

# **TopSpeed® C**

**For IBM® Personal Computers and Compatibles**

## **Library Reference**

**TopSpeed Corporation**

Copyright© 1990-1991, by TopSpeed Corporation. All rights reserved.

TopSpeed® is a registered trademark of TopSpeed Corporation.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Printed in the United Kingdom.

10 9 8 7 6 5 4 3 2 1

# Contents

<b>CHAPTER 1 .....</b>	<b>1</b>
Introduction to the C Library .....	1
Introduction .....	1
How to use this manual .....	1
Finding the function you want .....	2
How to use the library .....	4
Error and exception handling .....	10
OS2 considerations .....	14
Multi-thread considerations .....	14
<b>CHAPTER 2 .....</b>	<b>16</b>
A Guide to the C library functions .....	16
Input-output .....	16
Mathematical functions .....	27
Ctype (Character) test and conversion functions .....	29
Memory allocation .....	30
Huge data objects .....	31
Search and sort functions .....	33
Time and date functions .....	33
Standard process control functions .....	34
The TopSpeed process and task control module .....	36
Mouse control functions .....	39
The TopSpeed window module .....	40
The TopSpeed Graphics library .....	47
Operating system interface .....	50
<b>CHAPTER 3 C LIBRARY REFERENCE .....</b>	<b>55</b>
<b>APPENDIX A INCLUDE FILES .....</b>	<b>666</b>
<b>APPENDIX B IMPLEMENTATION DEPENDENT .....</b>	<b>668</b>
<b>INDEX .....</b>	<b>675</b>

# CHAPTER 1

## Introduction to the C Library

### Introduction

---

This manual describes the TopSpeed C function library, one of the most extensive function libraries available for industry-standard Personal Computers. All the functions described here are for use with either DOS or OS2 except those that access services specific to one or other system. All are fully link-compatible with object modules compiled under any of the TopSpeed language family.

#### The library offers:

- All ANSI standard function calls and data types;
- a complete text *windowing* system;
- an extensive *task and process control* pack, whose facilities include full pre-emptive DOS multi-tasking;
- a comprehensive *graphics* function set;
- a comprehensive *low-level operating* system interface to DOS and the BIOS.
- Portability from a wide range of other platforms, including most commonly-used UNIX, Microsoft C and Turbo-C functions.

The TopSpeed C library is also compile-compatible with TopSpeed C++, and all C library function calls are available to C++ programs.

This manual is the reference for the functions contained in the TopSpeed C library, and applies equally to their use by C or C++ programs.

### How to use this manual

---

The core of this manual is the Function Reference section, which contains an alphabetical list of all functions distributed with the TopSpeed C library. The Library Guide, which is the next chapter of this book, supplements the reference section, explaining what the main groups of functions are for and how they fit together.

## Finding the function you want

---

If you know the name of the function you require, you can proceed directly to the reference section (or to the alphabetical index at the back of this book).

Alternatively, if you want more general information or if you are not sure exactly which function to use, the Library Guide will assist you in your choice. Treat it as an introduction, or as a subject index for the reference section.

### **Ordering of functions**

Functions are for the most part described in strict alphabetical order, any leading underscore (‘\_’) characters being ignored for indexing purposes.

There are only minor exceptions to this rule. Two or three groups of similarly named functions (for example the `isalnum...isxdigit` family of character conversion functions and the `spawn` and `exec` family) are listed together rather than break up the description of a very closely-related group. The second exception relates to different variants of the same function — either memory model variants (see ‘Programming memory models: choosing the right pointer’ below) or a group of functions which take long double parameters in place of double parameters. In cases where there are two different variants of the same function, such as `_fstcpy` and `stcpy`, you will find a brief cross-reference under `_fstcpy` and the main description under `stcpy`. Generally this applies only to far, huge or near variants. In such cases you will always find the main description under the default variant of the function.

### **Finding the right window function**

Some of the Turbo C window functions appear in two versions with the same name and different definitions. This is because there are two versions of the text window module. One is the TopSpeed window module; the other is provided for compatibility with Turbo C. You can only link to one or other of these modules so that a name conflict cannot occur. See the section on the Turbo C window module for more details.

### **Finding other information**

The Guide also briefly describes any relevant globally-defined variables and constants, and any special type definitions contained in the header files concerned.

The reference pages contain most of the information on the functions including examples of use and any special considerations such as compatibility, the relevant include file, error status and a list of related

functions. In addition, the title line for each function contains a quick reference to the general introduction, so that if you need background information you can rapidly find it.

### **Compatibility considerations**

Care has been taken to provide you with the widest possible range of functions to ensure compatibility with other commonly-used libraries. Sometimes this involves a certain amount of duplication, since different libraries provide the same function under different names. You will pay no penalty for using a function that is duplicated in this manner, but you should try to avoid using two different versions of the same, or essentially the same function.

### **Quick reference conventions**

Function title lines are flagged with a quick reference key as follows:

A ANSI

T Turbo C window module

M Provided for Microsoft compatibility

W TopSpeed C text window module

U Unix-compatible function call

### **These serve a dual purpose:**

- They are a quick guide to the portability of each function — for more detail see ‘Portability: Choosing the right function’ in the next section
- Where a special library is required — for TopSpeed windows, graphics and Turbo C windows — they draw attention to it.

## How to use the library

---

Very few special steps have to be taken to use the TopSpeed C Library, other than ensuring that the correct *header file* has been included in the source code for your program, as described immediately below.

If you use the TopSpeed project system, any functions you use will be automatically be linked. You only need to take special steps to find the correct library if you have customized your project file and are using the windowing and graphics functions. In this case consult ‘Choosing the right library’ on page 12 below.

If you are using a standard memory model the correct function variant will automatically be linked; if, however, you are using a mixed memory model you should consult ‘Programming Memory Models’ on page 8 below.

For maximum portability, implementations of many functions in common use have been provided; consult ‘Programming for portability’ on page 6 if you are in doubt.

### **Including the correct header file**

When using functions from the TopSpeed Library you must identify each function used and specify its parameters. To ensure this is done correctly you should insert a statement similar to:

```
#include <header.h>
```

in your program, where *header.h* is the full name of a *header file* specified in the description of the function you want to use. This *preprocessor* statement should be at the top of your program, before any other code. This tells the compiler to import a file containing all the information needed to ensure that your calls to the function concerned will be compiled properly. The information comprises the function declaration, any relevant constants, and any necessary macros and type definitions. The reference section will tell you the header file to use for each function.

Some header files include other; for example *conio.h* includes the *window.h* file. Strictly speaking, if you include *conio.h* you need not include *window.h*. However no harm can result from doing so, since each header file is bracketed by preprocessor statements which prevent it being included twice. If you are in any doubt, include the header file specified in the reference section entry to the function you intend to use.

### **Locating your header files**

There is no need to specify a path in the file name and we advise against doing so. The TopSpeed environment and project system will find the header

file for you provided only that you specify the directory containing files with a '.h' extension when you set up the redirection file. The TopSpeed Environment comes to you with the redirection file and directory structure set up, so you should not need to take any special steps unless you have customized the directory structure to your needs.

If, however, you do relocate the header files supplied with the library, you should change the redirection file as explained in the User's Guide so that the compiler can find them.

If you specify a path it will be ignored unless you use the alternate form of the #include macro:

```
#include "C:\mydir\header.h"
```

In this case redirection will be bypassed and header.h will be read from C:\mydir.

**Warning:** Specifying a path for an include file will bypass the dependency-checking used by the TopSpeed project system (See the *TopSpeed User's Guide* for more information). We strongly recommend you use the redirection system rather than specify a path.

Some functions appear in more than one include file in order to accommodate the internal structure of the header files. The reference section specifies the header file which you would normally include in order to use the function concerned, but it may not be the only option. All header files contain macros which guard against double inclusion, so no damage can result if you accidentally include the same file twice.

### **A note for C++ users**

If you are using TopSpeed C++ the rules above apply without change. However there are specific header files for the C++ classes in addition to the normal C header files which are usually given the extension .hpp, rather than .h.

In some cases, for example where C++ declarations differ from C declarations, there are preprocessor directives in the ordinary C header files to provide for C++ compilation. The TopSpeed environment handles this automatically so you need take no action.

### **Programming for portability**

As C has developed, a number of different C environments have emerged, often with their own specialized functions.

A great deal of the confusion to which this gives rise has been removed with the adoption of an ANSI C standard which includes a standard set of library



functions. TopSpeed C was the first PC compiler to receive ANSI certification, which included certification of its complete implementation of the ANSI standard library functions.

We strongly recommend that if you are developing new programs you confine yourself to ANSI library functions. This will give you the maximum possible portability at virtually no sacrifice in efficiency. The only important exception is when you require functionality not catered for by the ANSI functions - for example if you want to use windowing or graphic functions, or low-level operating-system dependent functions.

Nevertheless the TopSpeed C library provides literally hundreds of functions selected to provide compatibility with other specialized environments. This ensures that if you are importing or porting applications from other platforms you can recompile and use the TopSpeed C library easily and without extensive reprogramming.

In some case this means there is considerable overlap. The same function has often been implemented slightly differently by in different environments. Similarly, certain functions are meaningful only with certain hardware (e.g., 80x86 machines) or under certain operating systems (e.g., OS2, UNIX).

The TopSpeed library contains functions defined in these specialized environments.

To assist you the library reference specifies which functions conform to the ANSI standard and which ones are for more specialized environments. The quick reference symbols in the function title summarize this information, and where necessary more detailed information is provided for each function under the heading 'Portability'

### **The following environments are distinguished:**

ANSI	Any function marked ANSI conforms to the proposed ANSI standard and may be expected to compile and perform as such. These functions are shown within a double box in the next chapter.
UNIX/DOS/OS2	This grouping describes common non-ANSI unctions. Such functions may behave differently in the different environments.
DOS/OS2	This grouping describes common functions supported by Turbo C, Microsoft C, etc.
80x86	This grouping describes functions specific to a particular processor's architecture Æ namely, the Intel 80x86 family.

IBM PC	This grouping describes functions that are specific to a particular BIOS or machine.
TopSpeed	This grouping describes proprietary functions, such as those for the windows, process, and mouse modules.

### **Programming memory models: choosing the pointer variant**

If you are using a standard memory model, you need take no special steps to ensure that the correct library functions will be used. The TopSpeed project system will set up preprocessor directives to ensure the correct declarations are given, and the linker will automatically find the correct calls.

However, if you are doing mixed-model programming, or if you wish to use huge pointers, you may need to take specific steps to ensure you are calling the correct functions. Many functions — for example the memory allocation and string-handling functions — are supplied in near, far and huge form so that you can do this.

### **Pointer conventions in standard functions**

The standard functions are always generic; that is, they will automatically compile and link correctly for the memory model you are using. For example if you use the function `malloc` then in a small model program it will return a near pointer and in a large model it will return a far pointer.

If you are using near pointers in a large model program, then when you assign to such a pointer the compiler will not realize that you require the version of `malloc` which returns a near pointer and you must explicitly invoke it yourself. You should therefore use the special near version of `malloc`, which is called `_nmalloc`, thus:

```
/* example for a large model program */
```

```
char * ordinaryPointer;  
char near * nearPointer;  
ordinaryPointer = malloc(100);  
nearPointer = _nmalloc(100);
```

You can, of course, use typecasts where appropriate; but you must be careful not to cast a far to a near pointer since this is not in general safe. For this reason it is unwise to use far pointers in small or medium models; it is better to go the other way round. There are cases such as Windows programming where this is demanded by the environment. However you should only embark on mixed model programming if you fully understand what you are doing.

Care must also be exercised with other functions taking pointer parameters such as `strcpy`. The far versions supplied are for use when the default is near pointers but far pointers are being used. We recommend that wherever possible you use the generic versions, casting from near to far where appropriate. You should try to not use the default functions with far pointers in a small model program. If you find you are attempting to do this, it almost certainly means you should be using the large or compact model. However in some cases — for example when programming with Microsoft Windows — the combination of small model with far pointers may not be avoidable, in which case you should use far version of the string functions for all far objects.

### **Huge pointers**

A full range of variants is supplied for all functions affected by using huge pointers: in other words, all functions that might have to deal with huge objects — objects greater than 64K, or for other reasons cross a segment boundary.

With huge objects the segment part of the pointer must be updated during pointer arithmetic if the offset overflows. For TopSpeed compilers there is no difference in the way huge and far pointers are actually stored, but they are handled differently in pointer arithmetic. There are some functions such as `farmalloc` which are supplied for compatibility reasons and which return far pointers to huge objects. The best thing is to use the correct huge variant, unless you are obliged to do otherwise because you are porting from a platform which does not support it.

The following functions will automatically compile and link correctly if huge pointers are specified globally by defining `_HUGE` before including your header files:

`memcpy`, `memccpy`, `memset`, `memcmp`, `memicmp`, `memchr`, `bsearch`, `lsearch`, `lfind`, `qsort`, `fread`, `fwrite`, `malloc`, `realloc`, `free`

If huge pointers are not specified globally, the following functions may be called explicitly.

`hmemcpy`, `hmemccpy`, `hmemset`, `hmemcmp`, `hmemicmp`, `hmemchr`, `halloc`, `hrealloc`, `hfree`

Arguments for these functions are identical to the standard versions with the exception of `halloc` and `hrealloc`:

Any function may be used with a huge data object as long as segment aligned objects are passed to a function. This may be achieved by using the global constant `_hugeshift` to increment the huge pointer:

```
extern unsigned _hugeshift;

void huge *buffer;
void huge *ptr;

buffer=malloc(0x20000);
ptr=buffer;

read(fh, ptr, 0x8000);
read(fh, ptr+0x8000, 0x8000);
ptr+=_hugeshift;
read(fh, ptr, 0x8000);
```

For further information about on memory models consult your TopSpeed Developer's Guide or TopSpeed TechKit.

### **Choosing the right precision: long double variants**

ANSI C provides for a new type, long double. This provides a slightly higher precision and a substantially larger mantissa — see the TopSpeed C language reference manual for details. It is used for large numbers (up to  $10^{4096}$ ), and also helps to avoid loss of precision in floating point expressions.

If you are working with high precision floating point numbers, which are declared in C as long double, you should use the TopSpeed C library long double variants of the standard mathematical and other functions. These are nearly all distinguished by the addition of a final 'l' — for example `cosl` is the long double version of `cos`. They take long double arguments and return long double results wherever appropriate. Otherwise they are the same as their similarly named double variants.

### **Choosing the right library**

If you are using the TopSpeed project system in the normal way you need take no special action. Any functions you use will automatically be found and linked, using information in the header files. More precisely, this will be done if your project uses the `#link` pragma.

In addition all the standard functions, and many others, are contained in the standard TopSpeed library which is always linked. The only exceptions are the TopSpeed Window functions, the TopSpeed Graphics functions, and the TopSpeed clipping window functions

You need take special action only if you are using one of these functions and if your project does not use the `#link` pragma

In this case you should explicitly include a line similar to:

```
#pragma link (RS_BORW.LIB)
```

This instruction tells the linker to use the library containing the small model object code for the Borland Turbo C Window Compatible functions, to be used in a real-mode application.

This would normally be done using macros as follows (See the TopSpeed Developer's Guide for more detail)

```
#pragma link (%O% %M% %C% BORW.LIB)
```

Here %O will expand to the mode (Real or Protected) and %M will expand to the memory model. It is best to use these macros if you want to ensure that the same project file will work if you change the global memory model or the mode.

The third component of the library name depends on the functions you use.

- The *Window* functions are contained in the %O% %M% %C% WIND.LIB libraries;
- The *Graphics* functions are contained in the %O% %M% %C% GRAPH.LIB libraries;
- The *Turbo C window* functions are contained in the %O% %M% %C% BORW.LIB libraries;

## Error and exception handling

---

The TopSpeed environment provides a very convenient error trapping and diagnosis system which leads you quickly to the source of any error or exception condition which causes your program to terminate. This is documented fully in the TopSpeed Developer's Guide. In outline, a special file called ERRORINF.\$\$\$ is generated containing detailed information on the cause and location of the error; using TopSpeed utilities such as the VID debugger you can then interrogate this file to find out what went wrong.

The story does not end here, however, because when a language-specific error occurs the C language and library themselves have to define whether the program should be terminated, and what action should be taken before handing over to the TopSpeed environment's error processor.

### **C error handling**

Error and exception handling in the C language is unfortunately not as developed or integrated as might be desired. The C user does not lose functionality as a result of this, but error handling can on occasions be complex, since there are a number of different error-handling mechanisms. Some of these are language features, some are part of the library and some

are specific only to set groups of library functions such as the floating point math routines and the file handlers.

The most important point to grasp is the distinction between *unrecoverable* (fatal) and *recoverable* errors.

### **Unrecoverable (fatal) and recoverable errors**

An unrecoverable error is one which forces program termination. Generally speaking this will be a condition such as a process corruption error which would make it dangerous or impossible to continue execution.

An unrecoverable error will always invoke the TopSpeed environment procedure described above, after carrying out a number of standardized cleaning up activities described below.

A recoverable error is a condition such as divide by zero, or an even milder condition such as end-of-file, which demands some exceptional action by the program, though the program can continue if the appropriate action is taken.

The C library functions deal with recoverable errors in a number of different ways, depending on the nature of the error. Most important, the user often has the option to establish special-purpose handlers to bypass the library default.

A comprehensive list of error and exception-handling procedures is given immediately below.

