handle %d is a device type\n", handle);
    else
        printf("Handle %d isn't a device type\n", handle);
    return 0;
}
```

isctrl

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isctrl(int c);`

Prototype in `ctype.h`

Remarks **isctrl** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii(c)** is true or *c* is EOF.

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isctrl** returns nonzero if *c* is a delete character or ordinary control character (0x7F or 0x00 to 0x1F).

Portability **isctrl** is available on UNIX machines and is compatible with ANSI C.

Example

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

int main(void)
{
```

```

char c = 'C';
if (isctrnl(c))
    printf("%c is a control character\n",c);
else
    printf("%c isn't a control character\n",c);
return 0;
}

```

isdigit

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isdigit(int c);`

Prototype in `ctype.h`

Remarks **isdigit** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii**(*c*) is true or *c* is EOF.

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isdigit** returns nonzero if *c* is a digit (0 to 9).

Portability **isdigit** is available on UNIX machines and is compatible with ANSI C. It is compatible with Kernighan and Ritchie.

Example

```

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

int main(void)
{
    char c = 'C';
    if (isdigit(c))
        printf("%c is a digit\n",c);
    else
        printf("%c isn't a digit\n",c);
    return 0;
}

```

isgraph

Function	Character classification macro.
Syntax	<pre>#include <ctype.h> int isgraph(int c);</pre>
Prototype in	ctype.h
Remarks	<p>isgraph is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when isascii(c) is true or <i>c</i> is EOF.</p> <p>You can make this macro available as a function by undefining (#undef) it.</p>
Return value	isgraph returns nonzero if <i>c</i> is a printing character, like isprint , except that a space character is excluded.
Portability	isgraph is available on UNIX machines and is compatible with ANSI C.
Example	<pre>#include <ctype.h> #include <stdio.h> int main(void) { char c = 'C'; if (isgraph(c)) printf("%c is a graphic character\n",c); else printf("%c isn't a graphic character\n",c); return 0; }</pre>

islower

Function	Character classification macro.
Syntax	<pre>#include <ctype.h> int islower(int c);</pre>
Prototype in	ctype.h
Remarks	<p>islower is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when isascii(c) is true or <i>c</i> is EOF.</p> <p>You can make this macro available as a function by undefining (#undef) it.</p>

- Return value** **islower** returns nonzero if *c* is a lowercase letter (*a* to *z*).
- Portability** **islower** is available on UNIX machines and is compatible with ANSI C. It is compatible with Kernighan and Ritchie.
- Example**
- ```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
 char c = 'C';

 if (islower(c))
 printf("%c is lower case\n",c);
 else
 printf("%c isn't lower case\n",c);
 return 0;
}
```

## isprint

---

- Function** Character classification macro.
- Syntax** `#include <ctype.h>`  
`int isprint(int c);`
- Prototype in** `ctype.h`
- Remarks** **isprint** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii**(*c*) is true or *c* is EOF.
- You can make this macro available as a function by undefining (**#undef**) it.
- Return value** **isprint** returns nonzero if *c* is a printing character (0x20 to 0x7E).
- Portability** **isprint** is available on UNIX machines and is compatible with ANSI C.
- Example**
- ```
#include <ctype.h>
#include <stdio.h>

int main(void)
{
    char c = 'C';
    if (isprint(c))
        printf("%c is a printable character\n",c);
    else
        printf("%c isn't a printable character\n",c);
}
```

ispunct

```
    return 0;  
}
```

ispunct

Function	Character classification macro.
Syntax	<pre>#include <ctype.h> int ispunct(int c);</pre>
Prototype in	ctype.h
Remarks	<p>ispunct is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when isascii(c) is true or <i>c</i> is EOF.</p> <p>You can make this macro available as a function by undefining (#undef) it.</p>
Return value	ispunct returns nonzero if <i>c</i> is a punctuation character (iscntrl or isspace).
Portability	ispunct is available on UNIX machines and is compatible with ANSI C.
Example	<pre>#include <ctype.h> #include <stdio.h> int main(void) { char c = 'C'; if (ispunct(c)) printf("%c is a punctuation character\n",c); else printf("%c isn't a punctuation character\n",c); return 0; }</pre>

isspace

Function	Character classification macro.
Syntax	<pre>#include <ctype.h> int isspace(int c);</pre>
Prototype in	ctype.h
Remarks	<p>isspace is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when isascii(c) is true or <i>c</i> is EOF.</p>

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isspace** returns nonzero if *c* is a space, tab, carriage return, new line, vertical tab, or formfeed (0x09 to 0x0D, 0x20).

Portability **isspace** is available on UNIX machines and is compatible with ANSI C. It is compatible with Kernighan and Ritchie.

Example

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

int main(void)
{
    char c = 'C';
    if (isspace(c))
        printf("%c is white space\n",c);
    else
        printf("%c isn't white space\n",c);
    return 0;
}
```

I-J

isupper

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isupper(int c);`

Prototype in `ctype.h`

Remarks **isupper** is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when **isascii**(*c*) is true or *c* is EOF.

You can make this macro available as a function by undefining (**#undef**) it.

Return value **isupper** returns nonzero if *c* is an uppercase letter (A to Z).

Portability **isupper** is available on UNIX machines and is compatible with ANSI C. It is compatible with Kernighan and Ritchie.

Example

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

int main(void)
{
    char c = 'C';
    if (isupper(c))
        printf("%c is upper case\n",c);
}
```

isupper

```
    else
        printf("%c isn't upper case\n",c);
    return 0;
}
```

isxdigit

Function Character classification macro.

Syntax `#include <ctype.h>`
`int isxdigit(int c);`

Prototype in `ctype.h`

Remarks `isxdigit` is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false. It is defined only when `isascii(c)` is true or `c` is EOF.

You can make this macro available as a function by undefining (`#undef`) it.

Return value `isxdigit` returns nonzero if `c` is a hexadecimal digit (0 to 9, *A to F*, *a to f*).

Portability `isxdigit` is available on UNIX machines and is compatible with ANSI C.

Example

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

int main(void)
{
    char c = 'C';
    if (isxdigit(c))
        printf("%c is hexadecimal\n",c);
    else
        printf("%c isn't hexadecimal\n",c);
    return 0;
}
```

itoa

Function Converts an integer to a string.

Syntax `#include <stdlib.h>`
`char *itoa(int value, char *string, int radix);`

Prototype in `stdlib.h`

Remarks `itoa` converts *value* to a null-terminated string and stores the result in *string*. With `itoa`, *value* is an integer.

radix specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. If *value* is negative and *radix* is 10, the first character of *string* is the minus sign (-).



The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (\0). `itoa` can return up to 17 bytes.

Return value `itoa` returns a pointer to *string*.

Portability `itoa` is unique to DOS.

See also `ltoa`, `ultoa`

Example

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

int main(void)
{
    int number = 12345;
    char string[25];
    itoa(number, string, 10);
    printf("integer = %d string = %s\n", number, string);
    return 0;
}
```

kbhit

Function Checks for currently available keystrokes.

Syntax `#include <conio.h>`
`int kbhit(void);`

Prototype in `conio.h`

Remarks `kbhit` checks to see if a keystroke is currently available. Any available keystrokes can be retrieved with `getch` or `getche`.

Return value If a keystroke is available, `kbhit` returns a nonzero value. Otherwise, it returns 0.

Portability `kbhit` is unique to DOS. This function is not Windows compatible.

See also `getch`, `getche`

Example `#include <conio.h>`

kbhit

```
int main(void)
{
    cprintf("Press any key to continue:");
    while (!kbhit()) /* do nothing */ ;
    cprintf("\r\nA key was pressed...\r\n");
    return 0;
}
```

keep

Function Exits and remains resident.

Syntax `#include <dos.h>`
`void keep(unsigned char status, unsigned size);`

Prototype In dos.h

Remarks **keep** returns to DOS with the exit status in *status*. The current program remains resident, however. The program is set to *size* paragraphs in length, and the remainder of the memory of the program is freed.

keep can be used when installing TSR programs. **keep** uses DOS function 0x31.

Return value None.

Portability **keep** is unique to DOS. This function is not Windows compatible.

See also **abort**, **exit**

Example `/* NOTE: This is an interrupt service routine. You canNOT compile this program with Test Stack Overflow turned on and get an executable file which will operate correctly. Due to the nature of this function the formula used to compute the number of paragraphs may not necessarily work in all cases. Use with care! Terminate Stay Resident (TSR) programs are complex and no other support for them is provided. Refer to the MS-DOS technical documentation for more information. */`

```
#include <dos.h>

#define INTR 0x1C      /* the clock tick interrupt */
#define ATTR 0x7900   /* screen attribute (blue on grey) */

/* reduce heaplength and stacklength to make smaller program in memory */
extern unsigned _heaplen = 1024;
extern unsigned _stklen = 512;

void interrupt (*oldhandler)(void);
void interrupt handler(void)
```

```

{
    unsigned int (far *screen)[80];
    static int count;

    /* for a color screen the video memory is at B800:0000;
       for a monochrome system use B000:000 */
    screen = MK_FP(0xB800,0);

    /* increase the counter and keep it
       within 0 to 9 */
    count++;
    count %= 10;

    /* put the number on the screen */
    screen[0][79] = count + '0' + ATTR;

    /* call the old interrupt handler */
    oldhandler();
}

int main(void)
{
    /* get the address of the current clock tick interrupt */
    oldhandler = getvect(INTR);

    /* install the new interrupt handler */
    setvect(INTR, handler);

    /* _psp is the starting address of the program in memory. The top
       of the stack is the end of the program. Using _SS and _SP
       together gets the end of the stack. You may want to allow a bit
       of safety space to insure that enough room is being allocated;
       that is, (_SS + ((_SP + safety space)/16) - _psp) */
    keep(0, (_SS + (_SP/16) - _psp));
    return 0;
}

```

K-M

labs

Function	Gives long absolute value.
Syntax	#include <math.h> long int labs(long int x);
Prototype in	math.h, stdlib.h
Remarks	labs computes the absolute value of the parameter <i>x</i> .
Return value	labs returns the absolute value of <i>x</i> .
Portability	labs is available on UNIX systems and is defined in ANSI C.

See also `abs`, `cabs`, `fabs`

Example

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

int main(void)
{
    long result;
    long x = -12345678L;
    result= labs(x);
    printf("number: %ld abs value: %ld\n", x, result);
    return 0;
}
```

ldexp

Function Calculates $x \times 2^{exp}$.

Syntax `#include <math.h>`
`double ldexp(double x, int exp);`

Prototype in `math.h`

Remarks **ldexp** calculates the **double** value $x \times 2^{exp}$.

Return value On success, **ldexp** returns the value it calculated, $x \times 2^{exp}$.
 Error handling for **ldexp** can be modified through the function **matherr**.

Portability **ldexp** is available on UNIX systems and is defined in ANSI C.

See also `exp`, `frexp`, `modf`

Example

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

int main(void)
{
    double value;
    double x = 2;

    /* ldexp raises 2 by a power of 3 then multiplies the result by 2 */
    value = ldexp(x,3);
    printf("The ldexp value is: %lf\n", value);
    return 0;
}
```

ldiv

Function Divides two **longs**, returning quotient and remainder.

Syntax `#include <stdlib.h>`
`ldiv_t ldiv(long int numer, long int denom);`

Prototype in `stdlib.h`

Remarks **ldiv** divides two **longs** and returns both the quotient and the remainder as an *ldiv_t* type. *numer* and *denom* are the numerator and denominator, respectively. The *ldiv_t* type is a structure of **longs** defined (with **typedef**) in `stdlib.h` as follows:

```
typedef struct {
    long int quot;    /* quotient */
    long int rem;    /* remainder */
} ldiv_t;
```

Return value **ldiv** returns a structure whose elements are *quot* (the quotient) and *rem* (the remainder).

Portability **ldiv** is compatible with ANSI C.

See also **div**

Example

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

int main(void)
{
    ldiv_t lx;
    lx = ldiv(100000L, 30000L);
    printf("100000 div 30000 = %ld remainder %ld\n", lx.quot, lx.rem);
    return 0;
}
```



lfind

Function	Performs a linear search.
Syntax	<pre>#include <stdlib.h> void *lfind(const void *key, const void *base, size_t *num, size_t width, int (*fcmp)(const void *, const void *));</pre>
Prototype in	stdlib.h
Remarks	<p>lfind makes a linear search for the value of <i>key</i> in an array of sequential records. It uses a user-defined comparison routine (<i>fcmp</i>).</p> <p>The array is described as having <i>*num</i> records that are <i>width</i> bytes wide, and begins at the memory location pointed to by <i>base</i>.</p>
Return value	lfind returns the address of the first entry in the table that matches the search key. If no match is found, lfind returns null. The comparison routine must return 0 if <i>*elem1</i> == <i>*elem2</i> , and nonzero otherwise (<i>elem1</i> and <i>elem2</i> are its two parameters).
Portability	lfind is unique to DOS.
See also	bsearch , lsearch , qsort
Example	<pre>#include <stdio.h> #include <stdlib.h> int compare(int *x, int *y) { return(*x - *y); } int main(void) { int array[5] = {35, 87, 46, 99, 12}; size_t nelem = 5; int key; int *result; key = 99; result = lfind(&key, array, &nelem, sizeof(int), (int(*) (const void *, const void *)) compare); if (result) printf("Number %d found\n",key); else printf("Number %d not found\n",key); return 0; }</pre>

line

- Function** Draws a line between two specified points.
- Syntax** `#include <graphics.h>`
`void far line(int x1, int y1, int x2, int y2);`
- Prototype in** `graphics.h`
- Remarks** **line** draws a line in the current color, using the current line style and thickness between the two points specified, $(x1,y1)$ and $(x2,y2)$, without updating the current position (CP).
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getlinesettings**, **linereel**, **lineto**, **setcolor**, **setlinestyle**, **setwritemode**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int xmax, ymax;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }

    setcolor(getmaxcolor());
    xmax = getmaxx();
```



line

```
ymax = getmaxy();

/* draw a diagonal line */
line(0, 0, xmax, ymax);

/* clean up */
getch();
closegraph();
return 0;
}
```

linere1

- Function** Draws a line a relative distance from the current position (CP).
- Syntax** `#include <graphics.h>`
`void far linere1(int dx, int dy);`
- Prototype in** `graphics.h`
- Remarks** **linere1** draws a line from the CP to a point that is a relative distance (*dx,dy*) from the CP. The CP is advanced by (*dx,dy*).
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getlinesettings, line, lineto, setcolor, setlinestyle, setwritemode**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 char msg[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
```

```

 getch();
 exit(1);
 }

 /* move the CP to location (20,30) */
 moveto(20,30);

 /* create and output a message at (20,30) */
 sprintf(msg, " (%d, %d)", getx(), gety());
 outtextxy(20,30, msg);

 /* draw line to a point a relative distance away from current CP */
 linerel(100, 100);

 /* create and output a message at CP */
 sprintf(msg, " (%d, %d)", getx(), gety());
 outtext(msg);

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

## lineto

---

- Function** Draws a line from the current position (CP) to  $(x,y)$ .
- Syntax** `#include <graphics.h>`  
`void far lineto(int x, int y);`
- Prototype in** graphics.h
- Remarks** **lineto** draws a line from the CP to  $(x,y)$ , then moves the CP to  $(x,y)$ .
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getlinesettings**, **line**, **linerel**, **setcolor**, **setlinestyle**, **setvisualpage**, **setwritemode**
- Example**
- ```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{

```



```

/* request autodetection */
int gdriver = DETECT, gmode, errorcode;
char msg[80];

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
}

/* move the CP to location (20,30) */
moveto(20, 30);

/* create and output a message at (20,30) */
sprintf(msg, " (%d, %d)", getx(), gety());
outtextxy(20,30, msg);

/* draw a line to (100,100) */
lineto(100, 100);

/* create and output a message at CP */
sprintf(msg, " (%d, %d)", getx(), gety());
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}

```

localeconv

Function	Returns a pointer to the current locale structure.
Syntax	#include <locale.h> struct lconv *localeconv(void);
Prototype in	locale.h
Remarks	This function sets up country-specific monetary and other numeric formats. However, Borland C++ currently only supports locale C.
Return value	Returns a pointer to the current locale structure. See locale.h for details.
Portability	localeconv is compatible with ANSI C.

See also **setlocale**

Example

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

int main(void)
{
    struct lconv ll;
    struct lconv *conv = &ll;

    /* read the locality conversion structure */
    conv = localeconv();

    /* display the structure */
    printf("Decimal Point           : %s\n", conv->decimal_point);
    printf("Thousands Separator       : %s\n", conv->thousands_sep);
    printf("Grouping                     : %s\n", conv->grouping);
    printf("International Currency symbol : %s\n", conv->int_curr_symbol);
    printf("$ thousands separator   : %s\n", conv->mon_thousands_sep);
    printf("$ grouping              : %s\n", conv->mon_grouping);
    printf("Positive sign                 : %s\n", conv->positive_sign);
    printf("Negative sign                  : %s\n", conv->negative_sign);
    printf("International fraction digits : %d\n", conv->int_frac_digits);
    printf("Fraction digits               : %d\n", conv->frac_digits);
    printf("Positive $ symbol precedes    : %d\n", conv->p_cs_precedes);
    printf("Positive sign space separation : %d\n", conv->p_sep_by_space);
    printf("Negative $ symbol precedes    : %d\n", conv->n_cs_precedes);
    printf("Negative sign space separation : %d\n", conv->n_sep_by_space);
    printf("Positive sign position        : %d\n", conv->p_sign_posn);
    printf("Negative sign position        : %d\n", conv->n_sign_posn);
    return 0;
}
```



localtime

Function Converts date and time to a structure.

Syntax `#include <time.h>`
`struct tm *localtime(const time_t *timer);`

Prototype in time.h

Remarks **localtime** accepts the address of a value returned by **time** and returns a pointer to the structure of type *tm* containing the broken-down time. It corrects for the time zone and possible daylight saving time.

The global long variable *timezone* should be set to the difference in seconds between GMT and local standard time (in PST, *timezone* is

localtime

8×60×60). The global variable *daylight* should be set to nonzero *only* if the standard U.S. daylight saving time conversion should be applied.

The **tm** structure declaration from the `time.h` include file follows:

```
struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1 to 31), month (0 to 11), weekday (Sunday equals 0), year – 1900, day of year (0 to 365), and a flag that is nonzero if daylight saving time is in effect.

Return value **localtime** returns a pointer to the structure containing the broken-down time. This structure is a static that is overwritten with each call.

Portability **localtime** is available on UNIX systems, and it is compatible with ANSI C.

See also **asctime**, **ctime**, **ftime**, **gmtime**, **stime**, **time**, **tzset**

Example

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

int main(void)
{
    time_t timer;
    struct tm *tblock;

    /* gets time of day
    timer = time(NULL);

    /* converts date/time to a structure */
    tblock = localtime(&timer);
    printf("Local time is: %s", asctime(tblock));
    return 0;
}
```

lock

- Function** Sets file-sharing locks.
- Syntax** `#include <io.h>`
`int lock(int handle, long offset, long length);`
- Prototype in** io.h
- Remarks** **lock** provides an interface to the DOS 3.x file-sharing mechanism. SHARE.EXE must be loaded before using **lock**.
- lock** can be placed on arbitrary, nonoverlapping regions of any file. A program attempting to read or write into a locked region will retry the operation three times. If all three retries fail, the call fails with an error.
- Return value** **lock** returns 0 on success, -1 on error.
- Portability** **lock** is unique to DOS 3.x. Older versions of DOS do not support it.
- See also** **open, sopen, unlock**

Example

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
    int handle, status;
    long length;

    /* must have DOS SHARE.EXE loaded for file locking to function */
    handle = sopen("c:\\autoexec.bat", O_RDONLY, SH_DENYNO, S_IREAD);
    if (handle < 0) {
        printf("sopen failed\n");
        exit(1);
    }
    length = filelength(handle);
    status = lock(handle, 0L, length/2);
    if (status == 0)
        printf("lock succeeded\n");
    else
        printf("lock failed\n");
    status = unlock(handle, 0L, length/2);
    if (status == 0)
```



lock

```
        printf("unlock succeeded\n");
    else
        printf("unlock failed\n");
    close(handle);
    return 0;
}
```

log

Function	Calculates the natural logarithm of x .	
Syntax	<i>Real version:</i> <code>#include <math.h></code>	<i>Complex version:</i> <code>#include <complex.h></code>
Prototype in	<i>Real version:</i> <code>math.h</code>	<i>Complex version:</i> <code>complex.h</code>
Remarks	log calculates the natural logarithm of x . The complex natural logarithm is defined by $\log(z) = \log(\text{abs}(z)) + i \arg(z)$	
Return value	On success, log returns the value calculated, $\ln(x)$. If the argument x passed to log is real and less than 0, the global variable <i>errno</i> is set to EDOM Domain error log(0) returns the value negative HUGE_VAL and sets <i>errno</i> to ERANGE. Error handling for log can be modified through the function matherr .	
Portability	The real version of log is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.	
See also	complex, exp, log10, sqrt	
Example	<pre>#include <math.h> #include <stdio.h> int main(void) { double result; double x = 8.6872; result = log(x); printf("The natural log of %lf is %lf\n", x, result); }</pre>	

```

    return 0;
}

```

log10

Function Calculates $\log_{10}(x)$.

Syntax *Real version:* `#include <math.h>`
`double log10(double x);` *Complex version:* `#include <complex.h>`
`complex log10(complex x);`

Prototype in *Real version:* `math.h` *Complex version:* `complex.h`

Remarks **log10** calculates the base 10 logarithm of x .

The complex common logarithm is defined by

$$\log_{10}(z) = \log(z) / \log(10)$$

Return value On success, **log10** returns the value calculated, $\log_{10}(x)$.

If the argument x passed to **log10** is real and less than 0, the global variable *errno* is set to

EDOM Domain error

log10(0) returns the value negative HUGE_VAL.

Error handling for **log10** can be modified through the function **matherr**.

Portability The real version of **log10** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

See also **complex, exp, log**

Example

```

#include <math.h>
#include <stdio.h>

int main(void)
{
    double result;
    double x = 800.6872;
    result = log10(x);
    printf("The common log of %lf is %lf\n", x, result);

    return 0;
}

```



longjmp

Function	Performs nonlocal goto.
Syntax	<pre>#include <setjmp.h> void longjmp(jmp_buf <i>jmpb</i>, int <i>retval</i>);</pre>
Prototype in	setjmp.h
Remarks	<p>A call to longjmp restores the task state captured by the last call to setjmp with the argument <i>jmpb</i>. It then returns in such a way that setjmp appears to have returned with the value <i>retval</i>.</p> <p>A task state is</p> <ul style="list-style-type: none"> ■ all segment registers (CS, DS, ES, SS) ■ register variables (SI, DI) ■ stack pointer (SP) ■ frame base pointer (BP) ■ flags <p>A task state is complete enough that setjmp and longjmp can be used to implement coroutines.</p> <p>setjmp must be called before longjmp. The routine that called setjmp and set up <i>jmpb</i> must still be active and cannot have returned before the longjmp is called. If this happens, the results are unpredictable.</p> <p>longjmp cannot pass the value 0; if 0 is passed in <i>retval</i>, longjmp will substitute 1.</p> <p>➔ You can't use setjmp and longjmp for implementing coroutines if your program is overlaid. Normally, setjmp and longjmp save and restore all the registers needed for coroutines, but the overlay manager needs to keep track of stack contents and assumes there is only one stack. When you implement coroutines there are usually either two stacks or two partitions of one stack, and the overlay manager will not track them properly.</p> <p>You can have background tasks which run with their own stacks or sections of stack, but you must ensure that the background tasks do not invoke any overlaid code, and you must not use the overlay versions of setjmp or longjmp to switch to and from background.</p>
Return value	None.
Portability	longjmp is available on UNIX systems and is defined in ANSI C. This function is not Windows compatible.

See also `ctrlbrk`, `setjmp`, `signal`

Example

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

void subroutine(jmp_buf);

int main(void)
{
    int value;
    jmp_buf jumper;
    value = setjmp(jumper);
    if (value != 0) {
        printf("Longjmp with value %d\n", value);
        exit(value);
    }
    printf("About to call subroutine ... \n");
    subroutine(jumper);
    return 0;
}

void subroutine(jmp_buf jumper) {
    longjmp(jumper, 1);
}
```

Program output

```
About to call subroutine ...
Longjmp with value 1
```

lowvideo

Function Selects low-intensity characters.

Syntax `#include <conio.h>`
`void lowvideo(void);`

Prototype in `conio.h`

Remarks **lowvideo** selects low-intensity characters by clearing the high-intensity bit of the currently selected foreground color.

This function does not affect any characters currently on the screen, only those displayed by functions that perform text mode, direct console output *after* this function is called.

Return value None.

lowvideo

Portability **lowvideo** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **highvideo**, **normvideo**, **textattr**, **textcolor**

Example

```
#include <conio.h>

int main(void)
{
    clrscr();
    highvideo();
    printf("High Intesity Text\r\n");
    lowvideo();
    gotoxy(1,2);
    printf("Low Intensity Text\r\n");
    return 0;
}
```

_lrotl

Function Rotates an **unsigned long** integer value to the left.

Syntax `#include <stdlib.h>`
`unsigned long _lrotl(unsigned long val, int count);`

Prototype in `stdlib.h`

Remarks `_lrotl` rotates the given *val* to the left *count* bits; *val* is an **unsigned long**.

Return value `_lrotl` returns the value of *val* left-rotated *count* bits.

Portability `_lrotl` is unique to DOS.

See also `_lrotr`, `_rotl`, `_rotr`

Example

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

int main(void)
{
    unsigned long result;
    unsigned long value = 100;
    result = _lrotl(value,1);
    printf("The value %lu rotated left one bit is: %lu\n", value, result);
    return 0;
}
```

_lrotr

Function Rotates an **unsigned long** integer value to the right.

Syntax `#include <stdlib.h>`
`unsigned long _lrotr(unsigned long val, int count);`

Prototype in `stdlib.h`

Remarks `_lrotr` rotates the given *val* to the right *count* bits; *val* is an **unsigned long**.

Return value `_lrotr` returns the value of *val* right-rotated *count* bits.

Portability `_lrotr` is unique to DOS.

See also `_lrotl`, `_rotl`, `_rotr`

Example

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

int main(void)
{
    unsigned long result;
    unsigned long value = 100;
    result = _lrotr(value,1);
    printf("The value %lu rotated right one bit is: %lu\n", value, result);
    return 0;
}
```

K-M

lsearch

Function Performs a linear search.

Syntax `#include <stdlib.h>`
`void *lsearch(const void *key, void *base, size_t *num, size_t width,`
`int (*fcmp)(const void *, const void *));`

Prototype in `stdlib.h`

Remarks `lsearch` searches a table for information. Because this is a linear search, the table entries do not need to be sorted before a call to `lsearch`. If the item that *key* points to is not in the table, `lsearch` appends that item to the table.

■ *base* points to the base (0th element) of the search table.

■ *num* points to an integer containing the number of entries in the table.

- *width* contains the number of bytes in each entry.
- *key* points to the item to be searched for (the *search key*).

The argument *fcmp* points to a user-written comparison routine, which compares two items and returns a value based on the comparison.

To search the table, **lsearch** makes repeated calls to the routine whose address is passed in *fcmp*.

On each call to the comparison routine, **lsearch** passes two arguments: *key*, a pointer to the item being searched for, and *elem*, a pointer to the element of *base* being compared.

fcmp is free to interpret the search key and the table entries in any way.

Return value **lsearch** returns the address of the first entry in the table that matches the search key.

If the search key is not identical to **elem*, *fcmp* returns a nonzero integer. If the search key is identical to **elem*, *fcmp* returns 0.

Portability **lsearch** is available on UNIX systems.

See also **bsearch**, **lfind**, **qsort**

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>          /* for strcmp declaration */

/* initialize number of colors */
char *colors[10] = { "Red", "Blue", "Green" };
int ncolors = 3;

int colorscmp(char **arg1, char **arg2) {
    return(strcmp(*arg1, *arg2));
}

int addelem(char *key) {
    int oldn = ncolors;
    lsearch(key, colors, (size_t *) &ncolors, sizeof(char *), (int(*)
        (const void *, const void *)) colorscmp);
    return(ncolors == oldn);
}

int main(void)
{
    int i;
    char *key = "Purple";
    if (addelem(key))
        printf("%s already in colors table\n", key);
    else {
        strcpy(colors[ncolors-1], key);
    }
}
```

```

        printf("%s added to colors table\n", key);
    }
    printf("The colors:\n");
    for (i = 0; i < ncolors; i++)
        printf("%s\n", colors[i]);
    return 0;
}

```

lseek

Function Moves file pointer.

Syntax #include <io.h>
 long lseek(int *handle*, long *offset*, int *fromwhere*);

Prototype in io.h

Remarks **lseek** sets the file pointer associated with *handle* to a new position *offset* bytes beyond the file location given by *fromwhere*. It is a good idea to set *fromwhere* using one of three symbolic constants (defined in io.h) instead of a specific number. The constants are

<i>fromwhere</i>		File location
SEEK_SET	(0)	File beginning
SEEK_CUR	(1)	Current file pointer position
SEEK_END	(2)	End-of-file

Return value **lseek** returns the offset of the pointer's new position measured in bytes from the file beginning. **lseek** returns $-1L$ on error, and the global variable *errno* is set to one of the following:

EBADF	Bad file number
EINVAL	Invalid argument

On devices incapable of seeking (such as terminals and printers), the return value is undefined.

Portability **lseek** is available on all UNIX systems.

See also **filelength**, **fseek**, **ftell**, **getc**, **open**, **sopen**, **ungetc**, **_write**, **write**

Example

```

#include <sys/stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)

```

K-M

lseek

```
{
    int handle;
    char msg[] = "This is a test";
    char ch;

    /* create a file */
    handle = open("TEST.$$$", O_CREAT | O_RDWR, S_IRREAD | S_IWRITE);

    /* write some data to the file */
    write(handle, msg, strlen(msg));

    /* seek to the beginning of the file */
    lseek(handle, 0L, SEEK_SET);

    /* reads chars from the file until EOF */
    do {
        read(handle, &ch, 1);
        printf("%c", ch);
    }
    while (!eof(handle));
    close(handle);

    return 0;
}
```

ltoa

- Function** Converts a **long** to a string.
- Syntax** `#include <stdlib.h>`
`char *ltoa(long value, char *string, int radix);`
- Prototype in** `stdlib.h`
- Remarks** **ltoa** converts *value* to a null-terminated string and stores the result in *string*. *value* is a long integer.
- radix* specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. If *value* is negative and *radix* is 10, the first character of *string* is the minus sign (-).
- ➡ The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (\0). **ltoa** can return up to 33 bytes.
- Return value** **ltoa** returns a pointer to *string*.
- Portability** **ltoa** is unique to DOS.
- See also** `itoa`, `ultoa`

```

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

           int main(void)
           {
               char string[25];
               long value = 123456789L;
               ltoa(value, string, 10);
               printf("number = %ld string = %s\n", value, string);
               return 0;
           }

```

malloc

Function Allocates main memory.

Syntax #include <stdlib.h> or #include <alloc.h>
void *malloc(size_t size);

Prototype in alloc.h, stdlib.h

Remarks **malloc** allocates a block of *size bytes* from the memory heap. It allows a program to allocate memory explicitly as it's needed, and in the exact amounts needed.

The heap is used for dynamic allocation of variable-sized blocks of memory. Many data structures, such as trees and lists, naturally employ heap memory allocation.

All the space between the end of the data segment and the top of the program stack is available for use in the small data models, except for a small margin immediately before the top of the stack. This margin is intended to allow the application some room to make the stack larger, in addition to a small amount needed by DOS.

In the large data models, all the space beyond the program stack to the end of available memory is available for the heap.

Return value On success, **malloc** returns a pointer to the newly allocated block of memory. If not enough space exists for the new block, it returns null. The contents of the block are left unchanged. If the argument *size* == 0, **malloc** returns null.

Portability **malloc** is available on UNIX systems and is defined in ANSI C.

See also **allocmem, calloc, coreleft, farcalloc, farmalloc, free, realloc**

Example

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

int main(void)
{
    char *str;

    /* allocate memory for string */
    if ((str = malloc(10)) == NULL) {
        printf("Not enough memory to allocate buffer\n");
        exit(1); /* terminate program if out of memory */
    }

    /* copy "Hello" into string */
    strcpy(str, "Hello");

    /* display string */
    printf("String is %s\n", str);

    /* free memory */
    free(str);
    return 0;
}
```

matherr

Function User-modifiable math error handler.

Syntax #include <math.h>
int matherr(struct exception *e);

Prototype in math.h

Remarks **matherr** is called when an error is generated by the math library.

matherr serves as a user hook (a function that can be customized by the user) that you can replace by writing your own math error handling routine—see the following example of a user-defined **matherr** implementation.

matherr is useful for trapping domain and range errors caused by the math functions. It does not trap floating-point exceptions, such as division by zero. See **signal** for trapping such errors.

You can define your own **matherr** routine to be a custom error handler (such as one that catches and resolves certain types of errors); this customized function overrides the default version in the C library. The

customized **matherr** should return 0 if it fails to resolve the error, or nonzero if the error is resolved. When **matherr** returns nonzero, no error message is printed and the global variable *errno* is not changed.

This is the **exception** structure (defined in math.h):

```
struct exception {
    int    type;
    char  *Function;
    double arg1, arg2, retval;
};
```

The members of the **exception** structure are shown in the following table:

Member	What it is (or represents)
<i>type</i>	The type of mathematical error that occurred; an enum type defined in the typedef <i>_mexcep</i> (see definition after this list).
<i>name</i>	A pointer to a null-terminated string holding the <i>name</i> of the math library function that resulted in an error.
<i>arg1</i> , <i>arg2</i>	The arguments (passed to the function <i>name</i> points to) that caused the error; if only one argument was passed to the function, it is stored in <i>arg1</i> .
<i>retval</i>	The default return value for matherr ; you can modify this value.

The **typedef** *_mexcep*, also defined in math.h, enumerates the following symbolic constants representing possible mathematical errors:

Symbolic constant	Mathematical error
DOMAIN	Argument was not in domain of function, such as log (-1).
SING	Argument would result in a singularity, such as pow (0, -2).
OVERFLOW	Argument would produce a function result greater than MAXDOUBLE, such as exp (1000).
UNDERFLOW	Argument would produce a function result less than MINDOUBLE, such as exp (-1000).
TLOSS	Argument would produce function result with total loss of significant digits, such as sin (10e70).

The symbolic constants MAXDOUBLE and MINDOUBLE are defined in values.h.

The source code to the default **matherr** is on the Borland C++ distribution disks.

The UNIX-style **matherr** default behavior (printing a message and terminating) is not ANSI compatible. If you desire a UNIX-style version of **matherr**, use MATHERR.C provided on the Borland C++ distribution disks.

Return value The default return value for **matherr** is 1 if the error is UNDERFLOW or TLOSS, 0 otherwise. **matherr** can also modify *e* → *retval*, which propagates back to the original caller.

When **matherr** returns 0 (indicating that it was not able to resolve the error), the global variable *errno* is set to 0 and an error message is printed.

When **matherr** returns nonzero (indicating that it was able to resolve the error), the global variable *errno* is not set and no messages are printed.

Portability **matherr** is available on many C compilers, but it is not compatible with ANSI C. A UNIX-style **matherr** that prints a message and terminates is provided in MATHERR.C on the Borland C++ distribution disks. **matherr** might not be supported in future versions of Borland C++.

Example

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

int matherr(struct exception *a)
{
    if (a->type == DOMAIN) {
        if (!strcmp(a->name,"sqrt")) {
            a->retval = sqrt(-(a->arg1));
            return 1;
        }
    }
    return 0;
}

int main(void)
{
    double x, y;
    x = -2.0;
    y = sqrt(x);
    printf("Matherr corrected value: %lf\n",y);
    return 0;
}
```

max

Function Returns the larger of two values.

Syntax `#include <stdlib.h>`
`(type) max(a, b);`

Prototype in `stdlib.h`

Remarks This macro compares two values and returns the larger of the two. Both arguments and the function declaration must be of the same type.

Return value **max** returns the larger of two values.

Portability **max** is unique to DOS.

See also **min**

Example

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

int main(void)
{
    int x = 5;
    int y = 6;
    int z;
    z = max(x, y);
    printf("The larger number is %d\n", z);
    return 0;
}
```

Program output

The larger number is 6

memccpy, _fmemccpy

Function Copies a block of *n* bytes.

Syntax `#include <mem.h>`
Near version: `void *memccpy(void *dest, const void *src, int c, size_t n);`
Far version: `void far * far _fmemccpy(void far *dest, const void far *src,`
`int c, size_t n)`

Prototype in `string.h`, `mem.h`

memccpy, _fmemccpy

- Remarks** **memccpy** copies a block of *n* bytes from *src* to *dest*. The copying stops as soon as either of the following occurs:
- The character *c* is first copied into *dest*.
 - *n* bytes have been copied into *dest*.
- Return value** **memccpy** returns a pointer to the byte in *dest* immediately following *c*, if *c* was copied; otherwise, **memccpy** returns null.
- Portability** **memccpy** is available on UNIX System V systems.
- See also** **memcpy**, **memmove**, **memset**
- Example**
- ```
#include <string.h>
#include <stdio.h>

int main(void)
{
 char *src = "This is the source string";
 char dest[50];
 char *ptr;
 ptr = memccpy(dest, src, 'c', strlen(src));
 if (ptr) {
 *ptr = '\0';
 printf("The character was found: %s\n", dest);
 }
 else
 printf("The character wasn't found\n");
 return 0;
}
```

## memchr, \_fmemchr

---

- Function** Searches *n* bytes for character *c*.
- Syntax** `#include <mem.h>`  
*Near version:* `void *memchr(const void *s, int c, size_t n);`  
*Far version:* `void far * _fmemchr(const void far *s, int c, size_t n);`
- Prototype in** `string.h`, `mem.h`
- Remarks** **memchr** searches the first *n* bytes of the block pointed to by *s* for character *c*.
- Return value** On success, **memchr** returns a pointer to the first occurrence of *c* in *s*; otherwise, it returns null.

**Portability** `memchr` is available on UNIX System V systems and is compatible with ANSI C.

**Example**

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

int main(void)
{
 char str[17];
 char *ptr;
 strcpy(str, "This is a string");
 ptr = memchr(str, 'r', strlen(str));
 if (ptr)
 printf("The character 'r' is at position: %d\n", ptr - str);
 else
 printf("The character was not found\n");
 return 0;
}
```

## memcmp, \_fmemcmp

K-M

**Function** Compares two blocks for a length of exactly  $n$  bytes.

**Syntax** `#include <mem.h>`  
*Near version:* `int memcmp(const void *s1, const void *s2, size_t n);`  
*Far version:* `int far _fmemcmp(const void far *s1, const void far *s2, size_t n)`

**Prototype in** `string.h`, `mem.h`

**Remarks** `memcmp` compares the first  $n$  bytes of the blocks  $s1$  and  $s2$  as **unsigned chars**.

**Return value** Since it compares bytes as **unsigned chars**, `memcmp` returns a value

- < 0 if  $s1$  is less than  $s2$
- = 0 if  $s1$  is the same as  $s2$
- > 0 if  $s1$  is greater than  $s2$

For example,

```
memcmp("\xFF", "\x7F", 1)
```

returns a value greater than 0.

**Portability** `memcmp` is available on UNIX System V systems and is compatible with ANSI C.

**See also** `memicmp`

## memcmp, \_fmemcmp

**Example**

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

int main(void)
{
 char *buf1 = "aaa";
 char *buf2 = "bbb";
 char *buf3 = "ccc";
 int stat;
 stat = memcmp(buf2, buf1, strlen(buf2));
 if (stat > 0)
 printf("buffer 2 is greater than buffer 1\n");
 else
 printf("buffer 2 is less than buffer 1\n");
 stat = memcmp(buf2, buf3, strlen(buf2));
 if (stat > 0)
 printf("buffer 2 is greater than buffer 3\n");
 else
 printf("buffer 2 is less than buffer 3\n");
 return 0;
}
```

## memcpy, \_fmemcpy

---

- Function** Copies a block of *n* bytes.
- Syntax** `#include <mem.h>`  
*Near version:* `void *memcpy(void *dest, const void *src, size_t n);`  
*Far version:* `void far *far _fmemcpy(void far *dest, const void far *src, size_t n);`
- Prototype in** string.h, mem.h
- Remarks** **memcpy** copies a block of *n* bytes from *src* to *dest*. If *src* and *dest* overlap, the behavior of **memcpy** is undefined.
- Return value** **memcpy** returns *dest*.
- Portability** **memcpy** is available on UNIX System V systems and is compatible with ANSI C.
- See also** **memccpy**, **memmove**, **memset**, **movedata**, **movmem**
- Example**
- ```
#include <stdio.h>
#include <string.h>

int main(void)
{
```

```

char src[] = "*****";
char dest[] = "abcdefghijklmnopqrstuvwxy0123456709";
char *ptr;
printf("destination before memcpy: %s\n", dest);
ptr = memcpy(dest, src, strlen(src));
if (ptr)
    printf("destination after memcpy: %s\n", dest);
else
    printf("memcpy failed\n");
return 0;
}

```

memicmp, _fmemicmp

Function Compares n bytes of two character arrays, ignoring case.

Syntax #include <mem.h>

Near version: int memicmp(const void *s1, const void *s2, size_t n);

Far version: int far _fmemicmp(const void far *s1, const void far *s2, size_t n)

K-M

Prototype in string.h, mem.h

Remarks **memicmp** compares the first n bytes of the blocks $s1$ and $s2$, ignoring character case (upper or lower).

Return value **memicmp** returns a value

- < 0 if $s1$ is less than $s2$
- = 0 if $s1$ is the same as $s2$
- > 0 if $s1$ is greater than $s2$

Portability **memicmp** is available on UNIX System V systems.

See also **memcmp**

Example

```

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

int main(void)
{
    char *buf1 = "ABCDE123";
    char *buf2 = "abcde456";
    int stat;
    stat = memicmp(buf1, buf2, 5);
    printf("The strings to position 5 are ");
    if (stat)
        printf("not ");
}

```

memicmp, _fmemicmp

```
    printf("the same\n");  
    return 0;  
}
```

memmove

Function	Copies a block of n bytes.
Syntax	<code>#include <mem.h></code> <code>void *memmove(void *dest, const void *src, size_t n);</code>
Prototype in	string.h, mem.h
Remarks	memmove copies a block of n bytes from <i>src</i> to <i>dest</i> . Even when the source and destination blocks overlap, bytes in the overlapping locations are copied correctly.
Return value	memmove returns <i>dest</i> .
Portability	memmove is available on UNIX System V systems and is compatible with ANSI C.
See also	memcpy, memcopy, movmem
Example	<pre>#include <string.h> #include <stdio.h> int main(void) { char *dest = "abcdefghijklmnopqrstuvwxyz0123456789"; char *src = "*****"; printf("destination prior to memmove: %s\n", dest); memmove(dest, src, 26); printf("destination after memmove: %s\n", dest); return 0; }</pre>

memset, _fmemset

Function	Sets n bytes of a block of memory to byte c .
Syntax	<code>#include <mem.h></code> <i>Near version:</i> <code>void *memset(void *s, int c, size_t n);</code> <i>Far version:</i> <code>void far * _fmemset (void far *s, int c, size_t n)</code>
Prototype in	string.h, mem.h

- Remarks** `memset` sets the first n bytes of the array s to the character c .
- Return value** `memset` returns s .
- Portability** `memset` is available on UNIX System V systems and is compatible with ANSI C.
- See also** `memccpy`, `memcpy`, `setmem`

Example

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

int main(void)
{
    char buffer[] = "Hello world\n";
    printf("Buffer before memset: %s\n", buffer);
    memset(buffer, '*', strlen(buffer) - 1);
    printf("Buffer after memset: %s\n", buffer);
    return 0;
}
```

K-M

min

- Function** Returns the smaller of two values.
- Syntax** `#include <stdlib.h>`
`(type) min(a , b);`
- Prototype in** `stdlib.h`
- Remarks** `min` compares two values and returns the smaller of the two. Both arguments and the function declaration must be of the same type.
- Return value** `min` returns the smaller of two values.
- Portability** `min` is unique to DOS.
- See also** `max`

Example

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

int main()
{
    int x = 5;
    int y = 6;
    printf("The smaller number is %d\n", min(x,y));
}
```



```

    return 0;
}

```

Program output

The smaller number is 5

mkdir

Function Creates a directory.

Syntax `#include <dir.h>`
`int mkdir(const char *path);`

Prototype in `dir.h`

Remarks `mkdir` creates a new directory from the given path name *path*.

Return value `mkdir` returns the value 0 if the new directory was created.

A return value of `-1` indicates an error, and the global variable `errno` is set to one of the following values:

<code>EACCES</code>	Permission denied
<code>ENOENT</code>	No such file or directory

See also `chdir`, `getcurdir`, `getcwd`, `rmdir`

Example

```

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>

int main(void)
{
    int status;
    clrscr();
    status = mkdir("asdfjklm");
    (!status) ? (printf("Directory created\n")) :
               (printf("Unable to create directory\n"));

    getch();
    system("dir");
    getch();
    status = rmdir("asdfjklm");
    (!status) ? (printf("Directory deleted\n")) :
               (perror("Unable to delete directory"));

    return 0;
}

```

MK_FP

- Function** Makes a far pointer.
- Syntax** `#include <dos.h>`
`void far * MK_FP(unsigned seg, unsigned ofs);`
- Prototype in** `dos.h`
- Remarks** **MK_FP** is a macro that makes a far pointer from its component segment (*seg*) and offset (*ofs*) parts.
- Return value** **MK_FP** returns a far pointer.
- Portability** **MK_FP** is unique to Borland C++.
- See also** **FP_OFF, FP_SEG, movedata, segread**
- Example**
- ```
#include <dos.h>
#include <graphics.h>

int main(void)
{
 int gd, gm, i;
 unsigned int far *screen;
 detectgraph(&gd, &gm);
 if (gd == HERCMONO)
 screen = MK_FP(0xB000, 0);
 else
 screen = MK_FP(0xB800, 0);
 for (i = 0; i < 26; i++)
 screen[i] = 0x0700 + ('a' + i);
 return 0;
}
```

K-M

## mktemp

---

- Function** Makes a unique file name.
- Syntax** `#include <dir.h>`  
`char *mktemp(char *template);`
- Prototype in** `dir.h`
- Remarks** **mktemp** replaces the string pointed to by *template* with a unique file name and returns *template*.

## mktemp

*template* should be a null-terminated string with six trailing Xs. These Xs are replaced with a unique collection of letters plus a period, so that there are two letters, a period, and three suffix letters in the new file name.

Starting with AA.AAA, the new file name is assigned by looking up the name on the disk and avoiding pre-existing names of the same format.

**Return value** If *template* is well-formed, **mktemp** returns the address of the *template* string. Otherwise, it returns null.

**Portability** **mktemp** is available on UNIX systems.

**Example**

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

int main(void)
{
 /* fname defines template for temporary file */
 char *fname = "TXXXXXX", *ptr;
 ptr = mktemp(fname);
 printf("%s\n", ptr);
 return 0;
}
```

## mktime

---

**Function** Converts time to calendar format.

**Syntax** `#include <time.h>`  
`time_t mktime(struct tm *t);`

**Prototype in** time.h

**Remarks** Converts the time in the structure pointed to by *t* into a calendar time with the same format used by the **time** function. The original values of the fields *tm\_sec*, *tm\_min*, *tm\_hour*, *tm\_mday*, and *tm\_mon* are not restricted to the ranges described in the *tm* structure. If the fields are not in their proper ranges, they are adjusted. Values for fields *tm\_wday* and *tm\_yday* are computed after the other fields have been adjusted.