### **Error-handling mechanisms in the C language and library**

Errors are handled by one of the following mechanisms:

#### **Program termination and the exit mechanism**

Unless otherwise specified, any nonrecoverable error, and any other error for which no handling procedure has been defined or established, invoke the exit function. This performs the following actions:

- It executes a stack of user-defined exit procedures which have been added by calling either of the functions `atexit` or `onexit`. If neither `atexit` nor `onexit` have been called during the course of the program, nothing is done.
- It invokes a standard cleanup procedure which endeavors to release any allocated memory, flush and close any opened files, delete any temporary files and restore any interrupt vectors which the program has altered. Static object destructors will be called at this point.
- It passes control over to the TopSpeed environment error-reporting

procedure described in the TopSpeed Developer's Guide.

If a further error occurs while any of the above are being done, the exit procedure will immediately hand over to the TopSpeed environment error-reporting mechanism without attempting any further cleaning up. This is essential to avoid infinite recursive calls to exit.

### **Recoverable errors**

Unless otherwise stated the default procedure for any recoverable error is terminate execution using the exit procedure described above.

There are a number of conditions for which the user may redefine the way recoverable errors are dealt with. The signal function lets you establish handlers which will be invoked when any of the following conditions occur:

SIGINT	interrupt - corresponds to DOS 3.x int 23H
SIGILL	illegal opcode.
SIGFPE	floating point error.
SIGSEGV	segment violation.
SIGTERM	Software termination signal from kill.
SIGABRT	abnormal termination triggered by abort call.

The constants SIGINT, etc are defined in file signal.h.

The most common use of this mechanism is to establish an exception handler for conditions such as divide by zero.

An important point to note is that the control-break mechanism, which by default passes through signal, can be intercepted by calls to functions such as ctrlbrk and EnableBreakCheck. If this is done the signal mechanism is bypassed.

### **Mathematical functions**

A number of library functions have their own special error-handling procedures. In such cases these are documented under the function description. Most mathematical functions invoke a special function called matherr. If this returns zero — the default — the calling function (where the error occurred) sets errno to the appropriate code and then returns an error value. If matherr returns a nonzero value errno is not set and instead of a default error the calling function will return the value supplied by matherr.

Since `matherr` is declared globally as a pointer to a function, you can redefine the response to mathematical errors by reassigning `matherr` to a user-defined function.

## **File handling functions**

In common with many other library functions the file handling functions deal with most recoverable exceptions through their return value. In addition there is a status variable defined in the `FILE` structure which lets you interrogate a file and find out if an error has occurred while processing it; for example, to find out if an end of file has been reached you call the `feof` function. The `ferror` function will report on a file's error status.

## **Other library functions**

Many library functions report abnormal or error conditions through their return value — for example, most of the memory allocation functions report an error in this way.

In addition files may set the global variable `errno` (see below).

In the event of a low-level error while executing a library function — for example if a text window module encounters a pointer to a non-existent window — the function will generally invoke `exit`.

You should consult the documentation of any function you use to find out exactly what it does if an exception is encountered.

## **The `errno` variable**

There is a global variable `errno` which can be set by library functions — for example the memory allocation functions — in the event of a recoverable exception. The program can interrogate `errno` either directly or via the functions `strerror` and `perror` which give access to a short description of the error's meaning.

A list of possible values of `errno` generated by TopSpeed C library functions is given in Appendix B on page 540

## **Trapping runtime checks**

A number of standard checks are carried out by TopSpeed C programs unless these are turned off by means of the relevant pragma — for example the 'index out of range' check for array access.

The default action in the event of an exception is to invoke the `exit` mechanism described above. However the user can override this default by reassigning the global variable `CnsHandler`, which defines a function that will be invoked when a runtime check encounters an error. The errors which



can be intercepted in this way is given in the description of the CnsHandler pointer variable.

### **Process termination under OS2**

The normal exit procedure is followed except in the case of an exception such as an segment over-run. In this case only user specified termination procedures are executed. Code should be limited since a recursive error will prevent OS2 from killing the process.

By default buffered streams will not be flushed and static C++ destructors will not be called.

## **OS2 considerations**

- Only three predefined streams and handles are available:  
**stdin, stdout and stderr.**
- The global variable `_osmode` indicates the operating mode:
  - 1   protected mode
  - 0   indicates real mode.
- See the next section on multi-thread programming and the previous section on termination under OS2.

## **Multi-thread considerations**

### **OS2 Multi-thread programming**

Under OS2 three layers of multi-thread programming exist.

The top layer is the JPI process module. All library modules and floating point are supported, and the resultant programs are portable to DOS.

The middle layer uses `_beginthread` and `_endthread`. All library modules and floating point are supported, but portability to DOS is lost.

Using this interface it is possible to determine the thread number of a child thread.

The lowest layer is the OS2 API. Using the OS2 `DosCreateThread` system call does not initialize the library, floating point emulator and stack checking

mechanism. Only those library functions that do not require locking or floating point may be used — for example string functions.

OS2 multithread applications should take the following points into account:

- All functions except DOS specific ones can be used without restrictions in a single thread process running under OS2. Some small differences in performance do exist, and are documented under the appropriate library function description.

### **General multi-thread considerations**

- All user C source and the library C sources must be compiled in the MT (multithread) model.
- OS2 Threads must be begun and terminated either using the library functions `_dosbeginthread` and `_dosendthread`, or by using the JPI process module. However DOS programs must use the process module.
- Access to I/O streams is controlled by semaphores. Therefore separate threads may access the same stream using the following functions:

`gets, puts, fgets, fputs, fread, fwrite, scanf, printf`

- Care should be taken in the use of `fflush`, `fseek` and `ftell`.

Near heap functions may not be used in DLL modules.

### **Single Byte Stream Access Functions**

These include `getc`, `fgetc`, `getchar`, `fgetchar`, `putc`, `fputc`, `putchar`, as well as `ungetc` and `fputchar`, and are not controlled by semaphore, leaving the user to control access to a particular stream, if necessary.

### **Functions that are not re-entrant**

- The archaic functions `ecvt` and `fcvt` are not re-entrant. The non-ANSI function `_seterrno` is not re-entrant
- The simple Turbo C window module is not re-entrant.
- DOS is not re-entrant, and locking should be used.

# CHAPTER 2

## A Guide to the C library functions

This chapter summarizes the functions provided with the TopSpeed C library. The information in this chapter qualifies the information given in the Library Reference section ( Chapter ), providing a guide to their selection and purpose.

### Input-output

---

Standard input-output uses stream variables, a construct closely associated with the C language since its inception and now incorporated into the ANSI standard. A C stream is an object of type FILE, which in turn is a structure defined in the stdio.h header file. A set of standard read and write functions is defined which take a FILE parameter. A FILE can be a standard stream (see below) or can be associated with a disk file or device. In addition the TopSpeed Library provides special input-output functions to redirect input and output to a window. These are described separately in the section dealing with TopSpeed Text windows, described later in this chapter. A FILE variable may also be associated with a unique integer *handle* which is used by a number of alternative (non-ANSI) input-output functions, commonly known as *low-level* functions (see below: 'High and low-level input-output').

#### Streams

##### Standard streams

The library provides the standard C streams:

stdout	The standard output - usually the screen
stdin	The standard input - usually the console
stderr	An error stream which remains linked to the console even when stdout is redirected.
stdaux	Standard auxiliary stream
stdprn	Standard printer stream

**Note** stdaux and stdprn are not available under OS2

These are assigned low-level handles of 0-4 respectively.

It also provides functions which always write to stdout, such as printf, or always read from stdin, such as scanf.

## **File streams**

Functions such as `fprintf` and `fscanf` take an arbitrary `FILE` as parameter. A `FILE` can be connected to a disk file via calls such as `fopen`.

## **Incore input-output**

Functions such as `sprintf` and `sscanf` look like standard input-output, but actually read and write directly from an area of memory.

## **High and low-level input-output**

Two main groups of input-output functions are traditionally associated with C libraries:

- The standard or *high-level* functions, which work with the C language standard streams and so offer redirection, and which supply buffering.
- *Low-level* functions which work at the operating system level, which are generally unbuffered

A reasonably comprehensive set of standard functions have now been incorporated into the ANSI standard, and their use is recommended if you want to ensure your programs are portable.

There are very few circumstances under which you would now want to use any other functions, but there are a number of older programs which still include low-level calls. In particular the UNIX world has a set of low-level input-output calls which were at one time in quite widespread use. Thus if you are porting from a UNIX application you may have need of them.

The TopSpeed library includes most commonly-used low-level calls for compatibility reasons. However high level calls are implemented efficiently and there is no reason to use low-level calls for new programs.

## **Standard input-output**

The TopSpeed C function library includes the following high-level stream I/O functions:

### **File open and close**

<code>fclose</code>	Closes a stream
<code>fcloseall</code>	Closes all open streams;
<code>fopen</code>	Opens a stream
<code>freopen</code>	Reassigns a <code>FILE</code> pointer

**File read**

<code>fgetc</code>	Reads a character from a stream
<code>fgetchar</code>	Reads a character from stdin
<code>fgets</code>	Reads a string from a stream
<code>fread</code>	Reads unformatted from a stream
<code>hfreadd</code>	Huge variant of <code>fread</code>
<code>fscanf</code>	Reads formatted data from a stream
<code>getc</code>	Reads a character from a stream
<code>getchar</code>	Reads a character from stdin
<code>gets</code>	Reads a line from stdin
<code>getw</code>	Reads a binary int from a stream
<code>scanf</code>	Formatted read from stdin
<code>sscanf</code>	Reads formatted data from string
<code>ungetc</code>	Puts a character back into the buffer
<code>vfscanf</code>	Like <code>fscanf</code> but processes pointer to argument list
<code>vscanf</code>	Like <code>scanf</code> but processes pointer to argument list
<code>vsscanf</code>	Like <code>sscanf</code> but processes pointer to argument list

**File write**

<code>fprintf</code>	Writes formatted data to a stream
<code>fputc</code>	Writes a character to a stream
<code>fputchar</code>	Writes a character to stdout
<code>fputs</code>	Writes a string to a stream
<code>fwrite</code>	Writes unformatted data to a stream
<code>hfwrite</code>	Huge variant of <code>fwrite</code>
<code>printf</code>	Writes formatted data to stdout
<code>putc</code>	Writes a character to a stream
<code>putchar</code>	Writes a character to stdout
<code>puts</code>	Writes a line to a stream
<code>putw</code>	Writes a binary int to a stream
<code>sprintf</code>	Writes formatted data to string

<code>vfprintf</code>	Writes formatted data to a stream
<code>vprintf</code>	Writes formatted data to stdout
<code>vsprintf</code>	Writes formatted data to a string

### **Position functions**

<code>fgetpos</code>	Gets a stream position indicator
<code>fsetpos</code>	Sets stream position indicator
<code>fseek</code>	Repositions FILE pointer
<code>ftell</code>	Gets current FILE pointer position
<code>rewind</code>	Repositions FILE pointer to stream start

### **File handle interface**

<code>fileno</code>	Gets stream file handle
<code>fdopen</code>	Opens a stream using its handle
<code>open</code>	opens stream and return handle

### **Error handling**

<code>clearerr</code>	Clears the error indicator for a stream
<code>feof</code>	Tests for end of file
<code>ferror</code>	Tests for error on a stream
<code>perror</code>	Maps error number into system message
<code>strerror</code>	convert system error number to a message

### **Buffer handling and flushing**

<code>fflush</code>	Flushes a stream
<code>flushall</code>	Flushes all streams
<code>setbuf</code>	Specify buffer for stream use
<code>setvbuf</code>	Specify buffer and buffering mode

### **Other file handling**

<code>remove</code>	Delete a file
<code>rename</code>	Rename a file
<code>tempnam</code>	Generates a temporary file name
<code>tmpfile</code>	Creates a temporary file
<code>tmpnam</code>	Generates a temporary file name
<code>rmtmp</code>	Removes temporary files created by tmpfile

Stream I/O functions treat files, devices and data items as streams of characters. When a file is opened, it is allocated a stream and associated with a structure of type `FILE` containing information about that stream. Streams are normally buffered and provide efficient I/O because data is only read or written in large blocks.

### **Lower Level I/O Functions**

This category includes two main types of function:

- functions using an integer *handle* to access a file or stream instead of the ANSI `FILE` parameter described above;
- all functions which deal with files or with input-output in an operating system-specific manner. For example the access function includes a mode parameter which relates only to the DOS file access conventions.

The functions are as follows:

#### **UNIX compatible low level functions**

<code>access</code>	Checks whether the file exists
<code>chmod</code>	Set file's access mode
<code>chsize</code>	Change length of specified file
<code>close</code>	Close file
<code>creat</code>	Create file
<code>createmp</code>	Create temporary file
<code>dup</code>	Duplicate a file handle
<code>up2</code>	Force two handles to refer to same file
<code>eof</code>	Check end of file
<code>fstat</code>	file status information
<code>filelength</code>	Return length of file
<code>getftime</code>	Return file time and date information
<code>ioctl</code>	Invoke DOS interrupt 44H (file handling)
<code>isatty</code>	Ascertains if a handle refers to character device
<code>locking</code>	Lock or unlock specified number of bytes
<code>lseek</code>	Position file pointer
<code>mktemp</code>	Generate temporary file name
<code>open</code>	Open a file and return handle
<code>read</code>	Read n bytes from file

setftime	Set file time and date information
setmode	Set file translation mode
tell	Return current file position
umask	?? not in reference manual
unlink	Delete the specified file
write	Write n bytes to file

### **DOS low level file functions**

_chmod	Set or get file access mode
_close	Close file
_creat	Create file and return handle
_dup	Duplicate a file handle
_dup2	Force two handles to refer to same file
_eof	Check end of file
_open	Open file and return handle
_read	Read n bytes from file
_write	Write n bytes to file

### **Translation Modes**

There are two possible *modes* of file use: *text* and *binary*.

#### **Text Mode**

When a file is opened in text mode, carriage return/linefeed pairs are translated to a single linefeed on input. On output the single linefeed is translated to a carriage return/linefeed pair. The ^Z character is treated as an end-of-file indicator.

#### **Binary Mode**

When a file is opened in binary mode, no such translations are performed. The ^Z (end of file) character has no special meaning in binary mode.

The default translation mode for a file is controlled by the global variable `_fmode`. The default setting for `_fmode` is `O_TEXT`.

The `stdin`, `stdout` and `stderr` streams are opened in text mode. The `stdaux` and `stdprn` streams are opened in binary mode.



## **Buffering**

### **Stream Buffering**

A newly opened stream is automatically allocated a buffer of BUFSIZ, defined in stdio.h.

The predefined streams stdin, stdout and stderr are buffered. The stdaux and stdprn streams are not. Any unbuffered stream will be temporarily allocated a buffer by the printf and scanf families of functions.

### **Line Buffering**

A line buffered stream is flushed when a linefeed character is encountered by fputc or fputchar.

## **File protection**

Streams may be opened as *read-only*, *write-only* or *read-write*. Any attempt to perform a forbidden I/O operation will result in an error.

## **Stream error and end of file conditions**

A variety of conditions can give rise to stream errors or *exception conditions*:

- An attempt to carry out a forbidden or illogical operation such as writing to a read-only file.
- A device or mechanical failure;
- Physical end of file;

A read/write stream will not allow a change of flow without an intervening call to fseek or rewind, or without encountering an EOF on a read operation. An attempt to read from a stream immediately after writing to it will cause an error return.

The physical end of file (i.e. the ^Z character in text mode) will cause the EOF flag \_F\_EOF to be set. All further calls will cause an error return until rewind, fseek or ungetc are called.

An End-of-File condition can be detected by the routine feof.

An attempt to put the value EOF (-1) onto a stream will cause an error return.

## Treatment of exception conditions

In most cases when an exception condition arises on a stream, input-output functions will return a code representing the error.

An error condition can also be detected by the routine `ferror`. The global variable `errno` contains the error code.

After an error, the state of the stream is undefined. A call to `clearerr` clears the error flag although the state of the stream will still be undefined.

## Initial State of Streams

An unused or reset stream is denoted by the flag `_F_RST`. A stream in this state will accept a buffer reassignment by `setbuf` or `setvbuf`.

## String handling functions

The C library string handling functions fall into two major categories. One set deals with C conventional *null-terminated* strings. The other set deals with *memory-buffer* arrays. These are held as character arrays but without a null terminator. The relevant functions therefore take an integer parameter saying how long the string is. The program is responsible for keeping track of string lengths.

### Null-terminated (C) string handling functions

These functions deal with move and compare operations on null-terminated strings.

#### Copy and set

<code>strcpy</code>	Copy string, return address of destination final null
<code>_fstpcpy</code>	Far version of <code>strcpy</code>
<code>strcpy</code>	Copy one string to another, return destination address
<code>_fstrcpy</code>	Far version of <code>strcpy</code>
<code>strcoll</code>	Same as <code>strcpy</code> - provided for ANSI compatibility
<code>_fstrcoll</code>	Far version of <code>strcoll</code> .
<code>strxfrm</code>	Same as <code>strncpy</code> - provided for ANSI compatibility
<code>_fstrxfrm</code>	Far version of <code>strxfrm</code> .
<code>strncpy</code>	Copy <i>n</i> characters
<code>_fstrncpy</code>	Far version of <code>strncpy</code> .
<code>strnset</code>	Set <i>n</i> characters to a given character

<code>_fstrnset</code>	Far version of <code>strnset</code> .
<code>strset</code>	Set string to a given character
<code>_fstrset</code>	Set far string to a given character

### **Concatenate (append)**

<code>strcat</code>	Concatenate two strings
<code>_fstrcat</code>	Concatenate two far strings
<code>strncat</code>	Concatenate n characters
<code>_fstrncat</code>	Far version of <code>strncat</code> .