**Return value** See Remarks.

**Portability** **mktime** is compatible with ANSI C.

**See also** **localtime**, **strftime**, **time**

**Example**

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

```

char *wday[] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
 "Saturday", "Unknown"};

int main(void)
{
 struct tm time_check;
 int year, month, day;

 /* input year, month, and day to find the weekday for */
 printf("Year: ");
 scanf("%d", &year);
 printf("Month: ");
 scanf("%d", &month);
 printf("Day: ");
 scanf("%d", &day);

 /* load the time_check structure with the data */
 time_check.tm_year = year - 1900;
 time_check.tm_mon = month - 1;
 time_check.tm_mday = day;
 time_check.tm_hour = 0;
 time_check.tm_min = 0;
 time_check.tm_sec = 1;
 time_check.tm_isdst = -1;

 /* call mktime to fill in the structure's weekday field */
 if (mktime(&time_check) == -1)
 time_check.tm_wday = 7;

 /* print out the day of the week */
 printf("That day is a %s\n", wday[time_check.tm_wday]);
 return 0;
}

```

## modf

---

|                     |                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Splits a <b>double</b> into integer and fractional parts.                                                                                            |
| <b>Syntax</b>       | #include <math.h><br>double modf(double <i>x</i> , double * <i>ipart</i> );                                                                          |
| <b>Prototype in</b> | math.h                                                                                                                                               |
| <b>Remarks</b>      | <b>modf</b> breaks the double <i>x</i> into two parts: the integer and the fraction. It stores the integer in <i>ipart</i> and returns the fraction. |
| <b>Return value</b> | <b>modf</b> returns the fractional part of <i>x</i> .                                                                                                |
| <b>Portability</b>  | <b>modf</b> is unique to DOS.                                                                                                                        |

## modf

**See also** [fmod](#), [ldexp](#)

**Example**

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

int main(void)
{
 double fraction, integer;
 double number = 100000.567;
 fraction = modf(number, &integer);
 printf("The whole and fractional parts of %lf are %lf and %lf\n", number,
 integer, fraction);
 return 0;
}
```

## movedata

---

**Function** Copies *n* bytes.

**Syntax** `#include <mem.h>`  
`void movedata(unsigned srcseg, unsigned srcoff, unsigned dstseg,  
 unsigned dstoff, size_t n);`

**Prototype in** `mem.h`, `string.h`

**Remarks** `movedata` copies *n* bytes from the source address (*srcseg:srcoff*) to the destination address (*dstseg:dstoff*).

`movedata` is a means of moving blocks of data that is independent of memory model.

**Return value** None.

**Portability** `movedata` is unique to DOS.

**See also** [FP\\_OFF](#), [memcpy](#), [MK\\_FP](#), [movmem](#), [segread](#)

**Example**

```
#include <mem.h>

#define COLOR_BASE 0xB800
#define BUFFER_SIZE 80*25*2

char buf[BUFFER_SIZE];

/* saves the contents of the color screen in buffer */
void save_color_screen(char *buffer)
{
 movedata(COLOR_BASE, 0, _DS, (unsigned)buffer, BUFFER_SIZE);
}
```

```
int main()
{
 save_color_screen(buf);
 return 0;
}
```

## moverel

---

- Function** Moves the current position (CP) a relative distance.
- Syntax** `#include <graphics.h>`  
`void far moverel(int dx, int dy);`
- Prototype in** graphics.h
- Remarks** **moverel** moves the current position (CP) *dx* pixels in the *x* direction and *dy* pixels in the *y* direction.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **moveto**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 char msg[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }
}
```

K-M

```

/* move the CP to location (20,30) */
moveto(20,30);

/* plot a pixel at the CP */
putpixel(getx(), gety(), getmaxcolor());

/* create and output a message at (20,30) */
sprintf(msg, "%d, %d", getx(), gety());
outtextxy(20,30, msg);

/* move to a point a relative distance away from the current CP */
moverel(100, 100);

/* plot a pixel at the CP */
putpixel(getx(), gety(), getmaxcolor());

/* create and output a message at CP */
sprintf(msg, "%d, %d", getx(), gety());
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}

```

## movetext

---

- Function** Copies text onscreen from one rectangle to another.
- Syntax** `#include <conio.h>`  
`int movetext(int left, int top, int right, int bottom, int destleft, int desttop);`
- Prototype in** conio.h
- Remarks** **movetext** copies the contents of the onscreen rectangle defined by *left*, *top*, *right*, and *bottom* to a new rectangle of the same dimensions. The new rectangle's upper left corner is position (*destleft*, *desttop*).
- All coordinates are absolute screen coordinates. Rectangles that overlap are moved correctly.
- movetext** is a text mode function performing direct video output.
- Return value** **movetext** returns nonzero if the operation succeeded. If the operation failed (for example, if you gave coordinates outside the range of the current screen mode), **movetext** returns 0.
- Portability** **movetext** can be used on IBM PCs and compatible systems. This function is not Windows compatible.

**See also** **gettext, puttext**

**Example**

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

int main(void)
{
 char *str = "This is a test string";
 clrscr();
 cputs(str);
 getch();
 movetext(1, 1, strlen(str), 2, 10, 10);
 getch();
 return 0;
}
```

## moveto

---

K-M

**Function** Moves the current position (CP) to  $(x,y)$ .

**Syntax** `#include <graphics.h>`  
`void far moveto(int x, int y);`

**Prototype in** `graphics.h`

**Remarks** **moveto** moves the current position (CP) to viewport position  $(x,y)$ .

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **moverel**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 char msg[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");
```



## moveto

```
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

/* move the CP to location (20,30) */
moveto(20,30);

/* plot a pixel at the CP */
putpixel(getx(), gety(), getmaxcolor());

/* create and output a message at (20,30) */
sprintf(msg, " (%d, %d)", getx(), gety());
outtextxy(20,30, msg);

/* move to (100,100) */
moveto(100,100);

/* plot a pixel at the CP */
putpixel(getx(), gety(), getmaxcolor());

/* create and output a message at CP */
sprintf(msg, " (%d, %d)", getx(), gety());
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}
```

## movmem

---

|                     |                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Moves a block of <i>length</i> bytes.                                                                                                                                                                           |
| <b>Syntax</b>       | <code>#include &lt;mem.h&gt;</code><br><code>void movmem(void *src, void *dest, unsigned length);</code>                                                                                                        |
| <b>Prototype in</b> | mem.h                                                                                                                                                                                                           |
| <b>Remarks</b>      | <b>movmem</b> moves a block of <i>length</i> bytes from <i>src</i> to <i>dest</i> . Even if the source and destination blocks overlap, the move direction is chosen so that the data is always moved correctly. |
| <b>Return value</b> | None.                                                                                                                                                                                                           |

**Portability** `movmem` is unique to Borland C++.

**See also** `memcpy`, `memmove`, `movedata`

**Example**

```
#include <mem.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
 char *source = "Borland International";
 char *destination;
 int length;
 length = strlen(source);
 destination = malloc(length + 1);
 movmem(source, destination, length);
 printf("%s\n", destination);
 return 0;
}
```

K-M

## norm

---

**Function** Returns the square of the absolute value.

**Syntax** `#include <complex.h>`  
`double norm(complex x);`

**Prototype in** `complex.h`

**Remarks** `norm` can overflow if either the real or imaginary part is sufficiently large.

**Return value** `norm(x)` returns the magnitude  $\text{real}(x) * \text{real}(x) + \text{imag}(x) * \text{imag}(x)$ .

**Portability** Complex functions require C++ and are not portable.

**See also** `arg`, `complex`, `polar`

**Example**

```
#include <complex.h>

int main(void)
{
 double x = 3.1, y = 4.2;
 complex z = complex(x,y);
 cout << "z = " << z << "\n";
 cout << " has real part = " << real(z) << "\n";
 cout << " and imaginary real part = " << imag(z) << "\n";
 cout << "z has complex conjugate = " << conj(z) << "\n";
 double mag = sqrt(norm(z));
}
```

## norm

```
double ang = arg(z);
cout << "The polar form of z is:\n";
cout << " magnitude = " << mag << "\n";
cout << " angle (in radians) = " << ang << "\n";
cout << "Reconstructing z from its polar form gives:\n";
cout << " z = " << polar(mag,ang) << "\n";
return 0;
}
```

## normvideo

---

**Function** Selects normal-intensity characters.

**Syntax** `#include <conio.h>`  
`void normvideo(void);`

**Prototype in** conio.h

**Remarks** **normvideo** selects normal characters by returning the text attribute (foreground and background) to the value it had when the program started.

This function does not affect any characters currently on the screen, only those displayed by functions (such as **cprintf**) performing direct console output functions *after* **normvideo** is called.

**Return value** None.

**Portability** **normvideo** works with IBM PCs and compatibles only; a corresponding function exists in Turbo Pascal. This function is not Windows compatible.

**See also** **highvideo**, **lowvideo**, **textattr**, **textcolor**

**Example**

```
#include <conio.h>

int main(void)
{
 clrscr();
 lowvideo();
 cprintf("LOW Intensity Text\r\n");
 highvideo();
 cprintf("HIGH Intensity Text\r\n");
 normvideo();
 cprintf("NORMAL Intensity Text\r\n");
 return 0;
}
```

## nosound

---

**Function** Turns PC speaker off.

**Syntax** `#include <dos.h>`  
`void nosound(void);`

**Prototype in** `dos.h`

**Remarks** Turns the speaker off after it has been turned on by a call to **sound**.

**Return value** None.

**Portability** **nosound** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

**See also** **delay, sound**

**Example** `#include <dos.h>`

```
/* Emits a 7-Hz tone for 5 seconds.
 True story: 7 Hz is the resonant frequency of a chicken's skull
 cavity. This was determined empirically in Australia, where a new
 factory generating 7-Hz tones was located too close to a chicken
 ranch. When the factory started up, all the chickens died. Your PC
 may not be able to emit a 7-Hz tone. */

int main()
{
 sound(7);
 delay(5000);
 nosound();
 return 0;
}
```

N-P

## \_open

---

**Function** Opens a file for reading or writing.

**Syntax** `#include <fcntl.h>`  
`int _open(const char *filename, int oflags);`

**Prototype in** `io.h`

## `_open`

**Remarks** `_open` opens the file specified by *filename*, then prepares it for reading or writing, as determined by the value of *oflags*. The file is always opened in binary mode.

For `_open`, the value of *oflags* in DOS 2.x is limited to `O_RDONLY`, `O_WRONLY`, and `O_RDWR`. For DOS 3.x, the following additional values can also be used:

*These symbolic constants are defined in `fcntl.h`.*

- `O_NOINHERIT` is included if the file is not passed to child programs.
- `O_DENYALL` allows only the current handle to have access to the file.
- `O_DENYWRITE` allows only reads from any other open to the file.
- `O_DENYREAD` allows only writes from any other open to the file.
- `O_DENYNONE` allows other shared opens to the file.

Only one of the `O_DENYxxx` values can be included in a single `_open` under DOS 3.x. These file-sharing attributes are in addition to any locking performed on the files.

The maximum number of simultaneously open files is defined by `HANDLE_MAX`.

**Return value** On successful completion, `_open` returns a nonnegative integer (the file handle). The file pointer, which marks the current position in the file, is set to the beginning of the file. On error, `_open` returns `-1` and the global variable *errno* is set to one of the following:

|                      |                        |
|----------------------|------------------------|
| <code>ENOENT</code>  | Path or file not found |
| <code>EMFILE</code>  | Too many open files    |
| <code>EACCES</code>  | Permission denied      |
| <code>EINVACC</code> | Invalid access code    |

**Portability** `_open` is unique to DOS.

**See also** `open`, `_read`, `sopen`

**Example**

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
 int handle;
 char msg[] = "Hello world\n";
 if ((handle = _open("TEST. $$$", O_RDWR)) == -1) {
 perror("Error:");
 return 1;
 }
}
```

```

 _write(handle, msg, strlen(msg));
 _close(handle);
 return 0;
}

```

# open

**Function** Opens a file for reading or writing.

**Syntax** `#include <fcntl.h>`  
`#include <sys/stat.h>`  
`int open(const char *path, int access [, unsigned mode]);`

**Prototype in** io.h

**Remarks** **open** opens the file specified by *path*, then prepares it for reading and/or writing as determined by the value of *access*.

To create a file in a particular mode, you can either assign to the global variable *\_fmode* or call **open** with the `O_CREAT` and `O_TRUNC` options ORed with the translation mode desired. For example, the call

```
open("xmp", O_CREAT|O_TRUNC|O_BINARY, S_IREAD)
```

will create a binary-mode, read-only file named XMP, truncating its length to 0 bytes if it already existed.

For **open**, *access* is constructed by bitwise ORing flags from the following two lists. Only one flag from the first list can be used (and one *must* be used); the remaining flags can be used in any logical combination.

**List 1: Read/write flags**

*These symbolic constants are defined in fcntl.h.*

- `O_RDONLY` Open for reading only.
- `O_WRONLY` Open for writing only.
- `O_RDWR` Open for reading and writing.

**List 2: Other access flags**

- `O_NDELAY` Not used; for UNIX compatibility.
- `O_APPEND` If set, the file pointer will be set to the end of the file prior to each write.
- `O_CREAT` If the file exists, this flag has no effect. If the file does not exist, the file is created, and the bits of *mode* are used to set the file attribute bits as in **chmod**.
- `O_TRUNC` If the file exists, its length is truncated to 0. The file attributes remain unchanged.



## open

|          |                                                                           |
|----------|---------------------------------------------------------------------------|
| O_EXCL   | Used only with O_CREAT. If the file already exists, an error is returned. |
| O_BINARY | This flag can be given to explicitly open the file in binary mode.        |
| O_TEXT   | This flag can be given to explicitly open the file in text mode.          |

If neither O\_BINARY nor O\_TEXT is given, the file is opened in the translation mode set by the global variable *\_fmode*.

If the O\_CREAT flag is used in constructing *access*, you need to supply the *mode* argument to **open** from the following symbolic constants defined in `sys\stat.h`.

| Value of <i>mode</i> | Access permission            |
|----------------------|------------------------------|
| S_IWRITE             | Permission to write          |
| S_IREAD              | Permission to read           |
| S_IREAD S_IWRITE     | Permission to read and write |

**Return value** On successful completion, **open** returns a nonnegative integer (the file handle). The file pointer, which marks the current position in the file, is set to the beginning of the file. On error, **open** returns `-1` and the global variable *errno* is set to one of the following:

|         |                           |
|---------|---------------------------|
| ENOENT  | No such file or directory |
| EMFILE  | Too many open files       |
| EACCES  | Permission denied         |
| EINVACC | Invalid access code       |

**Portability** **open** is available on UNIX systems. On UNIX version 7, the *O\_type* mnemonics are not defined. UNIX System III uses all of the *O\_type* mnemonics except O\_BINARY and O\_TEXT.

**See also** **chmod**, **chsize**, **close**, **\_creat**, **creat**, **creatnew**, **creattemp**, **dup**, **dup2**, **fdopen**, **filelength**, **fopen**, **freopen**, **getftime**, **lseek**, **lock**, **\_open**, **read**, **sopen**, **\_write**, **write**

**Example**

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
 int handle;
 char msg[] = "Hello world";
 if ((handle = open("TEST.$$$", O_CREAT | O_TEXT)) == -1) {
 perror("Error:");
 }
}
```

```

 return 1;
 }
 write(handle, msg, strlen(msg));
 close(handle);
 return 0;
}

```

## outport

---

**Function** Outputs a word to a hardware port.

**Syntax** `#include <dos.h>`  
`void outport(int portid, int value);`

**Prototype in** dos.h

**Remarks** **outport** works just like the 80x86 instruction **out**. It writes the low byte of the word given by *value* to the output port specified by *portid* and writes the high byte of the word to *portid* +1.

**Return value** None.

**Portability** **outport** is unique to the 8086 family.

**See also** **inport**, **inportb**, **outportb**

**Example**

```

#include <stdio.h>
#include <dos.h>

int main(void)
{
 int value = 64;
 int port = 0;
 outportb(port, value);
 printf("Value %d sent to port number %d\n", value, port);
 return 0;
}

```



## outportb

---

**Function** Outputs a byte to a hardware port.

**Syntax** `#include <dos.h>`  
`void outportb(int portid, unsigned char value);`

**Prototype in** dos.h



## outportb

**Remarks** **outportb** is a macro that writes the byte given by *value* to the output port specified by *portid*.

If **outportb** is called when `dos.h` has been included, it will be treated as a macro that expands to inline code. If you don't include `dos.h`, or if you do include `dos.h` and **#undef** the macro **outportb**, you'll get the **outportb** function.

**Return value** None.

**Portability** **outportb** is unique to the 8086 family.

**See also** **inport**, **inportb**, **outport**

**Example**

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

int main(void)
{
 int port = 0;
 char value = 'C';
 outportb(port, value);
 printf("Value %c sent to port number %d\n", value, port);
 return 0;
}
```

## outtext

---

**Function** Displays a string in the viewport.

**Syntax** `#include <graphics.h>`  
`void far outtext(char far *textstring);`

**Prototype in** `graphics.h`

**Remarks** **outtext** displays a text string in the viewport, using the current justification settings and the current font, direction, and size.

**outtext** outputs *textstring* at the current position (CP). If the horizontal text justification is `LEFT_TEXT` and the text direction is `HORIZ_DIR`, the CP's x-coordinate is advanced by **textwidth**(*textstring*). Otherwise, the CP remains unchanged.

To maintain code compatibility when using several fonts, use **textwidth** and **textheight** to determine the dimensions of the string.



If a string is printed with the default font using **outtext**, any part of the string that extends outside the current viewport is truncated.

**outtext** is for use in graphics mode; it will not work in text mode.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **gettextsettings**, **outtextxy**, **settextjustify**, **textheight**, **textwidth**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* move the CP to the center of the screen */
 moveto(midx, midy);

 /* output text starting at the CP */
 outtext("This ");
 outtext("is ");
 outtext("a ");
 outtext("test.");

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```



## outtextxy

---

**Function** Displays a string at a specified location.

**Syntax** `#include <graphics.h>`  
`void far outtextxy(int x, int y, char far *textstring);`

**Prototype in** graphics.h

**Remarks** **outtextxy** displays a text string in the viewport at the given position ( $x, y$ ), using the current justification settings and the current font, direction, and size.

To maintain code compatibility when using several fonts, use **textwidth** and **textheight** to determine the dimensions of the string.

➡ If a string is printed with the default font using **outtext** or **outtextxy**, any part of the string that extends outside the current viewport is truncated.

**outtext** is for use in graphics mode; it will not work in text mode.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **gettextsettings**, **outtext**, **textheight**, **textwidth**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }
}
```

```

 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* output text at center of the screen; CP doesn't get changed */
 outtextxy(midx, midy, "This is a test.");

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

## \_OvrInitEms

---

**Function** Initializes expanded memory swapping for the overlay manager.

**Syntax** `#include <dos.h>`  
`int cdecl far _OvrInitEms(unsigned emsHandle, unsigned firstPage,`  
`unsigned pages);`

**Prototype in** dos.h

**Remarks** **\_OvrInitEms** checks for the presence of expanded memory by looking for an EMS driver and allocating memory from it. If *emsHandle* is zero, the overlay manager allocates EMS pages and uses them for swapping. If *emsHandle* is not zero, then it should be a valid EMS handle; the overlay manager will use it for swapping. In that case, you can specify *firstPage*, where the swapping can start inside that area.

In both cases, a nonzero *pages* parameter gives the limit of the usable pages by the overlay manager.

**Return value** **\_OvrInitEms** returns 0 if the overlay manager is able to use expanded memory for swapping.

**Portability** **\_OvrInitEms** is unique to Borland C++.

**See also** **\_OvrInitExt**, *\_ovrbuffer* (global variable)

**Example**

```

#include <dos.h>

int main(void)
{
 /* ask overlay manager to check for expanded memory and allow it to use 16
 pages (256K) available only in medium, large, and huge memory models */
 _OvrInitEms (0, 0, 16);
}

```

N-P

## `_OvrInitExt`

```
 return 0;
}
```

## `_OvrInitExt`

---

- Function** Initializes extended memory swapping for the overlay manager.
- Syntax** `#include <dos.h>`  
`int cdecl far _OvrInitExt(unsigned long startAddress,  
 unsigned long length);`
- Prototype in** `dos.h`
- Remarks** `_OvrInitExt` checks for the presence of extended memory, using the known methods to detect the presence of other programs using extended memory, and allocates memory from it. If *startAddress* is zero, the overlay manager determines the start address and uses, at most, the size of the overlays. If *startAddress* is not zero, then the overlay manager uses the extended memory above that address.
- In both cases, a nonzero *length* parameter gives the limit of the usable extended memory by the overlay manager.
- Return value** `_OvrInitExt` returns 0 if the overlay manager is able to use extended memory for swapping.
- Portability** `_OvrInitExt` is unique to Borland C++.
- See also** `_OvrInitEms`, `_ovrbuffer` (global variable)
- Example**
- ```
#include <dos.h>

int main(void)
{
    /* use the extended memory from the linear address 0x200000L (2MB), as much as
       necessary */
    _OvrInitExt (0x200000L, 0);
    return 0;
}
```

parsfnm

Function Parses file name.

Syntax `#include <dos.h>`
`char *parsfnm(const char *cmdline, struct fcb *fcb, int opt);`

Prototype in dos.h

Remarks **parsfnm** parses a string pointed to by *cmdline* for a file name. The string is normally a command line. The file name is placed in a file control block (FCB) as a drive, file name, and extension. The FCB is pointed to by *fcb*.

The *opt* parameter is the value documented for AL in the DOS parse system call. See your DOS reference manuals under system call 0x29 for a description of the parsing operations performed on the file name.

Return value On success, **parsfnm** returns a pointer to the next byte after the end of the file name. If there is any error in parsing the file name, **parsfnm** returns null.

Portability **parsfnm** is unique to DOS.

Example

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
    char line[80];
    struct fcb blk;

    /* get file name */
    printf("Enter drive and file name (no path - ie. a:file.dat)\n");
    gets(line);

    /* put file name in fcb */
    if (parsfnm(line, &blk, 1) == NULL)
        printf("Error in parsfm call\n");
    else
        printf("Drive #%d Name: %11s\n", blk.fcb_drive, blk.fcb_name);
    return 0;
}
```

N-P

peek

Function	Returns the word at memory location specified by <i>segment:offset</i> .
Syntax	<code>#include <dos.h></code> <code>int peek(unsigned <i>segment</i>, unsigned <i>offset</i>);</code>
Prototype in	<code>dos.h</code>
Remarks	<p>peek returns the word at the memory location <i>segment:offset</i>.</p> <p>If peek is called when <code>dos.h</code> has been included, it is treated as a macro that expands to inline code. If you don't include <code>dos.h</code>, or if you do include it and #undef peek, you'll get the function rather than the macro.</p>
Return value	peek returns the word of data stored at the memory location <i>segment:offset</i> .
Portability	peek is unique to the 8086 family.
See also	harderr, peekb, poke
Example	<pre>#include <stdio.h> #include <conio.h> #include <dos.h> int main(void) { int value = 0; printf("The current status of your keyboard is:\n"); value = peek(0x0040, 0x0017); if (value & 1) printf("Right shift on\n"); else printf("Right shift off\n"); if (value & 2) printf("Left shift on\n"); else printf("Left shift off\n"); if (value & 4) printf("Control key on\n"); else printf("Control key off\n"); if (value & 8) printf("Alt key on\n"); else printf("Alt key off\n"); if (value & 16) printf("Scroll lock on\n"); else printf("Scroll lock off\n"); }</pre>

```

    if (value & 32)
        printf("Num lock on\n");
    else
        printf("Num lock off\n");
    if (value & 64)
        printf("Caps lock on\n");
    else
        printf("Caps lock off\n");
    return 0;
}

```

peekb

- Function** Returns the byte of memory specified by *segment:offset*.
- Syntax** `#include <dos.h>`
`char peekb(unsigned segment, unsigned offset);`
- Prototype in** dos.h
- Remarks** **peekb** returns the byte at the memory location addressed by *segment:offset*.
 If **peekb** is called when dos.h has been included, it is treated as a macro that expands to inline code. If you don't include dos.h, or if you do include it and **#undef peekb**, you'll get the function rather than the macro.
- Return value** **peekb** returns the byte of information stored at the memory location *segment:offset*.
- Portability** **peekb** is unique to the 8086 family.
- See also** **peek, pokeb**
- Example**
- ```

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

int main(void)
{
 int value = 0;
 printf("The current status of your keyboard is:\n");
 value = peekb(0x0040, 0x0017);
 if (value & 1)
 printf("Right shift on\n");
 else
 printf("Right shift off\n");
 if (value & 2)
 printf("Left shift on\n");
}

```



```

else
 printf("Left shift off\n");
if (value & 4)
 printf("Control key on\n");
else
 printf("Control key off\n");
if (value & 8)
 printf("Alt key on\n");
else
 printf("Alt key off\n");
if (value & 16)
 printf("Scroll lock on\n");
else
 printf("Scroll lock off\n");
if (value & 32)
 printf("Num lock on\n");
else
 printf("Num lock off\n");
if (value & 64)
 printf("Caps lock on\n");
else
 printf("Caps lock off\n");
return 0;
}

```

## perror

---

- Function** Prints a system error message.
- Syntax** `#include <stdio.h>`  
`void perror(const char *s);`
- Prototype in** `stdio.h`
- Remarks** **perror** prints to the *stderr* stream (normally the console) the system error message for the last library routine that produced the error.
- First the argument *s* is printed, then a colon, then the message corresponding to the current value of the global variable *errno*, and finally a newline. The convention is to pass the file name of the program as the argument string.
- The array of error message strings is accessed through the global variable *sys\_errlist*. The global variable *errno* can be used as an index into the array to find the string corresponding to the error number. None of the strings includes a newline character.

The global variable `sys_nerr` contains the number of entries in the array.

Refer to `errno`, `sys_errlist`, and `sys_nerr` in Chapter 3, "Global variables," for more information.

|                     |                                                                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return value</b> | None.                                                                                                                                                                                  |
| <b>Portability</b>  | <b>perror</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                   |
| <b>See also</b>     | <b>clearerr</b> , <b>eof</b> , <b>_strerror</b> , <b>strerror</b>                                                                                                                      |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;  int main(void) {     FILE *fp;     fp = fopen("perror.dat", "r");     if (!fp)         perror("Unable to open file for reading");     return 0; }</pre> |

## pieslice

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Draws and fills in pie slice.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Syntax</b>       | <pre>#include &lt;graphics.h&gt; void far pieslice(int x, int y, int stangle, int endangle, int radius);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Prototype in</b> | graphics.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Remarks</b>      | <p><b>pieslice</b> draws and fills a pie slice centered at <math>(x,y)</math> with a radius given by <i>radius</i>. The slice travels from <i>stangle</i> to <i>endangle</i>. The slice is outlined in the current drawing color and then filled using the current fill pattern and fill color.</p> <p>The angles for <b>pieslice</b> are given in degrees. They are measured counter-clockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.</p> <p>➡ If you are using a CGA or monochrome adapter, the examples in this book of how to use graphics functions may not produce the expected results. If your system runs on a CGA or monochrome adapter, use the value 1 (one) instead of the symbolic color constant, and consult the second example under <b>arc</b> on how to use the <b>pieslice</b> function.</p> |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |



**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **fillellipse**, **fill\_patterns** (enumerated type), **graphresult**, **sector**, **setfillstyle**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy;
 int stangle = 45, endangle = 135, radius = 100;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* set fill style and draw a pie slice */
 setfillstyle(EMPTY_FILL, getmaxcolor());
 pieslice(midx, midy, stangle, endangle, radius);

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

## poke

---

|                     |                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Stores an integer value at a memory location given by <i>segment:offset</i> .                                                                                                                                                                                                                                                                    |
| <b>Syntax</b>       | <code>#include &lt;dos.h&gt;</code><br><code>void poke(unsigned <i>segment</i>, unsigned <i>offset</i>, int <i>value</i>);</code>                                                                                                                                                                                                                |
| <b>Prototype in</b> | dos.h                                                                                                                                                                                                                                                                                                                                            |
| <b>Remarks</b>      | <b>poke</b> stores the integer <i>value</i> at the memory location <i>segment:offset</i> .<br><br>If this routine is called when dos.h has been included, it will be treated as a macro that expands to inline code. If you don't include dos.h, or if you do include it and <b>#undef poke</b> , you'll get the function rather than the macro. |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                            |
| <b>Portability</b>  | <b>poke</b> is unique to the 8086 family.                                                                                                                                                                                                                                                                                                        |
| <b>See also</b>     | <b>harderr, peek, pokeb</b>                                                                                                                                                                                                                                                                                                                      |
| <b>Example</b>      | <pre>#include &lt;dos.h&gt; #include &lt;conio.h&gt;  int main(void) {     clrscr();     printf("Make sure the scroll lock key is off and press any key\r\n");     getch();     poke(0x0000,0x0417,16);     printf("The scroll lock is now on\r\n");     return 0; }</pre>                                                                       |

N-P

## pokeb

---

|                     |                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Stores a byte value at memory location <i>segment:offset</i> .                                                                      |
| <b>Syntax</b>       | <code>#include &lt;dos.h&gt;</code><br><code>void pokeb(unsigned <i>segment</i>, unsigned <i>offset</i>, char <i>value</i>);</code> |
| <b>Prototype in</b> | dos.h                                                                                                                               |
| <b>Remarks</b>      | <b>pokeb</b> stores the byte <i>value</i> at the memory location <i>segment:offset</i> .                                            |

## pokeb

If this routine is called when `dos.h` has been included, it will be treated as a macro that expands to inline code. If you don't include `dos.h`, or if you do include it and **#undef `pokeb`**, you'll get the function rather than the macro.

**Return value** None.

**Portability** `pokeb` is unique to the 8086 family.

**See also** `peekb`, `poke`

**Example**

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

int main(void)
{
 clrscr();
 cprintf("Make sure the scroll lock key is off and press any key\r\n");
 getch();
 pokeb(0x0000, 0x0417, 16);
 cprintf("The scroll lock is now on\r\n");
 return 0;
}
```

## polar

---

**Function** Returns a complex number with a given magnitude and angle.

**Syntax** `#include <complex.h>`  
`complex polar(double mag, double angle);`

**Prototype in** `complex.h`

**Remarks** `polar(mag, angle)` is the same as `complex(mag*cos(angle), mag*sin(angle))`.

**Return value** The complex number with the given magnitude (absolute value) and angle (argument).

**Portability** Complex functions require C++ and are not portable.

**See also** `arg`, `complex`, `norm`

**Example**

```
#include <complex.h>

int main()
{
 double x = 3.1, y = 4.2;
 complex z = complex(x,y);
 cout << "z = " << z << "\n";
 cout << " has real part = " << real(z) << "\n";
}
```

```

cout << " and imaginary real part = " << imag(z) << "\n";
cout << "z has complex conjugate = " << conj(z) << "\n";
double mag = sqrt(norm(z));
double ang = arg(z);
cout << "The polar form of z is:\n";
cout << " magnitude = " << mag << "\n";
cout << " angle (in radians) = " << ang << "\n";
cout << "Reconstructing z from its polar form gives:\n";
cout << " z = " << polar(mag,ang) << "\n";
return 0;
}

```

## poly

---

**Function** Generates a polynomial from arguments.

**Syntax** `#include <math.h>`  
`double poly(double x, int degree, double coeffs[]);`

**Prototype in** `math.h`

**Remarks** **poly** generates a polynomial in  $x$ , of degree *degree*, with coefficients  $coeffs[0]$ ,  $coeffs[1]$ , ...,  $coeffs[degree]$ . For example, if  $n = 4$ , the generated polynomial is

$$coeffs[4]x^4 + coeffs[3]x^3 + coeffs[2]x^2 + coeffs[1]x + coeffs[0]$$

**Return value** **poly** returns the value of the polynomial as evaluated for the given  $x$ .

**Portability** **poly** is available on UNIX systems.

**Example**

```

#include <stdio.h>
#include <math.h>

/* polynomial: x**3 - 2x**2 + 5x - 1 */
int main(void)
{
 double array[] = { -1.0, 5.0, -2.0, 1.0 };
 double result;
 result = poly(2.0, 3, array);
 printf("The polynomial: x**3 - 2.0x**2 + 5x - 1 at 2.0 is %lf\n", result);
 return 0;
}

```

## pow

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                           |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Calculates $x$ to the power of $y$ .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                           |
| <b>Syntax</b>       | <i>Real version:</i><br><code>#include &lt;math.h&gt;</code><br><code>double pow(double x, double y);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <i>Complex version:</i><br><code>#include &lt;complex.h&gt;</code><br><code>complex pow(complex x, complex y);</code><br><code>complex pow(complex x, double y);</code><br><code>complex pow(double x, complex y);</code> |
| <b>Prototype in</b> | <i>Real version:</i><br><code>math.h</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <i>Complex version:</i><br><code>complex.h</code>                                                                                                                                                                         |
| <b>Remarks</b>      | <b>pow</b> calculates $x^y$ .<br><br>The complex <b>pow</b> is defined by<br>$\mathbf{pow}(base, expon) = \mathbf{exp}(expon \log(base))$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                           |
| <b>Return value</b> | On success, <b>pow</b> returns the value calculated, $x^y$ .<br><br>Sometimes the arguments passed to <b>pow</b> produce results that overflow or are in calculable. When the correct value would overflow, <b>pow</b> returns the value HUGE_VAL. Results of excessively large magnitude can cause the global variable <i>errno</i> to be set to<br>ERANGE     Result out of range<br><br>If the argument $x$ passed to <b>pow</b> is real and less than 0, and $y$ is not a whole number, the global variable <i>errno</i> is set to<br>EDOM        Domain error<br><br>If the arguments $x$ and $y$ passed to <b>pow</b> are both 0, <b>pow</b> returns 1.<br><br>Error handling for <b>pow</b> can be modified through the function <b>matherr</b> . |                                                                                                                                                                                                                           |
| <b>Portability</b>  | The real version of <b>pow</b> is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                           |
| <b>See also</b>     | <b>complex, exp, pow10, sqrt</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                           |
| <b>Example</b>      | <pre>#include &lt;math.h&gt; #include &lt;stdio.h&gt;  int main(void) {     double x = 2.0, y = 3.0;     printf("%lf raised to %lf is %lf\n", x, y, pow(x, y)); }</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                           |

```

 return 0;
}

```

## pow10

---

|                     |                                                                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Calculates 10 to the power of <i>p</i> .                                                                                                                                                                          |
| <b>Syntax</b>       | <code>#include &lt;math.h&gt;</code><br><code>double pow10(int <i>p</i>);</code>                                                                                                                                  |
| <b>Prototype in</b> | <code>math.h</code>                                                                                                                                                                                               |
| <b>Remarks</b>      | <b>pow10</b> computes $10^p$ .                                                                                                                                                                                    |
| <b>Return value</b> | On success, <b>pow10</b> returns the value calculated, $10^p$ .<br><br>The result is actually calculated to <b>long double</b> accuracy. All arguments are valid, though some can cause an underflow or overflow. |
| <b>Portability</b>  | Available on UNIX systems.                                                                                                                                                                                        |
| <b>See also</b>     | <b>exp</b> , <b>pow</b>                                                                                                                                                                                           |
| <b>Example</b>      | <pre> #include &lt;math.h&gt; #include &lt;stdio.h&gt;  int main(void) {     double p = 3.0;     printf("Ten raised to %lf is %lf\n", p, pow10(p));     return 0; } </pre>                                        |

## printf

---

|                     |                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Writes formatted output to <code>stdout</code> .                                                                                                                                                                                                           |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int printf(const char *<i>format</i>[, <i>argument</i>, ...]);</code>                                                                                                                                       |
| <b>Prototype in</b> | <code>stdio.h</code>                                                                                                                                                                                                                                       |
| <b>Remarks</b>      | <b>printf</b> accepts a series of arguments, applies to each a format specifier contained in the format string given by <i>format</i> , and outputs the formatted data to <i>stdout</i> . There must be the same number of format specifiers as arguments. |



## printf

**The format string** The format string, present in each of the **...printf** function calls, controls how each function will convert, format, and print its arguments. *There must be enough arguments for the format; if there are not, the results will be unpredictable and likely disastrous.* Excess arguments (more than required by the format) are merely ignored.

The format string is a character string that contains two types of objects—*plain characters* and *conversion specifications*:

- Plain characters are simply copied verbatim to the output stream.
- Conversion specifications fetch arguments from the argument list and apply formatting to them.

### **Format specifiers**

**...printf** format specifiers have the following form:

```
% [flags] [width] [.prec] [F|N|h|l|L] type
```

Each conversion specification begins with the percent character (%). After the % come the following, in this order:

- an optional sequence of flag characters, [flags]
- an optional width specifier, [width]
- an optional precision specifier, [.prec]
- an optional input-size modifier, [F|N|h|l|L]
- the conversion-type character, [type]

### **Optional format string components**

These are the general aspects of output formatting controlled by the optional characters, specifiers, and modifiers in the format string:

| Character or specifier | What it controls or specifies                                                                                                                                                     |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| flags                  | Output justification, numeric signs, decimal points, trailing zeros, octal and hex prefixes                                                                                       |
| width                  | Minimum number of characters to print, padding with blanks or zeros                                                                                                               |
| precision              | Maximum number of characters to print; for integers, minimum number of digits to print                                                                                            |
| size                   | Override default size of argument:<br><b>N</b> = near pointer<br><b>F</b> = far pointer<br><b>h</b> = <b>short int</b><br><b>l</b> = <b>long</b><br><b>L</b> = <b>long double</b> |

*...printf  
conversion-type  
characters*

The following table lists the **...printf** conversion-type characters, the type of input argument accepted by each, and in what format the output appears.

The information in this table of type characters is based on the assumption that no flag characters, width specifiers, precision specifiers, or input-size modifiers were included in the format specifier. To see how the addition of the optional characters and specifiers affects the **...printf** output, refer to the tables following this one.



## printf

---

| Type character    | Input argument | Format of output                                                                                                                                                        |
|-------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Numerics</b>   |                |                                                                                                                                                                         |
| <b>d</b>          | integer        | <b>signed decimal int.</b>                                                                                                                                              |
| <b>i</b>          | integer        | <b>signed decimal int.</b>                                                                                                                                              |
| <b>o</b>          | integer        | <b>unsigned octal int.</b>                                                                                                                                              |
| <b>u</b>          | integer        | <b>unsigned decimal int.</b>                                                                                                                                            |
| <b>x</b>          | integer        | <b>unsigned hexadecimal int</b> (with <b>a, b, c, d, e, f</b> ).                                                                                                        |
| <b>X</b>          | integer        | <b>unsigned hexadecimal int</b> (with <b>A, B, C, D, E, F</b> ).                                                                                                        |
| <b>f</b>          | floating-point | <b>signed</b> value of the form [-]dddd.dddd.                                                                                                                           |
| <b>e</b>          | floating-point | <b>signed</b> value of the form [-]d.dddd or e [+/-]ddd.                                                                                                                |
| <b>g</b>          | floating-point | <b>signed</b> value in either <b>e</b> or <b>f</b> form, based on given value and precision.<br><br>Trailing zeros and the decimal point are printed only if necessary. |
| <b>E</b>          | floating-point | Same as <b>e</b> , but with <b>E</b> for exponent.                                                                                                                      |
| <b>G</b>          | floating-point | Same as <b>g</b> , but with <b>E</b> for exponent if <b>e</b> format used.                                                                                              |
| <b>Characters</b> |                |                                                                                                                                                                         |
| <b>c</b>          | character      | Single character.                                                                                                                                                       |
| <b>s</b>          | string pointer | Prints characters until a null-terminator is pressed or precision is reached.                                                                                           |
| <b>%</b>          | none           | The % character is printed.                                                                                                                                             |
| <b>Pointers</b>   |                |                                                                                                                                                                         |
| <b>n</b>          | pointer to int | Stores (in the location pointed to by the input argument) a count of the characters written so far.                                                                     |
| <b>p</b>          | pointer        | Prints the input argument as a pointer; format depends on which memory model was used. It will be either XXXX:YYYY or YYYY (offset only).                               |

---

**Conventions** Certain conventions accompany some of these specifications, as summarized in the following table:

| Characters           | Conventions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>e</b> or <b>E</b> | The argument is converted to match the style [-] <i>d.ddd...e[+/-]ddd</i> , where <ul style="list-style-type: none"> <li>■ one digit precedes the decimal point.</li> <li>■ the number of digits after the decimal point is equal to the precision.</li> <li>■ the exponent always contains at least two digits.</li> </ul>                                                                                                                                                                                                                        |
| <b>f</b>             | The argument is converted to decimal notation in the style [-] <i>ddd.ddd...</i> , where the number of digits after the decimal point is equal to the precision (if a nonzero precision was given).                                                                                                                                                                                                                                                                                                                                                |
| <b>g</b> or <b>G</b> | The argument is printed in style <b>e</b> , <b>E</b> or <b>f</b> , with the precision specifying the number of significant digits. Trailing zeros are removed from the result, and a decimal point appears only if necessary.<br><br>The argument is printed in style <b>e</b> or <b>f</b> (with some restraints) if <b>g</b> is the conversion character, and in style <b>E</b> if the character is <b>G</b> . Style <b>e</b> is used only if the exponent that results from the conversion is either greater than the precision or less than -4. |
| <b>x</b> or <b>X</b> | For <b>x</b> conversions, the letters <b>a</b> , <b>b</b> , <b>c</b> , <b>d</b> , <b>e</b> , and <b>f</b> appear in the output; for <b>X</b> conversions, the letters <b>A</b> , <b>B</b> , <b>C</b> , <b>D</b> , <b>E</b> , and <b>F</b> appear.                                                                                                                                                                                                                                                                                                  |



Infinite floating-point numbers are printed as +INF and -INF. An IEEE Not-a-Number is printed as +NAN or -NAN.

#### Flag characters

The flag characters are minus (-), plus (+), sharp (#), and blank (). They can appear in any order and combination.

| Flag  | What it specifies                                                                                                                  |
|-------|------------------------------------------------------------------------------------------------------------------------------------|
| -     | Left-justifies the result, pads on the right with blanks. If not given, right-justifies result, pads on left with zeros or blanks. |
| +     | Signed conversion results always begin with a plus (+) or minus (-) sign.                                                          |
| blank | If value is nonnegative, the output begins with a blank instead of a plus; negative values still begin with a minus.               |
| #     | Specifies that <i>arg</i> is to be converted using an "alternate form." See the following table.                                   |



Plus (+) takes precedence over blank () if both are given.

#### Alternate forms

If the # flag is used with a conversion character, it has the following effect on the argument (*arg*) being converted:



| Conversion character | How # affects <i>arg</i>                                                                                                                                      |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>c,s,d,i,u</b>     | No effect.                                                                                                                                                    |
| <b>0</b>             | 0 is prepended to a nonzero <i>arg</i> .                                                                                                                      |
| <b>x or X</b>        | 0x (or 0X) is prepended to <i>arg</i> .                                                                                                                       |
| <b>e, E, or f</b>    | The result always contains a decimal point even if no digits follow the point. Normally, a decimal point appears in these results only if a digit follows it. |
| <b>g or G</b>        | Same as <b>e</b> and <b>E</b> , with the addition that trailing zeros are not removed.                                                                        |

**Width specifiers** The width specifier sets the minimum field width for an output value.

Width is specified in one of two ways: directly, through a decimal digit string, or indirectly, through an asterisk (\*). If you use an asterisk for the width specifier, the next argument in the call (which must be an **int**) specifies the minimum output field width.

In no case does a nonexistent or small field width cause truncation of a field. If the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

| Width specifier | How output width is affected                                                                                                                                                                    |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>n</i>        | At least <i>n</i> characters are printed. If the output value has less than <i>n</i> characters, the output is padded with blanks (right-padded if <b>-</b> flag given, left-padded otherwise). |
| <i>0n</i>       | At least <i>n</i> characters are printed. If the output value has less than <i>n</i> characters, it is filled on the left with zeros.                                                           |
| *               | The argument list supplies the width specifier, which must precede the actual argument being formatted.                                                                                         |

**Precision specifiers** A precision specification always begins with a period (.) to separate it from any preceding width specifier. Then, like width, precision is specified either directly through a decimal digit string, or indirectly through an asterisk (\*). If you use an asterisk for the precision specifier, the next argument in the call (treated as an **int**) specifies the precision.

If you use asterisks for the width or the precision, or for both, the width argument must immediately follow the specifiers, followed by the precision argument, then the argument for the data to be converted.

| Precision specifier | How output precision is affected                                                                                                                                                                                                                                        |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (none given)        | Precision set to default:<br>default = 1 for <i>d, i, o, u, x, X</i> types<br>default = 6 for <i>e, E, f</i> types<br>default = all significant digits for <i>g, G</i> types<br>default = print to first null character for <i>s</i> types; no effect on <i>c</i> types |
| .0                  | For <i>d, i, o, u, x</i> types, precision set to default; for <i>e, E, f</i> types, no decimal point is printed.                                                                                                                                                        |
| . <i>n</i>          | <i>n</i> characters or <i>n</i> decimal places are printed. If the output value has more than <i>n</i> characters, the output might be truncated or rounded. (Whether this happens depends on the type character.)                                                      |
| *                   | The argument list supplies the precision specifier, which must precede the actual argument being formatted.                                                                                                                                                             |



If an explicit precision of zero is specified, *and* the format specifier for the field is one of the integer formats (that is, *d, i, o, u, x*), *and* the value to be printed is 0, no numeric characters will be output for that field (that is, the field will be blank).

| Conversion character | How precision specification (. <i>n</i> ) affects conversion                                                                                                                                                                                            |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>d</b>             | <i>.n</i> specifies that at least <i>n</i> digits are printed. If the input argument has less than <i>n</i> digits, the output value is left-padded with zeros. If the input argument has more than <i>n</i> digits, the output value is not truncated. |
| <b>i</b>             |                                                                                                                                                                                                                                                         |
| <b>o</b>             |                                                                                                                                                                                                                                                         |
| <b>u</b>             |                                                                                                                                                                                                                                                         |
| <b>x</b>             |                                                                                                                                                                                                                                                         |
| <b>X</b>             | <i>.n</i> specifies that <i>n</i> characters are printed after the decimal point, and the last digit printed is rounded.                                                                                                                                |
| <b>e</b>             |                                                                                                                                                                                                                                                         |
| <b>E</b>             |                                                                                                                                                                                                                                                         |
| <b>f</b>             | <i>.n</i> specifies that at most <i>n</i> significant digits are printed.                                                                                                                                                                               |
| <b>g</b>             |                                                                                                                                                                                                                                                         |
| <b>G</b>             | <i>.n</i> has no effect on the output.                                                                                                                                                                                                                  |
| <b>c</b>             |                                                                                                                                                                                                                                                         |
| <b>s</b>             | <i>.n</i> specifies that no more than <i>n</i> characters are printed.                                                                                                                                                                                  |

#### Input-size modifier

The input-size modifier character (*F, N, h, l, or L*) gives the size of the subsequent input argument:

*F* = far pointer

*N* = near pointer

## printf

***h*** = short int  
***l*** = long  
***L*** = long double

The input-size modifiers (*F*, *N*, *h*, *l*, and *L*) affect how the ...**printf** functions interpret the data type of the corresponding input argument *arg*. *F* and *N* apply only to input *args* that are pointers (*%p*, *%s*, and *%n*). *h*, *L*, and *L* apply to input *args* that are numeric (integers and floating-point).

Both *F* and *N* reinterpret the input *arg*. Normally, the *arg* for a *%p*, *%s*, or *%n* conversion is a pointer of the default size for the memory model. *F* says "interpret *arg* as a far pointer." *N* says "interpret *arg* as a near pointer."

*h*, *l*, and *L* override the default size of the numeric data input arguments: *l* and *L* apply to integer (*d*, *i*, *o*, *u*, *x*, *X*) and floating-point (*e*, *E*, *f*, *g*, and *G*) types, while *h* applies to integer types only. Neither *h* nor *l* affect character (*c*, *s*) or pointer (*p*, *n*) types.

---

| Input-size modifier | How <i>arg</i> is interpreted                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>F</i>            | <i>arg</i> is read as a <b>far</b> pointer.                                                                                                                                                                                        |
| <i>N</i>            | <i>arg</i> is read as a <b>near</b> pointer. <i>N</i> cannot be used with any conversion in huge model.                                                                                                                            |
| <i>h</i>            | <i>arg</i> is interpreted as a <b>short int</b> for <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , or <i>X</i> .                                                                                                           |
| <i>l</i>            | <i>arg</i> is interpreted as a <b>long int</b> for <i>d</i> , <i>i</i> , <i>o</i> , <i>u</i> , <i>x</i> , or <i>X</i> ; <i>arg</i> is interpreted as a <b>double</b> for <i>e</i> , <i>E</i> , <i>f</i> , <i>g</i> , or <i>G</i> . |
| <i>L</i>            | <i>arg</i> is interpreted as a <b>long double</b> for <i>e</i> , <i>E</i> , <i>f</i> , <i>g</i> , or <i>G</i> .                                                                                                                    |

---

**Return value** **printf** returns the number of bytes output. In the event of error, **printf** returns EOF.

**Portability** **printf** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie. This function is not Windows compatible.

**See also** **cprintf**, **ecvt**, **fprintf**, **fread**, **fscanf**, **putc**, **puts**, **putw**, **scanf**, **sprintf**, **vprintf**, **vsprintf**

**Example**

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

#define I 555
#define R 5.5

int main(void)
{
```

```

int i,j,k,l;
char buf[7];
char *prefix = buf;
char tp[20];
printf("prefix 6d 6o 8x 10.2e "
 "10.2f\n");
strcpy(prefix,"%");
for (i = 0; i < 2; i++) {
 for (j = 0; j < 2; j++)
 for (k = 0; k < 2; k++)
 for (l = 0; l < 2; l++) {
 if (i==0) strcat(prefix,"-");
 if (j==0) strcat(prefix,"+");
 if (k==0) strcat(prefix,"#");
 if (l==0) strcat(prefix,"0");
 printf("%5s |",prefix);
 strcpy(tp,prefix);
 strcat(tp,"6d |");
 printf(tp,I);
 strcpy(tp,"");
 strcpy(tp,prefix);
 strcat(tp,"6o |");
 printf(tp,I);
 strcpy(tp,"");
 strcpy(tp,prefix);
 strcat(tp,"8x |");
 printf(tp,I);
 strcpy(tp,"");
 strcpy(tp,prefix);
 strcat(tp,"10.2e |");
 printf(tp,R);
 strcpy(tp,prefix);
 strcat(tp,"10.2f |");
 printf(tp,R);
 printf(" \n");
 strcpy(prefix,"%");
 }
 }
 }
return 0;
}

```





## printf

### Program output

| prefix | 6d     | 6o     | 8x       | 10.2e      | 10.2f      |  |
|--------|--------|--------|----------|------------|------------|--|
| %-+#0  | +555   | 01053  | 0x22b    | +5.50e+00  | +5.50      |  |
| %-+#   | +555   | 01053  | 0x22b    | +5.50e+00  | +5.50      |  |
| %-+0   | +555   | 1053   | 22b      | +5.50e+00  | +5.50      |  |
| %-+    | +555   | 1053   | 22b      | +5.50e+00  | +5.50      |  |
| %-#0   | 555    | 01053  | 0x22b    | 5.50e+00   | 5.50       |  |
| %-#    | 555    | 01053  | 0x22b    | 5.50e+00   | 5.50       |  |
| %-0    | 555    | 1053   | 22b      | 5.50e+00   | 5.50       |  |
| %-     | 555    | 1053   | 22b      | 5.50e+00   | 5.50       |  |
| %+#0   | +00555 | 001053 | 0x00022b | +05.50e+00 | +000005.50 |  |
| %+#    | +555   | 01053  | 0x22b    | +5.50e+00  | +5.50      |  |
| %+0    | +00555 | 001053 | 0000022b | +05.50e+00 | +000005.50 |  |
| %+     | +555   | 1053   | 22b      | +5.50e+00  | +5.50      |  |
| %#0    | 000555 | 001053 | 0x00022b | 005.50e+00 | 0000005.50 |  |
| %#     | 555    | 01053  | 0x22b    | 5.50e+00   | 5.50       |  |
| %0     | 000555 | 001053 | 0000022b | 005.50e+00 | 0000005.50 |  |
| %      | 555    | 1053   | 22b      | 5.50e+00   | 5.50       |  |

## putc

---

**Function** Outputs a character to a stream.

**Syntax** `#include <stdio.h>`  
`int putc(int c, FILE *stream);`

**Prototype in** `stdio.h`

**Remarks** `putc` is a macro that outputs the character *c* to the stream given by *stream*.

**Return value** On success, `putc` returns the character printed, *c*. On error, `putc` returns EOF.

**Portability** `putc` is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**See also** `fprintf`, `fputc`, `fputchar`, `fputs`, `fwrite`, `getc`, `getchar`, `printf`, `putchar`, `putw`, `vprintf`

**Example**

```
#include <stdio.h>

int main(void)
{
 char msg[] = "Hello world\n";
 int i = 0;
 while (msg[i])
 putc(msg[i++], stdout);
}
```

```

 return 0;
}

```

## putc

---

**Function** Outputs character to screen.

**Syntax** `#include <conio.h>`  
`int putch(int c);`

**Prototype in** conio.h

**Remarks** **putc** outputs the character *c* to the current text window. It is a text mode function performing direct video output to the console. **putc** does not translate linefeed characters (`\n`) into carriage-return/linefeed pairs.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable *directvideo*.