### **Compare**

<code>strcmp</code>	Compare two strings
<code>_fstrcmp</code>	Compare two far strings
<code>strcmpi</code>	Compare regardless of case (macro version)
<code>stricmp</code>	Compare regardless of case (function version)
<code>_fstricmp</code>	Far version of <code>stricmp</code> .
<code>strnicmp</code>	Compare n characters regardless of case
<code>_fstrnicmp</code>	Far version of <code>strnicmp</code> .
<code>strncmp</code>	Compare n characters
<code>_fstrncmp</code>	Far version of <code>strncmp</code> .

### **Search**

<code>strchr</code>	Find character in string
<code>_fstrchr</code>	Far version of <code>strchr</code>
<code>strcspn</code>	Finds one of a set of characters in string
<code>_fstrcspn</code>	Far version of <code>strcspn</code> .
<code>strpbrk</code>	Find character from one string in another
<code>_fstrpbrk</code>	Far version of <code>strpbrk</code> .
<code>strrchr</code>	Find last occurrence of character
<code>_fstrrchr</code>	Far version of <code>strrchr</code> .
<code>strspn</code>	Find first character with no match in given character set
<code>_fstrspn</code>	Far version of <code>strspn</code>
<code>strstr</code>	Find first occurrence of substring in a string
<code>_fstrstr</code>	Far version of <code>strstr</code> .
<code>strtok</code>	Find next token in string

`_fstrtok` Far version of `strtok`.

### Conversion

<code>atoi</code>	Convert string to integer
<code>atol</code>	Convert string to long
<code>atoul</code>	Convert string to unsigned long
<code>strtol</code>	Convert string to long
<code>strtoul</code>	Convert string to unsigned long
<code>itoa</code>	Convert integer to string
<code>ltoa</code>	Convert long to string
<code>ultoa</code>	Convert unsigned long to string
<code>atof,atofl;</code>	Convert string to floating point
<code>strtod,strtodl</code>	Convert string to double floating point
<code>strlwr</code>	Convert to lower case
<code>_fstrlwr</code>	Far version of <code>strlwr</code> .
<code>strupr</code>	Convert to upper case
<code>_fstrupr</code>	Far version of <code>strupr</code> .
<code>fcvt</code>	Convert floating point to character string
<code>gcvt,gcvtl</code>	Convert double to character string
<code>ecvt</code>	Convert double to character string, specifying decimal and sign information

### Miscellaneous

<code>strlen</code>	Return string length
<code>_fstrlen</code>	Return far string length
<code>strdup</code>	Duplicate string
<code>_nstrdup</code>	Near version of <code>strdup</code> (use when far is default)
<code>_fstrdup</code>	Far version of <code>strdup</code> .
<code>strrev</code>	Reverse string
<code>_fstrrev</code>	Far version of <code>strrev</code> .

### Memory buffer functions

These functions deal with move and compare operations in buffers that are not null-terminated. They all therefore take a parameter of type `size_t`, which is unsigned and defines the number of characters to be moved, set or copied.

**Character handling functions**

memcpy	Copy n characters
_fmemcpy	Copy n characters between far strings
hmemcpy	Copy n characters between huge strings
movedata	Copy n characters from different segments
movmem	Copy characters, safeguarding against overlapping
memccpy	Copy until a given character
_fmemccpy	Copy until a given character between far strings
hmemccpy	Copy until a given character between huge strings
memset	Set n characters to a given character
_fmemset	Set n characters to a given character in a far object
hmemset	Set n characters to a given character in a huge object
setmem	Same as memset; for compatibility with earlier releases
emmove	Move n characters
emcmp	Compare n characters
fmemcmp	Compare n characters in far strings
memcmp	Compare n characters in huge strings
memicmp	Compare n characters ignoring case
_fmemicmp	Compare n characters ignoring case, in far strings
hmemicmp	Compare n characters ignoring case in huge strings
memchr	Find a character
_fmemchr	Find a character in a far object
hmemchr	Find a character in a huge object

**Word handling functions**

memwcpy	Copy n words
memwmove	Copy n words, safeguarding against overlapping
memwset	Set n words
swab	Copy words exchanging byte order

## Mathematical functions

---

The ANSI standard specifies a wide range of mathematical functions, which take double parameters and return double results except where otherwise specified. All have their conventional meanings.

### long double variants

TopSpeed C supplies long double (high precision) variants of all the standard math functions, as explained in ‘choosing the right precision’ on page .

The TopSpeed C mathematical functions are:

### Transcendental

#### Trigonometric

cos	Cosine
sin,sinl	Sine
tan,tanl	Tangent
hypot,hypotl	Hypotenuse of triangle given two sides

#### Hyperbolic

tanh,tanhl	Hyperbolic tangent
sinh,sinhl	Hyperbolic sine
cosh,coshl	Hyperbolic cosine

#### Inverse trigonometric

acos,acosl	Arc cosine
asin,asinl	Arc sine
atan,atanl	Arc tangent
atan2,atan2l	Arc tangent of a quotient of two numbers

### Power, exponential, logarithm etc

log,logl	Natural logarithm
log10,log10l	Logarithm to base 10
exp,expl	Exponential
pow,powl	x to the power y
pow10	10 to the power y
sqrt,sqrtl	Square root

poly,polyl                      Calculate value of specified polynomial

### **Absolute values and rounding**

abs                                Integer absolute value  
 cabs,cabsl                      Absolute value of complex number  
 fabs,fabsl                      Floating point absolute value  
 labs,labsl                      Long integer absolute value  
 ceil,ceil                        Round up to integer  
 floor,floorl                      Round down to integer

### **Division and remainder**

div                                Integer divide and extract remainder  
 ldiv                               Long integer divide and extract remainder  
 modf,modfl                      Break into integer and fractional parts  
 fmod,fmodl                      Floating point remainder after division

### **Bitwise rotation operations**

\_rotr                              Bitwise rotate left, long integer  
 \_rotr                              Bitwise rotate right, long integer  
 \_rotr                              Bitwise rotate left  
 \_rotr                              Bitwise rotate right

### **Miscellaneous**

max                                Maximum of two numbers (macro)  
 min                                Minimum of two numbers (macro)  
 frexp,frexp1                      Calculate mantissa and exponent (inverse of ldexp)  
 ldexp,ldexp1                      Calculate from mantissa and exponent (inverse of frexp)  
 rand                                Return pseudorandom integer  
 random                            Returns pseudorandom number in a specified range  
 randomize                        Set pseudorandom seed with system time  
 srand                               Set pseudorandom seed with specified number

### **Floating point coprocessor handler**

\_clear87                        Clear 80x87 status word and return its former value  
 \_control87                      Set 80x87 control word and return its former value

<code>_fpreset</code>	Reinitialize 80x87 processor
<code>_status87</code>	Get 80x87 status word

### **Error handler**

<code>matherr</code>	redefinable math function error handler
----------------------	---

## **Ctype (Character) test and conversion functions**

---

### **Character test functions**

These functions are used for testing character attributes. Each returns a nonzero value (true) if the character passed as parameter matches the given attribute, and zero otherwise

<code>isalnum</code>	Alphanumeric
<code>isalpha</code>	Alphabetic
<code>isascii</code>	ASCII
<code>isctrl</code>	Control character
<code>isdigit</code>	Numerics
<code>isgraph</code>	Printable except space
<code>islower</code>	Lower case
<code>isprint</code>	Printable including space
<code>ispunct</code>	Punctuation character
<code>isspace</code>	Whitespace
<code>isupper</code>	Upper case
<code>isxdigit</code>	Hex digit

### **Character conversion functions**

<code>toascii</code>	Convert integer to ascii by clearing upper bits
<code>tolower</code>	Test and convert if uppercase
<code>_tolower</code>	Convert to lowercase
<code>toupper</code>	Test and convert if lowercase
<code>_toupper</code>	Convert to upper case

### **Multibyte characters**

<code>mblen</code>	Determine number of bytes in multibyte character
<code>mbtowc</code>	Convert multibyte character to a code



<code>mbstowcs</code>	Convert multibyte character sequence to code sequence
<code>wctomb</code>	Convert code to multibyte character
<code>wcstombs</code>	Convert code sequence to multibyte character sequence

## Memory allocation

---

Memory allocation functions provide free memory for new data objects on the heap of unallocated memory. Allocated space remains reserved until explicitly freed with a call to the relevant free function.

Previously allocated memory may be expanded by calling a reallocation function, which may have to move the allocated space to another part of memory if contiguous space is not available. The low level expansion functions such as `_expand` attempt to expand an object without moving it, reporting failure if insufficient contiguous space is available.

Functions are also provided to find out how much memory is available and to test the integrity of the heap to see if it has become corrupted.

Some attention needs to be paid to calling the correct function if mixed memory models are used (See ‘Memory Models’ in the TopSpeed Developer’s Guide) or huge objects are allocated (see the section on huge pointers in this manual)

In small and medium models the heap is located in the default data segment and can be referenced with a near pointer. In all other memory models the heap comprises multiple far data segments requiring a far pointer.

The default memory allocation functions (such as `malloc`) use whichever heap is the default for the memory model in use.

Therefore if you use far pointers in a small or medium model or near pointers in any other model, the heap specific functions `_fmalloc` and `_nmalloc` should be used.

When allocating from the far heap,

- all available RAM can be allocated.
- blocks larger than 64K can be allocated.
- far pointers must be used to access blocks allocated with functions such as `farcalloc`.

## Huge data objects

---

Because of the segmented addressing structure of the 80x86 family of microprocessors, special care has to be taken when using pointers which address objects larger than 64K. If such a pointer oversteps a segment boundary, its segment register should be incremented. (see the Developer's Guide for further information). For this reason pointers to such objects require special attention from the compiler and should be declared as huge. Matters are complicated further by the fact that some C environments do not provide huge pointers but nevertheless provide functions which can handle huge objects.

Because the TopSpeed C library provides a large degree of compatibility with other libraries there is therefore a large degree of overlap in some of the memory allocation functions provided. For example farmalloc is a Turbo C function which allocates a huge object but returns a far pointer; its actual operation is otherwise identical to hmalloc.

### Allocation

calloc	Allocate items from default heap
_ncalloc	Allocate items from near heap
_fcalloc	Allocate far item from far heap
farcalloc	Allocate huge item from far heap, return far pointer
malloc	Allocate characters from default heap
_nmalloc	Allocate characters from near heap
_fmalloc	Allocate far item from far heap, return far pointer
farmalloc	Allocate huge item from far heap, return far pointer
halloc	Allocate huge item from far heap, return huge pointer

### Deallocation

free	Free space in default heap
_ffree	Free space in far heap
farfree	Free huge space in default heap referenced by far pointer
_nfree	Free space in near heap
hfree	Free huge space in default heap referenced by huge pointer

**Available space**

coreleft	Space available in default heap
nearcoreleft	Space available in near heap
farcoreleft	Space available in far heap
_freect	Space available for multiple allocations in default heap
_memavl	Number of bytes available on near heap
_memmax	Size of largest object that can be allocated

**Reallocation**

realloc	Reallocate in default heap
_nrealloc	Reallocate in near heap
_frealloc	Reallocate far object in far heap
farrealloc	Reallocate huge object in far heap, return far pointer
hrealloc	Reallocate huge object in far heap, return huge pointer
_expand	Try to expand object where it is in default heap
_nexpand	Try to expand object where it is in near heap
_fexpand	Try to expand object where it is in far heap

**Heap information**

_heapwalk	Check default heap object by object
_nheapwalk	Check near heap object by object
_fheapwalk	Check far heap object by object
_nmsize	Block size allocated when memory was requested
_fmsize	Block size allocated when memory was requested
_msize	Block size allocated when memory was requested

**Heap integrity**

_heapchk	Check integrity of default heap
_nheapchk	Check integrity of near heap
_fheapchk	Check integrity of far heap
_heapset	Check and fill integrity of default heap
_nheapset	Check and fill integrity of near heap
_fheapset	Check and fill integrity of far heap

### **Stack size**

stackavail	Return number of bytes available in current stack.
------------	--

## **Search and sort functions**

---

### **Search functions**

lfind	Linear search for key
hlfind	Huge variant of lfind
lsearch	Linear search for key, add if not found
hlsearch	Huge variant of lsearch
bsearch	Binary search on sorted array
hbsearch	Huge variant of bsearch

### **Sort functions**

qsort	Sort array (using quicksort algorithm)
hqsort	Huge variant of qsort

## **Time and date functions**

---

### **System date and time**

getdate	Get system date
setdate	Set system date
gettime	Get system time
settime	Set system time
_strdate	Get system date as string
_strtime	Get system time as string

### **File timestamp**

get_ftime	Get file-modified time
ftime	Set file-modified time

### **Conversion**

time	Get current system time as long integer
stime	Set system time from long integer

ctime	Convert from long integer time to string
asctime	Convert from DOS time structure to string
gmtime	Convert from string to DOS time structure
strftime	Convert from DOS time structure to string, formatted
localtime	Convert string to time structure with local correction
ftime	Local time function provided for Microsoft compatibility
mktime	Convert time to calendar value

### **DOS time and date handling**

_dos_getdate	Get system date
_dos_gettime	Get current system time
_dos_setftime	Set date and time file was last written
_dos_settime	Set system time
_dos_setdate	Set system date

### **Miscellaneous**

clock	Return elapsed CPU time for a process
difftime	Difference between two times
tzset	Set time from environment

## **Standard process control functions**

---

### **Process control functions**

Process functions, most of them ANSI standard, provide an interface to DOS and OS2 process control.

### **System command**

system Execute system command, passed as ASCII string

### **Signal functions**

Signal functions provide an interface for handling exceptions and error conditions. The signal function establishes a handler function which is called when a hardware or software generated exception occurs. The raise function will create a software-generated exception condition

signal	Establish an exception handler
raise	Generate a software exception

### **Child process execute**

The following functions all execute a child process which overlays the calling process. Each variant specifies a different use of the passed parameters.

execl  
execle  
execlp  
execlpe  
execv  
execve  
execvp  
execvpe

### **Child process spawn**

These functions spawn a child process which may or may not overlay the calling process depending on the parameters passed. Each variant specifies a different use of the passed parameters.

spawnl  
spawnle  
spawnlp  
spawnlpe  
spawnv  
spawnve  
spawnvp  
spawnvpe

### **Exit handling**

atexit	Add an exit handler (ANSI)
onexit	Identical to atexit
exit	Terminate process, calling user-defined exit handlers
_exit	Terminate process without calling user-defined exit handlers
abort	Terminate and signal abnormal exit

assert

Conditional abort

## The TopSpeed process and task control module

---

The TopSpeed process module provides multi-thread execution within a single process, under both OS2 and DOS. Each thread is allocated a stack workspace and given a priority. The workspace must be large enough to accommodate the stack requirements of the thread. A minimum of 2000 bytes is recommended.

A process always has one thread, thread 1, which is the process itself when execution is begun. Other threads may be started by a call to StartProcess. Before any new threads are executed, the scheduler must be started with a call to StartScheduler.

Threads may communicate by *signals*, implemented as semaphores. A signal is initialized by a call to Init. A thread may then wait for a signal by calling WAIT. A signal may be sent to any waiting thread by a call to SEND.

Threads may protect non re-entrant code or global data objects by locking a section of code. Calls to Lock may be nested, but a corresponding call to Unlock must always be made to ensure that no deadlocks occur. Under DOS, calls to functions that enter the operating system or BIOS should be protected by locks.

If a library functions is affected by multiple thread processes this fact is documented where that function is described in the reference chapter. But you can assume that all standard process resources such as stream I/O, console I/O and memory allocation are re-entrant and may be used without restriction in a multiple thread process.

The TopSpeed function library includes the following functions from the process module:

awaited	Ask if a process is waiting for a signal
delay	Generate a delay
getpid	Return the process ID
_getTID	Return thread number of currently executing thread.
init	Initialize a signal
Lock	Seize processor until Unlock
notify	Conditionally schedule WAIT-bound task
ProcessDelay	Same as delay with different name for linking with PASCAL programs

SEND	Unconditionally restart WAIT-bound task
StartProcess	Create a new process
StartScheduler	Start the scheduler
StopProcess	Stop current process
StopScheduler	Stop the scheduler
Unlock	Relinquish processor use seized by lock
WAIT	Sleep until matching SEND occurs

### **reading a Multiple Thread Program**

The Mthread memory model must be used when creating a multiple thread program. This must be selected from the project menu or by editing the project file. This will ensure that the re-entrant library is used. The model uses extra large data and far calls.