**Return value** On success, **putc** returns the character printed, *c*. On error, it returns EOF.

**Portability** **putc** works with IBM PCs and compatibles only. This function is not Windows compatible.

**See also** **cprintf**, **cputs**, **getch**, **getche**, **putc**, **putchar**

**Example**

```

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

int main(void)
{
 char ch = 0;
 printf("Input a string:");
 while ((ch != '\r')) {
 ch = getch();
 putch(ch);
 }
 return 0;
}

```



# putchar

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Outputs character on stdout.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int putchar(int c);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>      | <b>putchar</b> ( <i>c</i> ) is a macro defined to be <b>putc</b> ( <i>c</i> , <i>stdout</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Return value</b> | On success, <b>putchar</b> returns the character <i>c</i> . On error, <b>putchar</b> returns EOF.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Portability</b>  | <b>putchar</b> is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>See also</b>     | <b>fputchar</b> , <b>getc</b> , <b>getchar</b> , <b>printf</b> , <b>putc</b> , <b>putch</b> , <b>puts</b> , <b>putw</b> , <b>vprintf</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;  /* define some box drawing characters */ #define LEFT_TOP 0xDA #define RIGHT_TOP 0xBF #define HORIZ 0xC4 #define VERT 0xB3 #define LEFT_BOT 0xC0 #define RIGHT_BOT 0xD9  int main(void) {     char i, j;      /* draw the top of the box */     putchar(LEFT_TOP);     for (i=0; i&lt;10; i++)         putchar(HORIZ);     putchar(RIGHT_TOP);     putchar('\n');      /* draw the middle */     for (i=0; i&lt;4; i++) {         putchar(VERT);         for (j=0; j&lt;10; j++)             putchar(' ');         putchar(VERT);         putchar('\n');     }      /* draw the bottom */     putchar(LEFT_BOT);</pre> |

```

 for (i=0; i<10; i++)
 putchar(HORIZ);
 putchar(RIGHT_BOT);
 putchar('\n');
 return 0;
}

```

## putenv

---

**Function** Adds string to current environment.

**Syntax** `#include <stdlib.h>`  
`int putenv(const char *name);`

**Prototype in** `stdlib.h`

**Remarks** **putenv** accepts the string *name* and adds it to the environment of the *current* process. For example,

```
putenv("PATH=C:\\BC");
```

**putenv** can also be used to modify or delete an existing *name*. Delete an existing entry by making the variable value empty (for example, `MYVAR=` ).

**putenv** can be used only to modify the current program's environment. Once the program ends, the old environment is restored.

Note that the string given to **putenv** must be static or global. Unpredictable results will occur if a local or dynamic string given to **putenv** is used after the string memory is released.

**Return value** On success, **putenv** returns 0; on failure, -1.

**Portability** **putenv** is available on UNIX systems.

**See also** **getenv**

**Example**

```

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

int main(void)
{
 char *path, *ptr;
 int i = 0;

 /* get the current path environment */
 ptr = getenv("PATH");

```



```

/* set up new path */
path = malloc(strlen(ptr)+15);
strcpy(path,"PATH=");
strcat(path,ptr);
strcat(path,";c:\\temp");

/* replace current path and display current environment */
putenv(path);
while (environ[i])
 printf("%s\n",environ[i++]);
return 0;
}

```

## putimage

---

**Function** Outputs a bit image to screen.

**Syntax** `#include <graphics.h>`  
`void far putimage(int left, int top, void far *bitmap, int op);`

**Prototype in** graphics.h

**Remarks** `putimage` puts the bit image previously saved with `getimage` back onto the screen, with the upper left corner of the image placed at *(left,top)*. *bitmap* points to the area in memory where the source image is stored.

The *op* parameter to `putimage` specifies a combination operator that controls how the color for each destination pixel onscreen is computed, based on the pixel already onscreen and the corresponding source pixel in memory.

The enumeration `putimage_ops`, as defined in `graphics.h`, gives names to these operators.

| Name     | Value | Description                    |
|----------|-------|--------------------------------|
| COPY_PUT | 0     | Copy                           |
| XOR_PUT  | 1     | Exclusive or                   |
| OR_PUT   | 2     | Inclusive or                   |
| AND_PUT  | 3     | And                            |
| NOT_PUT  | 4     | Copy the inverse of the source |

In other words, `COPY_PUT` copies the source bitmap image onto the screen, `XOR_PUT` XORs the source image with that already onscreen, `OR_PUT` ORs the source image with that onscreen, and so on.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** `getimage`, `imagesize`, `putpixel`, `setvisualpage`

**Example**

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

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main()
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 void *arrow;
 int x, y, maxx;
 unsigned int size;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 maxx = getmaxx();
 x = 0;
 y = getmaxy() / 2;
 draw_arrow(x, y);

 /* calculate the size of the image and allocate space for it */
 size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);
 arrow = malloc(size);

 /* grab the image */
 getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

 /* repeat until a key is pressed */
 while (!kbhit())
 {
 /* erase old image */
 putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
 x += ARROW_SIZE;
 }
}
```



## putimage

```
 if (x >= maxx)
 x = 0;

 /* plot new image */
 putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
 }

 free(arrow);
 closegraph();
 return 0;
}

void draw_arrow(int x, int y) {
 moveto(x, y);
 linerel(4*ARROW_SIZE, 0);
 linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
 linerel(0, 2*ARROW_SIZE);
 linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
```

## putpixel

---

|                     |                                                                                                                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Plots a pixel at a specified point.                                                                                                                                                                                            |
| <b>Syntax</b>       | <pre>#include &lt;graphics.h&gt; void far putpixel(int x, int y, int color);</pre>                                                                                                                                             |
| <b>Prototype in</b> | graphics.h                                                                                                                                                                                                                     |
| <b>Remarks</b>      | <b>putpixel</b> plots a point in the color defined by <i>color</i> at (x,y).                                                                                                                                                   |
| <b>Return value</b> | None.                                                                                                                                                                                                                          |
| <b>Portability</b>  | This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.                                                 |
| <b>See also</b>     | <b>getpixel, putimage</b>                                                                                                                                                                                                      |
| <b>Example</b>      | <pre>#include &lt;graphics.h&gt; #include &lt;stdlib.h&gt; #include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;dos.h&gt;  #define PIXEL_COUNT 1000 #define DELAY_TIME 100 /* in milliseconds */  int main() {</pre> |

```

/* request autodetection */
int gdriver = DETECT, gmode, errorcode;
int i, x, y, color, maxx, maxy, maxcolor, seed;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

maxx = getmaxx() + 1;
maxy = getmaxy() + 1;
maxcolor = getmaxcolor() + 1;

while (!kbhit())
{
 /* seed the random number generator */
 seed = random(32767);
 srand(seed);
 for (i=0; i<PIXEL_COUNT; i++) {
 x = random(maxx);
 y = random(maxy);
 color = random(maxcolor);
 putpixel(x, y, color);
 }
 delay(DELAY_TIME);
 srand(seed);
 for (i=0; i<PIXEL_COUNT; i++) {
 x = random(maxx);
 y = random(maxy);
 color = random(maxcolor);
 if (color == getpixel(x, y))
 putpixel(x, y, 0);
 }
}

/* clean up */
getch();
closegraph();
return 0;
}

```



## puts

---

|                     |                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Outputs a string to stdout.                                                                                                  |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int puts(const char *s);</code>                                               |
| <b>Prototype in</b> | stdio.h                                                                                                                      |
| <b>Remarks</b>      | <b>puts</b> copies the null-terminated string <i>s</i> to the standard output stream stdout and appends a newline character. |
| <b>Return value</b> | On successful completion, <b>puts</b> returns a nonnegative value. Otherwise, it returns a value of EOF.                     |
| <b>Portability</b>  | <b>puts</b> is available on UNIX systems and is defined in ANSI C. This function is not Windows compatible.                  |
| <b>See also</b>     | <b>cputs, fputs, gets, printf, putchar</b>                                                                                   |

## puttext

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Copies text from memory to the text mode screen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Syntax</b>       | <code>#include &lt;conio.h&gt;</code><br><code>int puttext(int <i>left</i>, int <i>top</i>, int <i>right</i>, int <i>bottom</i>, void *<i>source</i>);</code>                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Prototype in</b> | conio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Remarks</b>      | <p><b>puttext</b> writes the contents of the memory area pointed to by <i>source</i> out to the onscreen rectangle defined by <i>left</i>, <i>top</i>, <i>right</i>, and <i>bottom</i>.</p> <p>All coordinates are absolute screen coordinates, not window-relative. The upper left corner is (1,1).</p> <p><b>puttext</b> places the contents of a memory area into the defined rectangle sequentially from left to right and top to bottom.</p> <p><b>puttext</b> is a text mode function performing direct video output.</p> |
| <b>Return value</b> | <b>puttext</b> returns a nonzero value if the operation succeeds; it returns 0 if it fails (for example, if you gave coordinates outside the range of the current screen mode).                                                                                                                                                                                                                                                                                                                                                 |
| <b>Portability</b>  | <b>puttext</b> works only on IBM PCs and BIOS-compatible systems. This function is not Windows compatible.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>See also</b>     | <b>gettext, movetext, window</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

## putw

- Function** Puts an integer on a stream.
- Syntax** `#include <stdio.h>`  
`int putw(int w, FILE *stream);`
- Prototype in** `stdio.h`
- Remarks** **putw** outputs the integer *w* to the given stream. **putw** neither expects nor causes special alignment in the file.
- Return value** On success, **putw** returns the integer *w*. On error, **putw** returns EOF. Since EOF is a legitimate integer, use **ferror** to detect errors with **putw**.
- Portability** **putw** is available on UNIX systems.
- See also** **getw**, **printf**

**Example**

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

#define FNAME "test.$$$"

int main(void)
{
 FILE *fp;
 int word;

 /* place the word in a file */
 fp = fopen(FNAME, "wb");
 if (fp == NULL) {
 printf("Error opening file %s\n", FNAME);
 exit(1);
 }
 word = 94;
 putw(word, fp);
 if (ferror(fp))
 printf("Error writing to file\n");
 else
 printf("Successful write\n");
 fclose(fp);

 /* reopen the file */
 fp = fopen(FNAME, "rb");
 if (fp == NULL) {
 printf("Error opening file %s\n", FNAME);
 exit(1);
 }
}
```

N-P

```

/* extract the word */
word = getw(fp);
if (ferror(fp))
 printf("Error reading file\n");
else
 printf("Successful read: word = %d\n", word);

/* clean up */
fclose(fp);
unlink(FNAME);
return 0;
}

```

## qsort

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Sorts using the quicksort algorithm.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Syntax</b>       | <pre>#include &lt;stdlib.h&gt; void qsort(void *base, size_t nelem, size_t width,            int (*fcmp)(const void *, const void *));</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Prototype in</b> | stdlib.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Remarks</b>      | <p><b>qsort</b> is an implementation of the “median of three” variant of the quicksort algorithm. <b>qsort</b> sorts the entries in a table by repeatedly calling the user-defined comparison function pointed to by <i>fcmp</i>.</p> <ul style="list-style-type: none"> <li>■ <i>base</i> points to the base (0th element) of the table to be sorted.</li> <li>■ <i>nelem</i> is the number of entries in the table.</li> <li>■ <i>width</i> is the size of each entry in the table, in bytes.</li> </ul> <p><i>fcmp</i>, the comparison function, accepts two arguments, <i>elem1</i> and <i>elem2</i>, each a pointer to an entry in the table. The comparison function compares each of the pointed-to items (<i>*elem1</i> and <i>*elem2</i>), and returns an integer based on the result of the comparison.</p> <pre> *elem1 &lt; *elem2 fcmp returns an integer &lt; 0 *elem1 == *elem2 fcmp returns 0 *elem1 &gt; *elem2 fcmp returns an integer &gt; 0 </pre> <p>In the comparison, the less-than symbol (&lt;) means the left element should appear before the right element in the final, sorted sequence. Similarly, the greater-than (&gt;) symbol means the left element should appear after the right element in the final, sorted sequence.</p> |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

**Portability** `qsort` is available on UNIX systems and is defined in ANSI C.

**See also** `bsearch`, `lsearch`

**Example**

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

int sort_function(const void *a, const void *b);
char list[5][4] = { "cat", "car", "cab", "cap", "can" };

int main()
{
 int x;
 qsort((void *)list, 5, sizeof(list[0]), sort_function);
 for (x = 0; x < 5; x++)
 printf("%s\n", list[x]);
 return 0;
}

int sort_function(const void *a, const void *b) {
 return(strcmp(a,b));
}
```

**Program output**

```
cab
can
cap
car
cat
```

Q-R

## raise

---

**Function** Sends a software signal to the executing program.

**Syntax** `#include <signal.h>`  
`int raise(int sig);`

**Prototype in** `signal.h`

**Remarks** `raise` sends a signal of type `sig` to the program. If the program has installed a signal handler for the signal type specified by `sig`, that handler will be executed. If no handler has been installed, the default action for that signal type will be taken.

The signal types currently defined in `signal.h` are noted here:

## raise

| Signal  | Meaning                             |
|---------|-------------------------------------|
| SIGABRT | Abnormal termination (*)            |
| SIGFPE  | Bad floating-point operation        |
| SIGILL  | Illegal instruction (#)             |
| SIGINT  | Control break interrupt             |
| SIGSEGV | Invalid access to storage (#)       |
| SIGTERM | Request for program termination (*) |

Signal types marked with a (\*) aren't generated by DOS or Borland C++ during normal operation. However, they can be generated with **raise**. Signals marked by (#) *can't* be generated asynchronously on 8088 or 8086 processors but *can* be generated on some other processors (see **signal** for details).

**Return value** **raise** returns 0 if successful, nonzero otherwise.

**Portability** **raise** is available on UNIX systems and is defined in ANSI C.

**See also** **abort**, **signal**

**Example**

```
#include <signal.h>

int main()
{
 int a, b;
 a = 10;
 b = 0;
 if (b == 0)
 raise(SIGFPE); /* preempt divide by zero error */
 a = a / b;
 return 0;
}
```

## rand

**Function** Random number generator.

**Syntax** `#include <stdlib.h>`  
`int rand(void);`

**Prototype in** `stdlib.h`

**Remarks** **rand** uses a multiplicative congruential random number generator with period  $2^{32}$  to return successive pseudorandom numbers in the range from 0 to `RAND_MAX`. The symbolic constant `RAND_MAX` is defined in `stdlib.h`; its value is  $2^{15} - 1$ .

**Return value** **rand** returns the generated pseudorandom number.

**Portability** **rand** is available on UNIX systems and is defined in ANSI C.

**See also** **random, randomize, srand**

**Example**

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

int main(void)
{
 int i;
 printf("Ten random numbers from 0 to 99\n\n");
 for(i=0; i<10; i++)
 printf("%d\n", rand() % 100);
 return 0;
}
```

## randbrd

---

**Function** Reads random block.

**Syntax** `#include <dos.h>`  
`int randbrd(struct fcb *fcb, int rcnt);`

**Prototype in** dos.h

**Remarks** **randbrd** reads *rcnt* number of records using the open file control block (FCB) pointed to by *fcb*. The records are read into memory at the current disk transfer address (DTA). They are read from the disk record indicated in the random record field of the FCB. This is accomplished by calling DOS system call 0x27.

The actual number of records read can be determined by examining the random record field of the FCB. The random record field is advanced by the number of records actually read.

**Return value** The following values are returned, depending on the result of the **randbrd** operation:

- 0 All records are read.
- 1 End-of-file is reached and the last record read is complete.
- 2 Reading records would have wrapped around address 0xFFFF (as many records as possible are read).
- 3 End-of-file is reached with the last record incomplete.

**Portability** **randbrd** is unique to DOS. This function is not Windows compatible.

**See also** `getdta`, `randbwr`, `setdta`

```
Example #include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
 char far *save_dta;
 char line[80], buffer[256];
 struct fcb blk;
 int i, result;

 /* get user input file name for dta */
 printf("Enter drive and file name (no path - i.e. a:file.dat)\n");
 gets(line);

 /* put file name in fcb */
 if (!parsfnm(line, &blk, 1)) {
 printf("Error in call to parsfnm\n");
 exit(1);
 }
 printf("Drive #%d File: %s\n\n", blk.fcb_drive, blk.fcb_name);

 /* open file with DOS fcb open file */
 bdosptr(0x0F, &blk, 0);

 /* save old dta and set new one */
 save_dta = getdta();
 setdta(buffer);

 /* set up information for the new dta */
 blk.fcb_recsz = 128;
 blk.fcb_random = 0L;
 result = randbrd(&blk, 1);

 /* check results from randbrd */
 if (!result)
 printf("Read OK\n\n");
 else {
 perror("Error during read");
 exit(1);
 }

 /* read in data from the new dta */
 printf("The first 128 characters are:\n");
 for (i=0; i<128; i++)
 putchar(buffer[i]);

 /* restore previous dta */
 setdta(save_dta);
}
```

```

 return 0;
}

```

## randbwr

---

- Function** Writes random block.
- Syntax** `#include <dos.h>`  
`int randbwr(struct fcb *fcb, int rcnt);`
- Prototype in** dos.h
- Remarks** **randbwr** writes *rcnt* number of records to disk using the open file control block (FCB) pointed to by *fcb*. This is accomplished using DOS system call 0x28. If *rcnt* is 0, the file is truncated to the length indicated by the random record field.
- The actual number of records written can be determined by examining the random record field of the FCB. The random record field is advanced by the number of records actually written.
- Return value** The following values are returned, depending upon the result of the **randbwr** operation:
- 0 All records are written.
  - 1 There is not enough disk space to write the records (no records are written).
  - 2 Writing records would have wrapped around address 0xFFFF (as many records as possible are written).
- Portability** **randbwr** is unique to DOS.
- See also** **randbrd** This function is not Windows compatible.
- Example**

```

#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
 char far *save_dta;
 char line[80];
 char buffer[256] = "RANDBWR test!";
 struct fcb blk;
 int result;

 /* get new file name from user */
 printf("Enter a file name to create (no path - ie. a:file.dat\n");

```



## randbwr

```
gets(line);

/* parse the new file name to the dta */
parsfnm(line,&blk,1);
printf("Drive #%d File: %s\n", blk.fcb_drive, blk.fcb_name);

/* request DOS services to create file */
if (bdosptr(0x16, &blk, 0) == -1) {
 perror("Error creating file");
 exit(1);
}

/* save old dta and set new dta */
save_dta = getdta();
setdta(buffer);

/* write new records */
blk.fcb_recsz = 256;
blk.fcb_random = 0L;
result = randbwr(&blk, 1);

if (!result)
 printf("Write OK\n");
else {
 perror("Disk error");
 exit(1);
}

/* request DOS services to close the file */
if (bdosptr(0x10, &blk, 0) == -1) {
 perror("Error closing file");
 exit(1);
}

/* reset the old dta */
setdta(save_dta);
return 0;
}
```

## random

---

**Function** Random number generator.

**Syntax** #include <stdlib.h>  
int random(int *num*);

**Prototype in** stdlib.h

- Remarks** **random** returns a random number between 0 and (*num*-1). **random(*num*)** is a macro defined in `stdlib.h`. Both *num* and the random number returned are integers.
- Return value** **random** returns a number between 0 and (*num*-1).
- Portability** A corresponding function exists in Turbo Pascal.
- See also** **rand, randomize, srand**
- Example**
- ```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main()      /* prints a random number in the range 0 to 99 */
{
    randomize();
    printf("Random number in the 0-99 range: %d\n", random(100));
    return 0;
}
```

randomize

- Function** Initializes random number generator.
- Syntax**
- ```
#include <stdlib.h>
#include <time.h>
void randomize(void);
```
- Prototype in** `stdlib.h`
- Remarks** **randomize** initializes the random number generator with a random value. Because **randomize** is implemented as a macro that calls the **time** function prototyped in `time.h`, we recommend that you also include `time.h` when you use this routine.
- Return value** None.
- Portability** A corresponding function exists in Turbo Pascal.
- See also** **rand, random, srand**
- Example**
- ```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
    int i;
```

randomize

```
randomize();
printf("Ten random numbers from 0 to 99\n\n");
for(i=0; i<10; i++)
    printf("%d\n", rand() % 100);
return 0;
}
```

_read

- Function** Reads from file.
- Syntax** `#include <io.h>`
`int _read(int handle, void *buf, unsigned len);`
- Prototype in** `io.h`
- Remarks** `_read` attempts to read *len* bytes from the file associated with *handle* into the buffer pointed to by *buf*. `_read` is a direct call to the DOS read system call.
- When a file is opened in text mode, `_read` does not remove carriage returns.
- handle* is a file handle obtained from a `creat`, `open`, `dup`, or `dup2` call.
- On disk files, `_read` begins reading at the current file pointer. When the reading is complete, it increments the file pointer by the number of bytes read. On devices, the bytes are read directly from the device.
- The maximum number of bytes that `_read` can read is 65,534, since 65,535 (0xFFFF) is the same as -1, the error return indicator.
- Return value** On successful completion, `_read` returns a positive integer indicating the number of bytes placed in the buffer. On end-of-file, `_read` returns zero. On error, it returns -1, and the global variable *errno* is set to one of the following:
- | | |
|--------|-------------------|
| EACCES | Permission denied |
| EBADF | Bad file number |
- Portability** `_read` is unique to DOS.
- See also** `_open`, `read`, `_write`
- Example**
- ```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
```

```

#include <sys\stat.h>

int main(void)
{
 void *buf;
 int handle, bytes;
 buf = malloc(10);

 /* Looks for a file in the current directory named TEST.$$$ and attempts to
 read 10 bytes from it. To use this example you should create the file
 TEST.$$$ */
 if ((handle = open("TEST.$$$", O_RDONLY | O_BINARY,
 S_IWRITE | S_IREAD)) == -1) {
 printf("Error Opening File\n");
 exit(1);
 }
 if ((bytes = read(handle, buf, 10)) == -1) {
 printf("Read Failed.\n");
 exit(1);
 }
 else {
 printf("Read: %d bytes read.\n", bytes);
 }
 return 0;
}

```

## read

Q-R

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reads from file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Syntax</b>       | <code>#include &lt;io.h&gt;</code><br><code>int read(int <i>handle</i>, void *<i>buf</i>, unsigned <i>len</i>);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Prototype in</b> | io.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Remarks</b>      | <p><b>read</b> attempts to read <i>len</i> bytes from the file associated with <i>handle</i> into the buffer pointed to by <i>buf</i>.</p> <p>For a file opened in text mode, <b>read</b> removes carriage returns and reports end-of-file when it reaches a <i>Ctrl-Z</i>.</p> <p><i>handle</i> is a file handle obtained from a <b>creat</b>, <b>open</b>, <b>dup</b>, or <b>dup2</b> call.</p> <p>On disk files, <b>read</b> begins reading at the current file pointer. When the reading is complete, it increments the file pointer by the number of bytes read. On devices, the bytes are read directly from the device.</p> |

The maximum number of bytes that **read** can read is 65,534, since 65,535 (0xFFFF) is the same as -1, the error return indicator.

**Return value** On successful completion, **read** returns an integer indicating the number of bytes placed in the buffer. If the file was opened in text mode, **read** does not count carriage returns or *Ctrl-Z* characters in the number of bytes read.

On end-of-file, **read** returns 0. On error, **read** returns -1 and sets the global variable *errno* to one of the following:

|        |                   |
|--------|-------------------|
| EACCES | Permission denied |
| EBADF  | Bad file number   |

**Portability** **read** is available on UNIX systems.

**See also** **open**, **\_read**, **write**

**Example**

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
 void *buf;
 int handle, bytes;
 buf = malloc(10);

 /* Looks for a file in the current directory named TEST.$$$ and attempts to
 read 10 bytes from it. To use this example you should create the file
 TEST.$$$ */
 if ((handle = open("TEST.$$$", O_RDONLY | O_BINARY,
 S_IWRITE | S_IREAD)) == -1) {
 printf("Error Opening File\n");
 exit(1);
 }
 if ((bytes = read(handle, buf, 10)) == -1) {
 printf("Read Failed.\n");
 exit(1);
 }
 else {
 printf("Read: %d bytes read.\n", bytes);
 }
 return 0;
}
```

## real

|                     |                                                                                                                                                                                                                                                                                           |                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <b>Function</b>     | Returns the real part of a complex number or converts a BCD number back to <b>float</b> , <b>double</b> or <b>long double</b> .                                                                                                                                                           |                                                                                     |
| <b>Syntax</b>       | <i>As defined in <b>complex</b>:</i><br>#include <complex.h><br>double real(complex <i>x</i> );                                                                                                                                                                                           | <i>As defined in <b>bcd</b>:</i><br>#include <bcd.h><br>double real(bcd <i>x</i> ); |
| <b>Prototype in</b> | complex.h                                                                                                                                                                                                                                                                                 | bcd.h                                                                               |
| <b>Remarks</b>      | The data associated to a complex number consists of two floating-point numbers. <b>real</b> returns the one considered to be the real part.<br><br>You can also use <b>real</b> to convert a binary coded decimal number back to a <b>float</b> , <b>double</b> , or <b>long double</b> . |                                                                                     |
| <b>Return value</b> | The real part of part of the complex number.                                                                                                                                                                                                                                              |                                                                                     |
| <b>Portability</b>  | Complex functions require C++ and are not portable.                                                                                                                                                                                                                                       |                                                                                     |
| <b>See also</b>     | <b>bcd</b> , <b>complex</b> , <b>imag</b>                                                                                                                                                                                                                                                 |                                                                                     |

**Example 1**

```
#include <complex.h>

int main(void)
{
 double x = 3.1, y = 4.2;
 complex z = complex(x,y);
 cout << "z = " << z << "\n";
 cout << " has real part = " << real(z) << "\n";
 cout << " and imaginary real part = " << imag(z) << "\n";
 cout << "z has complex conjugate = " << conj(z) << "\n";
 return 0;
}
```

**Example 2**

```
#include <bcd.h>
#include <iostream.h>

int main(void)
{
 bcd x = 3.1;
 cout << "The bcd number x = " << x << "\n";
 cout << "Its binary equivalent is " << real(x) << "\n";
 return 0;
}
```

## realloc

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reallocates main memory.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Syntax</b>       | <code>#include &lt;stdlib.h&gt;</code><br><code>void *realloc(void *block, size_t size);</code>                                                                                                                                                                                                                                                                                                                                                                |
| <b>Prototype in</b> | stdlib.h, alloc.h                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Remarks</b>      | <p><b>realloc</b> attempts to shrink or expand the previously allocated block to <i>size</i> bytes. The <i>block</i> argument points to a memory block previously obtained by calling <b>malloc</b>, <b>calloc</b>, or <b>realloc</b>. If <i>block</i> is a null pointer, <b>realloc</b> works just like <b>malloc</b>.</p> <p><b>realloc</b> adjusts the size of the allocated block to <i>size</i>, copying the contents to a new location if necessary.</p> |
| <b>Return value</b> | <b>realloc</b> returns the address of the reallocated block, which can be different than the address of the original block. If the block cannot be reallocated or <i>size</i> == 0, <b>realloc</b> returns null.                                                                                                                                                                                                                                               |
| <b>Portability</b>  | <b>realloc</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>See also</b>     | <b>calloc</b> , <b>farrealloc</b> , <b>free</b> , <b>malloc</b>                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;alloc.h&gt; #include &lt;string.h&gt;  int main(void) {     char *str;      /* allocate memory for string */     str = malloc(10);      /* copy "Hello" into string */     strcpy(str, "Hello");     printf("String is %s\n Address is %p\n", str, str);     str = realloc(str, 20);     printf("String is %s\n New address is %p\n", str, str);      /* free memory */     free(str);     return 0; }</pre>        |

# rectangle

---

**Function** Draws a rectangle.

**Syntax** `#include <graphics.h>`  
`void far rectangle(int left, int top, int right, int bottom);`

**Prototype in** graphics.h

**Remarks** **rectangle** draws a rectangle in the current line style, thickness, and drawing color.

*(left,top)* is the upper left corner of the rectangle, and *(right,bottom)* is its lower right corner.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **bar, bar3d, setcolor, setlinestyle**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int left, top, right, bottom;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 left = getmaxx() / 2 - 50;
 top = getmaxy() / 2 - 50;
```



## rectangle

```
right = getmaxx() / 2 + 50;
bottom = getmaxy() / 2 + 50;

/* draw a rectangle */
rectangle(left,top,right,bottom);

/* clean up */
getch();
closegraph();
return 0;
}
```

## registerbgidriver

---

- Function** Registers a user-loaded or linked-in graphics driver code with the graphics system.
- Syntax** `#include <graphics.h>`  
`int registerbgidriver(void (*driver)(void));`
- Prototype in** `graphics.h`
- Remarks** **registerbgidriver** enables a user to load a driver file and “register” the driver. Once its memory location has been passed to **registerbgidriver**, **initgraph** uses the registered driver. A user-registered driver can be loaded from disk onto the heap, or converted to an .OBJ file (using BINOBJ.EXE) and linked into the .EXE.
- Calling **registerbgidriver** informs the graphics system that the driver pointed to by *driver* was included at link time. This routine checks the linked-in code for the specified driver; if the code is valid, it registers the code in internal tables. Linked-in drivers are discussed in detail in UTIL.DOC, included with your distribution disks.
- By using the name of a linked-in driver in a call to **registerbgidriver**, you also tell the compiler (and linker) to link in the object file with that public name.
- Return value** **registerbgidriver** returns a negative graphics error code if the specified driver or font is invalid. Otherwise, **registerbgidriver** returns the driver number.
- If you register a user-supplied driver, you *must* pass the result of **registerbgidriver** to **initgraph** as the drive number to be used.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** `graphresult`, `initgraph`, `installuserdriver`, `registerbgifont`

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;

 /* register a driver that was added into GRAPHICS.LIB */
 errorcode = registerbgidriver(EGAVGA_driver);

 /* report any registration errors */
 if (errorcode < 0) {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 /* draw a line */
 line(0, 0, getmaxx(), getmaxy());

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

## registerbgifont

---

- Function** Registers linked-in stroked font code.
- Syntax** `#include <graphics.h>`  
`int registerbgifont(void (*font)(void));`
- Prototype in** graphics.h
- Remarks** Calling **registerbgifont** informs the graphics system that the font pointed to by *font* was included at link time. This routine checks the linked-in code for the specified font; if the code is valid, it registers the code in internal tables. Linked-in fonts are discussed in detail under BGIOBJ in UTIL.DOC included with your distribution disks.
- By using the name of a linked-in font in a call to **registerbgifont**, you also tell the compiler (and linker) to link in the object file with that public name.
- If you register a user-supplied font, you *must* pass the result of **registerbgifont** to **setttextstyle** as the font number to be used.
- Return value** **registerbgifont** returns a negative graphics error code if the specified font is invalid. Otherwise, **registerbgifont** returns the font number of the registered font.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **graphresult, initgraph, installuserdriver, registerbgdriver, setttextstyle**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;

    /* register a font file that was added into GRAPHICS.LIB */
    errorcode = registerbgifont(triplex_font);

    /* report any registration errors */
    if (errorcode < 0) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
    }
}
```

```

    getch();
    exit(1);          /* terminate with an error code */
}

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* select the registered font */
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

/* output some text */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, "The TRIPLEX FONT");

/* clean up */
getch();
closegraph();
return 0;
}

```

remove

- Function** Removes a file.
- Syntax** `#include <stdio.h>`
`int remove(const char *filename);`
- Prototype in** `stdio.h`
- Remarks** **remove** deletes the file specified by *filename*. It is a macro that simply translates its call to a call to **unlink**. If your file is open, be sure to close it before removing it.
- ➡ The string pointed to by *filename* may include a full DOS path.
- Return value** On successful completion, **remove** returns 0. On error, it returns -1, and the global variable *errno* is set to one of the following:

remove

ENOENT No such file or directory
EACCES Permission denied

Portability **remove** is available on UNIX systems and is defined in ANSI C.

See also **unlink**

Example #include <stdio.h>

```
int main(void)
{
    char file[80];

    /* prompt for file name to delete */
    printf("File to delete: ");
    gets(file);

    /* delete the file */
    if (remove(file) == 0)
        printf("Removed %s.\n", file);
    else
        perror("remove");
    return 0;
}
```

rename

Function Renames a file.

Syntax #include <stdio.h>
int rename(const char *oldname, const char *newname);

Prototype in stdio.h

Remarks **rename** changes the name of a file from *oldname* to *newname*. If a drive specifier is given in *newname*, the specifier must be the same as that given in *oldname*.

Directories in *oldname* and *newname* need not be the same, so **rename** can be used to move a file from one directory to another. Wildcards are not allowed.

Return value On successfully renaming the file, **rename** returns 0. In the event of error, -1 is returned, and the global variable *errno* is set to one of the following:

ENOENT No such file or directory
EACCES Permission denied
ENOTSAM Not same device

Portability `rename` is compatible with ANSI C.

Example

```
#include <stdio.h>

int main(void)
{
    char oldname[80], newname[80];

    /* prompt for file to rename and new name */
    printf("File to rename: ");
    gets(oldname);
    printf("New name: ");
    gets(newname);

    /* rename the file */
    if (rename(oldname, newname) == 0)
        printf("Renamed %s to %s.\n", oldname, newname);
    else
        perror("rename");
    return 0;
}
```

restorecrtmode

Function Restores the screen mode to its pre-`initgraph` setting.

Syntax `#include <graphics.h>`
`void far restorecrtmode(void);`

Prototype in `graphics.h`

Remarks `restorecrtmode` restores the original video mode detected by `initgraph`.

This function can be used in conjunction with `setgraphmode` to switch back and forth between text and graphics modes. `textmode` should not be used for this purpose; use it only when the screen is in text mode, to change to a different text mode.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also `getgraphmode`, `initgraph`, `setgraphmode`

Example

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

```

#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int x, y;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press any key to exit graphics:");
    getch();

    /* restore system to text mode */
    restorecrtmode();
    printf("We're now in text mode.\n");
    printf("Press any key to return to graphics mode:");
    getch();

    /* return to graphics mode */
    setgraphmode(getgraphmode());

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "We're back in graphics mode.");
    outtextxy(x, y+textheight("W"), "Press any key to halt:");

    /* clean up */
    getch();
    closegraph();
    return 0;
}

```

rewind

Function Repositions a file pointer to the beginning of a stream.

Syntax `#include <stdio.h>`
`void rewind(FILE *stream);`

Prototype in `stdio.h`

Remarks `rewind(stream)` is equivalent to `fseek(stream, 0L, SEEK_SET)`, except that `rewind` clears the end-of-file and error indicators, while `fseek` only clears the end-of-file indicator.

After `rewind`, the next operation on an update file can be either input or output.

Return value None.

Portability `rewind` is available on all UNIX systems and is defined in ANSI C.

See also `fopen`, `fseek`, `ftell`

Example See `fseek`

Example

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

int main(void)
{
    FILE *fp;
    char *fname = "TXXXXXX", *newname, first;
    newname = mktemp(fname);
    fp = fopen(newname, "w+");
    fprintf(fp, "abcdefghijklmnopqrstuvwxyz");
    rewind(fp);
    fscanf(fp, "%c", &first);
    printf("The first character is: %c\n", first);
    fclose(fp);
    remove(newname);
    return 0;
}
```


rmdir

Function	Removes a DOS file directory.				
Syntax	<pre>#include <dir.h> int rmdir(const char *path);</pre>				
Prototype in	dir.h				
Remarks	<p>rmdir deletes the directory whose path is given by <i>path</i>. The directory named by <i>path</i></p> <ul style="list-style-type: none"> ■ must be empty. ■ must not be the current working directory. ■ must not be the root directory. 				
Return value	<p>rmdir returns 0 if the directory is successfully deleted. A return value of -1 indicates an error, and the global variable <i>errno</i> is set to one of the following:</p> <table> <tr> <td>EACCES</td> <td>Permission denied</td> </tr> <tr> <td>ENOENT</td> <td>Path or file function not found</td> </tr> </table>	EACCES	Permission denied	ENOENT	Path or file function not found
EACCES	Permission denied				
ENOENT	Path or file function not found				
Portability	rmdir is available on UNIX systems and is defined in ANSI C.				
See also	chdir, getcurdir, getcwd, mkdir				
Example	<pre>#include <stdio.h> #include <conio.h> #include <process.h> #include <dir.h> #define DIRNAME "testdir.\$\$\$" int main(void) { int stat; stat = mkdir(DIRNAME); if (!stat) printf("Directory created\n"); else { printf("Unable to create directory\n"); exit(1); } getch(); system("dir/p"); getch(); stat = rmdir(DIRNAME); if (!stat)</pre>				

```

        printf("\nDirectory deleted\n");
    else {
        perror("\nUnable to delete directory\n");
        exit(1);
    }
    return 0;
}

```

_rotl

Function Bit-rotates an **unsigned** integer value to the left.

Syntax `#include <stdlib.h>`
`unsigned _rotl(unsigned value, int count);`

Prototype in `stdlib.h`

Remarks `_rotl` rotates the given *value* to the left *count* bits. The value rotated is an **unsigned** integer.

Return value `_rotl` returns the value of *value* left-rotated *count* bits.

Portability `_rotl` is unique to DOS.

See also `_lrotl`, `_lrotr`, `_rotr`

Example

```

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

int main(void)
{
    unsigned value, result;
    value = 32767;
    result = _rotl(value, 1);
    printf("The value %u rotated left one bit is: %u\n", value, result);
    return 0;
}

```

Q-R

_rotr

Function Bit-rotates an **unsigned** integer value to the right.

Syntax `#include <stdlib.h>`
`unsigned _rotr(unsigned value, int count);`

Prototype in `stdlib.h`

`_rotr`

- Remarks** `_rotr` rotates the given *value* to the right *count* bits. The value rotated is an **unsigned integer**.
- Return value** `_rotr` returns the value of *value* right-rotated *count* bits.
- Portability** `_rotr` is unique to DOS.
- See also** `_lrotl`, `_lrotr`, `_rotl`
- Example**
- ```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
 unsigned value, result;
 value = 32767;
 result = _rotr(value, 1);
 printf("The value %u rotated right one bit is: %u\n", value, result);
 return 0;
}
```

## `sbrk`

---

- Function** Changes data segment space allocation.
- Syntax** `#include <alloc.h>`  
`void *sbrk(int incr);`
- Prototype in** `alloc.h`
- Remarks** `sbrk` adds *incr* bytes to the break value and changes the allocated space accordingly. *incr* can be negative, in which case the amount of allocated space is decreased.
- `sbrk` will fail without making any change in the allocated space if such a change would result in more space being allocated than is allowable.
- Return value** Upon successful completion, `sbrk` returns the old break value. On failure, `sbrk` returns a value of `-1`, and the global variable `errno` is set to
- `ENOMEM` Not enough core
- Portability** `sbrk` is available on UNIX systems. This function is not Windows compatible.
- See also** `brk`
- Example**
- ```
#include <stdio.h>
#include <alloc.h>
```

```

int main(void)
{
    printf("Changing allocation with sbrk()\n");
    printf("Before sbrk() call: %lu bytes free\n", (unsigned long) coreleft());
    sbrk(1000);
    printf("After sbrk() call: %lu bytes free\n", (unsigned long) coreleft());
    return 0;
}

```

scanf

Function Scans and formats input from the stdin stream.

Syntax #include <stdio.h>
int scanf(const char **format*[, address, ...]);

Prototype in stdio.h

Remarks **scanf** scans a series of input fields, one character at a time, reading from the stdin stream. Then each field is formatted according to a format specifier passed to **scanf** in the format string pointed to by *format*. Finally, **scanf** stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

The format string The format string present in **scanf** and the related functions **cscanf**, **fscanf**, **sscanf**, **vscanf**, **vfscanf**, and **vsscanf** controls how each function scans, converts, and stores its input fields. There must be enough address arguments for the given format specifiers; if not, the results are unpredictable and likely disastrous. Excess address arguments (more than required by the format) are merely ignored.

➡ **scanf** often leads to unexpected results if you diverge from an expected pattern. You need to remember to teach **scanf** how to synchronize at the end of a line. The combination of **gets** or **fgets** followed by **sscanf** is safe and easy, and therefore preferred.

The format string is a character string that contains three types of objects: whitespace characters, non-whitespace characters, and format specifiers.

- The whitespace characters are blank, tab (**\t**) or newline (**\n**). If a **...scanf** function encounters a whitespace character in the format string, it will read, but not store, all consecutive whitespace characters up to the next non-whitespace character in the input.

- The non-whitespace characters are all other ASCII characters except the percent sign (%). If a **...scanf** function encounters a non-whitespace character in the format string, it will read, but not store, a matching non-whitespace character.
- The format specifiers direct the **...scanf** functions to read and convert characters from the input field into specific types of values, then store them in the locations given by the address arguments.

Trailing whitespace is left unread (including a newline), unless explicitly matched in the format string.

Format specifiers

...scanf format specifiers have the following form:

```
% [*] [width] [F|N] [h|l|L] type_character
```

Each format specifier begins with the percent character (%). After the % come the following, in this order:

- an optional assignment-suppression character, [*]
- an optional width specifier, [width]
- an optional pointer size modifier, [F|N]
- an optional argument-type modifier, [h|l|L]
- the type character

Optional format string components

These are the general aspects of input formatting controlled by the optional characters and specifiers in the **...scanf** format string:

Character or specifier	What it controls or specifies
*	Suppresses assignment of the next input field.
width	Maximum number of characters to read; fewer characters might be read if the <code>...scanf</code> function encounters a whitespace or unconvertible character.
size	Overrides default size of address argument: <i>N</i> = near pointer <i>F</i> = far pointer
argument type	Overrides default type of address argument: <i>h</i> = short int <i>l</i> = long int (if the type character specifies an integer conversion) <i>l</i> = double (if the type character specifies a floating-point conversion) <i>L</i> = long double (valid only with floating-point conversions)

...scanf type characters

The following table lists the `...scanf` type characters, the type of input expected by each, and in what format the input will be stored.

The information in this table is based on the assumption that no optional characters, specifiers, or modifiers (***, *width*, or *size*) were included in the format specifier. To see how the addition of the optional elements affects the `...scanf` input, refer to the tables following this one.

Type character	Expected input	Type of argument
Numerics		
d	Decimal integer	Pointer to int (<code>int *arg</code>)
D	Decimal integer	Pointer to long (<code>long *arg</code>)
o	Octal integer	Pointer to int (<code>int *arg</code>)
O	Octal integer	Pointer to long (<code>long *arg</code>)
i	Decimal, octal, or hexadecimal integer	Pointer to int (<code>int *arg</code>)
l	Decimal, octal, or hexadecimal integer	Pointer to long (<code>long *arg</code>)
u	Unsigned decimal integer	Pointer to unsigned int (<code>unsigned int *arg</code>)
U	Unsigned decimal integer	Pointer to unsigned long (<code>unsigned long *arg</code>)
x	Hexadecimal integer	Pointer to int (<code>int *arg</code>)
X	Hexadecimal integer	Pointer to long (<code>long *arg</code>)
e, E	Floating point	Pointer to float (<code>float *arg</code>)
f	Floating point	Pointer to float (<code>float *arg</code>)
g, G	Floating point	Pointer to float (<code>float *arg</code>)
Characters		
s	Character string	Pointer to array of characters (<code>char arg[]</code>)
c	Character	Pointer to character (<code>char *arg</code>) if a field width <i>W</i> is given along with the <i>c</i> -type character (such as <code>%5c</code>). Pointer to array of <i>W</i> characters (<code>char arg[W]</code>)
%	% character	No conversion done; % is stored.
Pointers		
n		Pointer to int (<code>int *arg</code>). The number of characters read successfully up to <code>%n</code> is stored in this int .
p	Hexadecimal form <code>YYYY:ZZZZ</code> or <code>ZZZZ</code>	Pointer to an object (<code>far*</code> or <code>near*</code>) <code>%p</code> conversions default to the pointer size native to the memory model.

Input fields Any one of the following is an input field:

- all characters up to (but not including) the next whitespace character
- all characters up to the first one that cannot be converted under the current format specifier (such as an 8 or 9 under octal format)

- up to n characters, where n is the specified field width

Conventions Certain conventions accompany some of these format specifiers, as summarized here.

%c conversion

This specification reads the next character, including a whitespace character. To skip one whitespace character and read the next non-whitespace character, use %1s.

%Wc conversion (W = width specification)

The address argument is a pointer to an array of characters; the array consists of W elements (char *arg*[W]).

%s conversion

The address argument is a pointer to an array of characters (char *arg*[]).

The array size must be *at least* ($n+1$) bytes, where n equals the length of string s (in characters). A space or new line terminates the input field. A null-terminator is automatically appended to the string and stored as the last element in the array.

%[search_set] conversion

The set of characters surrounded by square brackets can be substituted for the s -type character. The address argument is a pointer to an array of characters (char *arg*[]).

These square brackets surround a set of characters that define a *search set* of possible characters making up the string (the input field).

If the first character in the brackets is a caret (^), the search set is inverted to include all ASCII characters except those between the square brackets. (Normally, a caret will be included in the inverted search set unless explicitly listed somewhere after the first caret.)

The input field is a string not delimited by whitespace. ...**scanf** reads the corresponding input field up to the first character it reaches that does not appear in the search set (or in the inverted search set). Two examples of this type of conversion are

- %[abcd] Searches for any of the characters a , b , c , and d in the input field.
- %[^abcd] Searches for any characters *except* a , b , c , and d in the input field.

You can also use a range facility shortcut to define a range of characters (numerals or letters) in the search set. For example, to catch all decimal digits, you could define the search set by using %[0123456789], or you could use the shortcut to define the same search set by using %[0-9].

To catch alphanumeric characters, use the following shortcuts:

<code>%[A-Z]</code>	Catches all uppercase letters.
<code>%[0-9A-Za-z]</code>	Catches all decimal digits and all letters (uppercase and lowercase).
<code>%[A-FT-Z]</code>	Catches all uppercase letters from <i>A</i> through <i>F</i> and from <i>T</i> through <i>Z</i> .

The rules covering these search set ranges are straightforward:

- The character prior to the hyphen (-) must be lexically less than the one after it.
- The hyphen must not be the first nor the last character in the set. (If it is first or last, it is considered to just be the hyphen character, not a range definer.)
- The characters on either side of the hyphen must be the ends of the range and not part of some other range.

Here are some examples where the hyphen just means the hyphen character, not a range between two ends:

<code>%[-+*/]</code>	The four arithmetic operations
<code>%[z-a]</code>	The characters <i>z</i> , <i>-</i> , and <i>a</i>
<code>%[+0-9-A-Z]</code>	The characters <i>+</i> and <i>-</i> and the ranges 0-9 and A-Z
<code>%[+0-9A-Z-]</code>	Also the characters <i>+</i> and <i>-</i> and the ranges 0-9 and A-Z
<code>%[^-0-9+A-Z]</code>	All characters except <i>+</i> and <i>-</i> and those in the ranges 0-9 and A-Z

%e, %E, %f, %g, and %G (floating-point) conversions

Floating-point numbers in the input field must conform to the following generic format:

```
[+/-] dddddddd [.] dddd [E | e] [+/-] ddd
```

where *[item]* indicates that *item* is optional, and *ddd* represents decimal, octal, or hexadecimal digits.

INF = infinity; NAN = not a number

In addition, *+INF*, *-INF*, *+NAN*, and *-NAN* are recognized as floating-point numbers. Note that the sign and capitalization are required.

%d, %i, %o, %x, %D, %I, %O, %X, %c, %n conversions

A pointer to **unsigned** character, **unsigned** integer, or **unsigned long** can be used in any conversion where a pointer to a character, integer, or **long** is allowed.

Assignment-suppression character

The assignment-suppression character is an asterisk (*); it is not to be confused with the C indirection (pointer) operator (also an asterisk).

If the asterisk follows the percent sign (%) in a format specifier, the next input field will be scanned but will not be assigned to the next address argument. The suppressed input data is assumed to be of the type specified by the type character that follows the asterisk character.

The success of literal matches and suppressed assignments is not directly determinable.

Width specifiers

The width specifier (*n*), a decimal integer, controls the maximum number of characters that will be read from the current input field.

If the input field contains fewer than *n* characters, ...**scanf** reads all the characters in the field, then proceeds with the next field and format specifier.

If a whitespace or nonconvertible character occurs before width characters are read, the characters up to that character are read, converted, and stored, then the function attends to the next format specifier.

A nonconvertible character is one that cannot be converted according to the given format (such as an 8 or 9 when the format is octal, or a *J* or *K* when the format is hexadecimal or decimal).

Width specifier	How width of stored input is affected
n	Up to <i>n</i> characters are read, converted, and stored in the current address argument.

Input-size and argument-type modifiers

The input-size modifiers (*N* and *F*) and argument-type modifiers (*h*, *l*, and *L*) affect how the ...**scanf** functions interpret the corresponding address argument *arglf*.

F and *N* override the default or declared size of *arg*.

h, *l*, and *L* indicate which type (version) of the following input data is to be used (*h* = **short**, *l* = **long**, *L* = **long double**). The input data will be converted to the specified version, and the *arg* for that input data should point to an object of the corresponding size (**short** object for %*h*, **long** or **double** object for %*l*, and **long double** object for %*L*).

Modifier	How conversion is affected
F	Overrides default or declared size; <i>arg</i> interpreted as far pointer.
N	Overrides default or declared size; <i>arg</i> interpreted as near pointer. Cannot be used with any conversion in huge model.
h	For <i>d, i, o, u, x</i> types, convert input to short int , store in short object. For <i>D, I, O, U, X</i> types, no effect. For <i>e, f, c, s, n, p</i> types, no effect.
l	For <i>d, i, o, u, x</i> types, convert input to long int , store in long object. For <i>e, f, g</i> types, convert input to double , store in double object. For <i>D, I, O, U, X</i> types, no effect. For <i>c, s, n, p</i> types, no effect.
L	For <i>e, f, g</i> types, convert input to a long double , store in long double object. L has no effect on other formats.

When scanf stops scanning

scanf might stop scanning a particular field before reaching the normal field-end character (whitespace), or might terminate entirely, for a variety of reasons.

scanf stops scanning and storing the current field and proceed to the next input field if any of the following occurs:

- An assignment-suppression character (*) appears after the percent character in the format specifier; the current input field is scanned but not stored.
- *width* characters have been read (*width* = width specification, a positive decimal integer in the format specifier).
- The next character read cannot be converted under the current format (for example, an *A* when the format is decimal).
- The next character in the input field does not appear in the search set (or does appear in an inverted search set).

When **scanf** stops scanning the current input field for one of these reasons, the next character is assumed to be unread and to be the first character of the following input field, or the first character in a subsequent read operation on the input.

scanf will terminate under the following circumstances:

- The next character in the input field conflicts with a corresponding non-whitespace character in the format string.
- The next character in the input field is EOF.

- The format string has been exhausted.

If a character sequence that is not part of a format specifier occurs in the format string, it must match the current sequence of characters in the input field; **scanf** will scan but not store the matched characters. When a conflicting character occurs, it remains in the input field as if it were never read.

Return value **scanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored.

If **scanf** attempts to read at end-of-file, the return value is EOF.

If no fields were stored, the return value is 0.

Portability **scanf** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie. This function is not Windows compatible.

See also **atoi**, **cscanf**, **fscanf**, **getc**, **printf**, **sscanf**, **vfscanf**, **vscanf**, **vsscanf**

Example

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

int main(void)
{
    char label[20];
    char name[20];
    int entries = 0;
    int loop, age;
    double salary;
    struct Entry_struct {
        char name[20];
        int age;
        float salary;
    } entry[20];

    /* input a label as character string restricted to 20 characters */
    printf("\n\nPlease enter a label for the chart: ");
    scanf("%20s", label);
    fflush(stdin); /* flush input stream in case of bad input */

    /* input number of entries as integer */
    printf("How many entries will there be? (less than 20) ");
    scanf("%d", &entries);

    /* flush the input stream in case of bad input */
    fflush(stdin);

    /* input a name, restricting input to only upper- or lowercase letters */
    for (loop=0; loop<entries; ++loop) {
```

scanf

```
printf("Entry %d\n", loop);
printf(" Name   : ");
scanf("[A-Za-z]", entry[loop].name);
fflush(stdin); /* flush input stream in case of bad input */

/* input an age as integer */
printf(" Age    : ");
scanf("%d", &entry[loop].age);
fflush(stdin); /* flush input stream in case of bad input */

/* input a salary as a float */
printf(" Salary : ");
scanf("%f", &entry[loop].salary);
fflush(stdin); /* flush input stream in case of bad input */
}

/* input name, age, and salary as string, integer, and double */
printf("\nPlease enter your name, age and salary\n");
scanf("%20s %d %lf", name, &age, &salary);

/* print out the data that was input */
printf("\n\nTable %s\n", label);
printf("Compiled by %s age %d $%15.2lf\n", name, age, salary);
printf("-----\n");
for (loop=0; loop<entries; ++loop)
    printf("%4d | %-20s | %5d | %15.2lf\n", loop + 1, entry[loop].name,
           entry[loop].age, entry[loop].salary);
printf("-----\n");
return 0;
}
```

searchpath

- Function** Searches the DOS path for a file.
- Syntax** `#include <dir.h>`
`char *searchpath(const char *file);`
- Prototype in** dir.h
- Remarks** **searchpath** attempts to locate *file*, searching along the DOS path, which is the PATH=... string in the environment. A pointer to the complete path-name string is returned as the function value.
- searchpath** searches for the file in the current directory of the current drive first. If the file is not found there, the PATH environment variable is fetched, and each directory in the path is searched in turn until the file is found, or the path is exhausted.

When the file is located, a string is returned containing the full path name. This string can be used in a call to access the file (for example, with **fopen** or **exec...**).

The string returned is located in a static buffer and is overwritten on each subsequent call to **searchpath**.

Return value **searchpath** returns a pointer to a file name string if the file is successfully located; otherwise, **searchpath** returns null.

Portability **searchpath** is unique to DOS.

See also **exec...**, **findfirst**, **findnext**, **spawn...**, **system**

Example

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

int main(void)
{
    char *p;

    /* looks for TLINK and returns a pointer to the path */
    p = searchpath("TLINK.EXE");
    printf("Search for TLINK.EXE : %s\n", p);

    /* looks for non-existent file */
    p = searchpath("NOTEXIST.FIL");
    printf("Search for NOTEXIST.FIL : %s\n", p);
    return 0;
}
```

Program output

```
Search for TLINK.EXE : C:\BIN\TLINK.EXE
Search for NOTEXIST.FIL : (NULL)
```

S

sector

Function Draws and fills an elliptical pie slice.

Syntax `#include <graphics.h>`
`void far sector(int x, int y, int stangle, int endangle, int xradius, int yradius);`

Prototype in graphics.h

Remarks Draws and fills an elliptical pie slice using (x,y) as the center point, *xradius* and *yradius* as the horizontal and vertical radii, respectively, and drawing from *stangle* to *endangle*. The pie slice is outlined using the current color,

sector

and filled using the pattern and color defined by **setfillstyle** or **setfillpattern**.

The angles for **sector** are given in degrees. They are measured counter-clockwise with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

If an error occurs while the pie slice is filling, **graphresult** returns a value of `-6` (`grNoScanMem`).

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **arc**, **circle**, **ellipse**, **getarcoords**, **getaspectratio**, **graphresult**, **pieslice**, **setfillpattern**, **setfillstyle**, **setgraphbufsize**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;
    int stangle = 45, endangle = 135;
    int xrad = 100, yrad = 50;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* loop through the fill patterns */
    for (i=EMPTY_FILL; i<USER_FILL; i++) {
        /* set the fill style */
        setfillstyle(i, getmaxcolor());
    }
}
```

```

    /* draw the sector slice */
    sector(midx, midy, stangle, endangle, xrad, yrad);
    getch();
}

/* clean up */
closegraph();
return 0;
}

```

segread

Function Reads segment registers.

Syntax `#include <dos.h>`
`void segread(struct SREGS *segs);`

Prototype in dos.h

Remarks **segread** places the current values of the segment registers into the structure pointed to by *segs*.

This call is intended for use with **intdosx** and **int86x**.

Return value None.

Portability **segread** is unique to the 8086 family of processors.

See also **FP_OFF**, **int86**, **int86x**, **intdos**, **intdosx**, **MK_FP**, **movedata**

Example

```

#include <stdio.h>
#include <dos.h>

int main(void)
{
    struct SREGS segs;
    segread(&segs);
    printf("Current segment register settings\n\n");
    printf("CS: %X DS: %X\n", segs.cs, segs.ds);
    printf("ES: %X SS: %X\n", segs.es, segs.ss);
    return 0;
}

```


setactivepage

- Function** Sets active page for graphics output.
- Syntax** `#include <graphics.h>`
`void far setactivepage(int page);`
- Prototype in** `graphics.h`
- Remarks** **setactivepage** makes *page* the active graphics page. All subsequent graphics output will be directed to that graphics page.
- The active graphics page might not be the one you see onscreen, depending on how many graphics pages are available on your system. Only the EGA, VGA, and Hercules graphics cards support multiple pages.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **setvisualpage**

Example

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

int main(void)
{
    /* select driver and mode that supports multiple pages */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;
    ht = textheight("W");
```

```

/* select the off screen page for drawing */
setactivepage(1);

/* draw a line on page #1 */
line(0, 0, getmaxx(), getmaxy());

/* output a message on page #1 */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(x, y, "This is page #1:");
outtextxy(x, y+ht, "Press any key to halt:");

/* select drawing to page #0 */
setactivepage(0);

/* output a message on page #0 */
outtextxy(x, y, "This is page #0.");
outtextxy(x, y+ht, "Press any key to view page #1:");
getch();

/* select page #1 as the visible page */
setvisualpage(1);

/* clean up */
getch();
closegraph();
return 0;
}

```

setallpalette

Function Changes all palette colors as specified.

Syntax `#include <graphics.h>`
`void far setallpalette(struct palettetype far *palette);`

Prototype in graphics.h

Remarks **setallpalette** sets the current palette to the values given in the **palettetype** structure pointed to by *palette*.

You can partially (or completely) change the colors in the EGA/VGA palette with **setallpalette**.

The MAXCOLORS constant and the **palettetype** structure used by **setallpalette** are defined in graphics.h as follows:

```

#define MAXCOLORS 15

struct palettetype {
    unsigned char size;

```

setallpalette

```
    signed char colors[MAXCOLORS + 1];  
};
```

size gives the number of colors in the palette for the current graphics driver in the current mode.

colors is an array of *size* bytes containing the actual raw color numbers for each entry in the palette. If an element of *colors* is -1 , the palette color for that entry is not changed.

The elements in the *colors* array used by **setallpalette** can be represented by symbolic constants defined in `graphics.h`.

Table 2.6
Actual color table

CGA		EGA/VGA	
Name	Value	Name	Value
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

Note that valid colors depend on the current graphics driver and current graphics mode.

Changes made to the palette are seen immediately onscreen. Each time a palette color is changed, all occurrences of that color onscreen will change to the new color value.



setallpalette cannot be used with the IBM-8514 driver.

Return value

If invalid input is passed to **setallpalette**, **graphresult** returns -11 (`grError`), and the current palette remains unchanged.

Portability

This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also

getpalette, **getpalettesize**, **graphresult**, **setbkcolor**, **setcolor**, **setpalette**

Example

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct palettetype pal;
    int color, maxcolor, ht;
    int y = 10;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    maxcolor = getmaxcolor();
    ht = 2 * textheight("W");

    /* grab a copy of the palette */
    getpalette(&pal);

    /* display the default palette colors */
    for (color=1; color<=maxcolor; color++) {
        setcolor(color);
        sprintf(msg, "Color: %d", color);
        outtextxy(1, y, msg);
        y += ht;
    }

    /* wait for a key */
    getch();

    /* black out the colors one by one */
    for (color=1; color<=maxcolor; color++) {
        setpalette(color, BLACK);
        getch();
    }

    /* restore the palette colors */
    setallpalette(&pal);
}

```

```

    /* clean up */
    getch();
    closegraph();
    return 0;
}

```

setaspectratio

Function Changes the default aspect ratio correction factor.

Syntax `#include <graphics.h>`
`void far setaspectratio(int xasp, int yasp);`

Prototype in graphics.h

Remarks **setaspectratio** changes the default aspect ratio of the graphics system. The graphics system uses the aspect ratio to make sure that circles are round onscreen. If circles appear elliptical, the monitor is not aligned properly. You could correct this in the hardware by realigning the monitor, but it's easier to change in the software by using **setaspectratio** to set the aspect ratio. To obtain the current aspect ratio from the system, call **getaspectratio**.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **circle**, **getaspectratio**

Example

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int xasp, yasp, midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */

```

```

{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;
setcolor(getmaxcolor());

/* get current aspect ratio settings */
getaspectratio(&xasp, &yasp);

/* draw normal circle */
circle(midx, midy, 100);
getch();

/* clear the screen */
cleardevice();

/* adjust the aspect for a wide circle */
setaspectratio(xasp/2, yasp);
circle(midx, midy, 100);
getch();

/* adjust the aspect for a narrow circle */
cleardevice();
setaspectratio(xasp, yasp/2);
circle(midx, midy, 100);

/* clean up */
getch();
closegraph();
return 0;
}

```

setbkcolor

Function Sets the current background color using the palette.

Syntax #include <graphics.h>
void far setbkcolor(int *color*);

Prototype in graphics.h

Remarks **setbkcolor** sets the background to the color specified by *color*. The argument *color* can be a name or a number, as listed in the following table:

setbkcolor

These symbolic names are defined in graphics.h.

Number	Name	Number	Name
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

For example, if you want to set the background color to blue, you can call

```
setbkcolor(BLUE) /* or */ setbkcolor(1)
```

On CGA and EGA systems, **setbkcolor** changes the background color by changing the first entry in the palette.



If you use an EGA or a VGA, and you change the palette colors with **setpalette** or **setallpalette**, the defined symbolic constants might not give you the correct color. This is because the parameter to **setbkcolor** indicates the entry number in the current palette rather than a specific color (unless the parameter passed is 0, which always sets the background color to black).

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **getbkcolor**, **setallpalette**, **setcolor**, **setpalette**

Example

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

int main(void)
{
    /* select driver and mode that supports multiple background colors */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int bkcol, maxcolor, x, y;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
```

```

{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
}

/* maximum color index supported */
maxcolor = getmaxcolor();

/* for centering text messages */
settextjustify(CENTER_TEXT, CENTER_TEXT);
x = getmaxx() / 2;
y = getmaxy() / 2;

/* loop through the available colors */
for (bkcol=0; bkcol<=maxcolor; bkcol++) {
    /* clear the screen */
    cleardevice();

    /* select a new background color */
    setbkcolor(bkcol);

    /* output a message */
    if (bkcol == WHITE)
        setcolor(EGA_BLUE);
    sprintf(msg, "Background color: %d", bkcol);
    outtextxy(x, y, msg);
    getch();
}

/* clean up */
closegraph();
return 0;
}

```

setblock

- Function** Modifies the size of a previously allocated block.
- Syntax** `#include <dos.h>`
`int setblock(unsigned segx, unsigned newsize);`
- Prototype in** dos.h
- Remarks** **setblock** modifies the size of a memory segment. *segx* is the segment address returned by a previous call to **allocmem**. *newsize* is the new, requested size in paragraphs.

setblock

Return value **setblock** returns `-1` on success. In the event of error, it returns the size of the largest possible block (in paragraphs), and the global variable `_doserrno` is set.

Portability **setblock** is unique to DOS. This function is not Windows compatible.

See also **allocmem**, **freemem**

Example

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

int main(void)
{
    unsigned int size, segp;
    int stat;
    size = 64; /* (64 x 16) = 1024 bytes */
    stat = allocmem(size, &segp);
    if (stat == -1)
        printf("Allocated memory at segment: %X\n", segp);
    else {
        printf("Failed: maximum number of paragraphs available is %d\n", stat);
        exit(1);
    }

    stat = setblock(segp, size * 2);
    if (stat == -1)
        printf("Expanded memory block at segment: %X\n", segp);
    else
        printf("Failed: maximum number of paragraphs available is %d\n", stat);
    freemem(segp);
    return 0;
}
```

setbuf

Function Assigns buffering to a stream.

Syntax `#include <stdio.h>`
`void setbuf(FILE *stream, char *buf);`

Prototype in `stdio.h`

Remarks **setbuf** causes the buffer *buf* to be used for I/O buffering instead of an automatically allocated buffer. It is used after *stream* has been opened.

512

If *buf* is null, I/O will be unbuffered; otherwise, it will be fully buffered. The buffer must be BUFSIZ bytes long (specified in `stdio.h`).

stdin and *stdout* are unbuffered if they are not redirected; otherwise, they are fully buffered. **setbuf** can be used to change the buffering style being used.

Unbuffered means that characters written to a stream are immediately output to the file or device, while *buffered* means that the characters are accumulated and written as a block.

setbuf produces unpredictable results unless it is called immediately after opening *stream* or after a call to **fseek**. Calling **setbuf** after *stream* has been unbuffered is legal and will not cause problems.

A common cause for error is to allocate the buffer as an automatic (local) variable and then fail to close the file before returning from the function where the buffer was declared.

Return value None.

Portability **setbuf** is available on UNIX systems and is defined in ANSI C.

See also **fflush**, **fopen**, **fseek**, **setvbuf**

Example `#include <stdio.h>`

```
/* bufsiz is defined in stdio.h */
char outbuf[BUFSIZ];

int main(void)
{
    /* attach buffer to standard output stream */
    setbuf(stdout, outbuf);

    /* put some characters into the buffer */
    puts("This is a test of buffered output.\n\n");
    puts("This output will go into outbuf\n");
    puts("and won't appear until the buffer\n");
    puts("fills up or we flush the stream.\n");

    /* flush the output buffer */
    fflush(stdout);
    return 0;
}
```

S

setcbk

Function	Sets control-break setting.
Syntax	<code>#include <dos.h></code> <code>int setcbk(int <i>cbkvalue</i>);</code>
Prototype in	dos.h
Remarks	<p>setcbk uses the DOS system call 0x33 to turn control-break checking on or off.</p> <p><i>value</i> = 0 Turns checking off (check only during I/O to console, printer, or communications devices).</p> <p><i>value</i> = 1 Turns checking on (check at every system call).</p>
Return value	setcbk returns <i>cbkvalue</i> , the value passed.
Portability	setcbk is unique to DOS.
See also	getcbrk
Example	<pre>#include <dos.h> #include <conio.h> #include <stdio.h> int main(void) { int break_flag; printf("Enter 0 to turn control break off\n"); printf("Enter 1 to turn control break on\n"); break_flag = getch() - 0; setcbk(break_flag); if (getcbrk()) printf("Cntrl-brk flag is on\n"); else printf("Cntrl-brk flag is off\n"); return 0; }</pre>

setcolor

Function Sets the current drawing color using the palette.

Syntax `#include <graphics.h>`
`void far setcolor(int color);`

Prototype in graphics.h

Remarks **setcolor** sets the current drawing color to *color*, which can range from 0 to **getmaxcolor**.

The current drawing color is the value to which pixels are set when lines, and so on are drawn. The following tables show the drawing colors available for the CGA and EGA, respectively.

Palette number	Constant assigned to color number (pixel value)		
	1	2	3
0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

Numeric value		Symbolic name	
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

You select a drawing color by passing either the color number itself or the equivalent symbolic name to **setcolor**. For example, in CGAC0 mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, either **setcolor(3)** or **setcolor(CGA_YELLOW)** selects a drawing color of yellow.

Return value None.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.



See also `getcolor`, `getmaxcolor`, `graphresult`, `setallpalette`, `setbkcolor`, `setpalette`

Example

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

int main(void)
{
    /* select driver and mode that supports multiple drawing colors */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int color, maxcolor, x, y;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* maximum color index supported */
    maxcolor = getmaxcolor();

    /* for centering text messages */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* loop through the available colors */
    for (color=1; color<=maxcolor; color++) {
        cleardevice(); /* clear the screen */
        setcolor(color); /* select new background color */

        /* output a message */
        sprintf(msg, "Color: %d", color);
        outtextxy(x, y, msg);
        getch();
    }

    /* clean up */
    closegraph();
    return 0;
}
```

`_setcursortype`

Function Selects cursor appearance.

Syntax `#include <conio.h>`
`void _setcursortype(int cur_t);`

Prototype in `<conio.h>`

Remarks Sets the cursor type to

<code>_NOCURSOR</code>	Turns off the cursor
<code>_SOLIDCURSOR</code>	Solid block cursor
<code>_NORMALCURSOR</code>	Normal underscore cursor

Return value None.

Portability It works only with IBM PCs and compatibles. This function is not Windows compatible.

Example

```
#include <conio.h>

int main(void)
{
    /* display the normal cursor */
    printf("\n\rNormal Cursor: ");
    getch();

    /* turn off the cursor */
    _setcursortype(_NOCURSOR);
    printf("\n\rNo Cursor   : ");
    getch();

    /* switch to a solid cursor */
    _setcursortype(_SOLIDCURSOR);
    printf("\n\rSolid Cursor : ");
    getch();

    /* switch back to the normal cursor */
    _setcursortype(_NORMALCURSOR);
    printf("\n\rNormal Cursor: ");
    getch();
    return 0;
}
```

setdate

Function Sets DOS date.

Syntax `#include <dos.h>`
`void setdate(struct date *datep);`

Prototype in dos.h

Remarks **setdate** sets the system date (month, day, and year) to that in the **date** structure pointed to by *datep*.

The **date** structure is defined as follows:

```
struct date {
    int da_year;      /* current year */
    char da_day;     /* day of the month */
    char da_mon;     /* month (1 = Jan) */
};
```

Return value None.

Portability **setdate** is unique to DOS.

See also **getdate**, **gettime**, **settime**

Example

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

int main(void)
{
    struct date reset;
    reset.da_year = 2001;
    reset.da_day = 1;
    reset.da_mon = 1;
    setdate(&reset);
    system("date");
    return 0;
}
```

setdisk

- Function** Sets current disk drive.
- Syntax** `#include <dir.h>`
`int setdisk(int drive);`
- Prototype in** `dir.h`
- Remarks** **setdisk** sets the current drive to the one associated with *drive*: 0 for A, 1 for B, 2 for C, and so on (equivalent to DOS call 0x0E).
- Return value** **setdisk** returns the total number of drives available.
- Portability** **setdisk** is unique to DOS.
- See also** **getdisk**
- Example**
- ```
#include <stdio.h>
#include <dir.h>

int main(void)
{
 int maxdrives;
 maxdrives = setdisk(2);
 printf("The number of logical drives are: %d\n", maxdrives);
 return 0;
}
```

## setdta

---

- Function** Sets disk-transfer address.
- Syntax** `#include <dos.h>`  
`void setdta(char far *dta);`
- Prototype in** `dos.h`
- Remarks** **setdta** changes the current setting of the DOS disk-transfer address (DTA) to the value given by *dta*.
- Return value** None.
- Portability** **setdta** is unique to DOS.
- See also** **getdta**



**Example**

```

#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
 char line[80], far *save_dta;
 char buffer[256] = "SETDTA test!";
 struct fcb blk;
 int result;

 /* get new file name from user */
 printf("Enter a file name to create:");
 gets(line);

 /* parse the new file name to the dta */
 parsfnm(line, &blk, 1);
 printf("%d %s\n", blk.fcb_drive, blk.fcb_name);

 /* request DOS services to create file */
 if (bdosptr(0x16, &blk, 0) == -1) {
 perror("Error creating file");
 exit(1);
 }

 /* save old dta and set new dta */
 save_dta = getdta();
 setdta(buffer);

 /* write new records */
 blk.fcb_recsize = 256;
 blk.fcb_random = 0L;
 result = randbwr(&blk, 1);
 printf("result = %d\n", result);

 if (!result)
 printf("Write OK\n");
 else {
 perror("Disk error");
 exit(1);
 }

 /* request DOS services to close the file */
 if (bdosptr(0x10, &blk, 0) == -1) {
 perror("Error closing file");
 exit(1);
 }

 /* reset the old dta */
 setdta(save_dta);
}

```

```

 return 0;
}

```

## setfillpattern

---

**Function** Selects a user-defined fill pattern.

**Syntax** `#include <graphics.h>`  
`void far setfillpattern(char far *upattern, int color);`

**Prototype in** `graphics.h`

**Remarks** **setfillpattern** is like **setfillstyle**, except that you use it to set a user-defined 8×8 pattern rather than a predefined pattern.

*upattern* is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel is plotted.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **getfillpattern**, **getfillsettings**, **graphresult**, **sector**, **setfillstyle**

**Example**

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int maxx, maxy;

 /* a user-defined fill pattern */
 char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x24, 0x24, 0x07, 0x00};

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 }
}

```

## setfillpattern

```
 getch();
 exit(1); /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();
setcolor(getmaxcolor());

/* select a user-defined fill pattern */
setfillpattern(pattern, getmaxcolor());

/* fill the screen with the pattern */
bar(0, 0, maxx, maxy);

/* clean up */
getch();
closegraph();
return 0;
}
```

## setfillstyle

---

- Function** Sets the fill pattern and color.
- Syntax** `#include <graphics.h>`  
`void far setfillstyle(int pattern, int color);`
- Prototype in** `graphics.h`
- Remarks** **setfillstyle** sets the current fill pattern and fill color. To set a user-defined fill pattern, do *not* give a *pattern* of 12 (USER\_FILL) to **setfillstyle**; instead, call **setfillpattern**.
- The enumeration *fill\_patterns*, defined in `graphics.h`, gives names for the predefined fill patterns, plus an indicator for a user-defined pattern.

| Name            | Value | Description                |
|-----------------|-------|----------------------------|
| EMPTY_FILL      | 0     | Fill with background color |
| SOLID_FILL      | 1     | Solid fill                 |
| LINE_FILL       | 2     | Fill with —                |
| LTSLASH_FILL    | 3     | Fill with ///              |
| SLASH_FILL      | 4     | Fill with ///, thick lines |
| BKSLASH_FILL    | 5     | Fill with \\\, thick lines |
| LTKSLASH_FILL   | 6     | Fill with \\               |
| HATCH_FILL      | 7     | Light hatch fill           |
| XHATCH_FILL     | 8     | Heavy crosshatch fill      |
| INTERLEAVE_FILL | 9     | Interleaving line fill     |
| WIDE_DOT_FILL   | 10    | Widely spaced dot fill     |
| CLOSE_DOT_FILL  | 11    | Closely spaced dot fill    |
| USER_FILL       | 12    | User-defined fill pattern  |

All but EMPTY\_FILL fill with the current fill color; EMPTY\_FILL uses the current background color.

If invalid input is passed to **setfillstyle**, **graphresult** returns -11 (grError), and the current fill pattern and fill color remain unchanged.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **bar**, **bar3d**, **fillpoly**, **floodfill**, **getfillsettings**, **graphresult**, **pieslice**, **sector**, **setfillpattern**

**Example**

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL", "LTSLASH_FILL",
 "SLASH_FILL", "BKSLASH_FILL", "LTKSLASH_FILL", "HATCH_FILL",
 "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL",
 "CLOSE_DOT_FILL", "USER_FILL" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int style, midx, midy;
 char stylestr[40];
```

## setfillstyle

```
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;
for (style = EMPTY_FILL; style < USER_FILL; style++)
{
 /* select the fill style */
 setfillstyle(style, getmaxcolor());

 /* convert style into a string */
 strcpy(stylestr, fname[style]);

 /* fill a bar */
 bar3d(0, 0, midx-10, midy, 0, 0);

 /* output a message */
 outtextxy(midx, midy, stylestr);

 /* wait for a key */
 getch();
 cleardevice();
}

/* clean up */
getch();
closegraph();
return 0;
}
```

## setftime

---

**Function** Sets file date and time.

**Syntax** #include <io.h>  
int setftime(int *handle*, struct ftime \**ftimep*);

**Prototype in** io.h

**Remarks** **setftime** sets the file date and time of the disk file associated with the open *handle* to the date and time in the **ftime** structure pointed to by *ftimep*.

The **ftime** structure is defined as follows:

```
struct ftime {
 unsigned ft_tsec: 5; /* two seconds */
 unsigned ft_min: 6; /* minutes */
 unsigned ft_hour: 5; /* hours */
 unsigned ft_day: 5; /* days */
 unsigned ft_month: 4; /* months */
 unsigned ft_year: 7; /* year - 1980 */
};
```

**Return value** **setftime** returns 0 on success.

In the event of an error, -1 is returned and the global variable *errno* is set to one of the following:

|         |                         |
|---------|-------------------------|
| EINVFNC | Invalid function number |
| EBADF   | Bad file number         |

**Portability** **setftime** is unique to DOS.

**See also** **getftime**

**Example**

```
#include <stdio.h>
#include <process.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
 struct ftime filet;
 FILE *fp;
 if ((fp = fopen("TEST.$$$", "w")) == NULL) {
 perror("Error:");
 exit(1);
 }

 fprintf(fp, "testing...\n");
 filet.ft_tsec = 1;
 filet.ft_min = 1;
 filet.ft_hour = 1;
 filet.ft_day = 1;
 filet.ft_month = 1;
 filet.ft_year = 21;
 system("dir TEST.$$$");
 setftime(fileno(fp), &filet);
 system("dir TEST.$$$");
 fclose(fp);
}
```

```

 unlink("TEST.$$$");
 return 0;
}

```

## setgraphbufsize

---

- Function** Changes the size of the internal graphics buffer.
- Syntax** `#include <graphics.h>`  
`unsigned far setgraphbufsize(unsigned bufsize);`
- Prototype in** `graphics.h`
- Remarks** Some of the graphics routines (such as **floodfill**) use a memory buffer that is allocated when **initgraph** is called, and released when **closegraph** is called. The default size of this buffer, allocated by **\_graphgetmem**, is 4,096 bytes.
- You might want to make this buffer smaller (to save memory space) or bigger (if, for example, a call to **floodfill** produces error -7: Out of flood memory).
- setgraphbufsize** tells **initgraph** how much memory to allocate for this internal graphics buffer when it calls **\_graphgetmem**.
- ➔ You *must* call **setgraphbufsize** before calling **initgraph**. Once **initgraph** has been called, all calls to **setgraphbufsize** are ignored until after the next call to **closegraph**.
- Return value** **setgraphbufsize** returns the previous size of the internal buffer.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **closegraph**, **\_graphfreemem**, **\_graphgetmem**, **initgraph**, **sector**
- Example**
- ```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define BUFSIZE 1000 /* internal graphics buffer size */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

```

```

int x, y, oldsize;
char msg[80];

/* set size of internal graphics buffer before calling initgraph */
oldsize = setgraphbufsize(BUFSIZE);

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

x = getmaxx() / 2;
y = getmaxy() / 2;

/* output some messages */
sprintf(msg, "Graphics buffer size: %d", BUFSIZE);
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(x, y, msg);
sprintf(msg, "Old graphics buffer size: %d", oldsize);
outtextxy(x, y+textheight("W"), msg);

/* clean up */
getch();
closegraph();
return 0;
}

```

setgraphmode

Function	Sets the system to graphics mode and clears the screen.
Syntax	#include <graphics.h> void far setgraphmode(int <i>mode</i>);
Prototype in	graphics.h
Remarks	setgraphmode selects a graphics mode different than the default one set by initgraph . <i>mode</i> must be a valid mode for the current device driver. setgraphmode clears the screen and resets all graphics settings to their defaults (current position, palette, color, viewport, and so on).

setgraphmode

You can use **setgraphmode** in conjunction with **restorecrtmode** to switch back and forth between text and graphics modes.

Return value If you give **setgraphmode** an invalid mode for the current device driver, **graphresult** returns a value of `-10` (`grInvalidMode`).

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **getgraphmode**, **getmoderange**, **graphresult**, **initgraph**, **restorecrtmode**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int x, y;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* output a message */
    setttextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press any key to exit graphics:");
    getch();

    /* restore system to text mode */
    restorecrtmode();
    printf("We're now in text mode.\n");
    printf("Press any key to return to graphics mode:");
    getch();

    /* return to graphics mode */
    setgraphmode(getgraphmode());
}
```

```

/* output a message */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(x, y, "We're back in graphics mode.");
outtextxy(x, y+textheight("W"), "Press any key to halt:");

/* clean up */
getch();
closegraph();
return 0;
}

```

setjmp

Function Sets up for nonlocal goto.

Syntax `#include <setjmp.h>`
`int setjmp(jmp_buf jmpb);`

Prototype in setjmp.h

Remarks **setjmp** captures the complete *task state* in *jmpb* and returns 0.

A later call to **longjmp** with *jmpb* restores the captured task state and returns in such a way that **setjmp** appears to have returned with the value *val*.

A task state is

- all segment registers (CS, DS, ES, SS)
- register variables (SI, DI)
- stack pointer (SP)
- frame base pointer (BP)
- flags

A task state is complete enough that **setjmp** can be used to implement coroutines.

setjmp must be called before **longjmp**. The routine that calls **setjmp** and sets up *jmpb* must still be active and cannot have returned before the **longjmp** is called. If it has returned, the results are unpredictable.

setjmp is useful for dealing with errors and exceptions encountered in a low-level subroutine of a program.

➔ You can't use **setjmp** and **longjmp** for implementing coroutines if your program is overlaid. Normally, **setjmp** and **longjmp** save and restore all

the registers needed for coroutines, but the overlay manager needs to keep track of stack contents and assumes there is only one stack. When you implement coroutines there are usually either two stacks or two partitions of one stack, and the overlay manager will not track them properly.

You can have background tasks which run with their own stacks or sections of stack, but you must ensure that the background tasks do not invoke any overlaid code, and you must not use the overlay versions of **setjmp** or **longjmp** to switch to and from background. When you avoid using overlay code or support routines, the existence of the background stacks does not disturb the overlay manager.

Return value **setjmp** returns 0 when it is initially called. If the return is from a call to **longjmp**, **setjmp** returns a nonzero value (as in the example).

Portability **setjmp** is available on UNIX systems and is defined in ANSI C. This function is not Windows compatible.

See also **longjmp**, **signal**

Example

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

void subroutine(void);

jmp_buf jumper;

int main()
{
    int value;
    value = setjmp(jumper);
    if (value != 0) {
        printf("Longjmp with value %d\n", value);
        exit(value);
    }
    printf("About to call subroutine ... \n");
    subroutine();
    return 0;
}

void subroutine(void) {
    longjmp(jumper, 1);
}
```

setlinestyle

Function Sets the current line width and style.

Syntax `#include <graphics.h>`
`void far setlinestyle(int linestyle, unsigned upattern, int thickness);`

Prototype in graphics.h

Remarks **setlinestyle** sets the style for all lines drawn by **line**, **lineto**, **rectangle**, **drawpoly**, and so on.

The *linesettingstype* structure is defined in graphics.h as follows:

```
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

linestyle specifies in which of several styles subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration *line_styles*, defined in graphics.h, gives names to these operators:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

thickness specifies whether the width of subsequent lines drawn will be normal or thick.

Name	Value	Description
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

upattern is a 16-bit pattern that applies only if *linestyle* is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. For example, a solid line corresponds to a *upattern* of 0xFFFF (all pixels drawn), while a dashed line can correspond to a *upattern* of 0x3333 or 0x0F0F. If the *linestyle* parameter to **setlinestyle** is not USERBIT_LINE (in other words, if

it is not equal to 4), you must still provide the *upattern* parameter, but it will be ignored.



The *linestyle* parameter does not affect arcs, circles, ellipses, or pie slices. Only the *thickness* parameter is used.

Return value If invalid input is passed to **setlinestyle**, **graphresult** returns -11, and the current line style remains unchanged.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **arc**, **bar3d**, **circle**, **drawpoly**, **ellipse**, **getlinesettings**, **graphresult**, **line**, **linereel**, **lineto**, **pieslice**, **rectangle**

Example

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE", "DASHED_LINE",
                  "USERBIT_LINE" };

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy, userpat;
    char stylestr[40];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* a user-defined line pattern */
    /* binary: "0000000000000001" */
    userpat = 1;
}
```

```

for (style=SOLID_LINE; style<=USERBIT_LINE; style++)
{
    /* select the line style */
    setlinestyle(style, userpat, 1);

    /* convert style into a string */
    strcpy(stylestr, lname[style]);

    /* draw a line */
    line(0, 0, midx-10, midy);

    /* draw a rectangle */
    rectangle(0, 0, getmaxx(), getmaxy());

    /* output a message */
    outtextxy(midx, midy, stylestr);

    /* wait for a key */
    getch();
    cleardevice();
}

/* clean up */
closegraph();
return 0;
}

```

setlocale

Function Selects a locale.

Syntax #include <locale.h>
char *setlocale(int *category*, char **locale*);

Prototype in locale.h

Remarks Borland C++ supports only the "C" locale at present, so invoking this function has no effect.

Possible values for the argument *category*:

```

LC_ALL
LC_COLLATE
LC_CTYPE
LC_MONETARY
LC_NUMERIC
LC_TIME

```

Return value

S

setlocale

If selection is successful, a string is returned to indicate the locale that was in effect prior to invoking the function. If it is not successful, a NULL pointer is returned.

Portability **setlocale** is compatible with ANSI C.

See also **localeconv**

Example

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

int main( void )
{
    char *old_locale;

    /* only locale supported in Borland C++ is "C" */
    old_locale = setlocale(LC_ALL,"C");
    printf("Old locale was %s\n",old_locale);
    return 0;
}
```

setmem

Function Assigns a value to a range of memory.

Syntax `#include <mem.h>`
`void setmem(void *dest, unsigned length, char value);`

Prototype in mem.h

Remarks **setmem** sets a block of *length* bytes, pointed to by *dest*, to the byte *value*.

Return value None.

Portability **setmem** is unique to the 8086 family.

See also **memset**, **strset**

Example

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

int main(void)
{
    char *dest;
    dest = calloc(21, sizeof(char));
    setmem(dest, 20, 'c');
    printf("%s\n", dest);
}
```

```

    return 0;
}

```

setmode

Function Sets mode of an open file.

Syntax `#include <fcntl.h>`
`int setmode(int handle, int amode);`

Prototype in `io.h`

Remarks **setmode** sets the mode of the open file associated with *handle* to either binary or text. The argument *amode* must have a value of either `O_BINARY` or `O_TEXT`, never both. (These symbolic constants are defined in `fcntl.h`.)

Return value **setmode** returns 0 if successful. On error it returns `-1` and sets the global variable *errno* to

`EINVAL` Invalid argument

Portability **setmode** is available on UNIX systems.

See also `_creat`, `creat`, `_open`, `open`

Example

```

#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
    int result;
    result = setmode(fileno(stdout), O_TEXT);
    if (result == -1)
        perror("Mode not available\n");
    else
        printf("Mode successfully switched\n");
    return 0;
}

```


setpalette

- Function** Changes one palette color.
- Syntax** `#include <graphics.h>`
`void far setpalette(int colornum, int color);`
- Prototype in** `graphics.h`
- Remarks** **setpalette** changes the *colornum* entry in the palette to *color*. For example, **setpalette**(0,5) changes the first color in the current palette (the background color) to actual color number 5. If *size* is the number of entries in the current palette, *colornum* can range between 0 and (*size* - 1).
- You can partially (or completely) change the colors in the EGA/VGA palette with **setpalette**. On a CGA, you can only change the first entry in the palette (*colornum* equals 0, the background color) with a call to **setpalette**.
- The *color* parameter passed to **setpalette** can be represented by symbolic constants defined in `graphics.h`.

CGA		EGA/VGA	
Name	Value	Name	Value
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

Note that valid colors depend on the current graphics driver and current graphics mode.

Changes made to the palette are seen immediately onscreen. Each time a palette color is changed, all occurrences of that color onscreen change to the new color value.



setpalette cannot be used with the IBM-8514 driver; use **setrgbpalette** instead.

- Return value** If invalid input is passed to **setpalette**, **graphresult** returns -11, and the current palette remains unchanged.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **getpalette**, **graphresult**, **setallpalette**, **setbkcolor**, **setcolor**, **setrgbpalette**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int color, maxcolor, ht;
    int y = 10;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    maxcolor = getmaxcolor();
    ht = 2 * textheight("W");

    /* display the default colors */
    for (color=1; color<=maxcolor; color++) {
        setcolor(color);
        sprintf(msg, "Color: %d", color);
        outtextxy(1, y, msg);
        y += ht;
    }

    /* wait for a key */
    getch();
}
```

setpalette

```
/* black out the colors one by one */
for (color=1; color<=maxcolor; color++) {
    setpalette(color, BLACK);
    getch();
}

/* clean up */
closegraph();
return 0;
}
```

setrgbpalette

- Function** Allows user to define colors for the IBM 8514.
- Syntax** `#include <graphics.h>`
`void far setrgbpalette(int colornum, int red, int green, int blue);`
- Prototype in** `graphics.h`
- Remarks** **setrgbpalette** can be used with the IBM 8514 and VGA drivers.
- colornum* defines the palette entry to be loaded, while *red*, *green*, and *blue* define the component colors of the palette entry.
- For the IBM 8514 display (and the VGA in 256K color mode), *colornum* is in the range 0 to 255. For the remaining modes of the VGA, *colornum* is in the range 0 to 15. Only the lower byte of *red*, *green*, or *blue* is used, and out of each byte, only the 6 most significant bits are loaded in the palette.
- ➡ For compatibility with other IBM graphics adapters, the BGI driver defines the first 16 palette entries of the IBM 8514 to the default colors of the EGA/VGA. These values can be used as is, or they can be changed using **setrgbpalette**.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **setpalette**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
```

```

{
 /* select driver and mode that supports use of setrgbpalette */
 int gdriver = VGA, gmode = VGAHI, errorcode;
 struct palettetype pal;
 int i, ht, y, xmax;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 /* grab a copy of the palette */
 getpalette(&pal);

 /* create gray scale */
 for (i=0; i<pal.size; i++)
 setrgbpalette(pal.colors[i], i*4, i*4, i*4);

 /* display the gray scale */
 ht = getmaxy() / 16;
 xmax = getmaxx();
 y = 0;
 for (i=0; i<pal.size; i++) {
 setfillstyle(SOLID_FILL, i);
 bar(0, y, xmax, y+ht);
 y += ht;
 }

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

## settextjustify

---

**Function** Sets text justification for graphics functions.

**Syntax** `#include <graphics.h>`  
`void far settextjustify(int horiz, int vert);`

**Prototype in** graphics.h

**Remarks** Text output after a call to **settextjustify** is justified around the current position (CP) horizontally and vertically, as specified. The default justification settings are LEFT\_TEXT (for horizontal) and TOP\_TEXT (for vertical). The enumeration *text\_just* in graphics.h provides names for the *horiz* and *vert* settings passed to **settextjustify**.

| Description  | Name        | Value | Action              |
|--------------|-------------|-------|---------------------|
| <i>horiz</i> | LEFT_TEXT   | 0     | left-justify text   |
|              | CENTER_TEXT | 1     | center text         |
|              | RIGHT_TEXT  | 2     | right-justify text  |
| <i>vert</i>  | BOTTOM_TEXT | 0     | justify from bottom |
|              | CENTER_TEXT | 1     | center text         |
|              | TOP_TEXT    | 2     | justify from top    |

If *horiz* is equal to LEFT\_TEXT and *direction* equals HORIZ\_DIR, the CP's *x* component is advanced after a call to **outtext(string)** by **textwidth(string)**.

**settextjustify** affects text written with **outtext** and cannot be used with text mode and stream functions.

**Return value** If invalid input is passed to **settextjustify**, **graphresult** returns -11, and the current text justification remains unchanged.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **gettextsettings**, **graphresult**, **outtext**, **settextstyle**

**Example**

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

/* function prototype */
void xat(int x, int y);

/* horizontal text justification settings */
char *hjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };
```

```

/* vertical text justification settings */
char *vjust[] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int midx, midy, hj, vj;
 char msg[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 midx = getmaxx() / 2;
 midy = getmaxy() / 2;

 /* loop through text justifications */
 for (hj=LEFT_TEXT; hj<=RIGHT_TEXT; hj++)
 for (vj=LEFT_TEXT; vj<=RIGHT_TEXT; vj++) {
 cleardevice();

 /* set the text justification */
 settextjustify(hj, vj);

 /* create a message string */
 sprintf(msg, "%s %s", hjust[hj], vjust[vj]);

 /* create crosshairs on the screen */
 xat(midx, midy);

 /* output the message */
 outtextxy(midx, midy, msg);
 getch();
 }

 /* clean up */
 closegraph();
 return 0;
}

void xat(int x, int y) /* draw an x at (x,y) */
{
 line(x-4, y, x+4, y);
}

```

```
 line(x, y-4, x, y+4);
}
```

## settextstyle

---

**Function** Sets the current text characteristics for graphics output.

**Syntax** #include <graphics.h>  
void far settextstyle(int *font*, int *direction*, int *charsize*);

**Prototype in** graphics.h

**Remarks** **settextstyle** sets the text font, the direction in which text is displayed, and the size of the characters. A call to **settextstyle** affects all text output by **outtext** and **outtextxy**.

The parameters *font*, *direction*, and *charsize* passed to **settextstyle** are described in the following:

*font*: One 8x8 bit-mapped font and several “stroked” fonts are available. The 8x8 bit-mapped font is the default. The enumeration *font\_names*, defined in graphics.h, provides names for these different font settings:

| Name            | Value | Description             |
|-----------------|-------|-------------------------|
| DEFAULT_FONT    | 0     | 8x8 bit-mapped font     |
| TRIPLEX_FONT    | 1     | Stroked triplex font    |
| SMALL_FONT      | 2     | Stroked small font      |
| SANS_SERIF_FONT | 3     | Stroked sans-serif font |
| GOTHIC_FONT     | 4     | Stroked gothic font     |

The default bit-mapped font is built into the graphics system. Stroked fonts are stored in \*.CHR disk files, and only one at a time is kept in memory. Therefore, when you select a stroked font (different from the last selected stroked font), the corresponding \*.CHR file must be loaded from disk.

To avoid this loading when several stroked fonts are used, you can link font files into your program. Do this by converting them into object files with the BGI OBJ utility, then registering them through **registerbgifont**, as described in UTIL.DOC, included with your distributions disks.

*direction*: Font directions supported are horizontal text (left to right) and vertical text (rotated 90 degrees counterclockwise). The default direction is **HORIZ\_DIR**.

| Name      | Value | Description   |
|-----------|-------|---------------|
| HORIZ_DIR | 0     | Left to right |
| VERT_DIR  | 1     | Bottom to top |

*charsize*: The size of each character can be magnified using the *charsize* factor. If *charsize* is nonzero, it can affect bit-mapped or stroked characters. A *charsize* value of 0 can be used only with stroked fonts.

- If *charsize* equals 1, **outtext** and **outtextxy** displays characters from the 8×8 bit-mapped font in an 8×8 pixel rectangle onscreen.
- If *charsize* equals 2, these output functions display characters from the 8×8 bit-mapped font in a 16×16 pixel rectangle, and so on (up to a limit of ten times the normal size).
- When *charsize* equals 0, the output functions **outtext** and **outtextxy** magnify the stroked font text using either the default character magnification factor (4) or the user-defined character size given by **setusercharsize**.

Always use **textheight** and **textwidth** to determine the actual dimensions of the text.

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **gettextsettings**, **graphresult**, **installuserfont**, **settextjustify**, **setusercharsize**, **textheight**, **textwidth**

#### Example

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

/* the names of the text styles supported */
char *fname[] = { "DEFAULT font", "TRIPLEX font", "SMALL font",
 "SANS SERIF font", "GOTHIC font" };

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int style, midx, midy;
 int size = 1;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");
```



## settextstyle

```
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;
settextjustify(CENTER_TEXT, CENTER_TEXT);

/* loop through the available text styles */
for (style=DEFAULT_FONT; style<=GOTHIC_FONT; style++) {
 cleardevice();
 if (style == TRIPLEX_FONT)
 size = 4;

 /* select the text style */
 settextstyle(style, HORIZ_DIR, size);

 /* output a message */
 outtextxy(midx, midy, fname[style]);
 getch();
}

/* clean up */
closegraph();
return 0;
}
```

## settime

---

**Function** Sets system time.

**Syntax** #include <dos.h>  
void settime(struct time \*timep);

**Prototype in** dos.h

**Remarks** **settime** sets the system time to the values in the **time** structure pointed to by *timep*. The **time** structure is defined as follows:

```
struct time {
 unsigned char ti_min; /* minutes */
 unsigned char ti_hour; /* hours */
 unsigned char ti_hund; /* hundredths of seconds */
};
```

```
 unsigned char ti_sec; /* seconds */
 };
```

**Return value** None.

**Portability** **settime** is unique to DOS.

**See also** **ctime**, **getdate**, **gettime**, **setdate**, **time**

**Example**

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

int main(void)
{
 struct time t;
 gettime(&t);
 printf("The current minute is: %d\n", t.ti_min);
 printf("The current hour is: %d\n", t.ti_hour);
 printf("The current hundredth of a second is: %d\n", t.ti_hund);
 printf("The current second is: %d\n", t.ti_sec);

 /* add 1 to minutes struct element, then call settime */
 t.ti_min++;
 settime(&t);
 return 0;
}
```

## setusercharsize

**Function** Varies character width and height for stroked fonts.

**Syntax** `#include <graphics.h>`  
`void far setusercharsize(int multx, int divx, int multy, int divy);`

**Prototype in** `graphics.h`

**Remarks** **setusercharsize** gives you finer control over the size of text from stroked fonts used with `graphics` functions. The values set by **setusercharsize** are active *only* if *charsize* equals 0, as set by a previous call to **setttextstyle**.