The following program illustrates the use of the process functions:

```
#define _JPI_WIN_
#include <process.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

#define WORK_SPACE_SIZE 0x1000

void thread1(void);           /* a thread must start execution */
                              /* with a function of this type */
void thread2(void);

SIGNAL exit_signal;

main()
{
    clrscr();
    StartScheduler();
    Init(&exit_signal);       /* initialize signal */
                              /* (must be global) */

                              /* start processes */
    StartProcess(thread1, WORK_SPACE_SIZE, 1);
    StartProcess(thread2, WORK_SPACE_SIZE, 1);
                              /* wait for stop signal */
    WAIT(exit_signal);
        StopScheduler(); /* stop scheduler */
        exit(0);
                              /* exit */
}

void thread1(void)
{
    void *ptr;
    int toggle=0xF;
```



```

gotoxy(0, 24);
cprintf("Thread1 is alive ");
while(!kbhit())
{
    Lock();

    /* two threads are sharing the same window,*/
    /* the full screen. To ensure this thread */
    /* is not re-scheduled before output of    */
    /* character is achieved, the calls to      */
    /* gotoxy and putch are in a locked region */
    gotoxy(18, 24);
    putch(toggle);
    Unlock();

    if(toggle == 0xF)
        toggle='\ ' ;
    else
        toggle=0xF;
    ptr=malloc(120);          /* memory allocation is */
                              /* locked internally */
    memset(ptr, 'C', 120);
    free(ptr);
};
getch();
/* wait for thread 1 to wait */
while(Awaited(exit_signal) == 0);
SEND(exit_signal);
return;
}

void thread2(void)
{
    void *ptr;
    int toggle=0xF;

    gotoxy(0, 0);
    cprintf("Thread2 is alive ");
    while(1)
    {
        Lock();
        gotoxy(18, 0);
        putch(toggle);
        Unlock();
        if(toggle == 0xF)
            toggle='\ ' ;
        else
            toggle=0xF;
        ptr=malloc(120);
        memset(ptr, 'C', 120);
        free(ptr);
    }
    return;
}

```

## Mouse control functions

---

The mouse functions provide an interface to the MicroSoft Mouse BIOS under DOS, and are intended to be used in conjunction with the Microsoft Mouse Programmer's reference. When using OS2, the API mouse functions must be used.

### **DOS Multi-thread considerations**

None of the mouse interface functions provide locking internally, since this would not guarantee the correct execution of a multi-thread process. You must use the functions Lock and Unlock to protect sections of code if more than one thread is attempting to access the mouse.

The TopSpeed C function library includes the following mouse functions:

<code>_ms_cursor</code>	Reveal or hide mouse cursor
<code>_ms_driversize</code>	Get space needed to store mouse driver state
<code>_ms_getmotion</code>	Get number of mickeys since last call
<code>_ms_getpage</code>	Get display page cursor is on
<code>_ms_getpress</code>	Get position and press information
<code>_ms_getrelease</code>	Get position and release information
<code>_ms_getsensitivity</code>	Get speed threshold information
<code>_ms_getstatus</code>	Get position and button status
<code>_ms_lightpen</code>	Turn lightpen emulation on/off
<code>_ms_reset</code>	Reset mouse driver and read status
<code>_ms_restoredriver</code>	Restore state saved by <code>_ms_savedriver</code>
<code>_ms_savedriver</code>	Save state for <code>_ms_savedriver</code> to restore
<code>_ms_setdouble</code>	Define double speed threshold
<code>_ms_setgraphcursor</code>	Define a graphic cursor
<code>_ms_setinterrupt</code>	Install interrupt handler, set call mask
<code>_ms_setmickeys</code>	Set mickey per pixel ratio
<code>_ms_setpage</code>	Set display page cursor appears on
<code>_ms_setposition</code>	Set position of mouse cursor
<code>_ms_setrange</code>	Set cursor range
<code>_ms_setsensitivity</code>	Set speed and double speed threshold
<code>_ms_settextcursor</code>	Define a text mode cursor
<code>_ms_swapinterrupt</code>	Set new handler and save old one

`_ms_updatescreen`

Define screen region for updating

## The TopSpeed window module

---

The TopSpeed window module provides a powerful collection of window management functions. With these functions you can display several virtual screens, or windows, on the physical screen. Multiple windows are treated in a stack-like manner. Two kinds of windows are supported: “normal” windows and palette windows, which are described on page .

### The JPI and Turbo C window modules

There are two window modules: the JPI window module providing multi-thread, multi-window facilities, and the simple Turbo C window module provided for compatibility with Turbo C. Only one window module may be linked with a program. The preprocessor statement

```
#define _CLIP_WIN_
```

forces inclusion of the Turbo C window module. The Turbo C window module is documented in the section on compatibility functions below.

### Generic console input-output

The window package is designed to integrate with the standard console input-output in a natural and obvious way through generic console input-output routines defined in the conio.h header file.

If a window module has been linked, all conio output will be redirected to the current window — the top window or the window selected by the use function (see below).

These are functions like `cprintf`, `cputs` and `putch`. End of line wrap will be managed automatically. Functions `cscanf` and `cgets` may similarly be used to read from a window.

If the window package is not linked, then the normal console I/O output functions send their output to the screen as normal.

If the window module has been linked but no window has been opened, the default window is the full screen. The following simple example will work the same, regardless of whether the window library has been linked.

### **Example**

```
#define _JPI_WIN_
```

```
#include <conio.h>

main()
{
    cprintf("Hello World\n");
    return(0);
}
```

To force the inclusion of the JPI window module function prototypes, types and definitions, the macro `_JPI_WIN_` must be defined before the inclusion of `conio.h`, or `window.h` must be included.

## Window Constants and Types

The following constants and types are defined and used in the window module.

```
/* Cell color attributes */
typedef enum {
    Black,      Blue,      Green,
    Cyan,      Red,      Magenta,
    Brown,      LightGray, DarkGray,
    LightBlue,  LightGreen, LightCyan,
    LightRed,   LightMagenta, Yellow,
    White
} Color ;

/* window definition struct */
typedef struct {
    abscoord X1, Y1, X2, Y2; /* opposite corners*/

    Color
        Foreground,
        Background; /* not Used if Palette */
    int CursorOn; /* is cursor active? */
    int WrapOn; /* is EOL wrap enabled?*/
    int Hidden; /* is window on view? */
    int FrameOn; /* has a frame? */
    framestr FrameDef; /* only Used if frame */
    Color FrameFore, FrameBack
        /* ...only Used if frame and */
        /* not Palette Window */
} windef;

/* Title position type */
typedef enum {
    NoTitle,
    LeftUpperTitle,
    CenterUpperTitle,
    RightUpperTitle,
    LeftLowerTitle,
    CenterLowerTitle,
    RightLowerTitle
} TitleMode;

/* Full window definition */
extern windef fullscreendef; /* default window handle */
extern wintype _fullscreen;
```

A window is defined by the type `windef`, and a variable or constant of this type is used to create a window. In `windef`, `X1`, `Y1` are the coordinates of the upper left corner and `X2`, `Y2` represent the lower right hand corner of the window. If `Hidden` is true, the window will not be displayed until the procedure `putontop` is called. `FrameOn` specifies whether the window has a frame.

A frame is specified by the type `framestr` Æ an array of 9 characters Æ where element 0 denotes the character of the upper left corner, element 1 is the character that denotes the upper bar, etc., as outlined in the declaration above. Two predefined frames are declared by the constants `SINGLEFRAME` and `DOUBLEFRAME`.

A window is created using the function `windowopen`, which returns a handle of type `wintype`. Any subsequent reference to that window is made by using this handle. `wintype` is a pointer to an internal window descriptor, which should be regarded as private to the window module.

The window manager operates with two different kinds of coordinates Æ coordinates relative to a window (`relcoord`) and coordinates relative to the screen (`abscoord`). Coordinate 0,0 is the upper left corner of the screen when using `abscoord`, and coordinate (1,1) is the upper left corner of a window when using `relcoord`.

In what follows, coordinates are relative to the current window unless otherwise specified. 'A point' refers to a coordinate pair `x,y`

### **Palette Windows**

A special kind of window, a palette window, allows you to use several color sets within the same window, and to change these color sets dynamically.

A palette (`PaletteDef`) is 0 to `PALETTEMAX` entries of color sets (`PaletteColorDef`). Each color set specifies a foreground and a background color. The relevant data structures are defined in `window.h`:

```
typedef struct      {
    Color Fore, Back;
} PaletteColorDef ;

typedef PaletteColorDef PaletteDef[PALETTEMAX];

#define NORMALPALETTECOLOR  0
#define FRAMEPALETTECOLOR   1
```

A window is defined in a type `windef` structure. This structure is passed to the `windowopen` function, and a handle to the newly opened window is returned. This handle is passed to any function that does not operate on the current window.

The current output window is either the last window to be opened, or a window whose handle has been passed to the functions `use` or `putontop`.

## **The window functions**

The TopSpeed function library includes the following functions from the JPI window module:

at	Get handle for window at a point, absolute
change	Change window size and position
cleol	Clear from cursor to end of line
clrscr	Clear current window
convertcoords	Convert relative to absolute
cursoroff	Turn cursor off
cursoron	Turn cursor on
delline	Delete line and scroll up
_directwrite	Write to window at a point
gotoxy	Go to a point
hide	Stop displaying a window
info	Get the window definition
insline	Insert a blank line and scroll down
obscuredat	Find out if window is obscured at a point
palettecolor	Get the current window palette
palettecolorused	Find out if a given color is used
paletteopen	Create a new palette window
putbeneath	Put a specified window below another
putontop	Put a specified window on top of the window stack
_rdbufferln	Get a buffer of character/attribute pairs
setframe	Change box attributes and title
setpalette	Set palette of current window and redisplay colors
setpalettecolor	Set palette color used in current window
settitle	Set title of current window
setwrap	Enable/disable automatic wrapping
snapshot	Update window buffer from screen
textbackground	Set text background color
textcolor	Set text foreground color
top	Get current top window
use	Redirect output to specified window

used	Get window last set by use, if there is one
wherex	Return x co-ordinate of current cursor pos
wherey	Return y co-ordinate of current cursor pos
windowclose	Close a specified window and remove it
windowopen	Open a new window on top of window stack
wrbufferln	Write character/attribute pairs to window

### **Generic conio Functions**

The TopSpeed function library includes the following generic console I/O functions:

cgets  
cprintf  
cputs  
cscanf  
getch  
getche  
kbhit  
putch  
ungetch

### **Example**

The following example opens a new window and writes to it.

```

#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD;
wintype W1;
main()

{
    clrscr();
    cursoroff();
    WD.X1 = 10; /* initialize window descriptor */
    WD.Y1 = 2 ;
    WD.X2 = 60;
    WD.Y2 = 8 ;
    WD.Foreground = White ;
    WD.Background = Red ;
    WD.CursorOn   = FALSE ;
    WD.WrapOn     = FALSE ;
    WD.Hidden     = FALSE ;
    WD.FrameOn    = TRUE ;
    strcpy(WD.FrameDef, SINGLEFRAME);
    WD.FrameFore  = White ;
    WD.FrameBack  = WD.Background ;
    W1 = windowopen(&WD) ; /* open window */
    setttitle(W1, " Window 1 ",
        CenterUpperTitle); /* set title */
    cprintf("Hello World\n");
                                /* output to current window */
    getch();
    windowclose(W1);           /* close window */
    clrscr();
    cursoron();
    cprintf("Terminate\n");
                                /* output now goes to */
                                /* full screen window */
    return(0);
}

```

The program WINDEMO.C, which you will find in your EXAMPLES directory, demonstrates a multiple window, multiple thread process.

### **Turbo C compatible windows**

The TopSpeed Turbo C window module is provided for compatibility with Borland's Turbo C compiler. This module is not re-entrant and must not be used with multi-thread programs. For such programs, use the TopSpeed window module.

To force the inclusion of the correct window module function prototypes, types and definitions, the macro `_CLIP_WIN_` must be defined before the inclusion of `conio.h`.



The TopSpeed function library includes the following Turbo C window functions:

clreol;  
clrscr;  
delline;  
gettext;  
gettextinfo;  
gotoxy;  
highvideo;  
insline;  
lowvideo;  
movetext;  
normvideo;  
puttext;  
textattr;  
textbackground;  
textcolor;  
textmode;  
wherex;  
wherey;  
window;

### **Input and Output Using the Turbo C Window Module**

When a program has been linked with the Turbo C window library, the normal console I/O functions automatically send their output to the default or to a user defined window.

Functions `cprintf`, `cputs` and `putch` may be used to write to a window. End of line wrap is managed automatically.

Functions `cscanf` and `cgets` may be used to read from a window.

### **Xample**

If no window has been opened, the default is the full screen. The following example will work the same, whether the Turbo C window library has been linked or not.

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

main()

{
    cprintf("Hello World\n");
    return(0);
}
```

### Example

A more complex example defines a new window and writes to it.

```
#define _CLIP_WIN_
#include <conio.h>
#include <string.h>

main()