With **setusercharsize**, you specify factors by which the width and height are scaled. The default width is scaled by *multx* : *divx*, and the default height is scaled by *multy* : *divy*. For example, to make text twice as wide and 50% taller than the default, set

```
multx = 2; divx = 1;
multy = 3; divy = 2;
```

**Return value** None.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **gettextsettings, graphresult, settextstyle**

**Example**

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

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 /* select a text style */
 settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

 /* move to the text starting position */
 moveto(0, getmaxy() / 2);

 /* output some normal text */
 outtext("Norm ");

 /* make the text 1/3 the normal width */
 setusercharsize(1, 3, 1, 1);
 outtext("Short ");

 /* make the text 3 times normal width */
 setusercharsize(3, 1, 1, 1);
 outtext("Wide");

 /* clean up */
 getch();
 closegraph();
 return 0;
}
```

## setvbuf

---

**Function** Assigns buffering to a stream.

**Syntax** `#include <stdio.h>`  
`int setvbuf(FILE *stream, char *buf, int type, size_t size);`

**Prototype in** `stdio.h`

**Remarks** **setvbuf** causes the buffer *buf* to be used for I/O buffering instead of an automatically allocated buffer. It is used after the given stream is opened. If *buf* is null, a buffer will be allocated using **malloc**; the buffer will use *size* as the amount allocated. The buffer will be automatically freed on close. The *size* parameter specifies the buffer size and must be greater than zero.



The parameter *size* is limited to a maximum of 32,767.

*stdin* and *stdout* are unbuffered if they are not redirected; otherwise, they are fully buffered. *Unbuffered* means that characters written to a stream are immediately output to the file or device, while *buffered* means that the characters are accumulated and written as a block.

The *type* parameter is one of the following:

- `_IOFBF` The file is *fully buffered*. When a buffer is empty, the next input operation will attempt to fill the entire buffer. On output, the buffer will be completely filled before any data is written to the file.
- `_IOLBF` The file is *line buffered*. When a buffer is empty, the next input operation will still attempt to fill the entire buffer. On output, however, the buffer will be flushed whenever a newline character is written to the file.
- `_IONBF` The file is *unbuffered*. The *buf* and *size* parameters are ignored. Each input operation will read directly from the file, and each output operation will immediately write the data to the file.

A common cause for error is to allocate the buffer as an automatic (local) variable and then fail to close the file before returning from the function where the buffer was declared.

**Return value** **setvbuf** returns 0 on success. It returns nonzero if an invalid value is given for *type* or *size*, or if there is not enough space to allocate a buffer.

**Portability** **setvbuf** is available on UNIX systems and is defined in ANSI C.

## setvbuf

**See also** `fflush`, `fopen`, `setbuf`

**Example**

```
#include <stdio.h>

int main(void)
{
 FILE *input, *output;
 char bufr[512];
 input = fopen("file.in", "r+b");
 output = fopen("file.out", "w");

 /* set up input stream for minimal disk access, using our own character buffer
 */
 if (setvbuf(input, bufr, _IOFBF, 512) != 0)
 printf("failed to set up buffer for input file\n");
 else
 printf("buffer set up for input file\n");

 /* set up output stream for line buffering using space that is obtained
 through an indirect call to malloc */
 if (setvbuf(output, NULL, _IOLBF, 132) != 0)
 printf("failed to set up buffer for output file\n");
 else
 printf("buffer set up for output file\n");

 /* perform file I/O here */

 /* close files */
 fclose(input);
 fclose(output);
 return 0;
}
```

## setvect

---

**Function** Sets interrupt vector entry.

**Syntax** `#include <dos.h>`  
`void setvect(int interruptno, void interrupt (*isr) ());`

**Prototype in** `dos.h`

**Remarks** Every processor of the 8086 family includes a set of interrupt vectors, numbered 0 to 255. The 4-byte value in each vector is actually an address, which is the location of an interrupt function.

**setvect** sets the value of the interrupt vector named by *interruptno* to a new value, *isr*, which is a far pointer containing the address of a new

interrupt function. The address of a C routine can only be passed to *isr* if that routine is declared to be an interrupt routine.



If you use the prototypes declared in *dos.h*, simply pass the address of an interrupt function to **setvect** in any memory model.

**Return value** None.

**Portability** **setvect** is unique to the 8086 family of processors.

**See also** **getvect**

**Example**

```
/* NOTE: This is an interrupt service routine. You canNOT compile this program
 with Test Stack Overflow turned on and get an executable file that will
 operate correctly. */
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0x1C /* the clock tick interrupt */

void interrupt (*oldhandler)(void);
int count=0;

void interrupt handler(void)
{
 count++; /* increase the global counter */
 oldhandler(); /* call the old routine */
}

int main()
{
 /* save the old interrupt vector */
 oldhandler = getvect(INTR);

 /* install the new interrupt handler */
 setvect(INTR, handler);

 /* loop until the counter exceeds 20 */
 while (count < 20)
 printf("count is %d\n",count);

 /* reset the old interrupt handler */
 setvect(INTR, oldhandler);
 return 0;
}
```

## setverify

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Sets the state of the verify flag in DOS.                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Syntax</b>       | <pre>#include &lt;dos.h&gt; void setverify(int <i>value</i>);</pre>                                                                                                                                                                                                                                                                                                                                                       |
| <b>Prototype in</b> | dos.h                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Remarks</b>      | <p><b>setverify</b> sets the current state of the verify flag to <i>value</i>.</p> <ul style="list-style-type: none"><li>■ A <i>value</i> of 0 = verify flag off.</li><li>■ A <i>value</i> of 1 = verify flag on.</li></ul> <p>The verify flag controls output to the disk. When verify is off, writes are not verified; when verify is on, all disk writes are verified to ensure proper writing of the data.</p>        |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Portability</b>  | <b>setverify</b> is unique to DOS.                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>See also</b>     | <b>getverify</b>                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;conio.h&gt; #include &lt;dos.h&gt;  int main(void) {     int verify_flag;     printf("Enter 0 to set verify flag off\n");     printf("Enter 1 to set verify flag on\n");     verify_flag = getch() - 0;     setverify(verify_flag);     if (getverify())         printf("DOS verify flag is on\n");     else         printf("DOS verify flag is off\n");     return 0; }</pre> |

# setviewport

---

**Function** Sets the current viewport for graphics output.

**Syntax** `#include <graphics.h>`  
`void far setviewport(int left, int top, int right, int bottom, int clip);`

**Prototype in** graphics.h

**Remarks** **setviewport** establishes a new viewport for graphics output.

The viewport's corners are given in absolute screen coordinates by (*left,top*) and (*right,bottom*). The current position (CP) is moved to (0,0) in the new window.

The parameter *clip* determines whether drawings are clipped (truncated) at the current viewport boundaries. If *clip* is nonzero, all drawings will be clipped to the current viewport.

**Return value** If invalid input is passed to **setviewport**, **graphresult** returns -11, and the current view settings remain unchanged.

**Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

**See also** **clearviewport**, **getviewsettings**, **graphresult**

**Example**

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

#define CLIP_ON 1 /* activates clipping in viewport */

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 }
}
```



## setviewport

```
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
}

setcolor(getmaxcolor());

/* message in default full-screen viewport */
outtextxy(0, 0, "* <-- (0, 0) in default viewport");

/* create a smaller viewport */
setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

/* display some text */
outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");

/* clean up */
getch();
closegraph();
return 0;
}
```

## setvisualpage

---

- Function** Sets the visual graphics page number.
- Syntax** `#include <graphics.h>`  
`void far setvisualpage(int page);`
- Prototype in** `graphics.h`
- Remarks** **setvisualpage** makes *page* the visual graphics page.
- Return value** None.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **graphresult**, **setactivepage**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports multiple pages */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;
```

```

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

x = getmaxx() / 2;
y = getmaxy() / 2;
ht = textheight("W");

/* select the off screen page for drawing */
setactivepage(1);

/* draw a line on page #1 */
line(0, 0, getmaxx(), getmaxy());

/* output a message on page #1 */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(x, y, "This is page #1:");
outtextxy(x, y+ht, "Press any key to halt:");

/* select drawing to page #0 */
setactivepage(0);

/* output a message on page #0 */
outtextxy(x, y, "This is page #0.");
outtextxy(x, y+ht, "Press any key to view page #1:");
getch();

/* select page #1 as the visible page */
setvisualpage(1);

/* clean up */
getch();
closegraph();
return 0;
}

```

setwritemode

Function	Sets the writing mode for line drawing in graphics mode.
Syntax	<pre>#include <graphics.h> void far setwritemode(int <i>mode</i>);</pre>
Prototype in	graphics.h
Remarks	<p>The following constants are defined:</p> <pre>COPY_PUT = 0 /* MOV */ XOR_PUT = 1 /* XOR */</pre> <p>Each constant corresponds to a binary operation between each byte in the line and the corresponding bytes onscreen. <code>COPY_PUT</code> uses the assembly language MOV instruction, overwriting with the line whatever is on the screen. <code>XOR_PUT</code> uses the XOR command to combine the line with the screen. Two successive XOR commands will erase the line and restore the screen to its original appearance.</p> <p>➔ setwritemode currently works only with line, linere1, lineto, rectangle, and drawpoly.</p>
Return value	None.
Portability	This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
See also	drawpoly , line , linere1 , lineto , putimage
Example	<pre>#include <graphics.h> #include <stdlib.h> #include <stdio.h> #include <conio.h> int main() { /* request autodetection */ int gdriver = DETECT, gmode, errorcode; int xmax, ymax; /* initialize graphics and local variables */ initgraph(&gdriver, &gmode, ""); /* read result of initialization */ errorcode = graphresult(); if (errorcode != grOk) /* an error occurred */ { printf("Graphics error: %s\n", grapherrormsg(errorcode));</pre>

```

    printf("Press any key to halt:");
    getch();
    exit(1);          /* terminate with an error code */
}

xmax = getmaxx();
ymax = getmaxy();

/* select XOR drawing mode */
setwritemode(XOR_PUT);

/* draw a line */
line(0, 0, xmax, ymax);
getch();

/* erase the line by drawing over it */
line(0, 0, xmax, ymax);
getch();

/* select overwrite drawing mode */
setwritemode(COPY_PUT);

/* draw a line */
line(0, 0, xmax, ymax);

/* clean up */
getch();
closegraph();
return 0;
}

```

signal

Function Specifies signal-handling actions.

Syntax `#include <signal.h>`
`void (*signal(int sig, void (*func) (int sig[, int subcode])))(int);`

Prototype in signal.h

Remarks **signal** determines how receipt of signal number *sig* will subsequently be treated. You can install a user-specified handler routine or use one of the two predefined handlers, SIG_DFL and SIG_IGN, in signal.h.

Function pointer	Meaning
SIG_DFL	Terminates the program
SIG_IGN	Ignore this type signal
SIG_ERR	Indicates an error return from signal

The signal types and their defaults are as follows:

Signal type	Meaning
SIGABRT	Abnormal termination. Default action is equivalent to calling <code>_exit(3)</code> .
SIGFPE	Arithmetic error caused by division by 0, invalid operation, and the like. Default action is equivalent to calling <code>_exit(1)</code> .
SIGILL	Illegal operation. Default action is equivalent to calling <code>_exit(1)</code> .
SIGINT	<i>CTRL-C</i> interrupt. Default action is to do an INT 23h.
SIGSEGV	Illegal storage access. Default action is equivalent to calling <code>_exit(1)</code> .
SIGTERM	Request for program termination. Default action is equivalent to calling <code>_exit(1)</code> .

`signal.h` defines a type called `sig_atomic_t`, the largest integer type the processor can load or store atomically in the presence of asynchronous interrupts (for the 8086 family, this is a 16-bit word; that is, a Borland C++ integer).

When a signal is generated by the **raise** function or by an external event, the following happens:

1. If a user-specified handler has been installed for the signal, the action for that signal type is set to `SIG_DFL`.
2. The user-specified function is called with the signal type as the parameter.

User-specified handler functions can terminate by a return or by a call to **abort**, **_exit**, **exit**, or **longjmp**.

Borland C++ implements an extension to ANSI C when the signal type is `SIGFPE`, `SIGSEGV`, or `SIGILL`. The user-specified handler function is called with one or two extra parameters. If `SIGFPE`, `SIGSEGV`, or `SIGILL` has been raised as the result of an explicit call to the **raise** function, the user-specified handler is called with one extra parameter, an integer specifying that the handler is being explicitly invoked. The explicit activation values for `SIGFPE`, `SIGSEGV` and `SIGILL` are as follows (see declarations in `float.h`):

SIGSEGV signal	Meaning
<code>SIGFPE</code>	<code>FPE_EXPLICITGEN</code>
<code>SIGSEGV</code>	<code>SEGV_EXPLICITGEN</code>
<code>SIGILL</code>	<code>ILL_EXPLICITGEN</code>

If SIGFPE is raised because of a floating-point exception, the user handler is called with one extra parameter that specifies the FPE_xxx type of the signal. If SIGSEGV, SIGILL, or the integer-related variants of SIGFPE signals (FPE_INTOVFLOW or FPE_INTDIV0) are raised as the result of a processor exception, the user handler is called with two extra parameters:

1. The SIGFPE, SIGSEGV, or SIGILL exception type (see float.h for all these types). This first parameter is the usual ANSI signal type.
2. An integer pointer into the stack of the interrupt handler that called the user-specified handler. This pointer points to a list of the processor registers saved when the exception occurred. The registers are in the same order as the parameters to an interrupt function; that is, BP, DI, SI, DS, ES, DX, CX, BX, AX, IP, CS, FLAGS. To have a register value changed when the handler returns, change one of the locations in this list. For example, to have a new SI value on return, do something like this:

```
*((int*)list_pointer + 2) = new_SI_value;
```

In this way, the handler can examine and make any adjustments to the registers that you want. (See Example 2 for a demonstration.)

The following SIGFPE-type signals can occur (or be generated). They correspond to the exceptions that the 8087 family is capable of detecting, as well as the "INTEGER DIVIDE BY ZERO" and the "INTERRUPT ON OVERFLOW" on the main CPU. (The declarations for these are in float.h.)

SIGFPE signal	Meaning
FPE_INTOVFLOW	INTO executed with OF flag set
FPE_INTDIV0	Integer divide by zero
FPE_INVALID	Invalid operation
FPE_ZERODIVIDE	Division by zero
FPE_OVERFLOW	Numeric overflow
FPE_UNDERFLOW	Numeric underflow
FPE_INEXACT	Precision
FPE_EXPLICITGEN	User program executed raise(SIGFPE)



The FPE_INTOVFLOW and FPE_INTDIV0 signals are generated by integer operations, and the others are generated by floating-point operations. Whether the floating-point exceptions are generated depends on the coprocessor control word, which can be modified with `_control87`. Denormal exceptions are handled by Borland C++ and not passed to a signal handler.

The following SIGSEGV-type signals can occur:

SEGV_BOUND	Bound constraint exception
SEGV_EXPLICITGEN	raise (SIGSEGV) was executed

The 8088 and 8086 processors *don't* have a bound instruction. The 186, 286, 386, and NEC V series processors *do* have this instruction. So, on the 8088 and 8086 processors, the SEGV_BOUND type of SIGSEGV signal won't occur. Borland C++ doesn't generate bound instructions, but they can be used in inline code and separately compiled assembler routines that are linked in.

The following SIGILL-type signals can occur:

ILL_EXECUTION	Illegal operation attempted.
ILL_EXPLICITGEN	raise (SIGILL) was executed.

The 8088, 8086, NEC V20, and NEC V30 processors *don't* have an illegal operation exception. The 186, 286, 386, NEC V40, and NEC V50 processors *do* have this exception type. So, on 8088, 8086, NEC V20, and NEC V30 processors, the ILL_EXECUTION type of SIGILL won't occur.

When the signal type is SIGFPE, SIGSEGV, or SIGILL, a return from a signal handler is generally not advisable because the state of the 8087 is corrupt, the results of an integer division are wrong, an operation that shouldn't have overflowed did, a bound instruction failed, or an illegal operation was attempted. The only time a return is reasonable is when the handler alters the registers so that a reasonable return context exists *or* the signal type indicates that the signal was generated explicitly (for example, FPE_EXPLICITGEN, SEGV_EXPLICITGEN, or ILL_EXPLICITGEN). Generally in this case you would print an error message and terminate the program using **_exit**, **exit**, or **abort**. If a return is executed under any other conditions, the program's action will probably be unpredictable upon resuming.

Return value If the call succeeds, **signal** returns a pointer to the previous handler routine for the specified signal type. If the call fails, **signal** returns SIG_ERR, and the external variable *errno* is set to EINVAL.

Portability **signal** is compatible with ANSI C.

See also **abort**, **_control87**, **ctrlbrk**, **exit**, **longjmp**, **raise**, **setjmp**

Example 1

```

/* This example installs a signal handler routine to be run when Ctrl-Break is
   pressed. */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void catcher(void)
{

```

```

    printf("\nNow in break routine\n");
    exit(1);
}

int main(void) {
    signal(SIGINT,catcher);
    for (;;)
        printf("\nIn main() program\n");
}

```

Example 2

/* This example installs a signal handler routine for SIGFPE, catches an integer overflow condition, makes an adjustment to AX register, and returns. This example program MAY cause your computer to crash, and will produce runtime errors depending on which memory model is used. */

```

#pragma inline
#include <stdio.h>
#include <signal.h>

void Catcher(int *reglist)
{
    printf("Caught it!\n");
    *(reglist + 8) = 3;          /* make return AX = 3 */
}

int main(void)
{
    signal(SIGFPE, Catcher);
    asm    mov    ax,07FFFH    /* AX = 32767 */
    asm    inc    ax          /* cause overflow */
    asm    into                   /* activate handler */

    /* The handler set AX to 3 on return. If that hadn't happened, there would
       have been another exception when the next 'into' was executed after the
       'dec' instruction. */
    asm    dec    ax          /* no overflow now */
    asm    into                   /* doesn't activate */
    return 0;
}

```

S

sin

Function Calculates sine.

Syntax *Real version:*
#include <math.h>
double sin(double x);

Complex version:
#include <complex.h>
complex sin(complex x);

Prototype in *Real version:*

Complex version:

sin

	math.h	complex.h
Remarks	sin computes the sine of the input value. Angles are specified in radians. Error handling for this routine can be modified through the function matherr .	
Return value	sin returns the sine of the input value. The complex sine is defined by $\sin(z) = (\exp(i * z) - \exp(-i * z)) / (2i)$	
Portability	The real version of sin is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.	
See also	acos, asin, atan, atan2, complex, cos, tan	
Example	<pre>#include <stdio.h> #include <math.h> int main(void) { double result, x = 0.5; result = sin(x); printf("The sin() of %lf is %lf\n", x, result); return 0; }</pre>	

sinh

Function	Calculates hyperbolic sine.	
Syntax	<i>Real version:</i> <pre>#include <math.h> double sinh(double x);</pre>	<i>Complex version:</i> <pre>#include <complex.h> complex sinh(complex x);</pre>
Prototype in	<i>Real version:</i> math.h	<i>Complex version:</i> complex.h
Remarks	sinh computes the hyperbolic sine, $(e^x - e^{-x})/2$. Error handling for sinh can be modified through the function matherr . The complex hyperbolic sine is defined by $\sinh(z) = (\exp(z) - \exp(-z)) / 2$	
Return value	sinh returns the hyperbolic sine of x .	

When the correct value overflows, **sinh** returns the value `HUGE_VAL` of appropriate sign. Also, the global variable `errno` is set to `ERANGE`. See **cosh**.

Portability The real version of **sinh** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

See also **acos, asin, atan, atan2, complex, cos, cosh, sin, tan, tanh**

Example

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

int main(void)
{
    double result, x = 0.5;
    result = sinh(x);
    printf("The hyperbolic sin() of %lf is %lf\n", x, result);
    return 0;
}
```

sleep

Function Suspends execution for an interval (seconds).

Syntax `#include <dos.h>`
`void sleep(unsigned seconds);`

Prototype in `dos.h`

Remarks With a call to **sleep**, the current program is suspended from execution for the number of seconds specified by the argument *seconds*. The interval is only accurate to the nearest hundredth of a second or the accuracy of the DOS clock, whichever is less accurate.

Return value None.

Portability **sleep** is available on UNIX systems. This function is not Windows compatible.

See also **delay**

Example

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

int main(void)
{
    int i;
    for (i=1; i<5; i++) {
```

sleep

```
        printf("Sleeping for %d seconds\n", i);
        sleep(i);
    }
    return 0;
}
```

sopen

Function Opens a shared file.

Syntax `#include <fcntl.h>`
`#include <sys\stat.h>`
`#include <share.h>`
`#include <io.h>`
`int sopen(char *path, int access, int shflag, int mode);`

Prototype in io.h

Remarks **sopen** opens the file given by *path* and prepares it for shared reading or writing, as determined by *access*, *shflag*, and *mode*.

sopen is a macro defined as

```
open(path, (access) | (shflag), mode)
```

For **sopen**, *access* is constructed by ORing flags bitwise from the following two lists. Only one flag from the first list can be used; the remaining flags can be used in any logical combination.

List 1: Read/write flags

O_RDONLY Open for reading only.
O_WRONLY Open for writing only.
O_RDWR Open for reading and writing.

List 2: Other access flags

O_NDELAY Not used; for UNIX compatibility.
O_APPEND If set, the file pointer is set to the end of the file prior to each write.
O_CREAT If the file exists, this flag has no effect. If the file does not exist, the file is created, and the bits of *mode* are used to set the file attribute bits as in **chmod**.
O_TRUNC If the file exists, its length is truncated to 0. The file attributes remain unchanged.
O_EXCL Used only with **O_CREAT**. If the file already exists, an error is returned.

- O_BINARY** This flag can be given to explicitly open the file in binary mode.
- O_TEXT** This flag can be given to explicitly open the file in text mode.

These `O_...` symbolic constants are defined in `fcntl.h`.

If neither `O_BINARY` nor `O_TEXT` is given, the file is opened in the translation mode set by the global variable `_fmode`.

If the `O_CREAT` flag is used in constructing *access*, you need to supply the *mode* argument to **sopen** from the following symbolic constants defined in `sys\stat.h`.

Value of <i>mode</i>	Access permission
<code>S_IWRITE</code>	Permission to write
<code>S_IREAD</code>	Permission to read
<code>S_IREAD S_IWRITE</code>	Permission to read/write

shflag specifies the type of file-sharing allowed on the file *path*. Symbolic constants for *shflag* are defined in `share.h`.

Value of <i>shflag</i>	What it does
<code>SH_COMPAT</code>	Sets compatibility mode
<code>SH_DENYRW</code>	Denies read/write access
<code>SH_DENYWR</code>	Denies write access
<code>SH_DENYRD</code>	Denies read access
<code>SH_DENYNONE</code>	Permits read/write access
<code>SH_DENYNO</code>	Permits read/write access

Return value On successful completion, **sopen** returns a nonnegative integer (the file handle), and the file pointer (that marks the current position in the file) is set to the beginning of the file. On error, it returns `-1`, and the global variable *errno* is set to

- `ENOENT` Path or file function not found
- `EMFILE` Too many open files
- `EACCES` Permission denied
- `EINVACC` Invalid access code

Portability **sopen** is available on UNIX systems. On UNIX version 7, the *O_type* mnemonics are not defined. UNIX System III uses all the *O_type* mnemonics except `O_BINARY`.

See also `chmod`, `close`, `creat`, `lock`, `lseek`, `_open`, `open`, `unlock`, `unmask`

Example `#include <io.h>`



sopen

```
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
    int handle;
    int status;
    handle = sopen("c:\\autoexec.bat", O_RDONLY, SH_DENYNO, S_IREAD);
    if (handle < 0)
    {
        printf("sopen failed\n");
        exit(1);
    }
    status = access("c:\\autoexec.bat", 6);
    if (status == 0)
        printf("read/write access allowed\n");
    else
        printf("read/write access not allowed\n");
    close(handle);
    return 0;
}
```

sound

Function Turns PC speaker on at specified frequency.

Syntax `#include <dos.h>`
`void sound(unsigned frequency);`

Prototype in dos.h

Remarks **sound** turns on the PC's speaker at a given frequency. *frequency* specifies the frequency of the sound in hertz (cycles per second). To turn the speaker off after a call to **sound**, call the function **nosound**.

Portability **sound** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **delay**, **nosound**

Example

```
/* Emits a 7-Hz tone for 10 seconds. NOTE: Your PC may not be able to emit a 7-Hz
tone. */
#include <dos.h>

int main(void)
```

```

{
    sound(7);
    delay(10000);
    nosound();
    return 0;
}

```

spawnl, spawnle, spawnlp, spawnlpe, spawnv, spawnve, spawnvp, and spawnvpe

Function Creates and runs child processes.

Syntax

```

#include <process.h>
#include <stdio.h>
int spawnl(int mode, char *path, char *arg0, arg1, ..., argn, NULL);
int spawnle(int mode, char *path, char *arg0, arg1, ..., argn, NULL,
            char *envp[]);

int spawnlp(int mode, char *path, char *arg0, arg1, ..., argn, NULL);
int spawnlpe(int mode, char *path, char *arg0, arg1, ..., argn, NULL, char
            *envp[]);

int spawnv(int mode, char *path, char *argv[]);
int spawnve(int mode, char *path, char *argv[], char *envp[]);

int spawnvp(int mode, char *path, char *argv[]);
int spawnvpe(int mode, char *path, char *argv[], char *envp[]);

```

Prototype in process.h

Remarks The functions in the **spawn...** family create and run (execute) other files, known as *child processes*. There must be sufficient memory available for loading and executing a child process.

The value of *mode* determines what action the calling function (the *parent process*) takes after the **spawn...** call. The possible values of *mode* are

P_WAIT	Puts parent process "on hold" until child process completes execution.
P_NOWAIT	Continues to run parent process while child process runs.
P_OVERLAY	Overlays child process in memory location formerly occupied by parent. Same as an exec... call.

➡ P_NOWAIT is currently not available; using it generates an error value.

path is the file name of the called child process. The **spawn...** function calls search for *path* using the standard DOS search algorithm:

- No extension or no period: Search for exact file name; if not successful, DOS adds .COM and searches again. If still not successful, it adds .EXE and searches again.
- Extension given: Search only for exact file name.
- Period given: Search only for file name with no extension.
- If *path* does not contain an explicit directory, **spawn...** functions that have the **p** suffix will search the current directory, then the directories set with the DOS PATH environment variable.

The suffixes *l*, *v*, *p*, and *e* added to the **spawn...** “family name” specify that the named function operates with certain capabilities.

- p** The function will search for the file in those directories specified by the PATH environment variable. Without the *p* suffix, the function will search only the current working directory.
- l** The argument pointers *arg0*, *arg1*, ..., *argn* are passed as separate arguments. Typically, the *l* suffix is used when you know in advance the number of arguments to be passed.
- v** The argument pointers *argv[0]*, ..., *arg[n]* are passed as an array of pointers. Typically, the *v* suffix is used when a variable number of arguments is to be passed.
- e** The argument *envp* can be passed to the child process, allowing you to alter the environment for the child process. Without the *e* suffix, child processes inherit the environment of the parent process.

Each function in the **spawn...** family *must* have one of the two argument-specifying suffixes (either *l* or *v*). The path search and environment inheritance suffixes (*p* and *e*) are optional.

For example,

- **spawnl** takes separate arguments, searches only the current directory for the child, and passes on the parent’s environment to the child.
- **spawnvpe** takes an array of argument pointers, incorporates PATH in its search for the child process, and accepts the *envp* argument for altering the child’s environment.

The **spawn...** functions must pass at least one argument to the child process (*arg0* or *argv[0]*): This argument is, by convention, a copy of *path*. (Using a different value for this 0th argument won’t produce an error.)

spawnl, spawnle, spawnlp, spawnlpe, spawnv, spawnve, spawnvp, and spawnvpe

Under DOS 3.x, *path* is available for the child process; under earlier versions, the child process cannot use the passed value of the 0th argument (*arg0* or *argv[0]*).

When the *l* suffix is used, *arg0* usually points to *path*, and *arg1*, ..., *argn* point to character strings that form the new list of arguments. A mandatory null following *argn* marks the end of the list.

When the *e* suffix is used, you pass a list of new environment settings through the argument *envp*. This environment argument is an array of character pointers. Each element points to a null-terminated character string of the form

envvar = *value*

where *envvar* is the name of an environment variable, and *value* is the string value to which *envvar* is set. The last element in *envp[]* is null. When *envp* is null, the child inherits the parents' environment settings.

The combined length of *arg0* + *arg1* + ... + *argn* (or of *argv[0]* + *argv[1]* + ... + *argv[n]*), including space characters that separate the arguments, must be < 128 bytes. Null-terminators are not counted.

When a **spawn...** function call is made, any open files remain open in the child process.

Return value

On a successful execution, the **spawn...** functions return the child process's exit status (0 for a normal termination). If the child specifically calls **exit** with a nonzero argument, its exit status can be set to a nonzero value.

On error, the **spawn...** functions return -1, and the global variable *errno* is set to

E2BIG	Arg list too long
EINVAL	Invalid argument
ENOENT	Path or file name not found
ENOEXEC	Exec format error
ENOMEM	Not enough core

Portability

The **spawn...** functions are unique to DOS.

See also

abort, atexit, _exit, exit, exec..., _freset, searchpath, system

Example 1

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

int main(void)
{
```



```

int result;
clrscr();
result = spawnl(P_WAIT, "bcc.exe", NULL);
if (result == -1) {
    perror("Error from spawnl");
    exit(1);
}
return 0;
}

```

Example 2

```

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

int main(void)
{
    int result;
    clrscr();
    result = spawnle(P_WAIT, "bcc.exe", NULL, NULL);
    if (result == -1) {
        perror("Error from spawnle");
        exit(1);
    }
    return 0;
}

```

sprintf

Function Writes formatted output to a string.

Syntax `#include <stdio.h>`
`int sprintf(char *buffer, const char *format[, argument, ...]);`

Prototype in `stdio.h`

Remarks **sprintf** accepts a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a string.

See *printf* for details on format specifiers.

sprintf applies the first format specifier to the first argument, the second to the second, and so on. There must be the same number of format specifiers as arguments.

Return value **sprintf** returns the number of bytes output. **sprintf** does not include the terminating null byte in the count. In the event of error, **sprintf** returns EOF.

Portability **printf** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

See also **fprintf, printf**

Example

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

int main(void)
{
    char buffer[80];
    printf(buffer, "An approximation of pi is %f\n", M_PI);
    puts(buffer);
    return 0;
}
```

sqrt

Function If the argument is real, calculates the positive square root of input value.

Syntax *Real version:*

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

Complex version:

```
#include <complex.h>
complex sqrt(complex x);
```

Prototype in *Real version:*
math.h

Complex version:
complex.h

Remarks **sqrt** calculates the positive square root of the input value.

Error handling for **sqrt** can be modified through the function **matherr**.

For complex numbers x , $\text{sqrt}(x)$ gives the complex root whose *arg* is $\text{arg}(x)/2$.

The complex square root is defined by

$$\text{sqrt}(z) = \text{sqrt}(\text{abs}(z)) (\cos(\text{arg}(z)/2) + i \sin(\text{arg}(z)/2))$$

Return value On success, **sqrt** returns the value calculated, the square root of x . If x is real and positive, the result is positive. If x is real and negative, the global variable *errno* is set to

EDOM Domain error

Portability The real version of **sqrt** is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.

See also **complex, exp, log, pow**

sqrt

Example

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

int main(void)
{
    double x = 4.0, result;
    result = sqrt(x);
    printf("The square root of %lf is %lf\n", x, result);
    return 0;
}
```

rand

Function Initializes random number generator.

Syntax `#include <stdlib.h>`
`void srand(unsigned seed);`

Prototype in `stdlib.h`

Remarks The random number generator is reinitialized by calling **srand** with an argument value of 1. It can be set to a new starting point by calling **srand** with a given *seed* number.

Return value None.

Portability **srand** is available on UNIX systems and is defined in ANSI C.

See also **rand, random, randomize**

Example

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

int main(void)
{
    int i;
    time_t t;
    srand((unsigned) time(&t));
    printf("Ten random numbers from 0 to 99\n\n");
    for(i=0; i<10; i++)
        printf("%d\n", rand() % 100);
    return 0;
}
```

sscanf

Function Scans and formats input from a string.

Syntax `#include <stdio.h>`
`int sscanf(const char *buffer, const char *format[, address, ...]);`

Prototype in `stdio.h`

Remarks **sscanf** scans a series of input fields, one character at a time, reading from a string. Then each field is formatted according to a format specifier passed to **sscanf** in the format string pointed to by *format*. Finally, **sscanf** stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

See **scanf** for details on format specifiers.

sscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See **scanf** for a discussion of possible causes.

Return value **sscanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If **sscanf** attempts to read at end-of-string, the return value is EOF.

Portability **sscanf** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

See also **fscanf**, **scanf**

Example

```
#include <stdio.h>

char buffer[] = "a 3.14159 12 a-string\n";

int main(void)
{
    char ch;
    float f;
    int i;
    char string[20];
    sscanf(buffer, "%c %f %d %s", &ch, &f, &i, string);
    printf("%c %f %d %s", ch, f, i, string);
    return 0;
}
```

stat

Function Gets information about a file.

Syntax `#include <sys\stat.h>`
`int stat(char *path, struct stat *statbuf)`

Prototype in `sys\stat.h`

Remarks **stat** stores information about a given file or directory in the **stat** structure.

statbuf points to the **stat** structure (defined in `sys\stat.h`). That structure contains the following fields:

<i>st_mode</i>	Bit mask giving information about the file's mode
<i>st_dev</i>	Drive number of disk containing the file
<i>st_rdev</i>	Same as <i>st_dev</i>
<i>st_nlink</i>	Set to the integer constant 1
<i>st_size</i>	Size of the file, in bytes
<i>st_atime</i>	Most recent time the file was modified
<i>st_mtime</i>	Same as <i>st_atime</i>
<i>st_ctime</i>	Same as <i>st_atime</i>

The **stat** structure contains three additional fields not mentioned here; they contain values that are not meaningful under DOS.

The bit mask that gives information about the mode of the file includes the following bits.

One of the following bits will be set:

<code>S_IFREG</code>	Set if an ordinary file is specified by <i>path</i> .
<code>S_IFDIR</code>	Set if <i>path</i> specifies a directory.

One or both of the following bits will be set:

<code>S_IWRITE</code>	Set if user has permission to write to file.
<code>S_IREAD</code>	Set if user has permission to read to file.

The bit mask contains user-execute bits; these are set according to the open file's extension. The bit mask also includes the read/write bits; these are set according to the file's permission mode.

Return value **stat** returns 0 if it successfully retrieves the information about the file. On error (failure to get the information), **stat** returns -1 and sets the global variable *errno* to

<code>ENOENT</code>	File or path not found
---------------------	------------------------

Portability **stat** is available on UNIX systems and is defined in ANSI C.

See also `access`, `chmod`, `fstat`

Example

```
#include <sys\stat.h>
#include <stdio.h>
#include <time.h>

#define FILENAME "TEST.$$$"

int main(void)
{
    struct stat statbuf;
    FILE *stream;

    /* open a file for update */
    if ((stream = fopen(FILENAME, "w+")) == NULL) {
        fprintf(stderr, "Cannot open output file.\n");
        return(1);
    }

    /* get information about the file */
    stat(FILENAME, &statbuf);
    fclose(stream);

    /* display the information returned */
    if (statbuf.st_mode & S_IFCHR)
        printf("Handle refers to a device.\n");
    if (statbuf.st_mode & S_IFREG)
        printf("Handle refers to an ordinary file.\n");
    if (statbuf.st_mode & S_IREAD)
        printf("User has read permission on file.\n");
    if (statbuf.st_mode & S_IWRITE)
        printf("User has write permission on file.\n");
    printf("Drive letter of file: %c\n", 'A'+statbuf.st_dev);
    printf("Size of file in bytes: %ld\n", statbuf.st_size);
    printf("Time file last opened: %s\n", ctime(&statbuf.st_ctime));
    return 0;
}
```

S

_status87

Function Gets floating-point status.

Syntax `#include <float.h>`
`unsigned int _status87(void);`

Prototype in `float.h`

_status87

- Remarks** **_status87** gets the floating-point status word, which is a combination of the 80x87 status word and other conditions detected by the 80x87 exception handler.
- Return value** The bits in the return value give the floating-point status. See `float.h` for a complete definition of the bits returned by **_status87**.
- Portability** **_status87** is unique to DOS.
- See also** **_clear87**, **_control87**, **_fpreset**
- Example**
- ```
#include <stdio.h>
#include <float.h>

int main(void)
{
 float x;
 double y = 1.5e-100;
 printf("Status 87 before error: %x\n", _status87());
 x = y; /* force an error to occur */
 y = x;
 printf("Status 87 after error : %x\n", _status87());
 return 0;
}
```

## **stime**

---

- Function** Sets system date and time.
- Syntax** `#include <time.h>`  
`int stime(time_t *tp);`
- Prototype in** `time.h`
- Remarks** **stime** sets the system time and date. *tp* points to the value of the time as measured in seconds from 00:00:00 GMT, January 1, 1970.
- Return value** **stime** returns a value of 0.
- Portability** **stime** is available on UNIX systems.
- See also** **asctime**, **ftime**, **gettime**, **gmtime**, **localtime**, **time**, **tzset**
- Example**
- ```
#include <stdio.h>
#include <time.h>
#include <dos.h>

int main(void)
{
```

```

time_t t;
struct tm *area;
t = time(NULL);
area = localtime(&t);
printf("Number of seconds since 1/1/1970 is: %ld\n", t);
printf("Local time is: %s", asctime(area));
t++;
area = localtime(&t);
printf("Add a second: %s", asctime(area));
t += 60;
area = localtime(&t);
printf("Add a minute: %s", asctime(area));
t += 3600;
area = localtime(&t);
printf("Add an hour: %s", asctime(area));
t += 86400L;
area = localtime(&t);
printf("Add a day: %s", asctime(area));
t += 2592000L;
area = localtime(&t);
printf("Add a month: %s", asctime(area));
t += 31536000L;
area = localtime(&t);
printf("Add a year: %s", asctime(area));
return 0;
}

```

strcpy

Function	Copies one string into another.
Syntax	<code>#include <string.h></code> <code>char *strcpy(char *dest, const char *src);</code>
Prototype in	string.h
Remarks	strcpy copies the string <i>src</i> to <i>dest</i> , stopping after the terminating null character of <i>src</i> has been reached.
Return value	strcpy returns <i>dest</i> + strlen(src) .
Portability	strcpy is available on UNIX systems.
See also	strcpy
Example	<code>#include <stdio.h></code> <code>#include <string.h></code>


```

int main(void)
{
    char string[10];
    char *str1 = "abcdefghi";
    stpcpy(string, str1);
    printf("%s\n", string);
    return 0;
}

```

strcat, _fstrcat

Function Appends one string to another.

Syntax `#include <string.h>`
Near version: `char *strcat(char *dest, const char *src);`
Far version: `char far * far _fstrcat(char far *dest, const char far *src)`

Prototype in string.h

Remarks **strcat** appends a copy of *src* to the end of *dest*. The length of the resulting string is **strlen(dest) + strlen(src)**.

Return value **strcat** returns a pointer to the concatenated strings.

Portability **strcat** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

Example

```

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

int main(void)
{
    char destination[25];
    char *blank = " ", *c = "C++", *turbo = "Turbo";
    strcpy(destination, turbo);
    strcat(destination, blank);
    strcat(destination, c);
    printf("%s\n", destination);
    return 0;
}

```

strchr, _fstrchr

Function Scans a string for the first occurrence of a given character.

Syntax `#include <string.h>`
Near version: `char *strchr(const char *s, int c);`
Far version: `char far * far _fstrchr(const char far *s, int c)`

Prototype in string.h

Remarks **strchr** scans a string in the forward direction, looking for a specific character. **strchr** finds the *first* occurrence of the character *c* in the string *s*. The null-terminator is considered to be part of the string, so that, for example,

```
strchr(strs,0)
```

returns a pointer to the terminating null character of the string *strs*.

Return value **strchr** returns a pointer to the first occurrence of the character *c* in *s*; if *c* does not occur in *s*, **strchr** returns null.

Portability **strchr** is available on UNIX systems and is defined in ANSI C.

See also **strcspn, strrchr**

Example

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

int main(void)
{
    char string[15];
    char *ptr, c = 'r';
    strcpy(string, "This is a string");
    ptr = strchr(string, c);
    if (ptr)
        printf("The character %c is at position: %d\n", c, ptr-string);
    else
        printf("The character was not found\n");
    return 0;
}
```

strcmp

Function	Compares one string to another.
Syntax	<pre>#include <string.h> int strcmp(const char *s1, const char *s2);</pre>
Prototype in	string.h
Remarks	strcmp performs an unsigned comparison of <i>s1</i> to <i>s2</i> , starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached.
Return value	strcmp returns a value that is < 0 if <i>s1</i> is less than <i>s2</i> == 0 if <i>s1</i> is the same as <i>s2</i> > 0 if <i>s1</i> is greater than <i>s2</i>
Portability	strcmp is available on UNIX systems and is defined in ANSI C.
See also	strncmpi, strcoll, stricmp, strncmp, strncmpi, strnicmp
Example	<pre>#include <string.h> #include <stdio.h> int main(void) { char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc"; int ptr; ptr = strcmp(buf2, buf1); if (ptr > 0) printf("buffer 2 is greater than buffer 1\n"); else printf("buffer 2 is less than buffer 1\n"); ptr = strcmp(buf2, buf3); if (ptr > 0) printf("buffer 2 is greater than buffer 3\n"); else printf("buffer 2 is less than buffer 3\n"); return 0; }</pre>

strcmpi

Function Compares one string to another, without case sensitivity.

Syntax `#include <string.h>`
`int strcmpi(const char *s1, const char *s2);`

Prototype in string.h

Remarks **strcmpi** performs an unsigned comparison of *s1* to *s2*, without case sensitivity (same as **stricmp**—implemented as a macro).

It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

The routine **strcmpi** is the same, respectively, as **stricmp**. **strcmpi** is implemented through a macro in string.h and translates calls from **strcmpi** to **stricmp**. Therefore, in order to use **strcmpi**, you must include the header file string.h for the macro to be available. This macro is provided for compatibility with other C compilers.

Return value **strcmpi** returns an **int** value that is

< 0 if *s1* is less than *s2*
 == 0 if *s1* is the same as *s2*
 > 0 if *s1* is greater than *s2*

Portability **strcmpi** is unique to DOS.

See also **strcmp**, **strcoll**, **stricmp**, **strncmp**, **strncmpi**, **strnicmp**

Example

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

int main(void)
{
    char *buf1 = "BBB", *buf2 = "bbb";
    int ptr;
    ptr = strcmpi(buf2, buf1);
    if (ptr > 0)
        printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0)
        printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0)
        printf("buffer 2 equals buffer 1\n");
    return 0;
}
```

strcoll

Function	Compares two strings.
Syntax	<pre>#include <string.h> int strcoll(char *s1, char *s2);</pre>
Prototype in	string.h
Remarks	strcoll compares the string pointed to by <i>s1</i> to the string pointed to by <i>s2</i> , according to the collating sequence set by setlocale .
Return value	strcoll returns a value that is < 0 if <i>s1</i> is less than <i>s2</i> == 0 if <i>s1</i> is the same as <i>s2</i> > 0 if <i>s1</i> is greater than <i>s2</i>
Portability	strcoll is compatible with ANSI C.
See also	strcmp, strcmpi, stricmp, strncmp, strncmpi, strnicmp, strxfrm
Example	<pre>#include <stdio.h> #include <string.h> int main(void) { char *two = "International"; char *one = "Borland"; int check; check = strcoll(one, two); if (check == 0) printf("The strings are equal\n"); if (check < 0) printf("%s comes before %s\n", one, two); if (check > 0) printf("%s comes before %s\n", two, one); return 0; }</pre>

strcpy

Function	Copies one string into another.
Syntax	<code>#include <string.h></code> <code>char *strcpy(char *dest, const char *src);</code>
Prototype in	string.h
Remarks	Copies string <i>src</i> to <i>dest</i> , stopping after the terminating null character has been moved.
Return value	strcpy returns <i>dest</i> .
Portability	strcpy is available on UNIX systems and is defined in ANSI C.
See also	stpcpy
Example	<pre>#include <stdio.h> #include <string.h> int main(void) { char string[10]; char *str1 = "abcdefghi"; strcpy(string, str1); printf("%s\n", string); return 0; }</pre>

strcspn, _fstrcspn

Function	Scans a string for the initial segment not containing any subset of a given set of characters.
Syntax	<code>#include <string.h></code> <i>Near version:</i> <code>size_t strcspn(const char *s1, const char *s2);</code> <i>Far version:</i> <code>size_t far far _fstrcspn(const char far *s1, const char far *s2)</code>
Prototype in	string.h
Return value	strcspn returns the length of the initial segment of string <i>s1</i> that consists entirely of characters <i>not</i> from string <i>s2</i> .
Portability	strcspn is available on UNIX systems and is defined in ANSI C.

strcspn, _fstrcspn

See also [strchr](#), [strrchr](#)

Example

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

int main(void)
{
    char *string1 = "1234567890";
    char *string2 = "747DC8";
    int length;
    length = strcspn(string1, string2);
    printf("Character where strings intersect is at postion %d\n", length);
    return 0;
}
```

strdup, _fstrdup

Function Copies a string into a newly created location.

Syntax `#include <string.h>`
Near version: `char *strdup(const char *s);`
Far version: `char far * _fstrdup(const char far *s)`

Prototype in `string.h`

Remarks **strdup** makes a duplicate of string *s*, obtaining space with a call to **malloc**. The allocated space is **(strlen(s) + 1)** bytes long. The user is responsible for freeing the space allocated by **strdup** when it is no longer needed.

Return value **strdup** returns a pointer to the storage location containing the duplicated string, or returns null if space could not be allocated.

Portability **strdup** is available on UNIX systems.

See also [free](#)

Example

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

int main(void)
{
    char *dup_str, *string = "abcde";
    dup_str = strdup(string);
    printf("%s\n", dup_str);
    free(dup_str);
}
```

```
    return 0;  
}
```

_strerror

Function Builds a customized error message.

Syntax #include <string.h>
char *_strerror(const char *s);

Prototype in string.h, stdio.h

Remarks **_strerror** allows you to generate customized error messages; it returns a pointer to a null-terminated string containing an error message.

- If *s* is null, the return value points to the most recently generated error message.
- If *s* is not null, the return value contains *s* (your customized error message), a colon, a space, the most-recently generated system error message, and a new line. *s* should be 94 characters or less.

_strerror is the same as **strerror** in version 1.0 of Turbo C.

Return value **_strerror** returns a pointer to a constructed error string. The error message string is constructed in a static buffer that is overwritten with each call to **_strerror**.

Portability **_strerror** is unique to DOS.

See also **perror**, **strerror**

Example #include <stdio.h>

```
int main(void)  
{  
    FILE *fp;  
  
    /* open a file for writing */  
    fp = fopen("TEST.$$$", "w");  
  
    /* force an error condition by attempting to read */  
    if (!fp) fgetc(fp);  
    if (ferror(fp))  
        /* display a custom error message */  
        printf("%s", _strerror("Custom"));  
    fclose(fp);  
  
    return 0;  
}
```



strerror

Function	Returns a pointer to an error message string.
Syntax	<pre>#include <string.h> char *strerror(int <i>errnum</i>);</pre>
Prototype in	stdio.h, string.h
Remarks	strerror takes an int parameter <i>errnum</i> , an error number, and returns a pointer to an error message string associated with <i>errnum</i> .
Return value	strerror returns a pointer to a constructed error string. The error message string is constructed in a static buffer that is overwritten with each call to strerror .
Portability	strerror is compatible with ANSI C.
See also	perror , _strerror
Example	<pre>#include <stdio.h> #include <errno.h> int main(void) { char *buffer; buffer = strerror(errno); printf("Error: %s\n", buffer); return 0; }</pre>

strftime

Function	Formats time for output.
Syntax	<pre>#include <time.h> size_t _cdecl strftime(char *s, size_t <i>maxsize</i>, const char *<i>fmt</i>, const struct tm *<i>t</i>);</pre>
Prototype in	time.h
Remarks	strftime formats the time in the argument <i>t</i> into the array pointed to by the argument <i>s</i> according to the <i>fmt</i> specifications. The format string consists of zero or more directives and ordinary characters. Like printf , a directive consists of the % character followed by a character that determines the substitution that is to take place. All ordinary characters are copied unchanged. No more than <i>maxsize</i> characters are placed in <i>s</i> .

Return value **strftime** returns the number of characters placed into *s*. If the number of characters required is greater than *maxsize*, **strftime** returns 0.

Format specifier	Substitutes
%%	Character %
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time
%d	Two-digit day of the month (01 to 31)
%H	Two-digit hour (00 to 23)
%I	Two-digit hour (01 to 12)
%j	Three-digit day of the year (001 to 366)
%m	Two-digit month as a decimal number
%M	Two-digit minute (00 to 59)
%p	AM or PM
%S	Two-digit second (00 to 59)
%U	Two-digit week number where Sunday is the first day of the week (00 to 52)
%w	Weekday where 0 is Sunday (0 to 6)
%W	Two-digit week number where Monday is the first day of the week (00 to 52)
%x	Date
%X	Time
%y	Two-digit year without century (00 to 99)
%Y	Year with century
%Z	Time zone name, or no characters if no time zone

Portability **strftime** is compatible with ANSI C.

See also **localtime**, **mktime**, **time**

Example

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

int main(void)
{
    struct tm *time_now;
    time_t secs_now;
    char str[80];
    tzset();
    time(&secs_now);
    time_now = localtime(&secs_now);
    strftime(str, 80, "It is %M minutes after %I o'clock (%Z) %A, %B %d 19%y",
            time_now);
    printf("%s\n", str);
}
```



stricmp, _fstricmp

```
        return 0;  
    }
```

stricmp, _fstricmp

- Function** Compares one string to another, without case sensitivity.
- Syntax** `#include <string.h>`
Near version: `int stricmp(const char *s1, const char *s2);`
Far version: `int far _fstricmp(const char far *s1, const char far *s2)`
- Prototype in** string.h
- Remarks** **stricmp** performs an unsigned comparison of *s1* to *s2*, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached. The comparison is not case sensitive.
- It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).
- The routines **stricmp** and **strcmpi** are the same; **strcmpi** is implemented through a macro in string.h that translates calls from **strcmpi** to **stricmp**. Therefore, in order to use **strcmpi**, you must include the header file string.h for the macro to be available.
- Return value** **stricmp** returns an **int** value that is
- < 0 if *s1* is less than *s2*
 - == 0 if *s1* is the same as *s2*
 - > 0 if *s1* is greater than *s2*
- Portability** **stricmp** is unique to DOS.
- See also** **strcmp**, **strcmpi**, **strcoll**, **strncmp**, **strncmpi**, **strnicmp**
- Example**
- ```
#include <string.h>
#include <stdio.h>

int main(void)
{
 char *buf1 = "BBB", *buf2 = "bbb";
 int ptr;
 ptr = strcmpi(buf2, buf1);
 if (ptr > 0)
 printf("buffer 2 is greater than buffer 1\n");
 if (ptr < 0)
 printf("buffer 2 is less than buffer 1\n");
}
```

```

 if (ptr == 0)
 printf("buffer 2 equals buffer 1\n");
 return 0;
}

```

## strlen, \_fstrlen

---

**Function** Calculates the length of a string.

**Syntax** `#include <string.h>`  
*Near version:* `size_t strlen(const char *s);`  
*Far version:* `size_t _fstrlen(const char far *s)`

**Prototype in** string.h

**Remarks** `strlen` calculates the length of `s`.

**Return value** `strlen` returns the number of characters in `s`, not counting the null-terminating character.

**Portability** `strlen` is available on UNIX systems and is defined in ANSI C.

**Example**

```

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

int main(void)
{
 char *string = "Borland International";
 printf("%d\n", strlen(string));
 return 0;
}

```

## strlwr, \_fstrlwr

---

**Function** Converts uppercase letters in a string to lowercase.

**Syntax** `#include <string.h>`  
*Near version:* `char *strlwr(char *s);`  
*Far version:* `char far * far _fstrlwr(char char far *s)`

**Prototype in** string.h

**Remarks** `strlwr` converts uppercase letters (*A* to *Z*) in string `s` to lowercase (*a* to *z*). No other characters are changed.

**Return value** `strlwr` returns a pointer to the string `s`.

**Portability** **strlwr** is unique to DOS.

**See also** **strupr**

**Example**

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

int main(void)
{
 char *string = "Borland International";
 printf("string prior to strlwr: %s\n", string);
 strlwr(string);
 printf("string after strlwr: %s\n", string);
 return 0;
}
```

## strncat, \_fstrncat

---

**Function** Appends a portion of one string to another.

**Syntax** `#include <string.h>`  
*Near version:* `char *strncat(char *dest, const char *src, size_t maxlen);`  
*Far version:* `char far * far _fstrncat(char far *dest, const char far *src, size_t maxlen)`

**Prototype in** string.h

**Remarks** **strncat** copies at most *maxlen* characters of *src* to the end of *dest* and then appends a null character. The maximum length of the resulting string is `strlen(dest) + maxlen`.

**Return value** **strncat** returns *dest*.

**Portability** **strncat** is available on UNIX systems and is defined in ANSI C.

**Example**

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

int main(void)
{
 char destination[25];
 char *source = " States";
 strcpy(destination, "United");
 strncat(destination, source, 7);
 printf("%s\n", destination);
 return 0;
}
```

## strncmp, \_fstrncmp

---

**Function** Compares a portion of one string to a portion of another.

**Syntax** `#include <string.h>`  
*Near version:* `int strncmp(const char *s1, const char *s2, size_t maxlen);`  
*Far version:* `int far _fstrncmp(const char far *s1, const char far *s2, size_t maxlen)`

**Prototype in** `string.h`

**Remarks** **strncmp** makes the same unsigned comparison as **strcmp**, but looks at no more than *maxlen* characters. It starts with the first character in each string and continues with subsequent characters until the corresponding characters differ or until it has examined *maxlen* characters.

**Return value** **strncmp** returns an **int** value based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

< 0 if *s1* is less than *s2*  
 == 0 if *s1* is the same as *s2*  
 > 0 if *s1* is greater than *s2*

**Portability** **strncmp** is available on UNIX systems and is defined in ANSI C.

**See also** **strcmp, strcoll, stricmp, strncmpi, strnicmp**

**Example**

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

int main(void)
{
 char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 = "ccc";
 int ptr;
 ptr = strncmp(buf2, buf1, 3);
 if (ptr > 0)
 printf("buffer 2 is greater than buffer 1\n");
 else
 printf("buffer 2 is less than buffer 1\n");
 ptr = strncmp(buf2, buf3, 3);
 if (ptr > 0)
 printf("buffer 2 is greater than buffer 3\n");
 else
 printf("buffer 2 is less than buffer 3\n");
 return(0);
}
```

## strncmpi

---

**Function** Compares a portion of one string to a portion of another, without case sensitivity.

**Syntax** `#include <string.h>`  
`int strncmpi(const char *s1, const char *s2, size_t n);`

**Prototype in** string.h

**Remarks** **strncmpi** performs a signed comparison of *s1* to *s2*, for a maximum length of *n* bytes, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until *n* characters have been examined. The comparison is not case sensitive. (**strncmpi** is the same as **strnicmp**—implemented as a macro). It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

The routines **strnicmp** and **strncmpi** are the same; **strncmpi** is implemented through a macro in string.h that translates calls from **strncmpi** to **strnicmp**. Therefore, in order to use **strncmpi**, you must include the header file string.h for the macro to be available. This macro is provided for compatibility with other C compilers.

**Return value** **strncmpi** returns an **int** value that is

< 0 if *s1* is less than *s2*  
== 0 if *s1* is the same as *s2*  
> 0 if *s1* is greater than *s2*

**Portability** **strncmpi** is unique to DOS.

**Example**

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

int main(void)
{
 char *buf1 = "BBBccc", *buf2 = "bbbccc";
 int ptr;
 ptr = strncmpi(buf2, buf1, 3);
 if (ptr > 0)
 printf("buffer 2 is greater than buffer 1\n");
 if (ptr < 0)
 printf("buffer 2 is less than buffer 1\n");
 if (ptr == 0)
 printf("buffer 2 equals buffer 1\n");
 return 0;
}
```

## strncpy, \_fstrncpy

---

**Function** Copies a given number of bytes from one string into another, truncating or padding as necessary.

**Syntax** `#include <stdio.h>`  
*Near version:* `char *strncpy(char *dest, const char *src, size_t maxlen);`  
*Far version:* `char far * far _fstrncpy(char far *dest, const char far *src, size_t maxlen)`

**Prototype in** `string.h`

**Remarks** **strncpy** copies up to *maxlen* characters from *src* into *dest*, truncating or null-padding *dest*. The target string, *dest*, might not be null-terminated if the length of *src* is *maxlen* or more.

**Return value** **strncpy** returns *dest*.

**Portability** **strncpy** is available on UNIX systems and is defined in ANSI C.

**Example**

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

int main(void)
{
 char string[10];
 char *str1 = "abcdefghi";
 strncpy(string, str1, 3);
 string[3] = '\0';
 printf("%s\n", string);
 return 0;
}
```

S

## strnicmp, \_fstrnicmp

---

**Function** Compares a portion of one string to a portion of another, without case sensitivity.

**Syntax** `#include <string.h>`  
*Near version:* `int strnicmp(const char *s1, const char *s2, size_t maxlen);`  
*Far version:* `int far _fstrnicmp(const char far *s1, const char far *s2, size_t maxlen)`

**Prototype in** `string.h`

**Remarks** **strnicmp** performs a signed comparison of *s1* to *s2*, for a maximum length of *maxlen* bytes, starting with the first character in each string and



## strnicmp, \_fstrnicmp

continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached. The comparison is not case sensitive.

It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

**Return value** **strnicmp** returns an **int** value that is

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

**Portability** **strnicmp** is unique to DOS.

**Example**

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

int main(void)
{
 char *buf1 = "BBBccc", *buf2 = "bbbccc";
 int ptr;
 ptr = strnicmp(buf2, buf1, 3);
 if (ptr > 0)
 printf("buffer 2 is greater than buffer 1\n");
 if (ptr < 0)
 printf("buffer 2 is less than buffer 1\n");
 if (ptr == 0)
 printf("buffer 2 equals buffer 1\n");
 return 0;
}
```

## strnset, \_fstrnset

---

**Function** Sets a specified number of characters in a string to a given character.

**Syntax** #include <string.h>

*Near version:* char \*strnset(char \*s, int ch, size\_t n);

*Far version:* char far \* \_fstrnset(char far \*s, int ch, size\_t n)

**Prototype in** string.h

**Remarks** **strnset** copies the character *ch* into the first *n* bytes of the string *s*. If *n* > **strlen(s)**, then **strlen(s)** replaces *n*. It stops when *n* characters have been set, or when a null character is found.

**Return value** **strnset** returns *s*.

**Portability** **strnset** is unique to DOS.

**Example**

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

int main(void)
{
 char *string = "abcdefghijklmnopqrstuvwxyz";
 char letter = 'x';
 printf("string before strnset: %s\n", string);
 strnset(string, letter, 13);
 printf("string after strnset: %s\n", string);
 return 0;
}
```

## strpbrk, \_fstrpbrk

---

**Function** Scans a string for the first occurrence of any character from a given set.

**Syntax** `#include <string.h>`  
*Near version:* `char *strpbrk(const char *s1, const char *s2);`  
*Far version:* `char far * far _fstrpbrk(const char far *s1, const char far *s2)`

**Prototype in** string.h

**Remarks** **strpbrk** scans a string, *s1*, for the first occurrence of any character appearing in *s2*.

**Return value** **strpbrk** returns a pointer to the first occurrence of any of the characters in *s2*. If none of the *s2* characters occurs in *s1*, it returns null.

**Portability** **strpbrk** is available on UNIX systems and is defined in ANSI C.

**Example**

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

int main(void)
{
 char *string1 = "abcdefghijklmnopqrstuvwxyz";
 char *string2 = "onm";
 char *ptr;
 ptr = strpbrk(string1, string2);
 if (ptr)
 printf("strpbrk found first character: %c\n", *ptr);
 else
 printf("strpbrk didn't find character in set\n");
 return 0;
}
```

## strchr, \_fstrchr

---

|                     |                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Scans a string for the last occurrence of a given character.                                                                                                                                                                                                                                                                                                     |
| <b>Syntax</b>       | <pre>#include &lt;string.h&gt; Near version: char *strchr(const char *s, int c); Far version: char far * _fstrchr(const char far *s, int c)</pre>                                                                                                                                                                                                                |
| <b>Prototype in</b> | string.h                                                                                                                                                                                                                                                                                                                                                         |
| <b>Remarks</b>      | <b>strchr</b> scans a string in the reverse direction, looking for a specific character. <b>strchr</b> finds the <i>last</i> occurrence of the character <i>c</i> in the string <i>s</i> . The null-terminator is considered to be part of the string.                                                                                                           |
| <b>Return value</b> | <b>strchr</b> returns a pointer to the last occurrence of the character <i>c</i> . If <i>c</i> does not occur in <i>s</i> , <b>strchr</b> returns null.                                                                                                                                                                                                          |
| <b>Portability</b>  | <b>strchr</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                             |
| <b>See also</b>     | <b>strcspn, strchr</b>                                                                                                                                                                                                                                                                                                                                           |
| <b>Example</b>      | <pre>#include &lt;string.h&gt; #include &lt;stdio.h&gt;  int main(void) {     char string[15];     char *ptr, c = 'r';     strcpy(string, "This is a string");     ptr = strchr(string, c);     if (ptr)         printf("The character %c is at position: %d\n", c, ptr-string);     else         printf("The character was not found\n");     return 0; }</pre> |

## strrev, \_fstrrev

---

|                     |                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Reverses a string.                                                                                                      |
| <b>Syntax</b>       | <pre>#include &lt;string.h&gt; Near version: char *strrev(char *s); Far version: char far * _fstrrev(char far *s)</pre> |
| <b>Prototype in</b> | string.h                                                                                                                |

- Remarks** **strrev** changes all characters in a string to reverse order, except the terminating null character. (For example, it would change *string*\0 to *gnirts*\0.)
- Return value** **strrev** returns a pointer to the reversed string.
- Portability** **strrev** is unique to DOS.
- Example**
- ```
#include <string.h>
#include <stdio.h>

int main(void)
{
    char *forward = "string";
    printf("Before strrev(): %s\n", forward);
    strrev(forward);
    printf("After strrev(): %s\n", forward);
    return 0;
}
```

strset, _fstrset

- Function** Sets all characters in a string to a given character.
- Syntax** `#include <string.h>`
Near version: `char *strset(char *s, int ch);`
Far version: `char far * far _fstrset(char far *s, int ch)`
- Prototype in** string.h
- Remarks** **strset** sets all characters in the string *s* to the character *ch*. It quits when the terminating null character is found.
- Return value** **strset** returns *s*.
- Portability** **strset** is unique to DOS.
- See also** **setmem**
- Example**
- ```
#include <stdio.h>
#include <string.h>

int main(void)
{
 char string[10] = "123456789";
 char symbol = 'c';
 printf("Before strset(): %s\n", string);
 strset(string, symbol);
 printf("After strset(): %s\n", string);
}
```

## strset, \_fstrset

```
 return 0;
 }
```

## strspn, \_fstrspn

---

|                     |                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Scans a string for the first segment that is a subset of a given set of characters.                                                                                                                                                                                                                                                                                       |
| <b>Syntax</b>       | <code>#include &lt;string.h&gt;</code><br><i>Near version:</i> <code>size_t strspn(const char *s1, const char *s2);</code><br><i>Far version:</i> <code>size_t far _fstrspn(const char far *s1, const char far *s2)</code>                                                                                                                                                |
| <b>Prototype in</b> | string.h                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Remarks</b>      | <b>strspn</b> finds the initial segment of string <i>s1</i> that consists entirely of characters from string <i>s2</i> .                                                                                                                                                                                                                                                  |
| <b>Return value</b> | <b>strspn</b> returns the length of the initial segment of <i>s1</i> that consists entirely of characters from <i>s2</i> .                                                                                                                                                                                                                                                |
| <b>Portability</b>  | <b>strspn</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                      |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt;<br/>#include &lt;string.h&gt;<br/>#include &lt;alloc.h&gt;<br/><br/>int main(void)<br/>{<br/>    char *string1 = "1234567890";<br/>    char *string2 = "123DC8";<br/>    int length;<br/>    length = strspn(string1, string2);<br/>    printf("Character where strings differ is at position %d\n", length);<br/>    return 0;<br/>}</pre> |

## strstr, \_fstrstr

---

|                     |                                                                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Scans a string for the occurrence of a given substring.                                                                                                                                                                       |
| <b>Syntax</b>       | <code>#include &lt;string.h&gt;</code><br><i>Near version:</i> <code>char *strstr(const char *s1, const char *s2);</code><br><i>Far version:</i> <code>char far * far _fstrstr(const char far *s1, const char far *s2)</code> |
| <b>Prototype in</b> | string.h                                                                                                                                                                                                                      |

- Remarks** `strstr` scans *s1* for the first occurrence of the substring *s2*.
- Return value** `strstr` returns a pointer to the element in *s1*, where *s2* begins (points to *s2* in *s1*). If *s2* does not occur in *s1*, `strstr` returns null.
- Portability** `strstr` is available on UNIX systems and is defined in ANSI C.

**Example**

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

int main(void)
{
 char *str1 = "Borland International", *str2 = "nation", *ptr;
 ptr = strstr(str1, str2);
 printf("The substring is: %s\n", ptr);
 return 0;
}
```

## strtod

---

- Function** Converts a string to a **double** value.
- Syntax** `#include <stdlib.h>`  
`double strtod(const char *s, char **endptr);`
- Prototype in** `stdlib.h`
- Remarks** `strtod` converts a character string, *s*, to a **double** value. *s* is a sequence of characters that can be interpreted as a **double** value; the characters must match this generic format:

```
[ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]
```

where

```
[ws] = optional whitespace
[sn] = optional sign (+ or -)
[ddd] = optional digits
[fmt] = optional e or E
[.] = optional decimal point
```

`strtod` also recognizes `+INF` and `-INF` for plus and minus infinity, and `+NAN` and `-NAN` for Not-a-Number.

For example, here are some character strings that `strtod` can convert to **double**:

```
+ 1231.1981 e-1
```

## strtod

```
502.85E2
+ 2010.952
```

**strtod** stops reading the string at the first character that cannot be interpreted as an appropriate part of a **double** value.

If *endptr* is not null, **strtod** sets *\*endptr* to point to the character that stopped the scan (*\*endptr = &stopper*). *endptr* is useful for error detection.

**Return value** **strtod** returns the value of *s* as a **double**. In case of overflow, it returns plus or minus HUGE\_VAL.

**Portability** **strtod** is available on UNIX systems and is defined in ANSI C.

**See also** **atof**

**Example**

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

int main(void)
{
 char input[80], *endptr;
 double value;
 printf("Enter a floating point number:");
 gets(input);
 value = strtod(input, &endptr);
 printf("The string is %s the number is %lf\n", input, value);
 return 0;
}
```

## strtok, \_fstrtok

---

**Function** Searches one string for tokens, which are separated by delimiters defined in a second string.

**Syntax** `#include <string.h>`  
*Near version:* `char *strtok(char *s1, const char *s2);`  
*Far version:* `char far * _fstrtok(char far *s1, const char far *s2)`

**Prototype in** string.h

**Remarks** **strtok** considers the string *s1* to consist of a sequence of zero or more text tokens, separated by spans of one or more characters from the separator string *s2*.

The first call to **strtok** returns a pointer to the first character of the first token in *s1* and writes a null character into *s1* immediately following the

returned token. Subsequent calls with null for the first argument will work through the string *s1* in this way, until no tokens remain.

The separator string, *s2*, can be different from call to call.

**Return value** **strtok** returns a pointer to the token found in *s1*. A null pointer is returned when there are no more tokens.

**Portability** **strtok** is available on UNIX systems and is defined in ANSI C.

**Example**

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

int main(void)
{
 char input[16] = "abc,d";
 char *p;

 /* strtok places a NULL terminator
 in front of the token, if found */
 p = strtok(input, ",");
 if (p) printf("%s\n", p);

 /* a second call to strtok using a NULL as the first parameter returns a
 pointer to the character following the token */
 p = strtok(NULL, ",");
 if (p) printf("%s\n", p);
 return 0;
}
```

## strtol

**Function** Converts a string to a **long** value.

**Syntax** `#include <stdlib.h>`  
`long strtol(const char *s, char **endptr, int radix);`

**Prototype in** `stdlib.h`

**Remarks** **strtol** converts a character string, *s*, to a **long** integer value. *s* is a sequence of characters that can be interpreted as a **long** value; the characters must match this generic format:

```
[ws] [sn] [0] [x] [ddd]
```

where

*[ws]* = optional whitespace  
*[sn]* = optional sign (+ or -)  
*[0]* = optional zero (0)



[*x*] = optional x or X  
 [*ddd*] = optional digits

**strtol** stops reading the string at the first character it doesn't recognize.

If *radix* is between 2 and 36, the long integer is expressed in base *radix*. If *radix* is 0, the first few characters of *s* determine the base of the value being converted.

| First character | Second character | String interpreted as |
|-----------------|------------------|-----------------------|
| 0               | 1-7              | Octal                 |
| 0               | x or X           | Hexadecimal           |
| 1-9             |                  | Decimal               |

If *radix* is 1, it is considered to be an invalid value. If *radix* is less than 0 or greater than 36, it is considered to be an invalid value.

Any invalid value for *radix* causes the result to be 0 and sets the next character pointer *\*endptr* to the starting string pointer.

If the value in *s* is meant to be interpreted as octal, any character other than 0 to 7 will be unrecognized.

If the value in *s* is meant to be interpreted as decimal, any character other than 0 to 9 will be unrecognized.

If the value in *s* is meant to be interpreted as a number in any other base, then only the numerals and letters used to represent numbers in that base will be recognized. (For example, if *radix* equals 5, only 0 to 4 will be recognized; if *radix* equals 20, only 0 to 9 and A to J will be recognized.)

If *endptr* is not null, **strtol** sets *\*endptr* to point to the character that stopped the scan (*\*endptr = &stopper*).

**Return value** **strtol** returns the value of the converted string, or 0 on error.

**Portability** **strtol** is available on UNIX systems and is defined in ANSI C.

**See also** **atoi, atol, strtoul**

**Example**

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

int main(void)
{
 char *string = "87654321", *endptr;
 long lnumber;

 /* strtol converts string to long integer */
 lnumber = strtol(string, &endptr, 10);
```

```

printf("string = %s long = %ld\n", string, lnumber);
return 0;
}

```

## strtoul

---

- Function** Converts a string to an **unsigned long** in the given radix.
- Syntax** `#include <stdlib.h>`  
`unsigned long strtoul(const char *s, char **endptr, int radix);`
- Prototype in** `stdlib.h`
- Remarks** **strtoul** operates the same as **strtol**, except that it converts a string *str* to an **unsigned long** value (where **strtol** converts to a **long**). Refer to the entry for **strtol** for more information.
- Return value** **strtoul** returns the converted value, an **unsigned long**, or 0 on error.
- Portability** **strtoul** is compatible with ANSI C.
- See also** **atol**, **strtol**
- Example**
- ```

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

int main(void)
{
    char *string = "87654321", *endptr;
    unsigned long lnumber;
    lnumber = strtoul(string, &endptr, 10);
    printf("string = %s long = %lu\n", string, lnumber);
    return 0;
}

```

strupr, _fstrupr

- Function** Converts lowercase letters in a string to uppercase.
- Syntax** `#include <string.h>`
Near version: `char *strupr(char *s);`
Far version: `char far * far _fstrupr(char far *s)`
- Prototype in** `string.h`
- Remarks** **strupr** converts lowercase letters (*a-z*) in string *s* to uppercase (*A-Z*). No other characters are changed.

strupr, _strupr

Return value **strupr** returns *s*.

Portability **strupr** is unique to DOS.

See also **strlwr**

Example

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

int main(void)
{
    char *string = "abcdefghijklmnopqrstuvwxyz", *ptr;

    /* converts string to uppercase characters */
    ptr = strupr(string);
    printf("%s\n", ptr);
    return 0;
}
```

strxfrm

Function Transforms a portion of a string.

Syntax `#include<string.h>`
`size_t strxfrm(char *s1, char *s2, size_t n);`

Prototype in string.h

Remarks **strxfrm** transforms the string pointed to by *s2* into the string *s1* for no more than *n* characters.

Return value Number of characters copied.

Portability **strxfrm** is compatible with ANSI C.

See also **strcoll**, **strncpy**

Example

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

int main(void)
{
    char *target;
    char *source = "Frank Borland";
    int length;

    /* allocate space for the target string */
    target = calloc(80, sizeof(char));
```

```

    /* copy source over to target and get length */
    length = strxfrm(target, source, 80);

    /* print out the results */
    printf("%s has the length %d\n", target, length);
    return 0;
}

```

swab

Function Swaps bytes.

Syntax `#include <stdlib.h>`
`void swab(char *from, char *to, int nbytes);`

Prototype in `stdlib.h`

Remarks **swab** copies *nbytes* bytes from the *from* string to the *to* string. Adjacent even- and odd-byte positions are swapped. This is useful for moving data from one machine to another machine with a different byte order. *nbytes* should be even.

Return value None.

Portability **swab** is available on UNIX systems.

Example

```

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

char source[15] = "rFna koBlrna d";
char target[15];

int main(void)
{
    swab(source, target, strlen(source));
    printf("This is target: %s\n", target);
    return 0;
}

```

system

Function	Issues a DOS command.
Syntax	<pre>#include <stdlib.h> int system(const char *<i>command</i>);</pre>
Prototype in	stdlib.h, process.h
Remarks	<p>system invokes the DOS COMMAND.COM file to execute a DOS command, batch file, or other program named by the string <i>command</i>, from inside an executing C program.</p> <p>To be located and executed, the program must be in the current directory or in one of the directories listed in the PATH string in the environment.</p> <p>The COMSPEC environment variable is used to find the COMMAND.COM file, so that file need not be in the current directory.</p>
Return value	If <i>command</i> is a NULL pointer, then system returns nonzero if a command processor is available. If command is not a NULL pointer, system returns zero if the command processor was successfully started. If an error occurred, a -1 is returned and <i>errno</i> is set to ENOENT, ENOMEM, E2BIG, or ENOEXEC.
Portability	system is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie. This function is not Windows compatible.
See also	exec..., _fpreset, searchpath, spawn...
Example	<pre>#include <stdlib.h> #include <stdio.h> int main(void) { printf("About to spawn command.com and run a DOS command\n"); system("dir"); return 0; }</pre>

tan

Function	Calculates the tangent.	
Syntax	<i>Real version:</i> #include <math.h> double tan(double x);	<i>Complex version:</i> #include <complex.h> complex tan(complex x);
Prototype in	<i>Real version:</i> math.h	<i>Complex version:</i> complex.h
Remarks	<p>tan calculates the tangent. Angles are specified in radians.</p> <p>Error handling for this routine can be modified through the function matherr.</p> <p>The complex tangent is defined by</p> $\tan(z) = \sin(z) / \cos(z)$	
Return value	tan returns the tangent of x , $\sin(x)/\cos(x)$.	
Portability	The real version of tan is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.	
See also	acos, asin, atan, atan2, complex, cos, sin	
Example	<pre>#include <stdio.h> #include <math.h> int main(void) { double result, x; x = 0.5; result = tan(x); printf("The sin of %lf is %lf\n", x, result); return 0; }</pre>	

tanh

Function	Calculates the hyperbolic tangent.	
Syntax	<i>Real version:</i> #include <math.h> double tanh(double <i>x</i>);	<i>Complex version:</i> #include <complex.h> complex tanh(complex <i>x</i>);
Prototype in	<i>Real version:</i> math.h	<i>Complex version:</i> complex.h
Remarks	<p>tanh computes the hyperbolic tangent, $\sinh(x)/\cosh(x)$.</p> <p>Error handling for tanh can be modified through the function matherr.</p> <p>The complex hyperbolic tangent is defined by</p> $\mathbf{tanh}(z) = \mathbf{sinh}(z) / \mathbf{cosh}(z)$	
Return value	tanh returns the hyperbolic tangent of <i>x</i> .	
Portability	The real version of tanh is available on UNIX systems and is defined in ANSI C. The complex version of this function requires C++ and probably is not portable.	
See also	complex, cos, cosh, sin, sinh, tan	
Example	<pre>#include <stdio.h> #include <math.h> int main(void) { double result, x; x = 0.5; result = tanh(x); printf("The hyperbolic tangent of %lf is %lf\n", x, result); return 0; }</pre>	

tell

Function	Gets the current position of a file pointer.
Syntax	#include <io.h> long tell(int <i>handle</i>);
Prototype in	io.h

- Remarks** `tell` gets the current position of the file pointer associated with *handle* and expresses it as the number of bytes from the beginning of the file.
- Return value** `tell` returns the current file pointer position. A return of `-1` (long) indicates an error, and the global variable *errno* is set to
- `EBADF` Bad file number
- Portability** `tell` is available on all UNIX systems.
- See also** `fgetpos`, `fseek`, `ftell`, `lseek`

Example

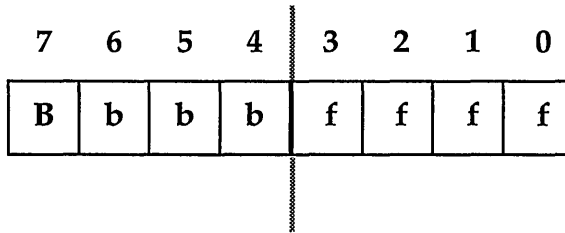
```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
    int handle;
    char msg[] = "Hello world";
    if ((handle = open("TEST.$$$", O_CREAT | O_TEXT | O_APPEND)) == -1) {
        perror("Error:");
        return 1;
    }
    write(handle, msg, strlen(msg));
    printf("The file pointer is at byte %ld\n", tell(handle));
    close(handle);
    return 0;
}
```

textattr

- Function** Sets text attributes.
- Syntax** `#include <conio.h>`
`void textattr(int newattr);`
- Prototype in** `conio.h`
- Remarks** `textattr` lets you set both the foreground and background colors in a single call. (Normally, you set the attributes with `textcolor` and `textbackground`.)
- This function does not affect any characters currently on the screen; it only affects those displayed by functions (such as `cprintf`) performing text mode, direct video output *after* this function is called.
- The color information is encoded in the *newattr* parameter as follows:

textattr



In this 8-bit *newattr* parameter,

ffff is the 4-bit foreground color (0 to 15).

bbb is the 3-bit background color (0 to 7).

B is the blink-enable bit.

If the blink-enable bit is on, the character blinks. This can be accomplished by adding the constant `BLINK` to the attribute.

If you use the symbolic color constants defined in `conio.h` for creating text attributes with `textattr`, note the following limitations on the color you select for the background:

- You can only select one of the first eight colors for the background.
- You must shift the selected background color left by 4 bits to move it into the correct bit positions.

These symbolic constants are listed in the following table:

Symbolic constant	Numeric value	Foreground or background?
BLACK	0	Both
BLUE	1	Both
GREEN	2	Both
CYAN	3	Both
RED	4	Both
MAGENTA	5	Both
BROWN	6	Both
LIGHTGRAY	7	Both
DARKGRAY	8	Foreground only
LIGHTBLUE	9	Foreground only
LIGHTGREEN	10	Foreground only
LIGHTCYAN	11	Foreground only
LIGHTRED	12	Foreground only
LIGHTMAGENTA	13	Foreground only
YELLOW	14	Foreground only
WHITE	15	Foreground only
BLINK	128	Foreground only

Return value None.

Portability **textattr** works only on IBM PCs and compatible systems. This function is not Windows compatible.

See also **gettextinfo**, **highvideo**, **lowvideo**, **normvideo**, **textbackground**, **textcolor**

Example

```
#include <conio.h>

int main(void)
{
    int i;
    clrscr();
    for (i = 0; i < 9; i++) {
        textattr(i + ((i+1) << 4));
        printf("This is a test\r\n");
    }
    return 0;
}
```

textbackground

Function Selects new text background color.

Syntax `#include <conio.h>`
`void textbackground(int newcolor);`

Prototype in conio.h

Remarks **textbackground** selects the background color. This function works for functions that produce output in text mode directly to the screen. *newcolor* selects the new background color. You can set *newcolor* to an integer from 0 to 7, or to one of the symbolic constants defined in conio.h. If you use symbolic constants, you must include conio.h.

Once you have called **textbackground**, all subsequent functions using direct video output (such as **cprintf**) will use *newcolor*. **textbackground** does not affect any characters currently onscreen.

The following table lists the symbolic constants and the numeric values of the allowable colors:

Symbolic constant	Numeric value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7

Return value None.

Portability **textbackground** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **gettextinfo**, **textattr**, **textcolor**

Example

```
#include <conio.h>

int main(void)
{
    int i, j;
    clrscr();
    for (i=0; i<9; i++) {
        for (j=0; j<80; j++)
            cprintf("C");
    }
}
```

```

        cprintf("\r\n");
        textcolor(i+1);
        textbackground(i);
    }
    return 0;
}

```

textcolor

Function Selects new character color in text mode.

Syntax #include <conio.h>
void textcolor(int *newcolor*);

Prototype in conio.h

Remarks **textcolor** selects the foreground character color. This function works for the console output functions. *newcolor* selects the new foreground color. You can set *newcolor* to an integer as given in the table below, or to one of the symbolic constants defined in conio.h. If you use symbolic constants, you must include conio.h.

Once you have called **textcolor**, all subsequent functions using direct video output (such as **cprintf**) will use *newcolor*. **textcolor** does not affect any characters currently onscreen.

The following table lists the allowable colors (as symbolic constants) and their numeric values:

Symbolic constant	Numeric value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15
BLINK	128

You can make the characters blink by adding 128 to the foreground color. The predefined constant BLINK exists for this purpose; for example,

```
textcolor(CYAN + BLINK);
```



Some monitors do not recognize the intensity signal used to create the eight "light" colors (8-15). On such monitors, the light colors will be displayed as their "dark" equivalents (0-7). Also, systems that do not display in color can treat these numbers as shades of one color, special patterns, or special attributes (such as underlined, bold, italics, and so on). Exactly what you'll see on such systems depends on your hardware.

Return value None.

Portability **textcolor** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **gettextinfo**, **highvideo**, **lowvideo**, **normvideo**, **textattr**, **textbackground**

Example

```
#include <conio.h>

int main(void)
{
    int i;
    for (i=0; i<15; i++) {
        textcolor(i);
        printf("Foreground Color\r\n");
    }
    return 0;
}
```

textheight

Function Returns the height of a string in pixels.

Syntax `#include <graphics.h>`
`int far textheight(char far *textstring);`

Prototype in graphics.h

Remarks The graphics function **textheight** takes the current font size and multiplication factor, and determines the height of *textstring* in pixels. This function is useful for adjusting the spacing between lines, computing viewport heights, sizing a title to make it fit on a graph or in a box, and so on.

For example, with the 8×8 bit-mapped font and a multiplication factor of 1 (set by **settextstyle**), the string *TurboC++* is 8 pixels high.



Use **textheight** to compute the height of strings, instead of doing the computations manually. By using this function, no source code modifications have to be made when different fonts are selected.

Return value **textheight** returns the text height in pixels.

Portability This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.

See also **gettextsettings**, **outtext**, **outtextxy**, **settextstyle**, **textwidth**

Example

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

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int y = 0;
    int i;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
```

textheight

```
if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

/* draw some text on the screen */
for (i=1; i<11; i++)
{
    /* select the text style, direction, and size */
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

    /* create a message string */
    sprintf(msg, "Size: %d", i);

    /* output the message */
    outtextxy(1, y, msg);

    /* advance to the next text line */
    y += textheight(msg);
}

/* clean up */
getch();
closegraph();
return 0;
}
```

textmode

Function	Puts screen in text mode.
Syntax	<code>#include <conio.h></code> <code>void textmode(int <i>newmode</i>);</code>
Prototype in	conio.h
Remarks	textmode selects a specific text mode.

You can give the text mode (the argument *newmode*) by using a symbolic constant from the enumeration type *text_modes* (defined in conio.h). If you use these constants, you must include conio.h.

The *text_modes* type constants, their numeric values, and the modes they specify are given in the following table:

Symbolic constant	Numeric value	Text mode
LASTMODE	-1	Previous text mode
BW40	0	Black and white, 40 columns
C40	1	Color, 40 columns
BW80	2	Black and white, 80 columns
C80	3	Color, 80 columns
MONO	7	Monochrome, 80 columns
C4350	64	EGA 43-line and VGA 50-line modes

When **textmode** is called, the current window is reset to the entire screen, and the current text attributes are reset to normal, corresponding to a call to **normvideo**.

Specifying LASTMODE to **textmode** causes the most recently selected text mode to be reselected.

textmode should be used only when the screen is in text mode (presumably to change to a different text mode). This is the only context in which **textmode** should be used. When the screen is in graphics mode, use **restorecrtmode** instead to escape temporarily to text mode.

Return value None.

Portability **textmode** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **gettextinfo**, **window**

Example

```
#include <conio.h>

int main(void)
{
    textmode(BW40);
    cprintf("ABC");
    getch();
    textmode(C40);
    cprintf("ABC");
    getch();
    textmode(BW80);
    cprintf("ABC");
    getch();
    textmode(C80);
    cprintf("ABC");
    getch();
    textmode(MONO);
    cprintf("ABC");
    getch();
}
```


textwidth

```
    return 0;  
}
```

textwidth

- Function** Returns the width of a string in pixels.
- Syntax** `#include <graphics.h>`
`int far textwidth(char far *textstring);`
- Prototype in** graphics.h
- Remarks** The graphics function **textwidth** takes the string length, current font size, and multiplication factor, and determines the width of *textstring* in pixels. This function is useful for computing viewport widths, sizing a title to make it fit on a graph or in a box, and so on.
- ➡ Use **textwidth** to compute the width of strings, instead of doing the computations manually. When you use this function, no source code modifications have to be made when different fonts are selected.
- Return value** **textwidth** returns the text width in pixels.
- Portability** This function is unique to Borland C++. It works only with IBM PCs and compatibles equipped with supported graphics display adapters. This function is not Windows compatible.
- See also** **gettextsettings, outtext, outtextxy, settextstyle, textheight**
- Example**
- ```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
 /* request autodetection */
 int gdriver = DETECT, gmode, errorcode;
 int x = 0, y = 0;
 int i;
 char msg[80];

 /* initialize graphics and local variables */
 initgraph(&gdriver, &gmode, "");

 /* read result of initialization */
 errorcode = graphresult();
 if (errorcode != grOk) /* an error occurred */
```

```

 {
 printf("Graphics error: %s\n", grapherrormsg(errorcode));
 printf("Press any key to halt:");
 getch();
 exit(1); /* terminate with an error code */
 }

 y = getmaxy() / 2;
 setttextjustify(LEFT_TEXT, CENTER_TEXT);
 for (i=1; i<11; i++)
 {
 /* select the text style, direction, and size */
 setttextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

 /* create a message string */
 sprintf(msg, "Size: %d", i);

 /* output the message */
 outtextxy(x, y, msg);

 /* advance to the end of the text */
 x += textwidth(msg);
 }

 /* clean up */
 getch();
 closegraph();
 return 0;
}

```

## time

---

|                     |                                                                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Gets time of day.                                                                                                                                                                                                 |
| <b>Syntax</b>       | <code>#include &lt;time.h&gt;</code><br><code>time_t time(time_t *timer);</code>                                                                                                                                  |
| <b>Prototype in</b> | time.h                                                                                                                                                                                                            |
| <b>Remarks</b>      | <b>time</b> gives the current time, in seconds, elapsed since 00:00:00 GMT, January 1, 1970, and stores that value in the location pointed to by <i>timer</i> , provided that <i>timer</i> is not a null pointer. |
| <b>Return value</b> | <b>time</b> returns the elapsed time in seconds, as described.                                                                                                                                                    |
| <b>Portability</b>  | <b>time</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                |
| <b>See also</b>     | <b>asctime</b> , <b>ctime</b> , <b>difftime</b> , <b>ftime</b> , <b>gettime</b> , <b>gmtime</b> , <b>localtime</b> , <b>settime</b> , <b>stime</b> , <b>tzset</b>                                                 |

## time

**Example**

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

int main(void)
{
 time_t t;
 t = time(NULL);
 printf("The number of seconds since January 1, 1970 is %ld",t);
 return 0;
}
```

## tmpfile

---

**Function** Opens a "scratch" file in binary mode.

**Syntax** `#include <stdio.h>`  
`FILE *tmpfile(void);`

**Prototype in** `stdio.h`

**Remarks** **tmpfile** creates a temporary binary file and opens it for update (*w + b*). The file is automatically removed when it's closed or when your program terminates.

**Return value** **tmpfile** returns a pointer to the stream of the temporary file created. If the file can't be created, **tmpfile** returns null.

**Portability** **tmpfile** is available on UNIX systems and is defined in ANSI C.

**See also** **fopen, tmpnam**

**Example**

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

int main(void)
{
 FILE *tempfp;
 tempfp = tmpfile();
 if (tempfp)
 printf("Temporary file created\n");
 else {
 printf("Unable to create temporary file\n");
 exit(1);
 }
 return 0;
}
```

# tmpnam

---

**Function** Creates a unique file name.

**Syntax** `#include <stdio.h>`  
`char *tmpnam(char *s);`

**Prototype in** `stdio.h`

**Remarks** **tmpnam** creates a unique file name, which can safely be used as the name of a temporary file. **tmpnam** generates a different string each time you call it, up to `TMP_MAX` times. `TMP_MAX` is defined in `stdio.h` as 65,535.

The parameter to **tmpnam**, *s*, is either null or a pointer to an array of at least `L_tmpnam` characters. `L_tmpnam` is defined in `stdio.h`. If *s* is null, **tmpnam** leaves the generated temporary file name in an internal static object and returns a pointer to that object. If *s* is not null, **tmpnam** places its result in the pointed-to array, which must be at least `L_tmpnam` characters long, and returns *s*.



If you do create such a temporary file with **tmpnam**, it is your responsibility to delete the file name (for example, with a call to **remove**). It is not deleted automatically. (**tmpfile** *does* delete the file name.)

**Return value** If *s* is null, **tmpnam** returns a pointer to an internal static object. Otherwise, **tmpnam** returns *s*.

**Portability** **tmpnam** is available on UNIX systems and is defined in ANSI C.

**See also** **tmpfile**

**Example**

```
#include <stdio.h>

int main(void)
{
 char name[13];
 tmpnam(name);
 printf("Temporary name: %s\n", name);
 return 0;
}
```

## toascii

---

|                     |                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Translates characters to ASCII format.                                                                                                                                                                        |
| <b>Syntax</b>       | <code>#include &lt;ctype.h&gt;</code><br><code>int toascii(int <i>c</i>);</code>                                                                                                                              |
| <b>Prototype in</b> | <code>ctype.h</code>                                                                                                                                                                                          |
| <b>Remarks</b>      | <b>toascii</b> is a macro that converts the integer <i>c</i> to ASCII by clearing all but the lower 7 bits; this gives a value in the range 0 to 127.                                                         |
| <b>Return value</b> | <b>toascii</b> returns the converted value of <i>c</i> .                                                                                                                                                      |
| <b>Portability</b>  | <b>toascii</b> is available on UNIX systems.                                                                                                                                                                  |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;ctype.h&gt;  int main(void) {     int number, result;     number = 511;     result = toascii(number);     printf("%d %d\n", number, result);     return 0; }</pre> |

## \_tolower

---

|                     |                                                                                                                                                                                                                                                 |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Translates characters to lowercase.                                                                                                                                                                                                             |
| <b>Syntax</b>       | <code>#include &lt;ctype.h&gt;</code><br><code>int _tolower(int <i>ch</i>);</code>                                                                                                                                                              |
| <b>Prototype in</b> | <code>ctype.h</code>                                                                                                                                                                                                                            |
| <b>Remarks</b>      | <b>_tolower</b> is a macro that does the same conversion as <b>tolower</b> , except that it should be used only when <i>ch</i> is known to be uppercase ( <i>A-Z</i> ).<br><br>To use <b>_tolower</b> , you must include <code>ctype.h</code> . |
| <b>Return value</b> | <b>_tolower</b> returns the converted value of <i>ch</i> if it is uppercase; otherwise, the result is undefined.                                                                                                                                |
| <b>Portability</b>  | <b>_tolower</b> is available on UNIX systems.                                                                                                                                                                                                   |
| <b>Example</b>      | <pre>#include &lt;string.h&gt; #include &lt;stdio.h&gt;</pre>                                                                                                                                                                                   |

```

#include <ctype.h>

int main(void)
{
 int length, i;
 char *string = "THIS IS A STRING.";

 /* We should be checking each character to make sure it is an uppercase before
 passing it to _tolower! The result of passing it a non-uppercase is
 undefined. */
 length = strlen(string);
 for (i = 0; i < length; i++) {
 string[i] = _tolower(string[i]);
 }
 printf("%s\n", string);
 return 0;
}

```

## tolower

---

**Function** Translates characters to lowercase.

**Syntax** `#include <ctype.h>`  
`int tolower(int ch);`

**Prototype in** `ctype.h`

**Remarks** **tolower** is a function that converts an integer *ch* (in the range EOF to 255) to its lowercase value (*a* to *z*; if it was uppercase, *A* to *Z*). All others are left unchanged.

**Return value** **tolower** returns the converted value of *ch* if it is uppercase; it returns all others unchanged.

**Portability** **tolower** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**Example**

```

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

int main(void)
{
 int length, i;
 char *string = "THIS IS A STRING";
 length = strlen(string);
 for (i = 0; i < length; i++) {
 string[i] = tolower(string[i]);
 }
}

```

## tolower

```
 printf("%s\n", string);
 return 0;
}
```

## tolower

---

**Function** Translates characters to uppercase.

**Syntax** #include <ctype.h>  
int tolower(int *ch*);

**Prototype in** ctype.h

**Remarks** tolower is a macro that does the same conversion as **tolower**, except that it should be used only when *ch* is known to be lowercase (*a* to *z*).

To use tolower, you must include ctype.h.

**Return value** tolower returns the converted value of *ch* if it is lowercase; otherwise, the result is undefined.

**Portability** tolower is available on UNIX systems.

**Example**

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

int main(void)
{
 int length, i;
 char *string = "this is a string";

 /* We should be checking each character to make sure it is lowercase before
 passing it to tolower. The result passing a non-lowercase is undefined. */
 length = strlen(string);
 for (i = 0; i < length; i++) {
 string[i] = tolower(string[i]);
 }
 printf("%s\n", string);
 return 0;
}
```

## toupper

---

**Function** Translates characters to uppercase.

**Syntax** `#include <ctype.h>`  
`int toupper(int ch);`

**Prototype in** `ctype.h`

**Remarks** **toupper** is a function that converts an integer *ch* (in the range EOF to 255) to its uppercase value (*A* to *Z*; if it was lowercase, *a* to *z*). All others are left unchanged.

**Return value** **toupper** returns the converted value of *ch* if it is lowercase; it returns all others unchanged.

**Portability** **toupper** is available on UNIX systems and is defined in ANSI C. It is compatible with Kernighan and Ritchie.

**Example**

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

int main(void)
{
 int length, i;
 char *string = "this is a string";
 length = strlen(string);
 for (i = 0; i < length; i++) {
 string[i] = toupper(string[i]);
 }
 printf("%s\n", string);
 return 0;
}
```

T-Z

## tzset

---

**Function** Sets value of global variables *daylight*, *timezone*, and *tzname*.

**Syntax** `#include <time.h>`  
`void tzset(void)`

**Prototype in** `time.h`



**Remarks** *tzset* sets the *daylight*, *timezone*, and *tzname* global variables based on the environment variable *TZ*. The library functions **ftime** and **localtime** use these global variables to correct Greenwich mean time (GMT) to whatever the local time zone is. The format of the *TZ* environment string follows:

```
TZ = zzz[+/-]d[d] [lll]
```

*zzz* is a three-character string representing the name of the current time zone. All three characters are required. For example, the string "PST" could be used to represent Pacific Standard Time.

[+/-]d[d] is a required field containing an optionally signed number with 1 or more digits. This number is the local time zone's difference from GMT in hours. Positive numbers adjust westward from GMT. Negative numbers adjust eastward from GMT. For example, the number 5 = EST, +8 = PST, and -1 = continental Europe. This number is used in the calculation of the global variable *timezone*. *timezone* is the difference in seconds between GMT and the local time zone.

*lll* is an optional three-character field that represents the local time zone daylight saving time. For example, the string "PDT" could be used to represent Pacific daylight saving time. If this field is present, it will cause the global variable *daylight* to be set nonzero. If this field is absent, *daylight* will be set to zero.

If the *TZ* environment string isn't present or isn't in the preceding form, a default *TZ* = "EST5EDT" is presumed for the purposes of assigning values to the global variables *daylight*, *timezone*, and *tzname*.

The global variable *tzname*[0] points to a three-character string with the value of the time-zone name from the *TZ* environment string. *tzname*[1] points to a three-character string with the value of the daylight saving time-zone name from the *TZ* environment string. If no daylight saving name is present, *tzname*[1] points to a null string.

**Return value** None.

**Portability** *tzset* is available on UNIX and XENIX systems.

**See also** **asctime**, **ctime**, **ftime**, **gmtime**, **localtime**, **stime**, **time**

**Example**

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

int main(void)
{
 time_t td;
 putenv("TZ=PST8PDT");
```

```

 tzset();
 time(&td);
 printf("Current time = %s\n", asctime(localtime(&td)));
 return 0;
}

```

## ultoa

---

**Function** Converts an **unsigned long** to a string.

**Syntax** `#include <stdlib.h>`  
`char *ultoa(unsigned long value, char *string, int radix);`

**Prototype in** `stdlib.h`

**Remarks** **ultoa** converts *value* to a null-terminated string and stores the result in *string*. *value* is an **unsigned long**.

*radix* specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. **ultoa** performs no overflow checking, and if *value* is negative and *radix* equals 10, it does not set the minus sign.



The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (`\0`). **ultoa** can return up to 33 bytes.