{
    clrscr();
    /* define new window */
    window(20, 4, 60, 12);
    /* output to new window */
    cprintf("Hello World");
    getch();
    clrscr();
    /* define new window */
    window(1, 1, 80, 25);
    /* output to full screen window */
    cprintf("Terminate\n");
    return(0);
}
```

## The TopSpeed Graphics library

---

The TopSpeed graphics library, which is compatible with the Microsoft graphics library, contains the following graphics functions:

<code>_arc</code>	Draw elliptical arc
<code>_clearscreen</code>	Fill an area with current background color
<code>_cube</code>	Draw cuboid shape - eg for bar charts
<code>_displaycursor</code>	Says if cursor stays off on exit from graphics
<code>_ellipse</code>	Draw whole ellipse
<code>_floodfill</code>	Fill an area using current color and fill mask
<code>_stackfill</code>	Same as floodfill but works faster with convex shapes and sparse fill masks
<code>getbkcolor</code>	Get current background color
<code>getcolor</code>	Get current color
<code>_getcurrentposition</code>	Get current graphics position

<code>_getfillmask</code>	Make copy of current fill mask
<code>_getimage</code>	Store a screen image in a buffer
<code>_getlinestyle</code>	Return mask to use for drawing lines
<code>_getlogcoord</code>	Convert physical to logical coordinates
<code>_getphyscoord</code>	Convert logical to physical coordinates
<code>_getpixel</code>	Get pixel at specified logical position
<code>_gettextcolor</code>	Get pixel value for current text color
<code>_gettextposition</code>	Get current text position
<code>_getvideoconfig</code>	Get current graphics configuration
<code>_imagesize</code>	Buffer size needed to store a screen rectangle
<code>_lineto</code>	Draw a line from current to specified point
<code>_moveto</code>	Move current to specified point
<code>_outtext</code>	Output text at current position
<code>_pie</code>	Draw a wedge
<code>_polygon</code>	Draw a polygon
<code>_putimage</code>	Display a stored image at a specified position
<code>_rectangle</code>	Draw a rectangle
<code>_remapallpalette</code>	Remap all pixel values to colors
<code>_remappalette</code>	Remap specified pixel value to colors
<code>_selectpalette</code>	Select a color palette
<code>_setactivepage</code>	Specify the graphics page
<code>_setbkcolor</code>	Set current background color
<code>_setcliprgn</code>	Define a clipping rectangle
<code>_setcolor</code>	Set current color
<code>_setfillmask</code>	Set current fill mask
<code>_setlinestyle</code>	Set mask to use for drawing lines
<code>_setlogorg</code>	Move logical origin
<code>_setpixel</code>	Set the pixel at a specified point
<code>_settextcolor</code>	Set current text color
<code>_settextposition</code>	Move current text position to specified point
<code>_settextwindow</code>	Specify a rectangular area for text output
<code>_setvideomode</code>	Set current video mode
<code>_setviewport</code>	Define a clipping rectangle

<code>_setvisualpage</code>	Select current video page
<code>_wrapon</code>	Say whether text that falls outside text window should be clipped or wrapped
<code>_enable_herc</code>	Inform system a Hercules card is present

To use the TopSpeed graphics functions, refer to Chapter 1 ‘Choosing the right library’ if you are not using the TopSpeed automatic link facility.

### **Graphics library multi-thread considerations**

None of the graphics library functions provide locking internally, since this would not guarantee the correct execution of a multi-thread process. Consequently, the user must use the functions Lock and Unlock to protect sections of code if more than one thread is attempting to access a graphics screen.

When console I/O functions `cprintf`, `cputs`, `cscanf` and `cgets` are used, locking is implemented internally.

### **I/O Using Graphics Modes**

The function `_outtext` outputs a string to the screen as documented. Formatted I/O is also available via the console I/O functions.

When `_setvideomode` is called with any graphics or text mode as an argument, output from `cprintf`, `cputs`, `cscanf`, `cgets`, `putch`, and `getche` is redirected to the currently defined graphics library text window or graphics viewport.

A viewport can be used to limit the boundaries for graphics calls to a specified area of the screen (called a clipping region). This is accomplished with function `_setviewport` which also makes the upper left corner of this region the logical origin. Only output that fits within the clipping region will be visible.

When `_setvideomode` is called with `_DEFAULTMODE` as an argument, output from the conio functions is redirected back to the current window (defined by any window module that may be linked with the program). If no window module has been used, the full screen will be used as the display area.

## Operating system interface

---

These functions provide an interface between your programs and DOS version 2.0 and later.

### Interrupts

<code>_disable</code>	Disable interrupts
<code>_enable</code>	Enable interrupts
<code>_dos_getvec</code>	Get address of an interrupt handler
<code>getvect</code>	Get address of an interrupt handler
<code>_dos_setvec</code>	Set interrupt vector
<code>setvec</code>	Set interrupt vector
<code>_chain_intr</code>	Chain interrupt handlers
<code>int86</code>	Invoke DOS interrupt
<code>int86x</code>	Invoke DOS interrupt specifying segment registers
<code>intdos</code>	Invoke DOS system call (INT 21H)
<code>intdosx</code>	Invoke DOS system call, specifying segment registers
<code>intr</code>	Execute software interrupt
<code>bdos</code>	DOS system call using DX and AL registers nly
<code>bdosptr</code>	Invoke DOS interrupt requiring a pointer argument
<code>geninterrupt</code>	Macro to generate an interrupt
<code>_hardresume</code>	Return to DOS after a hardware error
<code>_hardretn</code>	Return to application after a hardware error
<code>_dos_keep</code>	Install Reside and Stay Terminated program
<code>sound</code>	Start sounding on system speaker
<code>nosound</code>	stop soundint on system speaker
<code>keep</code>	Install Terminate and Stay Resident program

### Exception and error handling

<code>etjmp</code>	Save an environment so that setjmp can return
<code>longjmp</code>	Restore environment saved by setjmp
<code>ctrlbrk</code>	Establish control-break handler
<code>getcbrk</code>	Return current setting of control-break flag
<code>setcbrk</code>	Set value of control-break flag

EnableBreakCheck	Turn on check for ctrl-break user interrupt
DisableBreakCheck	Turn off check for ctrl-break user interrupt
CnsHandler	Runtime check handler
_harderr	Establish a hardware error handler
dosexterr	Return extended error information

### **Read and write main memory**

peek	Fetch word at specified address
peekb	Fetch byte at specified address
poke	Put word at specified address
pokeb	Put byte at specified address
FP_OFF	Offset portion of a far pointer
FP_SEG	Segment portion of a far pointer
MK_FP	Build far pointer from segment and offset
segread	Get current value of segment registers

### **Memory allocation**

_dos_allocmem	Allocate memory block via DOS system call
allocmem	Allocate block of memory
_dos_setblock	Change size of allocated block
setblock	Change size of allocated block of memory
_dos_freemem	Free memory block
freemem	Return memory allocated using allocmem

### **DOS file handling**

_dos_open	Open an existing file
_dos_clos	Close file
_dos_creat	Create file overwriting existing one if present
_dos_creatnew	Create file unless it exists
_dos_read	Read from file
_dos_write	Write to file
randbrd	Read random-access records from file
randbwr	Write random-access records to file
_dos_getfileattr	Get file or directory attributes

<code>_dos_setfileattr</code>	Set file or directory attributes
<code>_dos_gettime</code>	Get date and time of last file write
<code>getverify</code>	Get state of system verify flag
<code>setverify</code>	Set system verify flag
<code>parsfnm</code>	Parse file description string

### **DOS directory handling**

<code>getfat</code>	Get File Allocation info for specified drive
<code>getfatd</code>	Get File Allocation info for default drive
<code>_dos_findfirst</code>	Find first occurrence of file from wildcard
<code>findfirst</code>	Find first occurrence of file from wildcard
<code>_dos_findnext</code>	Find next occurrence of file from wildcard
<code>findnext</code>	Find next occurrence of file from wildcard
<code>getdfree</code>	Determine free space on a disk
<code>getdta</code>	Return location of Disk Transfer Area (DTA)
<code>setdta</code>	Set location of Disk Transfer Area (DTA)
<code>absread</code>	Read disc sector
<code>abswrite</code>	Write to disc sector
<code>_dos_getdiskfree</code>	Get disk drive information
<code>_dos_getdrive</code>	Get current default drive
<code>_dos_setdrive</code>	Change current default drive

### **Path control functions**

<code>getcurdir</code>	Fetch current directory
<code>mkdir</code>	Make new directory
<code>rmdir</code>	Remove specified directory
<code>chdir</code>	Change current working directory
<code>getcwd</code>	Get current path
<code>getdisk</code>	Get current drive number
<code>setdisk</code>	Set current drive
<code>_getdrive</code>	Get current drive number
<code>fnmerge</code>	Build a path name
<code>_makepath</code>	Build a path name
<code>_splitpath</code>	Break up a path name

<code>fnsplit</code>	Break up a path name
<code>searchpath</code>	Search specified path for file

### **DOS environment**

<code>putenv</code>	Add or modify environment variable
<code>getenv</code>	Search environment variables for a name
<code>_searchenv</code>	Find file in path given by environment variable
<code>country</code>	Get country dependent information
<code>getpsp</code>	Get Program Segment Prefix (PSP) location

### **Port I/O Functions**

The TopSpeed function library also includes the following port I/O functions:

<code>inp</code>	Read byte from a specified port
<code>inportb</code>	Read byte from a specified port
<code>inpw</code>	Read word from a specified port
<code>inportw</code>	Read word from a specified port
<code>outp</code>	Write byte to a specified port
<code>outportb</code>	Write byte to a specified port
<code>outpw</code>	Write word to a specified port
<code>outportw</code>	Write word to a specified port

### **BIOS interface**

These functions provide an interface between programs and the IBM PC BIOS.

<code>bioskey</code>	Access to keyboard services
<code>_bios_keybrd</code>	Access to keyboard services
<code>biosmemory</code>	Get available memory
<code>_bios_memsize</code>	Get available memory
<code>biosprint</code>	Access to printer services
<code>_bios_printer</code>	Access to printer services
<code>biostime</code>	Access to system clock
<code>_bios_timeofday</code>	Access to system clock
<code>biosdisk</code>	Access to disk services
<code>_bios_disk</code>	Access to disk services



biosequip	Equipment check
_bios_equiplist	Equipment check
bioscom	Access to serial port
_bios_serialcom	Access to serial port

### **OS2 interface**

The following functions, despite their names, are reserved for the tasking interface with OS2. They are the preferred method of controlling OS2 threads.

_dosbeginthread	Initiate a thread
_dosendthread	Terminate a thread
_dosfreestack	Find out how much stack a thread can use
_beginthread	The same as _dosbeginthread; provided for compatibility
_endthread	The same as _dosendthread; provided for compatibility

### **DOS re-entrancy control**

The following functions help avoid problems with non re-entrant code. The InProgramFlag indicates whether a process is executing within its own program code or within the operating system. If a program attempts to re-enter DOS (i.e. when the InProgramFlag is clear) a run-time error will occur.

However it is legal to re-enter DOS from a critical error handler. In this case a call to SetInProgramFlag with the argument 1 will allow library functions that call DOS to operate.

GetInProgramFlag    Fetch the state of the InProgramFlag

SetInProgramFlag    Fetch the state of the InProgramFlag

### **Locale functions**

localeconv

setlocale

# CHAPTER 3

## *TopSpeed C Library reference*

This chapter contains an alphabetic listing of the functions defined in the standard TopSpeed C library.

Functions are listed alphabetically with the minor exceptions described in Chapter 1: 'Finding the function you want'. Leading underscores are not used in ordering. Thus, the following three functions are ordered as shown:

```
_exit  
exp  
_expand
```

Where far, near or huge variants of default functions exist, the main description is kept with the default function but a brief reference is included in the correct alphabetical position.

Three families of functions are listed together: the ctype character handling functions such as isalnum, and the spawn and exec group of process control functions.

There are two versions of some window functions, as explained in Chapter 1: 'Finding the right function'. The quick reference key on the right of the function tells you which version comes from the TopSpeed window module and which from the Turbo C window module.

Data structures and unions used by library functions all figure in the index at the back of this manual under the name of the structure.

## **void abort(void);**

## **A**

<b>Header file</b>	stdlib.h
<b>See also</b>	exit, _exit, raise, signal, exec, spawn
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Causes all threads to terminate.

The abort function prints the following message to stderr:

### **Abnormal Program Termination**

and calls raise(SIGABRT).

The default SIGABRT action is for the process to terminate with an exit code of 3. This may be changed by a preceding call to the signal function. The default action is to cause a runtime error - see section on program termination.

### **Return value**

By default, abort does not return to the calling function, but returns an exit code of 3 to the calling process.

### **Example**

```

                                #include <stdio.h>
                                #include <stdlib.h>
                                int process(char
*name);
                                main(int argc, char
*argv[])
                                {
                                int exit_code;
                                if(argc != 2)
                                {
                                fprintf(stderr, "USAGE - PROG filename\n");
                                abort();
                                /* command line incorrect, abort */
                                }
                                exit_code=process(argv[1]);
                                return(exit_code);
                                }
```

## **int abs(int num);**

---

**A**

<b>Header file</b>	stdlib.h
<b>See also</b>	cabs, fabs, labs
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None

The abs function returns the absolute value of the integer argument (num).

### **Return value**

The function returns  $|\text{num}|$ . Thus, if num is positive, the function simply returns num; if num is negative, the function returns -num.

There is no error return.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    unsigned int n;
    int n=-10;
    an=abs(n);
    /* output absolute value of -10 */
    printf("Abs
value of %d is %u\n", n, an);
    return(0);
}
```

## **int absread(int drive, int nsects, int lsect, void \*buffer);**

---

<b>Header file</b>	dos.h
<b>See also</b>	abswrite, biosdisk
<b>Portability</b>	Some DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `absread` reads specific sectors of a disk, ignoring file and directory divisions. The function uses DOS interrupt 25H for this purpose.

<code>drive</code>	specifies the drive to be read, with drive A = 0, drive B = 1, etc.
<code>nsects</code>	specifies the number of sectors to read.
<code>lsect</code>	specifies the starting logical sector number.
<code>buffer</code>	points to the location at which the information being read will be stored.

The function can read up to 64K bytes of data in a single call.

### **Return value**

The function returns 0 if the read was successful; otherwise the function returns -1 and sets `errno` to the value of the AX register (which is returned by the DOS interrupt).

### **Example**

```
#include <dos.h>
#include <stdio.h>
#define BUFFER_SIZE 512
main()
{
    char
    buffer[BUFFER_SIZE];
    absread(0, 1,
    0, buffer); /* read sector 0 */
    abswrite(0, 1,
    0,buffer); /* write sector 0*/
    return(0);
}
```

---

**nt abswrite(int drive, int nsects, int lsect, const void \*buffer);**

---

<b>Header file</b>	dos.h
<b>See also</b>	absread, biosdisk
<b>Portability</b>	Some DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `abswrite` writes specific sectors of a disk, ignoring the logical structure of the disk and the file and directory divisions. The function uses DOS interrupt 26H for this purpose.

<code>drive</code>	specifies the drive to which to write, with drive A = 0, drive B = 1, etc.
<code>nsects</code>	specifies the number of sectors to write.
<code>lsect</code>	specifies the starting logical sector number.
<code>buffer</code>	points to the location at which the information to be written is stored.

The function can write up to 64K bytes of data in a single call.

**Return value**

The function returns 0 if the read was successful; otherwise the function returns -1 and sets `errno` to the value of the AX register (which is returned by the DOS interrupt).

**Example**

```
#include <dos.h>
#include <stdio.h>
#define
    BUFFER_SIZE 512
main()
{
    char
    buffer[BUFFER_SIZE];
    absread(0, 1,
    0, buffer); /* read sector 0 */
    abswrite(0, 1,
    0, buffer); /* write sector 0 */
    return(0);
}
```

## **int access(const char \*path, int mode);**

**U**

<b>Header file</b>	io.h
<b>See also</b>	chmod, fstat, stat.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	MSDOS not re-entrant.

The function access checks whether the specified file exists and, if it is not a directory, whether the file can be accessed in the specified mode.

path	specifies the file being checked.
mode	specifies the access mode being tested. This can take the following values:

<b>Value</b>	<b>Meaning</b>
0	Check for existence
2	Check for write permission
4	Check for read permission
6	Check for read and write permission

### **Return value**

Returns 0 if a file has access using the specified mode. The value -1 is returned if the access is not available, and errno is set to one of the following values:

EACCES	Access Denied.
ENOENT	File or path not found.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
main()
{
    int fh;
    char
        *file="TEMP.$$$";
    if(access(file, 2) == -1) /* write permit? */
        abort();
                                /* not present */
    fh=open(file,
        O_RDWR);
}
```

---

**double acos(double x);**

---

**A**

---

**long double acosl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	asin, atan, atan2, cos, matherr
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

Returns the arc cosine of x, which must be in the range -1 to +1.

**Return value**

Returns the function result. If x is outside of the range -1 to +1 `errno` is set to `EDOM`, an error message is printed to `stderr` and 0 is returned.

Error handling can be changed by installing a different `matherr` function.

**Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double result;
    float num;
    scanf("%f", &num);
    /* calculate arc cosine of input */
    result=acos((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num,
            errno);
    else
        printf(
            "arc cosine of %g is %g\n",
            num,
            result);
    return(0);
}
```



## **int allocmem(unsigned size, unsigned \*segp);**

---

<b>Header file</b>	dos.h
<b>See also</b>	coreleft, freemem, setblock, malloc
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function allocmem allocates a block of memory. The function uses DOS service 48H for this purpose.

size	specifies how many <i>paragraphs</i> (16-byte blocks) to allocate.
segp	points to a location that will contain the segment address of the block being allocated. If the desired block cannot be allocated, no value is assigned to the location to which segp points.

### **Return value**

The function returns -1 if successful; otherwise the number of paragraphs in the largest available block. If there is an error, the \_doserrno is set and the function sets errno to ENOMEM (not enough memory).

### **Example**

```
#include <dos.h>
#include <stdio.h>
main ( )
{
    char far *buffer;
    unsigned new_seg;
    int error;
    error=allocme (100, &new_seg);
    if (error)
    {
        fprintf(stderr, "Memory Allocation err\n");
        return (1);
    }
    buffer=MK_FP (new_seg, 0);
    printf ("Memory is at %Fp\n", buffer);
    freemem (new_seg);
    return (0);
}
```

**short far \_arc(****M**

```
short x1, short y1,  
short x2, short y2,  
short x3, short y3,  
short x4, short y4);
```

---

<b>Header file</b>	graph.h
<b>See also</b>	_ellipse, _lineto, _pie, _rectangle, _setcolor
<b>Portability</b>	DOS
<b>Multi-thread</b>	Function is not re-entrant.

The function `_arc` from the graphics module draws an elliptical arc within a bounding rectangle. The arc's center is the center of the bounding rectangle.

x1,y1	coordinates of the upper left point of the rectangle bounding the arc.
x2,y2	coordinates of the lower right point of the rectangle bounding the arc.
x3,y3	coordinates of a vector used to determine the starting point for the arc.
x4,y4	coordinates of a vector used to determine the arc end point.

When the arc is drawn, it begins at the point at which the arc intersects the vector specified by x3, y3, and ends at the point at which the arc intersects the vector specified by x4, y4. When the arc is drawn, it is drawn in a counterclockwise direction to the ending point.

**Note:** Coordinates for the points specified are for logical points.

**Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

**Example**

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

```

main()
{
    struct videoconfig v;
    short X1, Y1, X2, Y2, X3, Y3, X4, Y4;
    short xoffset, yoffset;
    struct xycoord arc_array[2][4];
    int n;
    _setvideomode(_MRESNOCOLOR);
    _getvideoconfig(&v);
    xoffset=v.numxpixels/16;
    yoffset=v.numypixels/16;

    arc_array[0][0].xcoord=v.numxpixels-1;
    arc_array[0][0].ycoord=0;
    arc_array[1][0].xcoord=v.numxpixels-1;
    arc_array[1][0].ycoord=v.numypixels-1;

    arc_array[0][1].xcoord=v.numxpixels-1;
    arc_array[0][1].ycoord=v.numypixels-1;
    arc_array[1][1].xcoord=0;
    arc_array[1][1].ycoord=v.numypixels-1;

    arc_array[0][2].xcoord=0;
    arc_array[0][2].ycoord=v.numypixels-1;
    arc_array[1][2].xcoord=0;
    arc_array[1][2].ycoord=0;

    arc_array[0][3].xcoord=0;
    arc_array[0][3].ycoord=0;
    arc_array[1][3].xcoord=v.numxpixels-1;
    arc_array[1][3].ycoord=0;
    n=0;
    while(n < 4)
    {
        X1=xoffset;
        Y1=yoffset;
        X2=v.numxpixels-xoffset;
        Y2=v.numypixels-yoffset;
        X3=arc_array[0][n].xcoord;
        Y3=arc_array[0][n].ycoord;
        X4=arc_array[1][n].xcoord;
        Y4=arc_array[1][n].ycoord;
        while((X1 < X2) && (Y1 < Y2))
        {
            _arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4);
            X1+=xoffset;
            Y1+=yoffset;
            X2-=xoffset;
            Y2-=yoffset;
        }
        getch();
        _clearscreen(_GCLEARSCREEN);
        ++n;
    }
    _setvideomode(_DEFAULTMODE);
    return(0);
}

```

## char \*asctime(const struct tm \*time);

**A**

**Header file**           time.h

**See also**             ctime, gmtime, localtime, time.

**Portability**         ANSI

**Multi-thread**        In a Multi-thread program a call to asctime will destroy only the result of a previous call from the same thread.

The function asctime converts time as a structure to a formatted character string containing exactly 26 characters. E.g.,

```
Tue Aug 23 12:02:22 1988\n\0
```

The value in structure tm can be obtained by a call to localtime or gmtime.

```
struct tm {
    int tm_sec;   /*seconds */

    int tm_min;   /*minutes */

    int tm_hour;  /*hours */

    int tm_mday;  /*day of month */

    int tm_mon;   /*month */

    int tm_year;  /*year */

    int tm_wday;  /*day of week */

    int tm_yday;  /*day of year */

    int tm_isdst; /*daylight saving flag */
};
```

**Note:** asctime uses a static buffer for the character string, so a call to asctime will destroy the result of a previous call to asctime or ctime.

ANSI does not support dates prior to 1970 and such values in structure tm will produce unpredictable results.

### Return value

The function returns a pointer to the character string result. There is no error return.

### Example

```
#include <stdio.h>
#include <time.h>
main()
{
```

```
    struct tm *new_time;
    time_t tt;
    char *time_string;
    time(&tt); /* time in seconds */
               /* local time as
    structure */
    new_time=localtime(&tt);
    time_string=asctime(new_time);
    printf("Time is %s\n", time_string);
               /* print time string */
    return(0);
}
```

---

**double asin(double x);**

---

**A**

---

**long double asinl(long double x);**

---

**Header file**           math.h**See also**            acos, atan, atan2, cos, matherr.**Portability**        ANSI**Multi-thread**       None.

The function asin returns the arc sine of x, which must be in the range -1 to +1.

**Return value**

asin returns the function result. If x is outside of the range -1 to +1, errno is set to EDOM, an error message is printed to stderr, and 0 is returned.

Error handling can be altered by installing a different matherr function.

**Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double result;
    float num;
    scanf("%f", &num);
    /* calculate arc sine of input */
    result=asin((double) num);
    if(errno)
        printf("Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("arc sine of %g is %g\n", num, result);
    return(0);
}
```

---

**void assert(expression);**

---

**A**

<b>Header file</b>	assert.h
<b>See also</b>	abort, raise, signal.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

If *expression* is false, assert prints a message to stdout and calls abort.

The message has the format:

Assertion failed: expression, file filename, line linenumber.

If the macro NDEBUG has been defined before the point of inclusion of assert.h, assert is defined as ((void) 0).

.i.NDEBUG macro;

**Return value**

See abort.

**Example**

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

void error_check(int test_value)
{
    assert(test_value != 0);
    assert(
        (test_value > -BUFFER_SIZE) &&
        (test_value < BUFFER_SIZE));
    printf("Test value %d OK\n", test_value);
        /* reaches here only if
*/
    /* assertions passed    */
    return;
}
```

**wintype at(abscoord X, abscoord Y);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

This function from the TopSpeed window module returns the handle for the window currently displayed at the *absolute* position X,Y.

X,Y                      coordinates of the location.

**Return value**

Returns the handle for the window displayed at the specified position if one is displayed. Returns null if no window is displayed at this position.

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>
windef WD1={
    10, 2, 60, 8, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
    wintype W1;
windef WD2={
    40, 6, 75, 18, Yellow, Red,
    FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
    LightGray, Blue};
wintype W2;
main()
{
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    W2 = windowopen(&WD2) ; /* open window */
    setttitle(W2, "Window 2", CenterUpperTitle);
    cprintf("Hello World sent to window 2\n");
    getch();
    putontop(at(60, 15));
    cprintf("This window visible at 60, 15");
    getch();
    windowclose(W1);
    windowclose(W2);
    putontop(_fullscreen);
    return(0);
}
```



---

**double atan(double x);**

---

**A**

---

**long double atanl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	acos, asin, atan2, cos, matherr.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function atan returns the arc tangent of x.

**Return value**

atan returns the function result.

**Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double result;
    float num;
    scanf("%f", &num);
    /* calculate arc      */
    /* tangent of input   */
    result=atan((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf(
            "arc tangent of %g is %g\n",
            num, result);
    return(0);
}
```

---

**double atan2(double x, double y);**

---

**A**

long double atan2l(long double y, long double x);

Header file **math.h**

See also **acos, asin, atan, cos, matherr.**

Portability **ANSI**

Multi-thread **None.**

The function `atan2` returns the arc tangent of  $y/x$ . The quadrant of the **Return value** is determined by the signs of both arguments.

**Return value**

`atan2` returns the function result. If both arguments of `atan2` are 0, `errno` is set to `EDOM`, an error message is printed to `stderr`, and the value 0 is returned.

Error handling can be altered by installing a different `matherr` function.

**Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double result;
    float x, y;
    scanf("%f %f", &x, &y);
    result=atan2((double) x, (double) y);
    if(errno)
        printf(
            "Input %g/%g causes error %d\n",
            (double) x, (double) y, errno);
    else
        printf(
            "arc tangent of %g/%g is %g\n",
            x, y, result);
    return(0);
}
```

## **int atexit(void (\* func)(void));**

**A**

<b>Header file</b>	stdlib.h
<b>See also</b>	exit, _exit, onexit.
<b>Portability</b>	ANSI.
<b>Multi-thread</b>	Access to the function stack is controlled by semaphore in Multi-thread program. While this ensures that any thread can call atexit, in a Multi-thread program the actual order of function calls would be undefined.

The function atexit places the function pointer func onto a stack to be called from the function exit when the process terminates. Functions are called on a ‘Last in First out’ basis and cannot take parameters or return any value. The maximum number of functions accommodated by the atexit stack is 32. Any further calls to atexit will return an error.

### **Return value**

atexit returns 0 if the function was successfully placed on the stack. If the stack was full, a nonzero value will be returned.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
void func1(void)
{
    printf("func1\n");
    return;
}
void func2(void)
{
    printf("func2\n");
    return;
}
void func3(void)
{
    printf("func3\n");
    return;
}
main()
{
    atexit(func1); /* push functions onto */
    atexit(func2); /* exit stack          */
    atexit(func3);
    printf("main\n");
    return(0);
}
/* Output is :
main
func3
func2
func1 */
```

---

**double atof(const char \*s);**

---

**A**

---

**long double atofl(const char \*nptr);**

---

<b>Header file</b>	math.h and stdlib.h
<b>See also</b>	atoi, atol, ecvt, fcvt, gcvt, strtod.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `atof` converts the string argument, `s`, to a double precision floating point value. The function `atofl` is the same as `atof` but returns a long double.

The input string is scanned until a character is encountered that cannot be recognized as part of a number. (This is usually the terminating null character).

The function expects the input string to have the following form:

[whitespace] [ {+|-} ] [digits] [.digits] [{D|d|E|e}] [{+|-}] [digits]

whitespace can consist of any space or tab characters and digits may be any decimal digit.

### Return value

The function returns the result of the conversion. If the input string cannot be interpreted as a number, the function returns 0.

### Example

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char *s="123456.78e10";
    double result;
    result=atof(s); /* convert real number */
                  /* and output it      */
    printf("%s %g\n", s, result);
    return(0);
}
```

---

**int atoi(const char \*s);**

---

**A**

<b>Header file</b>	stdlib.h
<b>See also</b>	atof, atol, atoul, strtol, strtoul.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `atoi` converts the string input `s` to an integer value.

The input string is scanned until a character is encountered that cannot be recognized as part of a number. (This is usually the terminating null character).

The function expects the input string to have the following form:

[whitespace] [ {+|-} ] [digits]

whitespace can consist of any space or tab characters and digits may be any decimal digit.

**Return value**

The function returns the result of the conversion. If the input string cannot be interpreted as a number, the function returns 0.

**Example**

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

main()
{
    char *s="12345";
    int result;

    result=atoi(s);
    /* convert and output integer */
    printf("%s %d\n", s, result);
    return(0);
}
```

## **long atol(const char \*s);**

**A**

**Header file**            `stdlib.h`

**See also**             `atof, atoi, atoul, strtol, strtoul.`

**Portability**         `ANSI`

**Multi-thread**        `None.`

The function `atol` converts the string input `s` to a long integer value.

The input string is scanned until a character is encountered that cannot be recognized as part of a number. (This is usually the terminating null character).

The function expects the input string to have the following form:

`[whitespace] [ {+|-} ] [digits]`

`whitespace` can consist of any space or tab characters and `digits` may be any decimal digit.

### **Return value**

The function returns the result of the conversion. If the input string cannot be interpreted as a number, the function returns 0.

### **Example**

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

main()
{
    char *s="-123456";
    long result;

    /* convert and output long */
    result=atol(s);
    printf("%s %ld\n", s, result);
    return(0);
}
```

## **unsigned long atoul(const char \*s);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	atof, atoi, atol, strtol, strtoul.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function atoul converts the string input s to an unsigned long value.

The input string is scanned until a character is encountered that cannot be recognized as part of a number. (This is usually the terminating null character).

The function expects the input string to have the following form:

[whitespace] [digits]

whitespace can consist of any space or tab characters and digits may be any decimal digit.

### **Return value**

The function returns the result of the conversion. If the input string cannot be interpreted as a number, the function returns 0.

### **Example**

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

main()
{
    char *s="123456";
    unsigned long result;
        /* convert and output */
        /* unsigned long      */
    result=atoul(s);
    printf("%s %lu\n", s, result);
    return(0);
}
```

## **int Awaited(SIGNAL s);**

---

<b>Header file</b>	process.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

The function `Awaited` returns a nonzero value if any process is waiting on the signal `s` Æ that is, if the counter associated with `s` is negative.

### **Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

### **Example**

See example program in 2, page 47 .



## **int bdos(int dosfunc, unsigned dosdx, unsigned dosal);**

---

<b>Header file</b>	dos.h
<b>See also</b>	intdos, intdosx
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function bdos calls the DOS function dispatcher (INT 21H) to invoke a specified DOS system call.

dosfunc	specifies the DOS system call to be invoked.
dosdx	contains the value to be stored in the DX register for the system call.
dosal	contains the value to be stored in the AL register for the system call.

Prior to calling INT 21H, the bdos function places the values of the second and third parameters in DX and AL.

### **Return value**

The function returns the value stored in the AX register after the interrupt.

### **Example**

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

main()
{
    char buffer[64];
    buffer[0]=61;
    /* keyboard input in small model */
    bdos(0xA, (unsigned) buffer, 0);
    printf(
        "Input - %.*s", buffer[1],
        &buffer[2]);
    return(0);
}
```

## **int bdosptr(int dosfunc, void \*dosdx, unsigned dosal);**

---

<b>Header file</b>	dos.h
<b>See also</b>	bdos, geninterrupt, int86, int86x, intdos, intdosx
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `bdosptr` provides access to those DOS system calls that require a pointer argument.

<code>dosfunc</code>	specifies the DOS system call to be invoked.
<code>dosdx</code>	points to the value of the DX register in small data models, and to DS:DX in large data models.
<code>dosal</code>	contains the value to be stored in the AL register for the system call.

### **Return value**

If successful, the function returns the value in AX; otherwise the function returns -1 and sets `errno` and `_doserrno` to the error code returned by DOS.

### **Example**

```
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
main()
{
    char buffer[64];
    buffer[0]=61;
    /* keyboard input in any model */
    bdosptr(0xA, buffer, 0);
    printf(
        "Input - %.*s",
        buffer[1], &buffer[2]);
    return(0);
}
```

```
int far _beginthread(  
    void (far cdecl *func)(void far *arg),  
    void far *stack,  
    unsigned stack_size,  
    void far *arg)
```

---

**Header file**            process.h

This function is very similar to \_dosbeginthread.

func                    the thread procedure, taking a far pointer to its arguments.

stack                   points to the base of the thread's stack.

stack\_size             the size of the stack segment.

arg                     pointer to the threads arguments.

**Return value**

Returns the thread number of the new thread. Returns -1 if the thread could not be created.

**Note** The stack must be a segment allocated from the operating system.

## **int bioscom(int cmd, char byte, int port);**

---

**Header file**            bios.h

**Portability**            IBM PC.

**Multi-thread**          DOS BIOS is not re-entrant

The function bioscom performs serial communications through port.

port                    specifies the port. COM1 = 0, COM2 = 1, etc.

cmd                    specifies the action desired. This can have any of the following values:

- |   |   |
|---|---|
| 0 | set communications parameters to the values given by byte |
| 1 | send the character in byte through the port.              |
| 2 | receive a character from the specified port.              |
| 3 | determine and return the status of the port.              |

byte                    specifies the communications settings (parity, baud rate, etc.). This information is represented by *ORing* the appropriate combination of values from the following groups. One value is taken from each group.

### **Baud Rate**

0x00	110 baud
0x20	150 baud
0x40	300 baud
0x60	600 baud
0x80	1200 baud
0xA0	2400 baud
0xC0	4800 baud
0xE0	9600 baud

### **Parity**

0x00	no parity
0x08	odd parity
0x18	even parity

### **Data Bits**

0x02	7 data bits
0x03	8 data bits

### **Stop Bits**

0x00	1 stop bit
0x04	2 stop bits

### Return value

The function returns a 16-bit value. The high-order byte represents status information:

Bit	Meaning
8	data ready
9	overflow error
10	parity error
11	framing error
12	break detected
13	Empty transmit holding register
14	Empty transmit shift register
15	Time out. Set if transmission was not possible.

The value of the low-order byte depends on the value of the cmd argument, as follows:

cmd == 0, 3

Low order bits as follows:

Bit	Meaning
0	Clear to send (CTS) changed
1	Data set ready changed
2	Trailing edge ring detector
3	Receive line signal detector hung
4	Clear to send
5	Data set ready
6	Ring indicator
7	Received line signal detect

cmd == 1

If bit 15 is set, this indicates an error. If it is clear, the **Return value** equals the byte written; otherwise the bits are interpreted as follows:

Bit(s)	Meaning
0—7	character to be written.
8	data ready



```
int biosdisk(  
    int cmd,  
    int drive,  
    int head,  
    int track,  
    int sector,  
    int nsecs,  
    void *buf);
```

---

**Header file**            bios.h

**See also**            absread, abswrite

**Portability**        IBM PC.

**Multi-thread**      DOS BIOS is not re-entrant

The function biosdisk carries out low-level disk operations by using interrupt 13H to communicate directly with the BIOS.

`drive`

specifies the drive on which the operations are to be carried out. Values 0, 1, ... correspond to floppy drives A, B, ..., respectively. Values 0x80, 0x81, ... correspond to the first and second hard disk drives (not partitions), respectively, with subsequent hard drives getting consecutive values.

`head`                specifies the number of the head to be used.

`track`              specifies a track.

`sector`             specifies a sector.

`nsecs`              specifies the number of sectors involved in the disk operation.

`buf`                points to a memory location that can be used as a source or target for information being written or read, or that can store information for comparison.

`cmd`                specifies the desired disk operation. The possible values this parameter can take depend on the type of machine being used. Values 0 through 5 are valid for any machine on which the biosdisk function works Æ i.e., PC, XT, AT, PS/2 and compatible computers. Values 6 through 20 are not valid for PCs. Values of cmd determine actions as follows:

This is formatted N1 with the space after cut down to fit the table in a page

## Values of the cmd parameter and their interpretation

---

0	Do hard reset using disk controller. All other parameters are ignored.
1	Return status of last disk operation. All other parameters are ignored.
2	Read information from one or more sectors into the storage to which buf points. Values of head, track, sect, and nsecs are used to determine where and how much to read. 512 bytes are read per sector.
3	Read information from the storage to which buf points into one or more sectors. Values of head, track, sect, and nsecs are used to determine where and how much to write. 512 bytes are written per sector.
4	Verify one or more disk sectors. Values of head, track, sect, and nsecs are used to determine where and how much to verify.
5	Format a track, specified by head and track. Sector headers are written to the specified track. A table of sector headers to be used are stored in the location which buf points to. (See your hardware Technical Reference Manual for information about this table.)
6	Format a track, flagging bad sectors.
7	Format the drive, beginning at a specified track.
8	Determine and return the drive parameters. This information is stored in the first four bytes of the location to which buf points.
9	Initialize drive-pair characteristics.
10	Do a long read Æ i.e., read four extra bytes per sector.
11	Do a long write Æ i.e., write four extra bytes per sector.
12	Do a disk seek.
13	Alternate a disk reset.
14	Read a sector buffer.
15	Write a sector buffer.
16	Test whether the specified drive is ready.
17	Recalibrate the specified drive.
18	Run a diagnostic on the controller RAM.
19	Run a diagnostic on the drive.
20	Run an internal diagnostic on the controller.



## Return value status information returned by biosdisk

---

0x00	Operation was successful.
0x01	Invalid command.
0x02	Address mark not found.
0x03	Attempt to write to write-protected disk.
0x04	Sector not found.
0x05	Hard disk reset failed.
0x06	Disk has been changed since last operation.
0x07	Drive parameter activity failed.
0x08	DMA overrun
0x09	Attempt to cross 64K boundary during DMA.
0x0A	Bad sector was detected.
0x0B	Bad track was detected.
0x0C	Invalid track
0x10	Error (CRC or ECC) on disk read.
0x11	Corrected (CRC or ECC) data error.
0x20	Controller has failed.
0x40	Seek operation was unsuccessful.
0x80	No response from drive.
0xAA	Hard disk drive not ready.
0xBB	Undefined hard disk error.
0xCC	Write fault.
0xE0	Status error.
0xFF	Attempt to sense drive was unsuccessful.

### Example

```
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
#define BUFFER_SIZE 512
main()
{
    char buffer[BUFFER_SIZE];
        /* read one sector      */
        /* from first hard disk */
    biosdisk(2, 0x80, 1, 2, 2, 1, buffer);
    return(0);
}
```

## int biosequip(void);

---

<b>Header file</b>	bios.h
<b>See also</b>	_bios_equiplist
<b>Portability</b>	IBM PC.
<b>Multi-thread</b>	DOS BIOS is not re-entrant

The function biosequip reports on the hardware configuration. The function uses BIOS interrupt 11H for this purpose.

### Return value

The equipment information is returned in the individual bits of a 16-bit value.

### Bit(s)    Meaning

0	Set to 1 if system boots from disk.
1	Set to 1 if a coprocessor is installed.
2-3	Indicates motherboard RAM size.
00	16K
01	32K
10	48K
11	64K
4-5	Initial video mode
00	Unused
01	Color card, 40x25 BW mode.
10	Color card, 80x25 BW mode.
11	Monochrome card, 80x25 BW mode.
6-7	Number of disk drives
00	1 drive
01	2 drives
10	3 drives
11	4 drives, but only if bit 0 is 1.
8	Set to 0 if machine does <i>not</i> have DMA; set to 1 otherwise.
9-11	Number of serial ports
12	Set to 1 if a game port is attached.
13	Set to 1 if a serial printer is attached.
14-15	Number of parallel printers installed.

**Example**

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

main()
{
    unsigned eq;
    eq=biosequip();
    if(eq&2)
        printf("Co-processor installed\n");
    return(0);
}
```

## **int bioskey(int cmd);**

---

<b>Header file</b>	bios.h
<b>See also</b>	_bios_keybrd
<b>Portability</b>	IBM PC.
<b>Multi-thread</b>	DOS BIOS is not re-entrant

The function bioskey performs the keyboard operation specified by cmd. The function uses BIO interrupt 16H for this purpose.

cmd	specifies the keyboard operation to carry out. This parameter can have the following values:
0	get the next character from the keyboard buffer. If there is no key in the buffer, the function waits until a key is pressed. The keystroke is removed from the buffer.
1	check the keyboard buffer to see whether a key has been pressed. If a key has been pressed, return the key but do not remove it from the buffer. If no key has been pressed, return with this information.
2	determine the current shift key status — i.e., determine whether any special keys (SHIFT, CTRL, ALT, Scroll Lock, Num Lock, Caps Lock, Insert) have been pressed.