**Return value** **ultoa** returns *string*.

**Portability** **ultoa** is unique to DOS.

**See also** **itoa**, **ltoa**

**Example**

```

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

int main(void)
{
 unsigned long lnumber = 3123456789L;
 char string[25];
 ultoa(lnumber, string, 10);
 printf("string = %s unsigned long = %lu\n", string, lnumber);
 return 0;
}

```

## ungetc

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Pushes a character back into input stream.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Syntax</b>       | <code>#include &lt;stdio.h&gt;</code><br><code>int ungetc(int c, FILE *stream);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Prototype in</b> | stdio.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Remarks</b>      | <b>ungetc</b> pushes the character <i>c</i> back onto the named input <i>stream</i> , which must be open for reading. This character will be returned on the next call to <b>getc</b> or <b>fread</b> for that <i>stream</i> . One character can be pushed back in all situations. A second call to <b>ungetc</b> without a call to <b>getc</b> will force the previous character to be forgotten. A call to <b>fflush</b> , <b>fseek</b> , <b>fsetpos</b> , or <b>rewind</b> erases all memory of any pushed-back characters.          |
| <b>Return value</b> | On success, <b>ungetc</b> returns the character pushed back; it returns EOF if the operation fails.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Portability</b>  | <b>ungetc</b> is available on UNIX systems and is defined in ANSI C.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>See also</b>     | <b>fgetc</b> , <b>getc</b> , <b>getchar</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;ctype.h&gt;  int main(void) {     int i=0;     char ch;     puts("Input an integer followed by a char:");      /* read chars until nondigit or EOF */     while((ch = getchar()) != EOF &amp;&amp; isdigit(ch))         i = 10 * i + ch - 48;    /* convert ASCII into int value */      /* if nondigit char was read, push it back into input buffer */     if (ch != EOF)         ungetc(ch, stdin);     printf("i = %d, next char in buffer = %c\n", i, getchar());     return 0; }</pre> |

# ungetch

---

**Function** Pushes a character back to the keyboard buffer.

**Syntax** `#include <conio.h>`  
`int ungetch(int ch);`

**Prototype in** `conio.h`

**Remarks** **ungetch** pushes the character *ch* back to the console, causing *ch* to be the next character read. The **ungetch** function fails if it is called more than once before the next read.

**Return value** **ungetch** returns the character *ch* if it is successful. A return value of EOF indicates an error.

**Portability** **ungetch** is available on UNIX systems. This function is not Windows compatible.

**See also** **getch, getche**

**Example**

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

int main(void)
{
 int i=0;
 char ch;
 puts("Input an integer followed by a char:");

 /* read chars until nondigit or EOF */
 while((ch = getche()) != EOF && isdigit(ch))
 i = 10 * i + ch - 48; /* convert ASCII into int value */

 /* if nondigit char was read, push it back into input buffer */
 if (ch != EOF)
 ungetch(ch);
 printf("\n\ni = %d, next char in buffer = %c\n", i, getch());
 return 0;
}
```

## unixtodos

---

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Function</b>     | Converts date and time from UNIX to DOS format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Syntax</b>       | <code>#include &lt;dos.h&gt;</code><br><code>void unixtodos(long <i>time</i>, struct date *<i>d</i>, struct time *<i>t</i>);</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Prototype in</b> | dos.h                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Remarks</b>      | <b>unixtodos</b> converts the UNIX-format time given in <i>time</i> to DOS format and fills in the <b>date</b> and <b>time</b> structures pointed to by <i>d</i> and <i>t</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Return value</b> | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Portability</b>  | <b>unixtodos</b> is unique to DOS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>See also</b>     | <b>dostounix</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>Example</b>      | <pre>#include &lt;stdio.h&gt; #include &lt;dos.h&gt;  char *month[] = { "---", "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",                  "Sep", "Oct", "Nov", "Dec"};  #define SECONDS_PER_DAY 86400L      /* number of secs in one day /  struct date dt; struct time tm;  int main(void) {     unsigned long val;      /* get today's date and time */     getdate(&amp;dt);     gettime(&amp;tm);     printf("Today is %d %s %d\n", dt.da_day, month[dt.da_mon], dt.da_year);      /* convert date and time to unix format (number of secs since Jan 1, 1970 */     val = dostounix(&amp;dt, &amp;tm);      /* subtract 42 days worth of seconds */     val -= (SECONDS_PER_DAY * 42);      /* convert back to dos time and date */     unixtodos(val, &amp;dt, &amp;tm);     printf("42 days ago it was %d %s %d\n", dt.da_day, month[dt.da_mon],           dt.da_year);     return 0; }</pre> |

# unlink

---

**Function** Deletes a file.

**Syntax** `#include <io.h>`  
`int unlink(const char *filename);`

**Prototype in** `dos.h, io.h, stdio.h`

**Remarks** **unlink** deletes a file specified by *filename*. Any DOS drive, path, and file name can be used as a *filename*. Wildcards are not allowed.

Read-only files cannot be deleted by this call. To remove read-only files, first use **chmod** or **\_chmod** to change the read-only attribute.



If your file is open, be sure to close it before unlinking it.

**Return value** On successful completion, **unlink** returns 0. On error, it returns -1 and the global variable *errno* is set to one of the following values:

ENOENT Path or file name not found  
 EACCES Permission denied

**Portability** **unlink** is available on UNIX systems.

**See also** **chmod, remove**

**Example**

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

int main(void)
{
 FILE *fp = fopen("junk.jnk","w");
 int status;
 fprintf(fp,"junk");
 status = access("junk.jnk",0);
 if (status == 0)
 printf("File exists\n");
 else
 printf("File doesn't exist\n");
 fclose(fp);
 unlink("junk.jnk");
 status = access("junk.jnk",0);
 if (status == 0)
 printf("File exists\n");
 else
 printf("File doesn't exist\n");
}
```

## unlock

```
 return 0;
}
```

## unlock

---

- Function** Releases file-sharing locks.
- Syntax** `#include <io.h>`  
`int unlock(int handle, long offset, long length);`
- Prototype in** `io.h`
- Remarks** **unlock** provides an interface to the DOS 3.x file-sharing mechanism. **unlock** removes a lock previously placed with a call to **lock**. To avoid error, all locks must be removed before a file is closed. A program must release all locks before completing.
- Return value** **unlock** returns 0 on success, -1 on error.
- Portability** **unlock** is unique to DOS 3.x. Older versions of DOS do not support this call.
- See also** **lock**, **sopen**

**Example**

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
 int handle, status;
 long length;
 handle = sopen("c:\\autoexec.bat", O_RDONLY, SH_DENYNO, S_IREAD);
 if (handle < 0) {
 printf("sopen failed\n");
 exit(1);
 }

 length = filelength(handle);
 status = lock(handle, 0L, length/2);
 if (status == 0)
 printf("lock succeeded\n");
 else
 printf("lock failed\n");
 status = unlock(handle, 0L, length/2);
}
```

```

 if (status == 0)
 printf("unlock succeeded\n");
 else
 printf("unlock failed\n");
 close(handle);
 return 0;
}

```

## va\_arg, va\_end, va\_start

---

**Function** Implement a variable argument list.

**Syntax** `#include <stdarg.h>`  
`void va_start(va_list ap, lastfix);`  
`type va_arg(va_list ap, type);`  
`void va_end(va_list ap);`

**Prototype in** `stdarg.h`

**Remarks** Some C functions, such as **vfprintf** and **vprintf**, take variable argument lists in addition to taking a number of fixed (known) parameters. The **va\_arg**, **va\_end**, and **va\_start** macros provide a portable way to access these argument lists. They are used for stepping through a list of arguments when the called function does not know the number and types of the arguments being passed.

The header file `stdarg.h` declares one type (*va\_list*) and three macros (**va\_start**, **va\_arg**, and **va\_end**).

**va\_list:** This array holds information needed by **va\_arg** and **va\_end**. When a called function takes a variable argument list, it declares a variable *ap* of type *va\_list*.

**va\_start:** This routine (implemented as a macro) sets *ap* to point to the first of the variable arguments being passed to the function. **va\_start** must be used before the first call to **va\_arg** or **va\_end**.

**va\_start** takes two parameters: *ap* and *lastfix*. (*ap* is explained under *va\_list* in the preceding paragraph; *lastfix* is the name of the last fixed parameter being passed to the called function.)

**va\_arg:** This routine (also implemented as a macro) expands to an expression that has the same type and value as the next argument being passed (one of the variable arguments). The variable *ap* to **va\_arg** should be the same *ap* that **va\_start** initialized.



➡ Because of default promotions, you can't use **char**, **unsigned char**, or **float** types with **va\_arg**.

The first time **va\_arg** is used, it returns the first argument in the list. Each successive time **va\_arg** is used, it returns the next argument in the list. It does this by first dereferencing *ap*, and then incrementing *ap* to point to the following item. **va\_arg** uses the *type* to both perform the dereference and to locate the following item. Each successive time **va\_arg** is invoked, it modifies *ap* to point to the next argument in the list.

**va\_end**: This macro helps the called function perform a normal return. **va\_end** might modify *ap* in such a way that it cannot be used unless **va\_start** is recalled. **va\_end** should be called after **va\_arg** has read all the arguments; failure to do so might cause strange, undefined behavior in your program.

**Return value** **va\_start** and **va\_end** return no values; **va\_arg** returns the current argument in the list (the one that *ap* is pointing to).

**Portability** **va\_arg**, **va\_start**, and **va\_end** are available on UNIX systems.

**See also** **v...printf**, **v...scanf**

### Example 1

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

/* calculate sum of a 0 terminated list */
void sum(char *msg, ...)
{
 int total = 0;
 va_list ap;
 int arg;
 va_start(ap, msg);
 while ((arg = va_arg(ap,int)) != 0) {
 total += arg;
 }
 printf(msg, total);
}

int main(void) {
 sum("The total of 1+2+3+4 is %d\n", 1,2,3,4,0);
 return 0;
}
```

### Program output

The total of 1+2+3+4 is 10

**Example 2**

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

void error(char *format,...)
{
 va_list argptr;
 printf("Error: ");
 va_start(argptr, format);
 vprintf(format, argptr);
 va_end(argptr);
}

int main(void) {
 int value = -1;
 error("This is just an error message\n");
 error("Invalid value %d encountered\n", value);
 return 0;
}
```

**Program output**

```
Error: This is just an error message
Error: Invalid value -1 encountered
```

## vfprintf

---

**Function** Writes formatted output to a stream.

**Syntax** #include <stdio.h>  
int vfprintf(FILE \*stream, const char \*format, va\_list arglist);

**Prototype in** stdio.h

**Remarks** The **v...printf** functions are known as *alternate entry points* for the **...printf** functions. They behave exactly like their **...printf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See **printf** for details on format specifiers.

**vfprintf** accepts a pointer to a series of arguments, applies to each argument a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a stream. There must be the same number of format specifiers as arguments.

**Return value** **vfprintf** returns the number of bytes output. In the event of error, **vfprintf** returns EOF.

**Portability** **vfprintf** is available on UNIX System V, and it is compatible with ANSI C.

## vfprintf

**See also** `printf`, `va_arg`, `va_end`, `va_start`

**Example**

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

FILE *fp;
int vfprintf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
 va_start(argptr, fmt);
 cnt = vfprintf(fp, fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber = 30;
 float fnumber = 90.0;
 char string[4] = "abc";
 fp = tmpfile();
 if (fp == NULL) {
 perror("tmpfile() call");
 exit(1);
 }
 vfprintf("%d %f %s", inumber, fnumber, string);
 rewind(fp);
 fscanf(fp, "%d %f %s", &inumber, &fnumber, string);
 printf("%d %f %s\n", inumber, fnumber, string);
 fclose(fp);
 return 0;
}
```

## vfscanf

---

- Function** Scans and formats input from a stream.
- Syntax** `#include <stdio.h>`  
`int vfscanf(FILE *stream, const char *format, va_list arglist);`
- Prototype in** `stdio.h`
- Remarks** The **v...scanf** functions are known as *alternate entry points* for the **...scanf** functions. They behave exactly like their **...scanf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See **scanf** for details on format specifiers.

**vfscanf** scans a series of input fields, one character at a time, reading from a stream. Then each field is formatted according to a format specifier passed to **vfscanf** in the format string pointed to by *format*. Finally, **vfscanf** stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

**vfscanf** might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See **scanf** for a discussion of possible causes.

**Return value** **vfscanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If **vfscanf** attempts to read at end-of-file, the return value is EOF.

**Portability** **vfscanf** is available on UNIX system V.

**See also** **scanf**, **scanf**, **va\_arg**, **va\_end**, **va\_start**

**Example**

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

FILE *fp;
int vfsf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
 va_start(argptr, fmt);
 cnt = vfscanf(fp, fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber = 30;
 float fnumber = 90.0;
 char string[4] = "abc";
 fp = tmpfile();
 if (fp == NULL) {
 perror("tmpfile() call");
 exit(1);
 }
 fprintf(fp, "%d %f %s\n", inumber, fnumber, string);
 rewind(fp);
 vfsf("%d %f %s", &inumber, &fnumber, string);
 printf("%d %f %s\n", inumber, fnumber, string);
}
```

```

 fclose(fp);
 return 0;
 }

```

## vprintf

---

**Function** Writes formatted output to stdout.

**Syntax** #include <stdarg.h>  
int vprintf(const char *\*format*, *va\_list* arglist);

**Prototype in** stdio.h

**Remarks** The **v...printf** functions are known as *alternate entry points* for the **...printf** functions. They behave exactly like their **...printf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See **printf** for details on format specifiers.

**vprintf** accepts a pointer to a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to stdout. There must be the same number of format specifiers as arguments.



When you use the **SS!=DS** flag, **vprintf** assumes that the address being passed is in the SS segment.

**Return value** **vprint** returns the number of bytes output. In the event of error, **vprint** returns EOF.

**Portability** **vprintf** is available on UNIX System V and is compatible with ANSI C. This function is not Windows compatible.

**See also** **printf**, **va\_arg**, **va\_end**, **va\_start**

**Example**

```

#include <stdio.h>

int vpf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
 va_start(argptr, format);
 cnt = vprintf(fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber = 30;
 float fnumber = 90.0;

```

```

char *string = "abc";
vprintf("%d %f %s\n", inumber, fnumber, string);
return 0;
}

```

## vscanf

---

**Function** Scans and formats input from stdin.

**Syntax** `#include <stdarg.h>`  
`int vscanf(const char *format, va_list arglist);`

**Prototype in** `stdio.h`

**Remarks** The **v...scanf** functions are known as *alternate entry points* for the **...scanf** functions. They behave exactly like their **...scanf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See **scanf** for details on format specifiers.

**vscanf** scans a series of input fields, one character at a time, reading from stdin. Then each field is formatted according to a format specifier passed to **vscanf** in the format string pointed to by *format*. Finally, **vscanf** stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

**vscanf** might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See **scanf** for a discussion of possible causes.

**Return value** **vscanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If **vscanf** attempts to read at end-of-file, the return value is EOF.

**Portability** **vscanf** is available on UNIX system V. This function is not Windows compatible.

**See also** **fscanf, scanf, va\_arg, va\_end, va\_start**

**Example**

```

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

int vscanf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
 printf("Enter an integer, a float, and a string (e.g., i,f,s,)\n");

```

## vscanf

```
 va_start(argptr, fmt);
 cnt = vscanf(fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber;
 float fnumber;
 char string[80];
 vscanf("%d, %f, %s", &inumber, &fnumber, string);
 printf("%d %f %s\n", inumber, fnumber, string);
 return 0;
}
```

## vsprintf

---

**Function** Writes formatted output to a string.

**Syntax** `#include <stdarg.h>`  
`int vsprintf(char *buffer, const char *format, va_list arglist);`

**Prototype in** `stdio.h`

**Remarks** The **v...printf** functions are known as *alternate entry points* for the **...printf** functions. They behave exactly like their **...printf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See **printf** for details on format specifiers.

**vsprintf** accepts a pointer to a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a string. There must be the same number of format specifiers as arguments.

**Return value** **vsprintf** returns the number of bytes output. In the event of error, **vsprintf** returns EOF.

**Portability** **vsprintf** is available on UNIX System V and is compatible with ANSI C.

**See also** **printf**, **va\_arg**, **va\_end**, **va\_start**

**Example**

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

char buffer[80];
int vspf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
```

```

 va_start(argptr, fmt);
 cnt = vsprintf(buffer, fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber = 30;
 float fnumber = 90.0;
 char string[4] = "abc";
 vsprintf("%d %f %s", inumber, fnumber, string);
 printf("%s\n", buffer);
 return 0;
}

```

## vsscanf

---

**Function** Scans and formats input from a stream.

**Syntax** #include <stdarg.h>  
int vsscanf(const char \*buffer, const char \*format, va\_list arglist);

**Prototype in** stdio.h

**Remarks** The **v...scanf** functions are known as *alternate entry points* for the **...scanf** functions. They behave exactly like their **...scanf** counterparts, but they accept a pointer to a list of arguments instead of an argument list.

See *scanf* for details on format specifiers.

**vsscanf** scans a series of input fields, one character at a time, reading from a stream. Then each field is formatted according to a format specifier passed to **vsscanf** in the format string pointed to by *format*. Finally, **vsscanf** stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

**vsscanf** might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See **scanf** for a discussion of possible causes.

**Return value** **vsscanf** returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If **vsscanf** attempts to read at end-of-string, the return value is EOF.

**Portability** **vsscanf** is available on UNIX system V.

T-Z



**See also** `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vfscanf`

**Example**

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

char buffer[80] = "30 90.0 abc";
int vssf(char *fmt, ...)
{
 va_list argptr;
 int cnt;
 fflush(stdin);
 va_start(argptr, fmt);
 cnt = vsscanf(buffer, fmt, argptr);
 va_end(argptr);
 return(cnt);
}

int main(void)
{
 int inumber;
 float fnumber;
 char string[80];
 vssf("%d %f %s", &inumber, &fnumber, string);
 printf("%d %f %s\n", inumber, fnumber, string);
 return 0;
}
```

## wherex

---

- Function** Gives horizontal cursor position within window.
- Syntax** `#include <conio.h>`  
`int wherex(void);`
- Prototype in** `conio.h`
- Remarks** **wherex** returns the x-coordinate of the current cursor position (within the current text window).
- Return value** **wherex** returns an integer in the range 1 to 80.
- Portability** **wherex** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.
- See also** `gettextinfo`, `gotoxy`, `wherey`
- Example**
- ```
#include <conio.h>

int main(void)
```

```

{
    clrscr();
    gotoxy(10,10);
    printf("Current location is X: %d Y: %d\r\n", wherex(), wherey());
    getch();
    return 0;
}

```

wherey

Function Gives vertical cursor position within window.

Syntax `#include <conio.h>`
`int wherey(void);`

Prototype in `conio.h`

Remarks **wherey** returns the y-coordinate of the current cursor position (within the current text window).

Return value **wherey** returns an integer in the range 1 to 25, 43, or 50.

Portability **wherey** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **gettextinfo**, **gotoxy**, **wherex**

Example

```

#include <conio.h>

int main(void)
{
    clrscr();
    gotoxy(10,10);
    printf("Current location is X: %d Y: %d\r\n", wherex(), wherey());
    getch();
    return 0;
}

```

window

Function Defines active text mode window.

Syntax `#include <conio.h>`
`void window(int left, int top, int right, int bottom);`

Prototype in `conio.h`

window

Remarks **window** defines a text window onscreen. If the coordinates are in any way invalid, the call to **window** is ignored.

left and *top* are the screen coordinates of the upper left corner of the window.

right and *bottom* are the screen coordinates of the lower right corner.

The minimum size of the text window is one column by one line. The default window is full screen, with these coordinates:

80-column mode: 1,1,80,25

40-column mode: 1,1,40,25

Return value None.

Portability **window** works with IBM PCs and compatibles only. A corresponding function exists in Turbo Pascal. This function is not Windows compatible.

See also **clreol**, **clrscr**, **delline**, **gettextinfo**, **gotoxy**, **insline**, **puttext**, **textmode**

Example

```
#include <conio.h>

int main(void)
{
    window(10,10,40,11);
    textcolor(BLACK);
    textbackground(WHITE);
    printf("This is a test\r\n");
    return 0;
}
```

_write

Function Writes to a file.

Syntax `#include <io.h>`
`int _write(int handle, void *buf, unsigned len);`

Prototype in io.h

Remarks **_write** attempts to write *len* bytes from the buffer pointed to by *buf* to the file associated with *handle*.

The maximum number of bytes that **_write** can write is 65,534, since 65,535 (0xFFFF) is the same as -1, which is the error return indicator for **_write**.

_write does not translate a linefeed character (LF) to a CR/LF pair because all its files are binary files.

If the number of bytes actually written is less than that requested, the condition should be considered an error and probably indicates a full disk.

For disk files, writing, always proceeds from the current file pointer. On devices, bytes are directly sent to the device.

For files opened with the **O_APPEND** option, the file pointer is not positioned to EOF by **_write** before writing the data.

Return value **_write** returns the number of bytes written. In case of error, **_write** returns -1 and sets the global variable *errno* to one of the following:

- EACCES Permission denied
- EBADF Bad file number

Portability **_write** is unique to DOS.

See also **lseek, _read, write**

Example

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
    void *buf;
    int handle, bytes;
    buf = malloc(200);

    /* Create a file TEST.$$$ in the current directory and write 200 bytes to it.
       If TEST.$$$ already exists, overwrite. */
    if ((handle = open("TEST. $$$", O_CREAT | O_WRONLY | O_BINARY,
                     S_IWRITE | S_IREAD)) == -1)
    {
        printf("Error Opening File\n");
        exit(1);
    }
    if ((bytes = _write(handle, buf, 200)) == -1) {
        printf("Write Failed.\n");
        exit(1);
    }
    printf("_write: %d bytes written.\n", bytes);
    return 0;
}
```



write

Function	Writes to a file.				
Syntax	<pre>#include <io.h> int write(int <i>handle</i>, void *<i>buf</i>, unsigned <i>len</i>);</pre>				
Prototype in	io.h				
Remarks	<p>write writes a buffer of data to the file or device named by the given <i>handle</i>. <i>handle</i> is a file handle obtained from a creat, open, dup, or dup2 call.</p> <p>This function attempts to write <i>len</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with <i>handle</i>. Except when write is used to write to a text file, the number of bytes written to the file will be no more than the number requested.</p> <p>The maximum number of bytes that write can write is 65,534, since 65,535 (0xFFFF) is the same as -1, which is the error return indicator for write.</p> <p>On text files, when write sees a linefeed (LF) character, it outputs a CR/LF pair.</p> <p>If the number of bytes actually written is less than that requested, the condition should be considered an error and probably indicates a full disk.</p> <p>For disks or disk files, writing, always proceeds from the current file pointer. For devices, bytes are sent directly to the device.</p> <p>For files opened with the O_APPEND option, the file pointer is positioned to EOF by write before writing the data.</p>				
Return value	<p>write returns the number of bytes written. A write to a text file does not count generated carriage returns. In case of error, write returns -1 and sets the global variable <i>errno</i> to one of the following:</p> <table> <tr> <td>EACCES</td> <td>Permission denied</td> </tr> <tr> <td>EBADF</td> <td>Bad file number</td> </tr> </table>	EACCES	Permission denied	EBADF	Bad file number
EACCES	Permission denied				
EBADF	Bad file number				
Portability	write is available on UNIX systems.				
See also	creat , lseek , open , read , _write				

Global variables

Borland C++ provides you with predefined global variables for many common needs, such as dates, times, command-line arguments, and so on. This chapter defines and describes them.

_8087

Function	Coprocessor chip flag.
Syntax	<code>extern int _8087;</code>
Declared in	<code>dos.h</code>
Remarks	<p>The <code>_8087</code> variable is set to a nonzero value (1, 2, or 3) if the startup code autodetection logic detects a floating-point coprocessor (an 8087, 80287, or 80387, respectively). The <code>_8087</code> variable is set to 0 otherwise.</p> <p>The autodetection logic can be overridden by setting the 87 environment variable to YES or NO. (The commands are <code>SET 87=YES</code> and <code>SET 87=NO</code>; it is essential that there be no spaces before or after the equal sign.) In this case, the <code>_8087</code> variable will reflect the override.</p> <p>Refer to Chapter 6, "Memory management," in the <i>Programmer's Guide</i> for more information about the 87 environment variable.</p>

`_argc`

`_argc`

Function	Keeps a count of command-line arguments.
Syntax	<code>extern int <i>_argc</i>;</code>
Declared in	<code>dos.h</code>
Remarks	<code>_argc</code> has the value of <code>argc</code> passed to main when the program starts.

`_argv`

Function	An array of pointers to command-line arguments.
Syntax	<code>extern char *<i>_argv</i>[];</code>
Declared in	<code>dos.h</code>
Remarks	<code>_argv</code> points to an array containing the original command-line arguments (the elements of <code>argv[]</code>) passed to main when the program starts.

`_ctype`

Function	An array of character attribute information.
Syntax	<code>extern char <i>_ctype</i>[]</code>
Declared in	<code>ctype.h</code>
Remarks	<code>_ctype</code> is an array of character attribute information indexed by ASCII value + 1. Each entry is a set of bits describing the character. This array is used by isdigit , isprint , and so on.

`daylight`

Function	Indicates whether daylight saving time adjustments will be made.
Syntax	<code>extern int <i>daylight</i>;</code>

Declared in time.h

Remarks *daylight* is used by the time and date functions. It is set by the **tzset**, **ftime**, and **localtime** functions to 1 for daylight saving time, 0 for standard time.

directvideo

Function Flag that controls video output.

Syntax extern int *directvideo*;

Declared in conio.h

Remarks *directvideo* controls whether your program's console output (from **cputs**, for example) goes directly to the video RAM (*directvideo* = 1) or goes via ROM BIOS calls (*directvideo* = 0).

The default value is *directvideo* = 1 (console output goes directly to video RAM). In order to use *directvideo* = 1, your system's video hardware must be identical to IBM display adapters. Setting *directvideo* = 0 allows your console output to work on any system that is IBM BIOS-compatible.

environ

Function Accesses DOS environment variables.

Syntax extern char * *environ*[];

Declared in dos.h

Remarks *environ* is an array of pointers to strings; it is used to access and alter the DOS environment variables. Each string is of the form

envvar = *varvalue*

where *envvar* is the name of an environment variable (such as **PATH**), and *varvalue* is the string value to which *envvar* is set (such as **C:\BIN;C:\DOS**). The string *varvalue* may be empty.

When a program begins execution, the DOS environment settings are passed directly to the program. Note that *env*, the third argument to **main**, is equal to the initial setting of *environ*.

The *environ* array can be accessed by **getenv**; however, the **putenv** function is the only routine that should be used to add, change or delete the *environ* array entries. This is because modification can resize and relocate the

process environment array, but *environ* is automatically adjusted so that it always points to the array.

See also `getenv`, `putenv`

`errno`, `_doserrno`, `sys_errlist`, `sys_nerr`

Function Enable **perror** to print error messages.

Syntax `extern int errno;`
`extern int _doserrno;`
`extern char * sys_errlist[];`
`extern int sys_nerr;`

Declared in `errno.h`, `stdlib.h` (`errno`, `_doserrno`, `sys_errlist`, `sys_nerr`)
`dos.h` (`_doserrno`)

Remarks `errno`, `sys_errlist`, and `sys_nerr` are used by **perror** to print error messages when certain library routines fail to accomplish their appointed tasks. `_doserrno` is a variable that maps many DOS error codes to `errno`; however, **perror** does not use `_doserrno` directly.

_doserrno: When a DOS system call results in an error, `_doserrno` is set to the actual DOS error code. `errno` is a parallel error variable inherited from UNIX.

errno: When an error in a math or system call occurs, `errno` is set to indicate the type of error. Sometimes `errno` and `_doserrno` are equivalent. At other times, `errno` does not contain the actual DOS error code, which is contained in `_doserrno`. Still other errors might occur that set only `errno`, not `_doserrno`.

sys_errlist: To provide more control over message formatting, the array of message strings is provided in `sys_errlist`. You can use `errno` as an index into the array to find the string corresponding to the error number. The string does not include any newline character.

sys_nerr: This variable is defined as the number of error message strings in `sys_errlist`.

The following table gives mnemonics and their meanings for the values stored in `sys_errlist`.

Mnemonic	Meaning
E2BIG	Arg list too long
EACCES	Permission denied
EBADF	Bad file number
ECONTR	Memory blocks destroyed
ECURDIR	Attempt to remove CurDir
EDOM	Domain error
EEXIST	File already exists
EFAULT	Unknown error
EINVACC	Invalid access code
EINVAL	Invalid argument
EINVDAT	Invalid data
EINVDRV	Invalid drive specified
EINVENV	Invalid environment
EINVMT	Invalid format
EINVFC	Invalid function number
EINVMEM	Invalid memory block address
EMFILE	Too many open files
ENMFILE	No more files
ENODEV	No such device
ENOENT	No such file or directory
ENOEXEC	Exec format error
ENOFILE	No such file or directory
ENOMEM	Not enough memory
ENOPATH	Path not found
ENOTSAM	Not same device
ERANGE	Result out of range
EXDEV	Cross-device link
EZERO	Error 0

The following list gives mnemonics for the actual DOS error codes to which *_doserrno* can be set. (This value of *_doserrno* may or may not be mapped (through *errno*) to an equivalent error message string in *sys_errlist*.)

Mnemonic	DOS error code
E2BIG	Bad environ
EACCES	Access denied
EACCES	Bad access
EACCES	Is current dir
EBADF	Bad handle
EFAULT	Reserved
EINVAL	Bad data
EINVAL	Bad function
EMFILE	Too many open
ENOENT	No such file or directory
ENOEXEC	Bad format

ENOMEM	Mcb destroyed
ENOMEM	Out of memory
ENOMEM	Bad block
EXDEV	Bad drive
EXDEV	Not same device

Refer to your DOS reference manual for more information about DOS error return codes.

Example

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

extern char *sys_errlist[];

main()
{
    int i = 0;

    while(sys_errlist[i++]) printf("%s\n", sys_errlist[i]);
    return 0;
}
```

_fmode

- Function** Determines default file-translation mode.
- Syntax** `extern int _fmode;`
- Declared in** `fcntl.h`
- Remarks** *_fmode* determines in which mode (text or binary) files will be opened and translated. The value of *_fmode* is `O_TEXT` by default, which specifies that files will be read in text mode. If *_fmode* is set to `O_BINARY`, the files are opened and read in binary mode. (`O_TEXT` and `O_BINARY` are defined in `fcntl.h`.)

In text mode, on input carriage-return/linefeed (CR/LF) combinations are translated to a single linefeed character (LF). On output, the reverse is true: LF characters are translated to CR/LF combinations.

In binary mode, no such translation occurs.

You can override the default mode as set by *_fmode* by specifying a *t* (for text mode) or *b* (for binary mode) in the argument *type* in the library routines **fopen**, **fdopen**, and **freopen**. Also, in the routine **open**, the argument *access* can include either `O_BINARY` or `O_TEXT`, which will explicitly define the file being opened (given by the **open** *pathname* argument) to be in either binary or text mode.

`_heaplen`

- Function** Holds the length of the near heap.
- Syntax** `extern unsigned _heaplen;`
- Declared in** `dos.h`
- Remarks** `_heaplen` specifies the size (in bytes) of the near heap in the small data models (tiny, small, and medium). `_heaplen` does not exist in the large data models (compact, large, and huge), as they do not have a near heap.
- In the small and medium models, the data segment size is computed as follows:
- $$\text{data segment [small,medium]} = \text{global data} + \text{heap} + \text{stack}$$
- where the size of the stack can be adjusted with `_stklen`.
- If `_heaplen` is set to 0, the program allocates 64K bytes for the data segment, and the effective heap size is
- $$64\text{K} - (\text{global data} + \text{stack}) \text{ bytes}$$
- By default, `_heaplen` equals 0, so you'll get a 64K data segment unless you specify a particular `_heaplen` value.
- In the tiny model, everything (including code) is in the same segment, so the data segment computations are adjusted to include the code plus 256 bytes for the program segment prefix (PSP).
- $$\text{data segment[tiny]} = 256 + \text{code} + \text{global data} + \text{heap} + \text{stack}$$
- If `_heaplen` equals 0 in the tiny model, the effective heap size is obtained by subtracting the PSP, code, global data, and stack from 64K.
- In the compact and large models, there is no near heap, and the stack is in its own segment, so the data segment is simply
- $$\text{data segment [compact,large]} = \text{global data}$$
- In the huge model, the stack is a separate segment, and each module has its own data segment.
- See also** `_stklen`

`_new_handler`

Function Traps new allocation miscues.

Syntax `typedef void (*pvf)();
pvf _new_handler;`

Or, as an alternative, you can set using the function `_set_new_handler`, like this:

`pvf set_new_handler(pvf p);`

Remarks `_new_handler` contains a pointer to a function that takes no arguments and returns **void**. If **operator new()** is unable to allocate the space required, it will call the function pointed to by `_new_handler`; if that function returns it will try the allocation again. By default, the function pointed to by `_new_handler` simply terminates the application. The application can replace this handler, however, with a function that can try to free up some space. This is done by assigning directly to `_new_handler` or by calling the function `set_new_handler`, which returns a pointer to the former handler. `_new_handler` is provided primarily for compatibility with C++ version 1.2. In most cases this functionality can be better provided by overloading **operator new()**.

`_openfd`

Function Array of access modes.

Syntax `extern unsigned int _openfd[]`

Declared in `io.h`

Remarks `_openfd` is an array of access modes for files and devices.

`_osmajor, _osminor`

Function Contain the major and minor DOS version numbers.

Syntax extern unsigned char `_osmajor`;
extern unsigned char `_osminor`;

Declared in dos.h

Remarks The major and minor version numbers are available individually through `_osmajor` and `_osminor`. `_osmajor` is the major version number, and `_osminor` is the minor version number. For example, if you are running DOS version 3.2, `_osmajor` will be 3, and `_osminor` will be 20.

These variables can be useful when you want to write modules that will run on DOS versions 2.x and 3.x. Some library routines behave differently depending on the DOS version number, while others only work under DOS 3.x. (For example, refer to `_open`, `creatnew`, and `ioctl` in the lookup section of this *Reference Guide*.)

`_ovrbuffer`

Function Change the size of the overlay buffer.

Syntax unsigned `_ovrbuffer` = *size*;

Declared in dos.h

Remarks The default overlay buffer size is twice the size of the largest overlay. This is adequate for some applications. But imagine that a particular function of a program is implemented through many modules, each of which is overlaid. If the total size of those modules is larger than the overlay buffer, a substantial amount of swapping will occur if the modules make frequent calls to each other.

The solution is to increase the size of the overlay buffer so that enough memory is available at any given time to contain all overlays that make frequent calls to each other. You can do this by setting the `_ovrbuffer` global variable to the required size in paragraphs. For example, to set the overlay buffer to 128K, include the following statement in your code:

```
unsigned _ovrbuffer = 0x2000;
```

`_ovrbuffer`

There is no general formula for determining the ideal overlay buffer size. Borland's Turbo Profiler can help provide a suitable value.

See also `_OvrInitEms`, `_OvrInitExt`

`_psp`

- Function** Contains the segment address of the program segment prefix (PSP) for the current program.
- Syntax** `extern unsigned int _psp;`
- Declared in** `dos.h`
- Remarks** The PSP is a DOS process descriptor; it contains initial DOS information about the program.
- Refer to the *DOS Programmer's Reference Manual* for more information on the PSP.

`_stklen`

- Function** Holds size of the stack.
- Syntax** `extern unsigned _stklen;`
- Declared in** `dos.h`
- Remarks** `_stklen` specifies the size of the stack for all six memory models. The minimum stack size allowed is 128 words; if you give a smaller value, `_stklen` is automatically adjusted to the minimum. The default stack size is 4K.
- In the small and medium models, the data segment size is computed as follows:
- $$\text{data segment [small,medium]} = \text{global data} + \text{heap} + \text{stack}$$
- where the size of the heap can be adjusted with `_heaplen`.
- In the tiny model, everything (including code) is in the same segment, so the data segment computations are adjusted to include the code plus 256 bytes for the program segment prefix (PSP).
- $$\text{data segment[tiny]} = 256 + \text{code} + \text{global data} + \text{heap} + \text{stack}$$

In the compact and large models, there is no near heap, and the stack is in its own segment, so the data segment is simply

```
data segment [compact,large] = global data
```

In the huge model, the stack is a separate segment, and each module has its own data segment.

See also `_heaplen`

Example

```
#include <stdio.h>

/* Set the stack size to be greater than the default. */
/* This declaration must go in the global data area. */
extern unsigned _stklen = 54321U;