### **Return value**

The value returned by the function depends on the action requested — i.e., on the value of cmd, as follows:

0	If the low-order byte is nonzero, this is interpreted as the ASCII code for the key pressed. If the low-order byte is zero, the high-order byte is interpreted as a scan code.
1	If no key has been pressed, the function returns 0. If a key has been pressed, the same information is returned as for a cmd value of 0.
2	Returns the shift key status as the result of <i>ORing</i> the following bits:

Bit	Meaning
0	Right shift key was pressed.
1	Left shift key was pressed.
2	CTRL key was pressed.
3	ALT key was pressed.
4	Scroll Lock is set to on.
5	Num Lock is set to on.
6	Caps Lock is set to on.
7	Insert is set to on.

**Example**

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

main()

{
    unsigned key_state;

    key_state=bioskey(2);
    if(key_state&0x10)
        printf("scroll lock toggled\n");
    return(0);
}
```

## **int biosmemory(void);**

---

<b>Header file</b>	bios.h
<b>See also</b>	_bios_memsizes
<b>Portability</b>	IBM PC.
<b>Multi-thread</b>	DOS BIOS is not re-entrant

The function biosmemory determines and returns the amount of RAM memory, *excluding* any special memory (such as adapter, extended, or expanded memory).

### **Return value**

The function returns the amount of RAM in Kbytes.

### **Example**

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

main()
{
    int size;

    size=biosmemory();
    printf("%dKBytes installed\n", size);
    return(0);
}
```

## **int biosprint(int cmd, int byte, int port);**

---

**Header file**            bios.h

**See also**             \_bios\_printer

**Portability**          IBM PC.

**Multi-thread**        DOS BIOS is not re-entrant

Accesses printer operations using BIOS interrupt 17H.

cmd                    specifies the printer command:

0                      print the character specified in the byte parameter.

1                      initialize the specified printer port.

2                      determine the printer's status.

byte                  specifies the character to print. This can be anything between 0 and 255.

port                  specifies the port associated with the printer: 0 = LPT1, 1 = LPT2, etc.

### **Return value**

The function returns information about the printer's status by *ORing* the proper combination of the following bits:

0                      Device timed out.

3                      I/O error

4                      Printer is selected

5                      Printer is out of paper.

6                      Acknowledge

7                      Not busy

### **Example**

```
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
main()
{
    unsigned status;
    status=biosprint(2, 0, 0);/* LPT1 status */
    printf("LPT1 status - %X\n", status);
    return(0);
}
```

## **long biostime(int cmd, long newtime);**

---

<b>Header file</b>	bios.h
<b>Portability</b>	IBM PC.
<b>Multi-thread</b>	DOS BIOS is not re-entrant

The function `biostime` provides access to the BIOS timer. The function uses BIOS interrupt 1AH for this purpose.

The BIOS timer indicates the number of clock ticks since midnight, with ticks about 0.055 seconds apart (18.2 ticks per second).

`cmd`

0	read the current timer value.
1	set the current timer to the value specified in <code>newtime</code> .
<code>newtime</code>	specifies the value to store in the timer.

### **Return value**

If `cmd = 0`, the function returns the number of ticks since midnight.

### **Example**

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

main()
{
    long current_time;

    current_time=biostime(0, 0L);
    printf(
        "Ticks since midnight - %ld\n",
        current_time);
    return(0);
}
```



## **unsigned \_bios\_disk(unsigned cmd, struct diskinfo\_t \*info);**

---

**Header file**            bios.h

**Portability**            IBM PC.

The function `_bios_disk` uses interrupt 13H to provide disk access.

`info`                    points to a structure that contains the parameters required for the desired action.

```
struct diskinfo_t {
    unsigned drive;
    unsigned head;
    unsigned track;
    unsigned sector;
    unsigned nsectors;
    void far *buffer;
};

cmd
```

specifies the action desired, and can have the following values:

<code>_DISK_RESET</code>	forces the disk controller to do a reset on the disk. This action requires no parameters.
<code>_DISK_READ</code>	reads a specified number of sectors from the disk. All members of the target structure for <code>info</code> are used to pass parameters for this action.
<code>_DISK_WRITE</code>	writes a specified amount of data to the disk. All members of the target structure for <code>info</code> are used to pass parameters for this action.
<code>_DISK_VERIFY</code>	checks the specified disk sectors and runs a <i>cyclic redundancy check</i> (CRC). Except for <code>buffer</code> , all members of the target structure for <code>info</code> are used to pass parameters for this action.
<code>_DISK_FORMAT</code>	formats a specified track on the disk. The track to be formatted is specified in the <code>head</code> and <code>track</code> members of <code>*info</code> . <code>buffer</code> points to sector markers, whose format depends on the drive type.
<code>_DISK_STATUS</code>	checks the status of the last disk operation, and returns this information. The possible Return values are shown in the following list.
0x00	no error.
0x01	invalid command.
0x02	address mark not found.
0x03	attempt to write to write-protected disk.
0x04	sector not found
0x05	reset failed

0x06	floppy disk removed
0x07	bad parameter information
0x08	DMA overflow
0x09	DMA error
0x0A	bad sector
0x10	CRC or ECC data error (uncorrectable)
0x11	corrected data (ECC) error
0x20	controller failure
0x40	seek error
0x80	time out
0xAA	drive not ready
0xBB	undefined error
0xCC	write error
0xE0	status error

### Return value

The returned value depends on the argument passed in `cmd`.

The outcome from checking the status of the last disk operation is returned in the high-order byte. Possible values are listed above.

If a disk read or write is successful, 0 is returned in the high-order byte of the returned value. In these cases, the number of sectors read or written is returned in the low-order byte. If an error occurred, the high order byte will have one of the values specified earlier. The function behaves in the same way for `_DISK_VERIFY`.

There is no **Return value** when formatting a disk.

### Example

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

#define BUFFER_SIZE 512

main()
{
    struct diskinfo_t di;
    char buffer[BUFFER_SIZE];
    di.drive=0x80;
    di.head=0;
    di.track=2;
    di.sector=2;
    di.nsectors=1;
    di.buffer=(void far *) buffer;
    _bios_disk(_DISK_READ, &di); /* read sector */ return(0);
}
```

## **unsigned \_bios\_equiplist(void);**

---

**Header file**            bios.h

**Portability**           IBM PC.

The \_bios\_equiplist function determines the hardware configuration, using DOS interrupt 11H.

### **Return value**

The function returns an unsigned value whose bits provide information about the configuration.

<b>Bit(s)</b>	<b>Meaning</b>
0	Specifies whether a disk drive is installed. Set to 1 if any drive is installed.
1	Specifies whether a math coprocessor is present (in which case the bit is set to 1).
2-3	Unused.
4-5	Unused.
6-7	Specifies the initial video mode, which can be any of:
<b>Value</b>	<b>Meaning.</b>
00	Reserved
01	Color, 40 x 25.
10	Color, 80 x 25.
11	Monochrome, 80 x 25.
8-9	Value is 1 less than the number of floppy drives(e.g., 2 drives yields a value of 1 in these bits).
10	Unused.
11-13	Number of RS-232 ports installed.
14	Specifies whether an internal modem is installed (in which case the bit is set to 1).
15-16	Specifies the number of printers installed.

**Example**

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

main()

{
    unsigned eq;

    eq=_bios_equiplist();
    if(eq&2)
        printf("Co-processor installed\n");
    return(0);
}
```

## **unsigned \_bios\_keybrd(unsigned cmd);**

---

**Header file**            bios.h

**Portability**           IBM PC.

The `_bios_keybrd` function accesses the keyboard, using DOS interrupt 16H. Depending on the action requested, a character may be removed from the keyboard buffer by a call to `_bios_keybrd`.

`cmd`                    Specifies the keyboard function being requested. This argument can have the following values:

`_KEYBRD_READ`       Reads the next character from the keyboard. The function waits if no character has been typed. If a character has been typed, this function removes the character from the input buffer.

`_KEYBRD_READY`      Checks whether a key has been pressed. If so, the function reads the character, but does not remove it from the input buffer.

`_KEYBRD_SHIFTSTATUS`   Returns information about any special keys (e.g., SHIFT, ALT, NUM LOCK) that have been pressed.

### **Return value**

The function returns a value whose components will be interpreted in a manner dependent on the value of `cmd`.

If a character was read, the character's ASCII value is returned in the low-order byte, and the character's keyboard scan code is returned in the high order byte.

If the function was checking whether a key was pressed, the **Return value** is either 0 (i.e., no key was pressed) or the same information as in the previous case.

If the shift status was checked, the low order byte represents the combination of special keys that were pressed, with each bit representing a different special key:

<b>Bit</b>	<b>Meaning</b>
0	Right shift key was pressed.
1	Left shift key was pressed.
2	CTRL key was pressed.
3	ALT key was pressed.
4	Scroll Lock is on.

- 5 Num Lock is on.
- 6 Caps Lock is on.
- 7 Insert mode is set.

**Example**

```
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
main()
{
    unsigned key_state;
    key_state=_bios_keybrd(2);
    if(key_state&0x10)
        printf("scroll lock toggled\n");
    return(0);
}
```

## **unsigned \_bios\_memsz(void);**

---

**Header file**            bios.h

**Portability**           IBM PC.

Determines the total amount of memory available, using interrupt 12H.

### **Return value**

The function returns the amount of memory installed on the system in Kbytes.

### **Example**

```
#include <bios.h>
#include <stdlib.h>
#include <stdio.h>
main()
{
    int size;

    size=_bios_memsz();
    printf("%dKBytes installed\n", size);
    return(0);
}
```

```
unsigned _bios_printer(  
    unsigned cmd,  
    unsigned port,  
    unsigned data);
```

---

**Header file**     bios.h

**Portability**     IBM PC.

The `_bios_printer` function sends the value specified in `data` to the printer, using interrupt 17H.

`data`               Specifies the value being sent to the printer.

`port`               Specifies the port to which the value is being sent.

`cmd`                Specifies the printer service being requested. This parameter can have the following values:

`_PRINTER_WRITE`

                    Sends the low order byte of `data` to the printer at the specified port.

`_PRINTER_INIT`     Initializes the printer at the specified port.

`_PRINTER_STATUS`

                    Gets the printer's status.

### Return value

The outcome of the function call is represented in the low order byte of the Return value. The following outcomes are possible:

Bit(s)	Meaning
1	Printer timed out
2	Unused
3	IO error
4	Printer selected
5	No paper
6	Printer acknowledge
7	Printer not busy. If busy, this bit is 0.



**Example**

```
#include <bios.h>
#include <stdio.h>
main()
{
    unsigned status;
        /* get status of LPT1 */
    status=_bios_printer(_PRINTER_STATUS, 2, 0);
    printf("LPT1 status - %X\n", status);
    return(0);
}
```

```

unsigned _bios_serialcom(
    unsigned cmd,
    unsigned port,
    unsigned data);

```

---

**Header file**      bios.h

**Portability**      IBM PC.

**Multi-thread**    DOS BIOS is not re-entrant.

The function `_bios_serialcom` uses interrupt 14H to send the information in data to the specified serial port.

**data**              Specifies the information being sent to the serial port. This value can be built using the appropriate combination of the following values from each of the following groups:

<code>_COM_110</code>	110 baud
<code>_COM_150</code>	150 baud
<code>_COM_300</code>	300 baud
<code>_COM_600</code>	600 baud
<code>_COM_1200</code>	1200 baud
<code>_COM_2400</code>	2400 baud
<code>_COM_4800</code>	4800 baud
<code>_COM_9600</code>	9600 baud

`_COM_NOPARITY`    No parity

`_COM_EVENPARITY` Even parity

`_COM_ODDPARITY`   Odd parity

`_COM_CHR7`              7 data bits

`_COM_CHR8`              8 data bits

`_COM_STOP1`            1 stop bit

`COM_STOP2`            2 stop bits

**port**              specifies the serial port. 0 represents COM1, 1 represents COM2, etc.

cmd specifies the services requested from the serial port:

\_COM\_INIT initializes port to specified parameter values.

\_COM\_SEND sends data to the specified port.

\_COM\_RECEIVE receives a character from the specified port.

\_COM\_STATUS returns the status of the specified port.

If the value of cmd is \_COM\_RECEIVE or \_COM\_STATUS, the value of data is not used.

### Return value

The function returns status information in the high order byte of its Return value. The low order byte's value depends on the value of cmd.

The high-order bits are interpreted as follows:

Bit	Meaning if On
8	data ready
9	overflow error
10	parity error
11	framing error
12	break detected
13	Empty transmit holding register
14	Empty transmit shift register

The value of the low-order byte depends on the value of the cmd argument, as follows:

### Value Returned Value

\_COM\_INIT The low-order byte provides the following information:

Bit	Meaning
0	Clear to send (CTS) changed
1	Data set ready changed
2	Trailing edge ring detector
3	Receive line signal detector changed
4	Clear to send
5	Data set ready
6	Ring indicator
7	Received line signal detect

<code>_COM_SEND</code>	If bit 15 is set, this indicates an error. If it is clear, the Return value equals the byte written.
<code>_COM_RECEIVE</code>	If no error, the value read is in the low-order byte; if there was an error, at least one of the bits in the high-order byte will be set.
<code>_COM_STATUS</code>	Same as for <code>COM_INIT</code> .

### Example

```
#include <bios.h>
main()
{
    _bios_serialcom(
        _COM_INIT,
        0,
        _COM_CHR7|_COM_NOPARITY);
        /* set COM1 to          */
        /* 7 data bits, no parity */
    return(0);
}
```

## **unsigned \_bios\_timeofday(unsigned mode, long \*time);**

---

**Header file**            bios.h

**Portability**           IBM PC.

The \_bios\_timeofday function uses interrupt 1AH to read or set the system clock value.

**time**                    specifies the current number of clock ticks.

**mode**                   Specifies whether to set the clock to the value specified in time or whether to read the system clock and store the count in time. The values of mode to accomplish these tasks are \_TIME\_SET\_CLOCK and \_TIME\_GET\_CLOCK, respectively.

### **Return value**

When setting the clock, there is no Return value. When reading the clock, a return value of 0 indicates that midnight has not been passed since the clock was last read; a return value of 1 indicates that midnight has been passed.

### **Example**

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

main()
{
    long current_time;

    current_time=_bios_timeofday(0, 0L);
    /* get status of LPT1 */
    printf("Ticks since midnight - %ld\n",
        current_time);
    return(0);
}
```

```
void *bsearch(  
    const void *key,  
    const void *base,  
    size_t num,  
    size_t width,  
    int (*compare)(const void *e1, const void *e2));
```

---

A

**Header file**            search.h and stdlib.h

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

**Variants**            hbsearch

**Portability**         ANSI

**Multi-thread**        None.

The function bsearch performs a binary search on a sorted array.

key                    is a pointer to the element being sought.

base                   is a pointer to the base of the array to be searched.

num                    is the number of array elements.

width                  is the size of each element.

**compare**    is a pointer to the user supplied function that will compare the array elements. The function will be passed pointers to two elements, e1 and e2. They must be compared and the following integer values returned:

If  $e1 < e2$             Return value  $< 0$

If  $e1 = e2$             Return value  $= 0$

If  $e1 > e2$             Return value  $> 0$

### Return value

The function returns a pointer to the first matching element if found, otherwise the value NULL is returned.