main()
{
    /* show the current stack length */
    printf("The stack length is: %u\n", _stklen);
    return 0;
}
```

timezone

Function Contains difference in seconds between local time and GMT.

Syntax `extern long timezone;`

Declared in `time.h`

Remarks *timezone* is used by the time-and-date functions.

This variable is calculated by the **tzset** function; it is assigned a long value that is the difference, in seconds, between the current local time and Greenwich mean time.

tzname

Function Array of pointers to time zone names.

Syntax `extern char * tzname[2]`

tzname

Declared in time.h

Remarks The global variable *tzname* is an array of pointers to strings containing abbreviations for time zone names. *tzname[0]* points to a three-character string with the value of the time zone name from the *TZ* environment string. The global variable *tzname[1]* points to a three-character string with the value of the daylight saving time zone name from the *TZ* environment string. If no daylight saving name is present, *tzname[1]* points to a null string.

_version

Function Contains the DOS version number.

Syntax extern unsigned int *_version*;

Declared in dos.h

Remarks *_version* contains the DOS version number, with the major version number in the low byte and the minor version number in the high byte. (For DOS version *x.y*, the *x* is the major version number, and *y* is the minor.)

_wscroll

Function Enables or disables scrolling in console I/O functions.

Syntax extern int *_wscroll*

Declared in conio.h

Remarks *_wscroll* is a console I/O flag. Its default value is 1. If you set *_wscroll* to 0, scrolling is disabled. This can be useful for drawing along the edges of a window without having your screen scroll.

_8087 (global variable) 555
 8086 processor 105
 interrupt vectors 231, 458
 interrupts 276, 277, 280
 80x87 coprocessors *See* numeric coprocessors
 0x11 BIOS interrupt 40
 0x12 BIOS interrupt 43
 0x13 BIOS interrupt 37
 0x16 BIOS interrupt 42
 0x17 BIOS interrupt 44
 0x19 DOS function 194
 0x31 DOS function 294
 0x21 DOS interrupt 278, 279
 0x23 DOS interrupt 86, 251
 0x24 DOS interrupt 249
 0x25 DOS interrupt 12
 0x26 DOS interrupt 14
 0x27 DOS system call 379
 0x28 DOS system call 381
 0x29 DOS system call 347
 0x33 DOS system call 185, 424
 0x44 DOS system call 282
 0x48 DOS system call 17
 0x59 DOS system call 95
 0x62 DOS system call 224
 0x1A BIOS interrupt 45
 0x4E DOS system call 143

A

abnormal program termination 378
 abort (function) 11
 abs (function) 11
 absolute disk sectors 12, 14
 absolute value *See also* values
 complex numbers 48
 square 335
 floating-point numbers 116
 integers 11

 long 295
 absread (function) 12
 abswrite (function) 14
 access
 DOS system calls 33, 34
 files *See* files, access
 flags 339, 472
 memory (DMA) 39, 40
 modes
 arrays of 562
 changing 54, 55
 program
 signal types 377
 invalid 378
 read/write 56, 171
 files 15, 80, 339, 472, 482
 permission 340
 access (function) 14
 acos (function) 16
 active page 462
 setting 414
 adapters
 graphics 88
 monochrome 18, 351
 address segment, of far pointer 159, 327
 addresses
 memory *See* memory, addresses
 passed to `__emit__` 104
 allocating memory *See* memory allocation
 allocmem (function) 17
 alphabetic ASCII codes, checking for 284
 alphanumeric ASCII codes, checking for 283
 angles (complex numbers) 19
 arc (function) 18
 coordinates 180
 arc cosine 16
 arc sine 21
 arc tangent 23, 24
 arcs, elliptical 102

- arg (function) 19
- _argc (global variable) 556
- argc (argument to main) 3
- ARGS.EXE 4
- argument list, variable 541
 - conversion specifications and 358
- arguments
 - command line, passing to main 3, 556
 - wildcards and 5
- _argv (global variable) 556
- argv (argument to main) 3
- arrays
 - of access modes 562
 - of character, attribute information 556
 - searching 47, 298
 - of time zone names 565
- ASCII codes
 - alphabetic 284
 - lowercase 288
 - uppercase 291
 - alphanumeric 283
 - control or delete 286
 - converting
 - characters to 530
 - date and time to 20
 - digits
 - 0 to 9 287
 - hexadecimal 292
 - integer value classification *See* macros, character classification
 - low 285
 - lowercase alphabetic 288
 - printing characters 288, 289
 - punctuation characters 290
 - uppercase alphabetic 291
 - whitespace 290
- asctime (function) 20
- asin (function) 21
- aspect ratio
 - correcting 418
 - getting 182
- assert (function) 22
- assertion 22
- assignment suppression
 - format specifiers 402, 403, 407, 408
- AT&T 6300 PC
 - detecting presence of 89
- atan2 (function) 24
- atan (function) 23
- atexit (function) 25
- atof (function) 26
- atoi (function) 27
- atol (function) 28
- attribute bits 339, 472
- attribute word 54, 78, 83
- attributes
 - characters, arrays of 556
 - files *See* files, attributes
 - text 517, 520, 521
- autodetection (graphics drivers) 88, 195, 266, 272
- AX register
 - hardware error handlers and 249

B

- background color *See* colors and palettes
- banker's rounding 32
- bar (function) 29
- bar3d (function) 30
- bars
 - three-dimensional 30
 - two-dimensional 29
- base 10 logarithm 307
- baud rate 35
- BCD (binary coded decimal) numbers 32, 387
- bcd (function) 32
- bdos (function) 33
- bdosptr (function) 34
- beeps 337, 474
- BGI *See* Borland Graphics Interface (BGI)
- BGIOBJ (graphics converter) 266
 - stroked fonts and 452
- binary coded decimal floating-point numbers
 - See* BCD (binary coded decimal) numbers
- binary files *See also* files
 - creat and 79
 - creattemp and 82
 - fdopen and 131
 - fopen and 155
 - freopen and 165
 - opening 131, 155, 165
 - and translating 560
 - setting 445

- temporary
 - naming 529
 - opening 528
- binary mode *See* binary files
- binary search 47
- BIOS
 - interrupts *See also* interrupts
 - 0x11 40
 - 0x12 43
 - 0x13 37
 - 0x16 42
 - 0x17 44
 - 0x1A 45
 - timer 45
- bioscom (function) 35
- biosdisk (function) 37
- biosequip (function) 40
- bioskey (function) 41
- biosmemory (function) 43
- biosprint (function) 44
- biostime (function) 45
- bit images
 - saving 206
 - storage requirements 264
 - writing to screen 370
- bit-mapped fonts 453
- bit mask 171
 - file mode 482
- bit rotation
 - long integer 310, 311
 - unsigned integer 399
- bits
 - attribute 78, 82, 339, 472
 - status (communications) 35
 - stop (communications) 35
- blink-enable bit 518
- block operations *See* editing, block operations
- blocks, memory *See* memory blocks
- Borland Graphics Interface (BGI)
 - device driver table 272
 - fonts 392
 - new 274
 - graphics drivers and 247, 266, 390
- BP register
 - hardware error handlers and 249
- break value 46, 400
- brk (function) 46

- bsearch (function) 47
- buffers
 - file 457
 - graphics, internal 436
 - keyboard, pushing character to 537
 - overlays
 - default size 563
 - streams and 422, 423, 457
 - clearing 149
 - flushing 128
 - writing 149
 - system-allocated, freeing 128
- bytes
 - copying 330
 - reading from hardware ports 270
 - returning from memory 349
 - status (disk drives) 38
 - storing in memory 353
 - swapping 513

C

- C++
 - binary coded decimal 32, 387
 - complex numbers *See* complex numbers
 - memory allocation 562
- cabs (function) 48
- calendar format (time) 328
- calloc (function) 49
- carry flag 276, 277, 278, 279
- ceil (function) 51
- CGA *See* Color Graphics Adapter (CGA)
- cgets (function) 51
- channels (device) 282
- characters
 - alphabetic 284
 - alphanumeric 283
 - array, global variable 556
 - attributes 517, 520, 521
 - blinking 518
 - classifying *See* macros, character classification
 - color, setting 517, 521
 - control or delete 286
 - converting to ASCII 530
 - device 286
 - digits (0 to 9) 287
 - displaying 360, 367, 404

- floating-point numbers and 26
- format specifiers *See* format specifiers, characters
- hexadecimal digits 292
- intensity
 - high 262
 - low 309
 - normal 336
- low ASCII 285
- lowercase 531
 - checking for 288
 - converting to 530, 531
- magnification, user-defined 455
- newline 374
- printing 288, 289
- punctuation 290
- pushing
 - to input stream 536
 - to keyboard buffer 537
- reading 404
 - from console 51
 - from keyboard 186, 187
 - from streams 135, 185, 187
 - stdin 136
- scanning in strings 491, 503, 504
 - segment subset 506
- searching
 - in block 320
 - in string 487
- size 453, 523, 526
- uppercase
 - checking for 291
 - converting to 532, 533
- whitespace 290
- writing
 - to screen 367
 - to streams 160, 366, 368
- chdir (function) 52
- child process 108, 475, *See also* processes
- _chmod (function) 54
- chmod (function) 55
- .CHR files 274, 452
- chsize (function) 57
- circle (function) 58
- circles, drawing 58
- _clear87 (function) 59
- cleardevice (function) 60
- clearerr (function) 62
- clearing
 - screens 60, 69, 437
 - to end of line 68
- clearviewport (function) 63
- clock (function) 64
- _close (function) 65
- close (function) 66
- closegraph (function) 67
- closing files *See* files, closing
- creol (function) 68
- clrscr (function) 69
- Color/Graphics Adapter (CGA) *See also* graphics; screens
 - colors *See* colors and palettes
 - detecting presence of 89
 - palettes 416, *See also* colors and palettes
 - problems 18, 351
- colors and palettes 241, 267, 415, *See also* graphics
 - background color 183, 419
 - setting 241, 419
 - text 517, 520
- CGA 416
 - changing 415, 446
 - color table 416, 419, 446
 - default 192
 - definition structure 192
 - drawing 188, 351, 425
 - setting 241
- EGA 416
- fill colors 141, 147
 - information on 201
 - pie slices 351, 412
 - setting 432
- fill patterns 141, 142, 147
 - defining 200, 202
 - by user 431, 432
 - information on 201
 - pie slices 351, 412
 - predefined 202
 - setting to default 241
- fill style 241
- filling graphics 147
- IBM 8514 448
- information on 218
 - returning 192

- maximum value 210
- pixels 223, 372
- problems with 18, 351
- rectangle 389
- setting 425, 448
 - background 419
 - character 517, 521
 - drawing 241
- size of 220
- VGA 416, 448
- COMMAND.COM 514
- command line
 - arguments, passing to main 556
 - errors *See* errors
- command-line compiler
 - Pascal calling conventions, option (-p) 7
- communications
 - parity 35
 - ports 35
 - checking for 40, 286
 - protocol settings 35
 - RS-232 35
 - stop bits 35
- compact memory model *See* memory models
- comparing
 - strings *See* strings, comparing
 - two values 319, 325
- comparison function, user-defined 376
- compiler, command-line *See* command-line compiler
- complex (function) 69
- complex numbers *See also* numbers;
 - trigonometric functions
 - absolute value 48
 - square of 335
 - angles 19
 - conjugate of 71
 - constructor for 69
 - imaginary portion 263
 - polar function 354
 - real portion 387
- COMSPEC environment variable 514
- concatenated strings 486, 498
- conditions, testing 22
- conj (function) 71
- conjugate (complex numbers) 71
- console *See also* hardware
 - checking for 286
 - output flag 557
 - reading and formatting
 - characters 51
 - input 84
- constructor (complex numbers) 69
- _control87 (function) 71
- control-break
 - handler 86
 - interrupt 251
 - returning 185
 - setting 424
 - software signal 378
- control characters, checking for 286
- control word, floating point 71
- conversion
 - binary coded decimal 32, 387
 - date and time 20
 - DOS to UNIX format 96
 - to string 85
 - to calendar format 328
 - to Greenwich mean time 238
 - to structure 303
 - UNIX to DOS format 538
- double
 - strings to 507
 - to integer and fraction 329
 - to mantissa and exponent 166
- floating point
 - strings to 26
 - to string 101, 130, 178
- format specifiers 358, 359, 360, 363
- integer
 - to ASCII 530
 - strings to 27
 - to string 292
- long integer
 - strings to 28, 509, 511
 - to string 314, 535
- lowercase to uppercase 511, 532, 533
- specifications (printf) 358
- strings
 - date and time to 85
 - to double 507
 - to floating point 26
 - integers to 292

- to integer 27
- to long integer 28, 509, 511
- to unsigned long integer 511
- unsigned long integer
 - strings to 511
 - to string 535
- uppercase to lowercase 497, 530, 531
- coordinates
 - arc, returning 180
 - current position 236, 237, 461
 - cursor position 240, 550, 551
 - screens
 - maximum 213, 214
 - text mode 226
 - x-coordinate 213, 236
 - y-coordinate 214, 237
- coprocessors *See* numeric coprocessors
- coreleft (function) 72
- coroutines
 - overlays and 308, 439
 - task states and 308, 439
- correction factor of aspect ratio 418
- cos (function) 73
- cosh (function) 74
- cosine 73
 - hyperbolic 74
 - inverse 16
- country (function) 75
- country-dependent data 75, 302, 443
- CP *See* current position (graphics)
- cprintf (function) 76
 - format specifiers 357
- cputs (function) 77
- _creat (function) 78
- creat (function) 79
- creatnew (function) 81
- createmp (function) 82
- cscanf (function) 84
 - format specifiers 401
- ctime (function) 85
- ctrlbrk (function) 86
- _ctype (global variable) 556
- currency symbols 75, 302, 443
- current position (graphics) 241
 - coordinates 236, 237, 461
 - justified text and 450
 - lines and 299, 300, 301

- moving 331, 333
- cursor
 - appearance, selecting 427
 - position in text window 240
 - returning 550, 551

D

- data
 - conversion *See* conversion
 - country-dependent, supporting 75, 302, 443
 - moving 330
 - reading from streams 162, 167, 544, 549
 - stdin 401, 547
 - returning from current environment 197
 - security 221
 - writing to current environment 369
- data bits (communications) 35
- data segment 50, 315, 561
 - allocation 46
 - changing 400
- date and time conversions *See* conversion, date and time
- dates *See also* time
 - file 204
 - setting 434
 - global variable 556
 - international formats 75
 - system 20, 85, 176, 238, 303
 - converting from DOS to UNIX 96
 - converting from UNIX to DOS 538
 - getting 191
 - setting 428, 484
- daylight (global variable) 556
 - setting value of 533
- daylight saving time *See also* time zones
 - adjustments 85, 556
 - setting 534
- deallocating memory *See* memory, freeing
- delay (function) 87
- deletion
 - characters, checking for 286
 - directories 398
 - file 393, 539
 - line 68, 87
- delline (function) 87
- detectgraph (function) 88

- detection
 - graphics adapter *88, 195*
 - graphics drivers *266*
- device *See also hardware*
 - channels *282*
 - character *286*
 - drivers *See also graphics drivers*
 - BGI *272*
 - DOS *282*
 - vendor-added *272*
 - errors *249*
 - type checking *285*
- DI register
 - hardware error handlers and *249*
- difftime (function) *92*
- direct memory access (DMA)
 - checking for presence of *39, 40*
- directories
 - creating *326*
 - current *108, 476*
 - changing *52*
 - returning *190*
 - deleting *398*
 - path *See paths*
 - searching *143, 145, 410*
- directvideo (global variable) *557*
- disable (function) *92*
- disk access errors *See errors*
- disk drives *See also hardware*
 - checking for presence of *40*
 - current number *194*
 - functions *37*
 - I/O operations *37*
 - setting *429*
 - status byte *38*
- disk operating system *See DOS*
- disk sectors
 - reading *12, 37*
 - writing *14, 37*
- disk transfer address (DTA)
 - DOS *379*
 - returning *196*
 - setting *429*
 - random blocks and *379*
- disks
 - errors *249, See also errors*
 - operations *37, 38*
 - space available *193*
 - writing to, verification *232, 460*
- div (function) *94*
- division, integers *94, 297*
- DMA *See direct memory access (DMA)*
- DOS
 - commands *514*
 - date and time *191*
 - converting to UNIX format *96*
 - converting UNIX to *538*
 - setting *428, 454, 484*
 - device drivers *282*
 - disk transfer address *See disk transfer address (DTA)*
 - environment
 - adding data to *369*
 - returning data from *197*
 - variables *108, 476*
 - accessing *557*
 - error codes *558, 559*
 - error information, extended *95*
 - file attributes *54, 78, 82*
 - search *144*
 - shared *338*
 - file-sharing mechanisms *See files, sharing*
 - functions
 - 0x19 *194*
 - 0x31 *294*
 - interrupts
 - 0x21 *278, 279*
 - 0x23 *86, 251*
 - 0x24 *249*
 - 0x25 *12*
 - 0x26 *14*
 - functions *231, 458*
 - handlers *249*
 - interface *278, 279*
 - path, searching for file in *410*
 - search algorithm *108*
 - system calls *249, 384*
 - 0x27 *379*
 - 0x28 *381*
 - 0x29 *347*
 - 0x33 *185, 424*
 - 0x44 *282*
 - 0x48 *17*
 - 0x59 *95*

- 0x62 224
- 0x4E 143
- accessing 33, 34
- memory models and 33, 34
- verify flag 232, 460
- version numbers 563, 566
- _doserrno (global variable) 558
- dosexterr (function) 95
- dostounix (function) 96
- double (data type) *See* floating point, double
- drawing color *See* colors and palettes
- drawpoly (function) 97
- drivers, graphics *See* graphics drivers
- drives *See* disk drives
- DTA *See* disk transfer address (DTA)
- dup2 (function) 100
- dup (function) 98
- dynamic memory allocation 49, 163, 315, 388

E

- echoing to screen 186, 187
- ecvt (function) 101
- editing
 - block operations
 - copying 319, 322, 324, 334
 - searching for character 320
- EGA *See* Enhanced Graphics Adapter (EGA)
- elapsed time *See* time
- ellipse (function) 102
- ellipses, drawing and filling 140
- elliptical arcs 102
- elliptical pie slices 411
- __emit__ (function) 104
- EMS *See* memory, expanded and extended
- enable (function) 105
- encryption 221
- end of file
 - checking 106, 132, 385
 - resetting 62
- end of line, clearing to 68
- Enhanced Graphics Adapter (EGA) *See also*
 - graphics; screens
 - colors *See* colors and palettes
 - detecting presence of 89
 - palette *See* graphics, palettes
- env (argument to main) 3

- environ (global variable) 4, 557
- environment *See* Programmer's Platform
 - variables 557
 - COMSPEC 514
 - PATH 108, 476
- environment, DOS *See* DOS
- eof (function) 106
- equations, polynomial 355
- errno (global variable) 558
- error handlers
 - hardware 249, 251, 254
 - math, user-modifiable 316
- errors
 - codes *See* errors, messages
 - detection, on stream 132, 133
 - DOS
 - extended information 95
 - mnemonics 558, 559
 - indicators, resetting 62
 - locked file 305
 - messages
 - graphics, returning 242
 - graphics drivers 247
 - pointer to, returning 242, 493, 494
 - printing 350, 558
 - read/write 133
 - European date formats 75
 - even parity (communications) 35
 - exception handlers, numeric coprocessors 59, 484
 - exceptions, floating point 71
 - exec... (functions) *See* processes
 - execl (function) 107
 - execlp (function) 107
 - execlpe (function) 107
 - execution, suspending 87, 471, *See also*
 - programs, stopping
 - execv (function) 107
 - execve (function) 107
 - execvp (function) 107
 - execvpe (function) 107
 - exit codes 11, *See also* programs, stopping
 - _exit (function) 113
 - exit (functions) 25, 114
 - exit status 113, 114, 294
 - exp (function) 115

expanded and extended memory *See* memory,
expanded and extended
exponents
 calculating 115, 356, 357
 double 166, 296
extended error information, DOS 95

F

fabs (function) 116
far heap *See also* heap
 allocating memory from 116, 125
 checking 119, 120
 nodes 122
 free blocks 120, 123
 memory in
 freeing 118
 measure of unused 118
 reallocating 127
 pointers 117, 126, 127
 walking through 124
far pointers *See* pointers, far
farcalloc (function) 116
farcoreleft (function) 118
 tiny memory model and 118
farfree (function) 118
 small and medium memory models and 118
farheapcheck (function) 119
farheapcheckfree (function) 120
farheapchecknode (function) 122
farheapfillfree (function) 123
farheapwalk (function) 124
farmalloc (function) 125
 tiny memory model and 126
farrealloc (function) 127
 tiny memory model and 127
FAT *See* file allocation table (FAT)
fatal errors *See* errors
FCB *See* file control block (FCB)
fclose (function) 128
fcloseall (function) 129
fcvt (function) 130
fdopen (function) 131
feof (function) 132
ferror (function) 133
fflush (function) 134
fgetc (function) 135
fgetchar (function) 136

fgetpos (function) 137
fgets (function) 138
fields
 input 404, 408
 random record 379, 381
figures, flood-filling *See* colors and palettes,
filling
file allocation table (FAT) 198, 199
file control block (FCB) 379, 381
file handles *See* files, handles
file modes *See also* text, modes
 arrays of access 562
 binary *See* binary files
 bit mask 482
 changing 54, 55
 default 78, 83
 global variables 560
 setting 445, 560
 text 131, 155, 165
 translation 79, 82, 560
file-sharing mechanisms *See* files, sharing
filelength (function) 139
fileno (function) 140
files
 access 14, *See also* file modes
 flags 339, 472
 modes (global variable) 562
 permission 55, 482
 read/write *See* access, read/write
 accessibility, determining 14
 ARGS.EXE 4
 attribute bits 339, 472
 attribute word 54
 attributes 80
 access mode 54
 file sharing 338
 searching directories and 144
 setting 78, 82
 binary *See* binary files
 buffers 457
 line 457
 .CHR 274, 452
 closing 65, 66, 128, 129, 165
 COMMAND.COM 514
 control block 379, 381
 creating *See* files, new
 date 204

- setting 434
- deleting 393, 539
- end of
 - checking 106, 132, 385
 - resetting 62
- graphics driver 266
- handles 65, 66, 340
 - duplicating 98, 100
 - linking to streams 131
 - returning 140
- header 10
- information on, returning 170, 482
- locking 305, 540
- modes *See* file modes
- names
 - parsing 347
 - unique 327, 529
- new 78, 79, 81, 82
- open, statistics on 170, 482
- opening 337, 339
 - for update 131, 155, 165
 - in binary mode 528
 - shared 472
 - streams and 155, 165
- overwriting 80
- pointers *See* pointers, file
- reading 80, 384, 385
 - and formatting input from 167, 401, 544, 547, 549
 - characters from 135, 185
 - data from 162
 - integers from 235
 - strings from 138
- renaming 394
- replacing 165
- rewriting 78, 79
- scratch 529
 - opening 528
- security 221
- sequential *See* text files
- sharing
 - attributes 338
 - locks 305, 540
 - opening shared files 472
 - permission 473
- size 57
 - returning 139

- statistics 170, 482
- temporary 529
 - opening 528
- text *See* text files
- time 204
 - setting 434
- unlocking 540
- WILDARGS.OBJ 5, 6
- writing 177, 552, 554
 - attributes 80
 - characters to 160
 - formatted output to 158, 357, 543, 546
 - strings to 161
- fill colors *See* colors and palettes, fill colors
- fill patterns *See* colors and palettes, fill patterns
- fill style (graphics) 241
- fillellipse (function) 140
- filling a figure *See* graphics, filling
- fillpoly (function) 142
- findfirst (function) 143
- findnext (function) 145
- flags
 - carry 276, 277, 278, 279
 - console output 557
 - DOS verify 232, 460
 - format specifiers 358, 359, 361
 - read/write 339, 472
 - video output 557
- floating point
 - absolute value of 116
 - binary coded decimal 32, 387
 - characters and 26
 - control word 71
 - conversion *See* conversion, floating point
 - displaying 360, 406
 - double
 - conversion *See* conversion, double
 - exponents 296
 - exceptions 71
 - format specifiers 360, 404, 406
 - infinity 71
 - math package 157
 - modes 71
 - precision 71
 - reading 404
 - software signal 378
 - status word 59, 483

- floodfill (function) 147
- floor (function) 148
- flushall (function) 149
- flushing streams 134, 149
- _fmemccpy (function) 319
- _fmemchr (function) 320
- _fmemcmp (function) 321
- _fmemcpy (function) 322
- _fmemicmp (function) 323
- _fmemset (function) 324
- fmod (function) 151
- _fmode (global variable) 560
- fnmerge (function) 151
- fnsplit (function) 153
- fonts 452, *See also* graphics
 - bit mapped 453
 - character size 453, 523, 526
 - characteristics 452
 - gothic 452
 - graphics text 241, 452
 - information on 228
 - height 523
 - ID numbers 274
 - linked-in 392
 - multiple 342, 452
 - new 274
 - sans-serif 452
 - settings 452
 - small 452
 - stroked 452, 453
 - fine tuning 455
 - linked-in code 392
 - maximum number 274
 - multiple 452
 - text 344
 - triplex 452
 - width 526
- fopen (function) 155
- foreground color *See* colors and palettes
- format specifiers
 - assignment suppression 402, 403, 407, 408
 - characters 360, 404
 - type 402, 403
 - conventions
 - display 360
 - reading 405
 - conversion type 358, 359, 360, 363
 - cprintf 357
 - cscanf 401
 - flags 358, 359, 361
 - alternate forms 361
 - floating-point 360, 404, 406
 - fprintf 357
 - fscanf 401
 - inappropriate character in 408
 - input fields and 404, 408
 - integers 360, 404
 - modifiers
 - argument-type 402, 403, 407
 - input-size 358, 359, 363
 - size 402, 403, 407
 - pointers 360, 404
 - precision 358, 359, 362, 363
 - printf 357
 - range facility shortcut 405
 - scanf 401
 - sprintf 357, 478
 - sscanf 401
 - strings 360, 404
 - vfprintf 357
 - vfscanf 401
 - vprintf 357
 - vsscanf 401
 - vsprintf 357
 - vsscanf 401
 - width
 - printf 358, 359, 362
 - scanf 402, 403, 407, 408
- format strings
 - input 401
 - output 358
- formatting *See also* format specifiers
 - console input 84
 - cprintf 76
 - cscanf 84
 - fprintf 158
 - fscanf 167
 - output 76
 - printf 357
 - scanf 401
 - sprintf 478
 - sscanf 481

stdin *See* stdin
 streams *See* input; output
 strings 478, 548
 time 494
 vfprintf 543
 vscanf 544
 vprintf 546
 vscanf 547
 vsprintf 548
 vsscanf 549
 FP_OFF (function) 156
 FP_SEG (function) 159
 _fpreset (function) 157
 fprintf (function) 158
 format specifiers 357
 fputc (function) 160
 fputchar (function) 160
 fputs (function) 161
 frame base pointers as task state 308, 439
 fread (function) 162
 free (function) 163
 free blocks *See* memory blocks
 freeing memory *See* memory, freeing
 freemem (function) 164
 freopen (function) 165
 frexp (function) 166
 fscanf (function) 167
 format specifiers 401
 fseek (function) 168
 fsetpos (function) 169
 fstat (function) 170
 _fstrcat (function) 486
 _fstrchr (function) 487
 _fstrcspn (function) 491
 _fstrdup (function) 492
 _fstricmp (function) 496
 _fstrlen (function) 497
 _fstrlwr (function) 497
 _fstrnbrk (function) 503
 _fstrncat (function) 498
 _fstrncmp (function) 499
 _fstrncpy (function) 501
 _fstrncmp (function) 501
 _fstrnset (function) 502
 _fstrrchr (function) 504
 _fstrrev (function) 504
 _fstrset (function) 505

_fstrspn (function) 506
 _fstrstr (function) 506
 _fstrtok (function) 508
 _fstrupr (function) 511
 ftell (function) 175
 ftime (function) 176
 functions *See also* specific function name
 comparison, user-defined 376
 trigonometric *See* trigonometric functions
 fwrite (function) 177

G

game port 40
 gcvt (function) 178
 geninterrupt (function) 179
 getarccoords (function) 180
 getaspectratio (function) 182
 getbkcolor (function) 183
 getc (function) 185
 getcbrk (function) 185
 getch (function) 186
 getchar (function) 187
 getche (function) 187
 getcolor (function) 188
 getcurdir (function) 190
 getcwd (function) 190
 getdate (function) 191
 getdefaultpalette (function) 192
 getdfree (function) 193
 getdisk (function) 194
 getdrivename (function) 195
 getdta (function) 196
 memory models and 196
 getenv (function) 197
 getfat (function) 198
 getfatd (function) 199
 getfillpattern (function) 200
 getfillsettings (function) 201
 getftime (function) 204
 getgraphmode (function) 205
 getimage (function) 206
 getlinesettings (function) 208
 getmaxcolor (function) 210
 getmaxmode (function) 212
 getmaxx (function) 213
 getmaxy (function) 214
 getmodename (function) 216

getmoderange (function) 217
 getpalette (function) 218
 getpalettesize (function) 220
 getpass (function) 221
 getpid (function) 222
 getpixel (function) 223
 getpsp (function) 224
 gets (function) 225
 gettext (function) 226
 gettextinfo (function) 227
 gettextsettings (function) 228
 gettime (function) 230
 getvect (function) 231
 getverify (function) 232
 getviewsettings (function) 233
 getw (function) 235
 getx (function) 236
 gety (function) 237
 global variables 555, *See also* variables
 _8087 555
 _argc 556
 _argv 556
 arrays
 access modes 562
 character 556
 command-line arguments 556
 _ctype 556
 daylight 556
 setting value of 533
 directvideo 557
 DOS environment 557
 _doserrno 558
 environ 4, 557
 errno 558
 file mode 560
 _fmode 560
 heap size 561
 _heaplen 561
 main function and 556
 memory models and 561
 _new_handler 562
 numeric coprocessors and 555
 _openfd 562
 _osmajor 563
 _osminor 563
 _ovrbuffer 563
 printing error messages 558
 program segment prefix (PSP) 564
 _psp 564
 _set_new_handler 562
 stack size 564
 _stklen 564
 sys_errlist 558
 sys_nerr 558
 time zones 556, 565
 setting value of 533
 timezone 565
 setting value of 533
 tzname 565
 setting value of 533
 _version 566
 video output flag 557
 GMT *See* Greenwich mean time (GMT)
 gmtime (function) 238
 gothic fonts 452
 goto, nonlocal 86, 308, 439
 gotoxy (function) 240
 graphdefaults (function) 240
 grapherrormsg (function) 242
 _graphfreemem (function) 243
 _graphgetmem (function) 245
 graphics *See also* colors and palettes; text
 active page 462
 setting 414
 adapters 88
 problems with 18, 351
 arcs 18
 coordinates of 180
 elliptical 102
 aspect ratio
 correcting 418
 getting 182
 bars 29, 30
 bit images *See* bit images
 buffer, internal 436
 circles, drawing 58
 coordinates *See* coordinates
 current position *See* current position
 (graphics)
 default settings 192, 240
 ellipses 140
 error messages 242

- filling *See* colors and palettes
- fonts *See* fonts
- functions, justifying text for 450
- I/O 461
- library, memory management of 243
- lines *See* lines
- memory
 - allocation of memory from 245
 - freeing 243
- mode *See* graphics drivers, modes
- pie slices 351, 411
- pixels, color 223, 372
- polygons 97, 142
- rectangle 389
- screens, clearing 60
- settings, default 192, 240
- system
 - closing down 67
 - initializing 266
- text fonts *See* fonts
- viewport 240
 - clearing 63
 - displaying strings in 342, 344
 - information 233
 - setting for output 461
- visual page 462
- graphics drivers *See also* device drivers
 - BGI and 247, 266, 390
 - device driver table 272
 - colors *See* colors and palettes
 - current 195
 - detecting 88, 195, 266, 272
 - error messages 247
 - file 266
 - initializing 266
 - loading 266, 390
 - modes 268, 395
 - maximum number for current driver 212
 - names 216
 - overriding 88
 - range 217
 - returning 205
 - setting 437
 - switching 438
 - text 226, 227
 - new 272

- palettes *See* graphics, palettes
- registering 390
- vendor added 272
- graphresult (function) 247
- Greenwich mean time (GMT) 85, 92, 176
 - converting to 238
 - time zones and 534
 - timezone (global variable) 565

H

- handlers 467
 - error *See* error handlers
 - exception 59, 484
 - interrupt 86
 - DOS 249
 - signal *See* signal handlers
- handles, file *See* files, handles
- harderr (function) 249
- hardresume (function) 251
- hardretn (function) 254
- hardware
 - checking
 - device type 285
 - for presence of 40, 286
 - checking for presence of 40
 - disk drives *See* disk drives
 - error handlers 249, 251, 254
 - I/O, controlling 282
 - interrupts 40, 105
 - keyboard *See* keyboard
 - ports 270
 - printer 44
 - reading from 270
 - writing to 341
- header files (explained) 10
- heap 73
 - allocating memory from 49, 163, 315, 388
 - checking 256
 - far *See* far heap
 - free blocks
 - checking 257
 - filling 259
 - length 561
 - memory freeing in 163
 - near, size of 561
 - nodes 258
 - reallocating memory in 388

- walking through 261
- heapcheck (function) 256
- heapcheckfree (function) 257
- heapchecknode (function) 258
- heapfillfree (function) 259
- _heaplen (global variable) 561
- heapwalk (function) 261
- Hercules card *See also* graphics; screens
 - detecting presence of 89
- hexadecimal digits, checking for 292
- high intensity 262
- highvideo (function) 262
- huge memory model *See* memory models
- hyperbolic cosine 74
- hyperbolic sine 470
- hyperbolic tangent 516
- hypot (function) 262
- hypotenuse 262

I

- IBM 8514
 - colors, setting 448
 - detecting presence of 89
- IBM 3270 PC, detecting presence of 89
- ID
 - font numbers 274
 - process 222
- illegal instruction, software signal 378
- imag (function) 263
- imagesize (function) 264
- imaginary numbers *See* complex numbers
- indicator
 - end-of-file 62, 106, 132, 385
 - error 62
- infinity, floating point 71
- initgraph (function) 266
- initializing
 - file pointers 397
 - graphics system 266
 - memory 324, 444
 - random number generator 383, 480
 - strings 502, 505
- inport (function) 270
- inportb (function) 270
- input *See also* I/O
 - console, reading and formatting 84

- fields 404
 - format specifiers and 408
 - from streams 167, 544, 549
 - formatting 167, 401, 544, 547, 549
 - pushing characters onto 536
 - stdin 401, 547
 - terminating from streams 408
- input ports *See* ports, I/O
- inline (function) 271
- installuserdriver (function) 272
- installuserfont (function) 274
- int86 (function) 276
- int86x (function) 277
- intdos (function) 278
- intdosx (function) 279
- integers
 - absolute value 11
 - ASCII and *See* ASCII codes
 - conversion *See* conversion, integer
 - displaying 360
 - division 94
 - long integers 297
 - format specifiers 360, 404
 - long
 - absolute value of 295
 - conversion *See* conversion, long integer
 - division 297
 - rotating 310, 311
 - reading 235, 404
 - rotating 310, 399
 - storing in memory 353
 - unsigned long, conversion *See* conversion, unsigned long integer
 - writing to stream 375
- integrated environment
 - wildcard expansion and 6
- intensity
 - high 262
 - low 309
 - normal 336
- internal graphics buffer 436
- international
 - country-dependent data 75
 - setting 302, 443
 - date formats 75
- interrupts
 - 8086 276, 277, 280

- BIOS *See* BIOS, interrupts
- control-break 185, 251, 424
- controlling 92, 179
- disabling 92
- DOS *See* DOS, interrupts
- DOS interface *See also* DOS, interrupts
- enabling 105
- handlers
 - DOS 86, 249
 - signal handlers and 467
- hardware 105
- non-maskable 92
- software 179, 276, 281
 - interface 277, 280
 - signal 378
- system equipment 40
- vectors 86
 - 8086 231, 458
 - setting 458
- intr (function) 280
- invalid access to storage 378
- inverse cosine 16
- inverse sine 21
- inverse tangent 23, 24
- ioctl (function) 282
- I/O *See also* input; output
 - buffers 422
 - characters, writing 366, 367, 368
 - controlling 282
 - disk 37
 - files *See* files, reading; files, writing
 - graphics 461
 - integers, writing 375
 - keyboard 186, 187
 - checking for keystrokes 293
 - ports
 - hardware 270
 - writing to 341
 - screen 76
 - writing to 77, 367
 - serial 35
 - streams 131, 155, 165, 536, *See also* streams, reading; streams, writing
 - strings *See* strings, reading; strings, writing
- isalnum (function) 283
- isalpha (function) 284
- isascii (function) 285

- isatty (function) 285
- iscntrl (function) 286
- isdigit (function) 287
- isgraph (function) 288
- islower (function) 288
- isprint (function) 289
- ispunct (function) 290
- isspace (function) 290
- isupper (function) 291
- isxdigit (function) 292
- itoa (function) 292

J

- Japanese date formats 75
- justifying text for graphics functions 450

K

- kbhit (function) 293
- keep (function) 294
- keyboard
 - buffer, pushing characters back into 537
 - I/O 186, 187
 - checking for 293
 - operations 42
 - reading characters from 186, 187
 - keystrokes, checking for 293

L

- labs (function) 295
- large memory model *See* memory models
- ldexp (function) 296
- ldiv (function) 297
- length
 - of files 57, 139
 - of strings 497
- lfind (function) 298
- libraries
 - entry headings 9
 - graphics, memory management and 243
- line (function) 299
- line-buffered files 457
- line styles *See* graphics, lines
- linear searches 298, 311
- linerel (function) 300
- lines *See also* graphics
 - blank, inserting 271

- clearing to end of 68
- deleting 68, 87
- drawing
 - between points 299
 - from current position 301
 - mode 464
 - relative to current position 300
- pattern of 208
- rectangles and 389
- style of 208, 441
- thickness of 208, 441
- lineto (function) 301
- linked-in fonts 392
- linked-in graphics drivers code 390
- literal values, inserting into code 104
- local standard time 85, 92, 176, 238, 303, *See also* time zones
- locale
 - current 302
 - selecting 443
- localeconv(function) 302
- localtime (function) 303
- lock (function) 305
- locks, file-sharing 305, 540
- log10 (function) 307
- log (function) 306
- logarithm
 - base 10 307
 - natural 306
- long integers *See* integers, long
- longjmp (function) 308
- low intensity 309
- lowercase
 - characters 530, 531
 - checking for 288
 - conversions 511, 532, 533
 - strings 497
- lowvideo (function) 309
- LPT1 and LPT2 *See* printers, ports
- _lrotl (function) 310
- _lrotr (function) 311
- lsearch (function) 311
- lseek (function) 313
- ltoa (function) 314

M

- machine language instructions
 - inserted into object code 104
- macros
 - assert 22
 - case conversion 530, 532
 - character classification 286, 290
 - case 283, 284, 288, 291
 - integers 283, 285, 287, 292
 - printable characters 288, 289
 - characters 368
 - ASCII conversion 530
 - comparing two values 319, 325
 - far pointer 327
 - file deletion 393
 - input ports 270
 - output ports 341
 - peek 348
 - peekb 349
 - poke 353
 - pokeb 353
 - toascii 530
 - variable argument list 541
- main (function) 3-7
 - arguments passed to 3, 556
 - example 4
 - wildcards 5
 - compiled with Pascal calling conventions 7
 - declared as C type 7
 - global variables and 556
 - value returned by 7
- malloc (function) 315
- mantissa 166, 329
- math coprocessors *See* numeric coprocessors
- math error handler, user-modifiable 316
- math package, floating-point 157
- matherr (function) 316
- max (function) 319
- maximum color value 210
- MCGA, detecting presence of 89
- medium memory model *See* memory models
- memccpy (function) 319
- memchr (function) 320
- memcmp (function) 321
- memcpy (function) 322
- memicmp (function) 323
- memmove (function) 324

- memory *See also* memory allocation
 - access (DMA) 39, 40
 - access error *See* errors
 - addresses
 - returning byte from 349
 - returning word from 348
 - storing byte at 353
 - storing integer at 353
 - bit images 264
 - saving to 206
 - blocks *See* memory blocks
 - copying 319, 322, 324, 334
 - in small and medium memory models 330
 - direct access (DMA) 39, 40
 - expanded and extended 345, 346
 - freeing
 - in DOS memory 164
 - in far heap 118
 - in graphics memory 243
 - in heap 163
 - in small and medium memory models 118
 - initializing 324, 444
 - measure of *See* memory allocation
 - models *See* memory models
 - random access *See* RAM
 - reallocating *See* memory allocation
 - screen segment, copying to 226
 - size 41
 - determining 43
- memory allocation 17, *See also* memory
 - data segment 46
 - changing 400
 - dynamic 49, 163, 315, 388
 - far heap *See* far heap
 - freeing 118, 164
 - graphics memory 243, 245
 - heap *See* heap
 - memory models and 49, 118, 126
 - _new_handler and 562
 - reallocating 127
 - _set_new_handler and 562
 - unused 72, 118
- memory blocks
 - adjusting size of 127, 421
 - in heap 388
 - file control 379, 381
 - free 120, 257
 - filling 123, 259
 - initializing 324, 444
 - random
 - reading 379
 - writing 381
 - searching 320
- memory models
 - disk transfer address and 196
 - DOS system calls and 33, 34
 - memory allocation and 50, 118, 126
 - moving data and 330
 - program segment prefix and 561, 564
 - size of near heap 561
 - stack size 564
 - tiny, restrictions 118, 126, 127
- memset (function) 324
- microprocessors 468
- midnight, number of seconds since 45
- min (function) 325
- MK_FP (function) 327
- mkdir (function) 326
- mktemp (function) 327
- mktime (function) 328
- mnemonics, error codes 558, 559
- modes
 - access *See* access modes
 - files *See* file modes
 - floating point, rounding 71
 - graphics *See* graphics drivers, modes
 - screen *See* screens, modes
 - text *See* text, modes
- modf (function) 329
- modifiers, format specifiers *See* format specifiers, modifiers
- modulo 151
- monochrome adapters 89, *See also* graphics graphics problems 18, 351
- monochrome EGA *See* Enhanced Graphics Adapter (EGA)
- movedata (function) 330
- moverel (function) 331
- movetext (function) 332
- moveto (function) 333
- movmem (function) 334
- multiple tasks 308, 439

N

- natural logarithm 306
- near heap 561
- new files 78, 79, 81, 82, *See also* files, opening
- _new_handler (global variable) 562
- newline character 374
- NMI 93
- no parity (communications) 35
- nodes, checking on heap 258
- non-maskable interrupt 92
- nonlocal goto 86, 308, 439
- norm (function) 335
- normal intensity 336
- normvideo (function) 336
- nosound (function) 337
- numbers *See also* complex numbers; floating point; integers
 - ASCII
 - checking for 287
 - BCD 32, 387
 - complex *See* complex numbers
 - pseudorandom 378
 - random 378, 382, 383
 - generating 480
 - rounding 51, 148
 - turning strings into 26
- numeric coprocessors
 - checking for presence of 41
 - control word 71
 - exception handler 59, 484
 - global variables 555
 - problems with 157
 - status word 59, 484

O

- object code
 - machine language instructions and 104
- odd parity (communications) 35
- offset, of far pointer 156, 327
- _open (function) 337
- open (function) 339
- _openfd (global variable) 562
- opening files *See* files, opening
- operating mode of screen *See* screens, modes
- _osmajor (global variable) 563
- _osminor (global variable) 563

- outport (function) 341
- outportb (function) 341
- output *See also* I/O
 - characters, writing 367
 - displaying 158, 357, 546
 - flag 557
 - flushing 134
 - formatting 76
 - ports *See* ports, I/O
 - to streams
 - formatting 158, 357, 546
- outtext (function) 342
- outtextxy (function) 344
- overlays
 - buffers
 - default size 563
 - expanded and extended memory and 345, 346
- overlays, coroutines and 308, 439
- overwriting files 80
- _ovrbuffer (global variable) 563
- _OvrInitEms (function) 345
- _OvrInitExt (function) 346

P

- p option (Pascal calling conventions)
 - main function and 7
- page
 - active 414
 - numbers, visual 462
- palettes *See* colors and palettes
- parent process 108, 475, *See also* processes
- parity (communications) 35
- parsfnm (function) 347
- parsing file names 347
- Pascal calling conventions
 - compiling main with 7
- passwords 221
- PATH environment variable 108, 476
- paths
 - directory 410
 - DOS 410
 - finding 190
 - names
 - creating 151
 - splitting 153
- patterns, fill *See* colors and palettes, fill patterns

- pause (suspended execution) 87, 471
- PC speaker 337, 474
- peek (function) 348
- peekb (function) 349
- permissions, file access *See* files, access
- perror (function) 350, 558
- PID (process ID) 222
- pie slices 351
 - elliptical 411
- pieslice (function) 351
- pixels, graphics color 223, 372
- pointers
 - to error messages 242, 493, 494
 - far 117, 126, 127
 - address segment 159, 327
 - creating 327
 - offset of 156, 327
 - file
 - initializing 397
 - moving 313
 - obtaining 137
 - resetting 168, 384, 385
 - returning 175
 - current position of 516
 - setting 169, 339, 340, 472
 - format specifiers 360, 404
 - frame base 308, 439
 - stack 308, 439
- poke (function) 353
- pokeb (function) 353
- polar (function) 354
- poly (function) 355
- polygons
 - drawing 97, 142
 - filling 142
- polynomial equation 355
- portability (explained) 10
- ports
 - checking for presence of 40
 - communications 35, 40, 286
 - hardware *See* hardware, ports
 - I/O 270, 341
 - macros 270, 341
 - writing to 341
- pow10 (function) 357
- pow (function) 356
- powers
 - calculating ten to 357
 - calculating values to 356
- precision
 - floating point 71
 - format specifiers 358, 359, 362, 363
- printable characters, checking for 288, 289
- printers *See also* hardware
 - checking for 40, 286
 - ports 44
 - printing directly 44
- printf (function) 357
 - conversion specifications 358
 - format specifiers 357
- printing, error messages 350, 558
- process ID 222
- processes
 - child 108, 475
 - exec... (functions)
 - suffixes 108
 - parent 108, 475
 - stopping 11
- profilers 563
- program segment prefix (PSP) 224
 - current program 564
 - memory models and 561, 564
- programs
 - loading and running 107
 - process ID 222
 - RAM resident 294
 - signal types 377
 - stopping 11, 25, 86
 - exit status 113, 114, 294
 - request for 378
 - suspended execution 87, 471
 - TSR 294
- protocol settings (communications) 35
- prototype (defined) 10
- pseudorandom numbers 378
- PSP *See* program segment prefix (PSP)
- _psp (global variable) 564
- punctuation characters, checking for 290
- putc (function) 366
- putch (function) 367
- putchar (function) 368
- putenv (function) 369
- putimage (function) 370

putpixel (function) 372
puts (function) 374
puttext (function) 374
putw (function) 375

Q

qsort (function) 376
quicksort 376
quotient 94, 297

R

raise (function) 377
RAM *See also* memory
 resident program 294
 size 41, 43
 unused 72
rand (function) 378
randbrd (function) 379
randbwr (function) 381
random (function) 382
random access memory *See* RAM
random block read 379
random block write 381
random number generator 378, 382
 initializing 383, 480
random numbers 378, 382, 383
random record field 379, 381
randomize (function) 383
range facility shortcut 405
_read (function) 384
read (function) 385
read access *See* access, read/write
read error 133
read/write flags 339, 472
real (function) 387
real numbers *See* floating point
realloc (function) 388
reallocating memory *See* memory allocation
records
 random fields 379, 381
 sequential 298
rectangle (function) 389
register variables, as task states 308, 439
registerbgidriver (function) 390

registerbgifont (function) 392
registers 249
 segment, reading 413
REGPACK structure 281
remainder 94, 151, 297
remove (function) 393
rename (function) 394
request for program termination 378, *See also*
 programs, stopping
restorecrmode (function) 395
restoring screen 374, 395
rewind (function) 397
rmdir (function) 398
rotation, bit
 long integer 310, 311
 unsigned integer 399
_rotl (function) 399
_rotr (function) 399
rounding 51, 148
 banker's 32
 modes, floating point 71
RS-232 communications 35

S

sans-serif font 452
sbrk (function) 400
scanf (function) 401
 format specifiers 401
 termination 408
scanf (functions) termination conditions 408
scratch files *See also* files
 naming 529
 opening 528
screens
 aspect ratio
 correcting 418
 getting 182
 bit images *See* bit images
 clearing 60, 69, 437
 coordinates, maximum 213, 214
 copying
 text from 332
 displaying strings 77
 echoing to 186, 187
 formatting output to 76

- graphics *See* graphics
- graphics drivers *See* graphics drivers
- modes
 - restoring 374, 395
 - text *See* text, modes
- saving 226
- segment, copying to memory 226
- writing characters to 367
- scrolling 566
- search key 312
- searches
 - appending and 311
 - binary 47
 - block, for characters 320
 - DOS
 - algorithms 108
 - path, for file 410
 - linear 298, 311
 - string
 - for character 487
 - for tokens 508
- searchpath (function) 410
- sector (function) 411
- sectors, disk *See* disks, sectors
- security, passwords 221
- seed number 480, *See also* random numbers
- segment prefix, program 224, 561, 564
- segments *See also* data segment
 - far pointer 159, 327
 - registers, reading 413
 - scanning for characters in strings 506
 - screen, copying to memory 226
- segread (function) 413
- sequential files *See* text files
- sequential records 298
- serial communications, I/O 35, *See also* communications
- setactivepage (function) 414
- setallpalette (function) 415
- setaspectratio (function) 418
- setbkcolor (function) 419
- setblock (function) 421
- setbuf (function) 422
- setcbk (function) 424
- setcolor (function) 425
- setcursortype (function) 427
- setdate (function) 428
- setdisk (function) 429
- setdta (function) 429
- setfillpattern (function) 431
- setfillstyle (function) 432
- setftime (function) 434
- setgraphbufsize (function) 436
- setgraphmode (function) 437
- setjmp (function) 439
- setlinestyle (function) 441
- setlocale (function) 443
- setmem (function) 444
- setmode (function) 445
- _set_new_handler (global variable) 562
- setpalette (function) 446
- setrgbpalette (function) 448
- settextjustify (function) 450
- settextstyle (function) 452
- settime (function) 454
- settings, graphics, default 192, 240
- setusercharsize (function) 455
- setvbuf (function) 457
- setvect (function) 458
- setverify (function) 460
- setviewport (function) 461
- setvisualpage (function) 462
- setwritemode (function) 464
- shared files *See* files, sharing
- signal (function) 465
- signals
 - handlers 377, 465
 - interrupt handlers and 467
 - returning from 468
 - user-specified 465
 - program 377
 - software *See* software signals
- sin (function) 469
- sine 469
 - hyperbolic 470
 - inverse 21
- sinh (function) 470
- size
 - color palette 220
 - file 57, 139
 - memory 41, 43
 - stack 564
- sleep (function) 471
- small fonts 452

- small memory model *See* memory models
- software interrupts *See* interrupts, software
- software signals 377, 378
- sopen (function) 472
- sorts, quick 376
- sound (function) 474
- sounds
 - turning off 337
 - turning on 474
- SP register
 - hardware error handlers and 249
- space on disk, finding 193
- spawn... (functions) *See* processes
 - suffixes 476
- spawnl (function) 475
- spawnle (function) 475
- spawnlp (function) 475
- spawnlpe (function) 475
- spawnv (function) 475
- spawnve (function) 475
- spawnvp (function) 475
- spawnvpe (function) 475
- speaker
 - turning off 337
 - turning on 474
- sprintf (function) 478
 - format specifiers 357, 478
- sqrt (function) 479
- square root 479
- srand (function) 480
- sscanf (function) 481
 - format specifiers 401
- stack 50, 73, 315
 - length 564
 - pointer, as task states 308, 439
 - size, global variable 564
- standard time 85, 92, 176, 238, *See also* time
 - zones
- stat (function) 482
- stat structure 171, 482
- _status87 (function) 483
- status bits (communications) 35
- status byte 38
- status word
 - floating point 59
 - floating-point 483
 - numeric coprocessors 59, 484
- stdaux 129, *See also* streams
- stderr 129, 165, *See also* streams
- stdin 129, 165, *See also* streams
 - buffers and 423
 - reading
 - characters from 136, 187
 - input from 401, 547
 - strings from 225
- stdout 129, 165, *See also* streams
 - buffers and 423
 - writing
 - characters to 160, 368
 - formatted output to 357, 546
 - strings to 374
- stdprn 129, *See also* streams
- stime (function) 484
- _stklen (global variable) 564
- stop bits (communications) 35
- storage, invalid access 378
- stpcpy (function) 485
- strcat (function) 486
- strchr (function) 487
- strcmp (function) 488
- strcmpi (function) 489
- strcoll (function) 490
- strcpy (function) 491
- strcspn (function) 491
- strdup (function) 492
- streams
 - closing 128, 129, 165
 - error and end-of-file indicators 62, 132, 133
 - flushing 134, 149
 - formatting input from 167, 544, 549
 - stdin 401, 547
 - I/O 131, 155, 165
 - pushing character onto 536
 - linking file handles to 131
 - opening 155, 165
 - pointers *See also* pointers
 - file 168, 169
 - initializing 397
 - reading
 - characters from 135, 185
 - data from 162
 - input from 167, 544, 549
 - stdin 401
 - integers from 235

- strings from 138
- replacing 165
- stdaux 129
- stderr 129, 165
- stdin *See* stdin
- stdout *See* stdout
- stdprn 129
- terminated input 408
- unbuffered 423, 457
- writing 149, 177
 - characters to 160, 366, 368
 - formatted output to 158, 357, 543
 - stdout 546
 - integers to 375
 - strings to 161, 374
- streams, buffered *See* buffers, streams and
- _strerror (function) 493
- strerror (function) 494
- strftime (function) 494
- stricmp (function) 496
- strings
 - appending 486
 - parts of 498
 - changing 512
 - comparing 321, 488, 490
 - ignoring case 323, 489, 496
 - parts of 499
 - ignoring case 500, 501
 - concatenated 486, 498
 - converting *See* conversion, strings
 - copying 485, 491
 - new location 492
 - truncating or padding 501
 - displaying 77, 342, 344, 360
 - duplicating 492
 - format *See* format strings
 - format specifiers 360, 404, *See also* format specifiers
 - formatting 478, 494, 548
 - height, returning 523
 - initializing 502, 505
 - length, calculating 497
 - lowercase 497
 - reading 404
 - and formatting 481
 - from console 51
 - from streams 138, 225
 - reversing 504
 - searching
 - for character 487
 - for character in set 503
 - for characters not in set 491
 - for last occurrence of character 504
 - for segment in set 506
 - for substring 506
 - for tokens 508
 - transforming 512
 - uppercase 511
 - width, returning 526
 - writing
 - to current environment 369
 - formatted output to 478, 548
 - to stdout 374
 - to streams 161
 - to the screen 77
- strlen (function) 497
- strlwr (function) 497
- strncat (function) 498
- strncmp (function) 499
- strncmpi (function) 500
- strncpy (function) 501
- strnicmp (function) 501
- strnset (function) 502
- stroked fonts *See* fonts, stroked
- strpbrk (function) 503
- strchr (function) 504
- strev (function) 504
- strset (function) 505
- strspn (function) 506
- strstr (function) 506
- strtod (function) 507
- strtok (function) 508
- strtol (function) 509
- strtoul (function) 511
- structures
 - graphics palette definition 192
 - REGPACK 281
 - stat 171, 482
- strupr (function) 511
- strxfrm (function) 512
- substrings, scanning for 506
- suffixes
 - exec... 108
 - spawn... 476

- suspended execution, program 87, 471
- swab (function) 513
- swapping bytes 513
- syntax errors *See* errors
- sys_errlist (global variable) 558
- sys_nerr (global variable) 558
- system
 - buffers 128
 - commands, issuing 514
 - date *See* dates
 - DOS calls *See* DOS, system calls
 - equipment interrupt 40, *See also* hardware
 - error messages 350, 558, *See also* errors
 - graphics 67, 266
 - time *See* time
- system (function) 514

T

- tables, searching 47, 311
- tan (function) 515
- tangent 515
 - hyperbolic 516
 - inverse 23, 24
- tanh (function) 516
- task states
 - defined 308, 439
 - register variables 308, 439
- tasks, multiple 308, 439
- tell (function) 516
- template (file names) 327
- temporary files *See also* files
 - naming 529
 - opening 528
- terminals *See also* hardware
 - checking for 285
- terminate and stay resident programs 294
- terminating
 - input from streams 408
 - program execution *See* programs, stopping
 - software signals 378
- termination function 25
- testing conditions 22
- text *See also* file modes; graphics; text files
 - attributes 517, 520, 521
 - background color, setting 517, 520
 - characteristics 452
 - colors 521, *See also* colors and palettes
 - copying *See also* editing, block operations
 - from one screen rectangle to another 332
 - to memory 226
 - to screen 374
 - fonts *See* fonts
 - intensity
 - high 262
 - low 309
 - normal 336
 - justifying 450
 - modes (screens) 374, *See also* file modes
 - character color 517, 521
 - coordinates 226
 - copying to memory 226
 - video information 227
 - screens *See* screens
 - strings, displaying 342, 344
 - window *See* windows, text
- text (screens)
 - modes 524, 551
- text files *See also* files; text
 - creat and 79
 - createmp and 82
 - fdopen and 131
 - fopen and 155
 - freopen and 165
 - _read and 384
 - reading 385
 - setting 445
 - mode 131, 155, 165, 560
- textattr (function) 517
- textbackground (function) 520
- textcolor (function) 521
- textheight (function) 523
- textmode (function) 524
- textwidth (function) 526
- three-dimensional bars 30
- time *See also* dates; time zones
 - BIOS timer 45
 - delays in program execution 87, 471
 - difference between two 92
 - elapsed 64, 92
 - returning 527
 - file 204
 - setting 434
 - formatting 494
 - global variables 533, 556, 565

- system 20, 85, 176, 238
 - converting from DOS to UNIX 96
 - converting from UNIX to DOS 538
 - local 303
 - returning 230
 - setting 454, 484
- time (function) 527
- time and date conversion *See* conversion, date and time
- time zones 176, 238, *See also* daylight saving time; time
 - arrays 565
 - differences between 92
 - global variables 556, 565
 - Greenwich mean time *See* Greenwich mean time (GMT)
 - local *See* local standard time
 - setting 85, 534
- timer, reading and setting 45
- timezone (global variable) 565
 - setting value of 533
- tiny memory model *See* memory models
- tmpfile (function) 528
- tmpnam (function) 529
- toascii (function) 530
- tokens, searching for in string 508
- _tolower (function) 530
- tolower (function) 531
- _toupper (function) 532
- toupper (function) 533
- translation mode 79, 82, 560
- triangles, hypotenuse 262
- trigonometric functions *See also* complex numbers
 - arc cosine 16
 - arc sine 21
 - arc tangent 23, 24
 - cosine 73
 - hyperbolic 74
 - inverse 16
 - hyperbolic tangent 516
 - sine 469
 - hyperbolic 470
 - inverse 21
 - tangent 515
 - hyperbolic 516
 - inverse 23, 24

- triplex font 452
- TSR programs 294
- Turbo Profiler 563
- two-dimensional bars 29
- type checking, device 285
- tzname (global variable) 565
 - setting value of 533
- tzset (function) 533

U

- U.S. date formats 75
- ultoa (function) 535
- unbuffered streams 423, 457
- ungetc (function) 536
- ungetch (function) 537
- UNIX
 - date and time
 - converting DOS to 96
 - converting to DOS format 538
 - unixtodos (function) 538
 - unlink (function) 539
 - unlock (function) 540
 - unsigned integers *See* integers
 - unsigned long integers *See* integers
 - uppercase
 - characters 291, 532, 533
 - checking for 291
 - conversions 497, 530, 531
 - strings 511
 - user-defined
 - comparison function 376
 - fill pattern 431, 432, *See also* graphics, fill patterns
 - user hook 316
 - user-loaded graphics driver code 390
 - user-modifiable math error handler 316
 - user-specified signal handlers 465

V

- va_arg (function) 541
- va_arg (variable argument macro) 541
- va_end (function) 541
- va_list (variable argument macro) 541
- va_start (function) 541
- va_start (variable argument macro) 541

- values *See also* absolute value
 - break 46, 400
 - calculating powers to 356, 357
 - comparing 319, 325
 - literal 104
- variables
 - argument list 541
 - environment 108, 476, 557
 - COMSPEC 514
 - global *See* global variables
 - register 308, 439
 - variable argument list
 - conversion specifications and 358
- vectors, interrupt *See* interrupts
- vendor-added device driver 272
- verify flag (DOS) 232, 460
- version numbers, DOS 563, 566
- vfprintf (function) 543
 - format specifiers 357
 - variable argument list 541
- vfscanf (function) 544
 - format specifiers 401
 - variable argument list 541
- VGA *See* Video Graphics Array Adapter (VGA)
- video *See also* hardware
 - checking for 286
 - information, text mode 227
 - mode *See also* graphics drivers, modes;
 - screens, modes
 - checking 41
 - output flag 557
- Video Graphics Array Adapter (VGA) *See also*
 - graphics; screens
 - color palettes 416, *See also* graphics, color palette
 - colors *See* colors and palettes
 - detecting presence of 89
- viewport *See* graphics, viewport
- visual graphics page 462
- vprintf (function) 546
 - format specifiers 357
 - variable argument list 541
- vscanf (function) 547
 - format specifiers 401
 - variable argument list 541
- vsprintf (function) 548
 - format specifiers 357

- variable argument list 541
- vsscanf (function) 549
 - format specifiers 401
 - variable argument list 541

W

- warning beeps 337, 474
- warnings *See* errors
- wherex (function) 550
- wherey (function) 551
- whitespace, checking for 290
- width
 - specifier *See* format specifiers, width string, returning 526
- WILDARGS.OBJ 5
- wildcards
 - expansion 5
 - by default 6
 - from the IDE 6
- window (function) 551
- windows
 - scrolling 566
 - text
 - cursor position 240, 550, 551
 - defining 551
 - deleting lines in 68, 87
 - inserting blank lines in 271
- words *See also* status word
 - floating-point control 71
 - reading from hardware ports 270
 - returning from memory 348
 - writing to hardware ports 341
- working directory *See* directories, current
- _write (function) 552
- write (function) 554
- write access *See* access, read/write
- write error 133, *See also* errors

X

- x aspect factor 182
- x-coordinate 213, 236, *See also* coordinates

Y

- y aspect factor 182
- y-coordinate 214, 237, *See also* coordinates

2.0

LIBRARY
REFERENCE

BORLAND[®] C++

B O R L A N D

CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 860001, SCOTTS VALLEY, CA 95067-0001, (408) 438-5300.
OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, SWEDEN AND THE UNITED KINGDOM ■ PART # 14MN-TCPO3 ■ BOR 1972