### Example

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

#define NUMBER_OF_ELEMENTS BUFFER_SIZE

int compare(void *n1, void *n2);

extern *data[];

main()
{
    double key=3.142;
    char *result;

    /* search for key in array data */

    result=bsearch(
        &key,
        data,
        NUMBER_OF_ELEMENTS,
        sizeof(double),
        compare);
    if(result)
        printf("Key Found\n");
    return(0);
}

int compare(void *n1, void *n2) {
    return(
        (int)((*(double *)n1)) -
        *(double *)n2));
}
```

---

**double cabs(struct complex z);**

---

---

**long double cabsl(z) (hypotl((z).x, (z).y));**

---

<b>Header file</b>	math.h
<b>See also</b>	abs, fabs, labs, hypot.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The `cabs` macro calculates the absolute value of the complex number `z`. This result is equivalent to `hypot(z.x, z.y)`. The function `cabsl` is equivalent to `hypotl(z.x, z.y)`. Both are implemented as macros.

The complex structure is defined as follows:

```
struct complex {  
    double x; /* real component */  
    double y; /* imaginary component */  
};
```

### Return value

The function returns  $\sqrt{z.x^2 + z.y^2}$ .

### Example

```
#include <stdio.h>  
#include <math.h>  
  
main()  
{  
    struct complex complex_number;  
    double result;  
  
    complex_number.x=3.5;  
    complex_number.y=4.0;  
    result=cabs(complex_number);  
    printf("Absolute value is %g\n", result);  
    return(0);  
}
```



---

**void \*calloc(size\_t number, size\_t size);**

---

**A****Header file**           stdlib.h and alloc.h**Variants**            \_fcalloc, \_ncalloc.**See also**            free, malloc, realloc, \_heapchk, \_heapset.**Portability**         ANSI**Multi-thread**        Far heap only protected by semaphore

The function calloc allocates a specifies number of elements, each of size bytes, from the heap. Each element is initialized to zero.

**number**               specifies how many elements are to have storage allocated for them.

**size**                 specifies the size of each element.

**Return value**

The function returns a pointer to the allocated storage. If there is insufficient memory available or if either num or size is zero, the value NULL is returned and errno is set to ENOMEM.

**Example**

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

double *allocate_reals(int number)
{
    double *pointer;

    pointer=calloc(number, sizeof(double));
    if(pointer == NULL) /* error */
        perror("Real Number Allocation");

    return(pointer);
}
```

---

**double ceil(double x);**

---

**A**

---

**long double ceill(long double x);**

---

**Header file**          math.h**See also**            floor.**Portability**        ANSI**Multi-thread**      None.

The function `ceil` rounds the double value `x` towards infinity, to the nearest integer. The function `ceill` is the same as `ceil` but returns a long double.

**Return value**

The function returns the double result of rounding. There is no error return.

**Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double x;
    x=ceil(21.79);
    printf("ceil of 21.79 is %g\n", x);
    return(0);
}
```

## **long double ceill(long double x);**

---

**Header file**           math.h

**See also**             floor.

**Portability**         ANSI

**Multi-thread**       None.

The function `ceil` rounds the double value `x` towards infinity, to the nearest integer. The function `ceill` is the same as `ceil` but returns a long double.

### **Return value**

The function returns the double result of rounding. There is no error return.

### **Example**

```
#include <stdio.h>
#include <math.h>
main()
{
    double x;
    x=ceil(21.79);
    printf("ceil of 21.79 is %g\n", x);
    return(0);
}
```

## **char \*cgets(char \*s);**

---

<b>Header file</b>	conio.h
<b>See also</b>	getch, getche.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Not re-entrant with Turbo C window module.

Reads a string of characters directly from the console.

s[0] must contain the maximum number of characters to be read. s[1] will contain the actual length of the string. The string will be stored starting at s[2].

cgets reads characters until a carriage return/line feed combination is read or the specified maximum number of characters is read. If a CR/LF combination is read, it is replaced by the terminating \0 character.

Therefore, the array s must be large enough to accommodate the maximum length byte, the actual length byte, a terminating \0 character plus the length of the string.

### **Return value**

The function returns s[2], a pointer to the start of the actual string. There is no error return.

### **Example**

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

char *store_string()
{
    char buffer[131];
    char *result;
    char *new_string;

    buffer[0]=128; /* max number of characters */
    result=cgets(buffer); /* get string */
    new_string=malloc(buffer[1]+1); /* allocate */
    /* characters actually read */
    strcpy(new_string, result);
    return(new_string);
}
```

---

**void \_chain\_intr(void ( interrupt far \*handler)());**

---

**Header file**            dos.h**See also**             \_dos\_getvect, \_dos\_keep, \_dos\_setvect**Portability**         IBM PC**The function**        \_chain\_intr chains one interrupt handler to another.**handler**            points to the interrupt handler to which the program will chain.

The environment for the new handler will appear as if the handler had been called directly after the interrupt occurred Æ i.e., as if no other handler had been invoked. This is possible (and desirable) because the address to which the interrupt handler(s) will return will already be on the stack when the handler(s) return.

**Return value**

None.

**void change(  
wintype win,  
abscoord X1, abscoord Y1,  
abscoord X2, abscoord Y2 );**

---

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function change from the TopSpeed window module alters the size and/or position of the specified window. The new coordinates determine the changes made to the window.

win	specifies the window being changed.
X1,Y1	coordinates of the point at the upper left corner of the window.
X2,Y2	coordinates of the point at the lower right corner of the window.

The coordinates are variables of type abscoord, which is defined in window.h as a synonym for unsigned.

The contents of the window will be moved with it. If the window is expanded, blanks are filled in; if the window is contracted, text is clipped.

### **Return value**

None.

### **Example**

```
#define _JPI_WIN_  
#include <conio.h>  
#include <string.h>  
#include <stdlib.h>
```

```
windex WD1={
    10, 2, 60, 8, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray}

wintype W1;

windex WD2={
    40, 6, 75, 18, Yellow, Red,
    FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
    LightGray, Blue};

wintype W2;

main()
{
    int x1, x2, y1, y2;

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    W2 = windowopen(&WD2) ; /* open window */
    setttitle(W2, "Window 2", CenterUpperTitle);
    cprintf("Hello World sent to window 2\n");

    x1=WD2.X1;
    x2=WD2.X2;
    y1=WD2.Y1;
    y2=WD2.Y2;
    while((x1 >= 0) && (y1 >= 0))
    {
        /* change window position */
        change(W2, x1, y1, x2, y2);
        delay(500);
        -x1;
        -x2;
        -y1;
        -y2;
    }
    getch();
    windowclose(W1);
    windowclose(W2);
    return(0);
}
```

## **int chdir(const char \*path);**

---

<b>Header file</b>	dir.h
<b>See also</b>	mkdir, getcwd, rmdir, system.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant.

The function `chdir` changes the current working directory to that specified by `path`.

This function can change the current working directory for any drive, but the default drive cannot be changed without a call to `system` or `setdisk`.

### **Return value**

The function returns 0 if successful. If the path name could not be found a value of -1 is returned and `errno` is set to `ENOENT`.

**OS2:** The function can be used to change the working directory on any drive, although the default drive is not changed. The working drive is local to a process and not system wide.

### **Example**

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

main()
{
    char buffer[131];
    char *result;

    buffer[0]=128;
    result=cgets(buffer); /* get new path */
    chdir(result);  r/* change directory to it */
    return(0);
}
```



## **int chmod(const char \*path, int mode);**

**U**

<b>Header file</b>	io.h (stat.h contains types)
<b>See also</b>	access, creat, fstat, open, stat.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant.

Changes the file specified by path to access type mode.

path specifies the file whose mode is to be changed.  
 mode specifies the new mode. This can be any of the following:

S_IWRITE	Writing permitted
S_IREAD	Reading permitted
S_IWRITE S_IREAD	Reading and writing permitted

Under DOS all files have read permission, so S\_IWRITE and S\_IREAD are equivalent.

### **Return value**

The function returns 0 if successful. If the path name could not be found a value of -1 is returned and errno is set to ENOENT.

### **Example**

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

main(int argc, char *argv[])
{
    if(argc != 2)
        abort();
    if(chmod(argv[1], S_IREAD) == -1)
        /* checks for existence */
        printf("File %s not found\n", argv[1]);
    else
        printf("Mode changed\n");
    return(0);
}
```

## **int \_chmod(const char \*path, int action, int attr);**

---

<b>Header file</b>	dos.h and io.h
<b>See also</b>	chmod, _creat
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant.

Sets or gets the access mode of the file specified by path.

path	specifies the file whose mode is to be changed.
action	specifies whether to set or get the mode. If action is 0, the function gets the access mode information; if function is 1, the function sets the access mode to the value specified by attr. This can be any of the following, as defined in dos.h:
FA_RDONLY	Read only
FA_HIDDEN	Hidden file
FA_SYSTEM	System file.

### **Return value**

The function returns the value of the access word if successful; otherwise the function returns -1 and sets errno to one of the following:

ENOENT	path or file name not found.
EACCES	access denied.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <stat.h>
main(int argc, char *argv[])
{
    if(argc != 2)
        abort();
    if(_chmod(argv[1], 0) == -1)
        printf("File %s not found\n", argv[1]);
    else
        printf("Mode changed\n");
    return(0);
}
```

## **int chsize(int handle, long size);**

**U**

<b>Header file</b>	io.h
<b>See also</b>	close, creat, open.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function `chsize` changes the length of the specified file to `size`. If the file is extended, null characters are appended. If the file is truncated, the data from the new end of the file to the original end of the file are lost.

`handle` specifies the handle for the file whose length is to be changed.

`size` specifies the new file size.

The file must be opened in a mode that permits writing.

### **Return value**

The function returns 0 if the size was successfully changed. If an error occurred, the value -1 is returned and `errno` is set to one of the following values:

EACCES	File is read-only or file is locked (DOS 3.0 and higher)
EBADF	Invalid file handle
ENOSPC	No space left on device

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <stat.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
    int fh;
    long file_size;
    if(argc != 3)
        abort();
    fh=open(
        argv[1],
        O_RDWR|O_CREAT,
        S_IREAD|S_IWRITE);
    if(fh < 0)/* create file */
    {
        printf("Could not create %s\n", argv[1]);
        abort();
    }
        /* get file size */
    sscanf(argv[2], "%ld", &file_size);
        /* change file size */
    if(chsize(fh, file_size))
        printf("Error occurred\n");
    close(fh);
    return(0);
}
```

## **unsigned \_clear87(void);**

---

**Header file**            float.h

**See also**             \_fpreset, \_control87, \_status87.

**Portability**          8086 Family

**Multi-thread**        None.

The function \_clear87 clears the floating point coprocessor status word.

### **Return value**

Returns the previous value of the coprocessor status word. The bits of the **Return value** are defined in the include file float.h.

### **Example**

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

clear_fp_status(void) /* print and reset status word */
{
    printf(
        "Status was %.4X, Now Reset\n",
        _clear87());
    return;
}
```

**void clearerr(FILE \*st);****A**

**Header file**           stdio.h

**See also**            eof, feof, ferror, perror, rewind.

**Portability**       ANSI

**Multi-thread**       Stream access is controlled by semaphore.

The function `clearerr` clears the error and EOF flags on stream `st`. Once an error has occurred on a stream, all operations on that stream will continue to return an error until `clearerr` or `rewind` are called.

**Example**

```
#include <stdio.h>

restore_stream(FILE *f)
{
    if(ferror(f)) /* check for error */
    {
        clearerr(f); /* clear error */
        printf("Error cleared\n");
    }
    return;
}
```

---

**void far \_clearscreen(short area);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getbkcolor, _setbkcolor
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_clearscreen` from the graphics module erases the contents of the specified area by filling the area with the current background color.

**area** specifies the area of the screen to be cleared. This can have any of the following values:

<code>_GCLEARSCREEN</code>	clear the entire screen.
<code>_GVIEWPORT</code>	clear the current viewport.
<code>_GWINDOW</code>	clear the current text window.

**Return value**

None.

**Example**

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

main()
{
    short X1=25, Y1=10, X2=500, Y2=200;

    _setvideomode(_ERESNOCOLOR);
    _ellipse(_GBORDER, X1, Y1, X2, Y2);
    _setviewport(10, 5, 400, 120);
    _rectangle(_GBORDER, 0, 0, 390, 115);
    _ellipse(_GFILLINTERIOR, 2, 5, 190, 80);
    getch();
    _clearscreen(_GVIEWPORT);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**time\_t clock(void);****A**

**Header file**           time.h

**See also**             time, difftime.

**Portability**          ANSI. The macro CLK\_TCK may also be used, although this is no longer supported by ANSI C.

**Multi-thread**        None.

The function clock returns a number that gives the number of seconds elapsed since the start of the current process, if divided by the macro CLOCKS\_PER\_SEC.

**Example**

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

main()
{
    double duration;

    getch();
    duration= ((double) clock())/CLOCKS_PER_SEC;      /* get seconds since
start of process */
    printf("%g seconds to press a key\n",
        duration);
    return;
}
```



## **int close(int handle);**

## **U**

<b>Header file</b>	io.h
<b>See also</b>	creat, dup, dup2, open.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Low level I/O functions are <i>not</i> controlled by semaphore, so the user must control access to handles among threads.

The function close closes file the associated with handle. This function should only be used to close a file that was opened with creat, dup, dup2, or open.

### **Return value**

The function returns 0 if successful. If an error occurred, the value -1 will be returned and errno will be set to EBADF, indicating that an invalid handle was passed to the function.

### **Example**

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

int find_file(char *path)
{
    /* inefficient way */
    /*to detect a file */

    int fh;
    fh=open(path, O_RDONLY);
    if(fh < 0)
        return(0);
    close(fh); /* close file if opened */
    return(1);
}
```

## **int \_close(int handle);**

---

<b>Header file</b>	io.h
<b>See also</b>	_creat, _open, _read, _write
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_close` closes the file specified by `handle`. The value of `handle` must be from a call to `_creat`, `creatnew`, `creattemp`, `_dup`, `_dup2`, or `_open`.

### **Return value**

If successful, the function returns 0; otherwise the function returns -1. If `handle` is not associated with a valid file name, `errno` is set to `EBADF` (bad file number).

### **Example**

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

main()
{
    int fh;

    fh=_open("TEMP.***", 2);
    if(fh != -1) {
        _close(fh);
        printf("TEMP.*** exists\n");
    }
    return(0);
}
```

**void clreol(void);****W**

<b>Header file</b>	window.h
<b>See also</b>	delline
<b>TopSpeed</b>	DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function clreol from the TopSpeed window module clears from the current cursor position to the end of line.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 16, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()
{
    int n=1;
    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    while(n < 14)
    {
        gotoxy(1, n);
        cputs("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
        ++n;
    }
    gotoxy(2, 4);
    printf("Press key to clear to end line ");
    getch();
    clreol();
    getch();
    return(0);
}
```

---

**void clreol(void);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	clrscr
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `clreol` from the Turbo C window module clears the screen from the cursor position to the end of line. This function acts within the boundaries of the current window.

**Return value**

None.

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#define COLS 40
#define LINES 8

main()
{
    int n=0;
    char block[COLS+2];

    while(n <= COLS){
        block[n]='X'; ++n;
    }
    block[n]='\0';
    clrscr();

    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    n=0;
    while(n < LINES)
    {
        cprintf(block); /* output to new window */
        ++n;
    }
    gotoxy(1, 4);
    cprintf("Press key to delete this line");
    getch();
    clreol();
    return(0);
}
```

---

**void clrscr(void);**

---

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `clrscr` from the TopSpeed window module clears the current window.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 16, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()

{
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    cprintf("Press a key to clear window ");
    getch();
    clrscr();
    return(0);
}
```

**void clrscr(void);**

---

**T****Header file** conio.h**See also** clreol**Portability** IBM PC and compatibles.**Multi-thread** Module is not re-entrant.

The function `clrscr` from the Turbo C window module clears the screen and sets the cursor position to 1,1. This function acts within the boundaries of the current window.

**Return value**

None.

**Example**

```
#define _CLIP_WIN_  
#include <conio.h>  
  
main()  
{  
    clrscr();  
    return(0);  
}
```

```
extern unsigned char (* CnsHandler)(
    unsigned code,
    unsigned IP,
    unsigned CS);
```

---

**Header file**        rtcheck.h

**Portability**        TopSpeed

**Multi-thread**       None.

The CnsHandler function pointer variable is a global function that handles the standard runtime checks specified below. By default CnsHandler invokes exit. The user may to replace the default CnsHandler with his/her own function by assigning to this variable.

When an error occurs the handler is passed an error code plus the value of the code segment and instruction pointer at the point where the error was detected.

The handler must returns a code which determines whether the program continues or terminates.

IP	The value of the instruction pointer at the point where the error was detected.
CS	The value of the code segment at the point where the error was detected.
code	This parameter defines the error which was detected. It is provided by the runtime check mechanism

whenCnsHandler is invoked. It has one of the following values:

Value of code	Meaning
_CNS_STACK	Stack -overflow
_CNS_NILPTR	Nil Ptr dereference
_CNS_INDEX	Index out of range
_CNS_OVERFLOW	Integer overflow
_CNS_RANGE	Integer out of range
_CNS_RETURN	No return in function
_CNS_CASE	No matching case
_CNS_DIVZERO	Long integer divide by zero
_CNS_PVC	Pure virtual function called (C++)

**Return value**

The function must return one of the following two constants defined in `rtcheck.h`:

**Return value****Meaning**`_CNS_CONTINUE`

Continue execution

`_CNS_TERMINATE`Terminate execution with call to `exit`



## **unsigned \_control87(unsigned new, unsigned mask);**

---

<b>Header file</b>	float.h
<b>See also</b>	_fpreset, _clear87, _status87.
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	None.

The function `_control87` sets the floating point coprocessor control word.

<code>new</code>	specifies the bit values for the new control word.
<code>mask</code>	specifies the mask to be used for setting the new control word bits.

If the value of `mask` is zero, then the function only returns the control word. If `mask` is nonzero then, for any bit that is set in `mask`, the corresponding bit in `new` is used to set the control word. I.e.,

```
control = ((control & ~mask) | (new & mask))
```

### **Return value**

The function returns the previous value of the control word. The bits are defined in the include file `float.h`.

### **Example**

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

/* reset control word */
void reset_control(void)
{
    _control87(CW_DEFAULT, UINT_MAX);
    return;
}
```

```
void convertcoords(  
    wintype win,  
    relcoord X,  
    relcoord y,  
    abscoord *X0,  
    abscoord *Y0);
```

---

**W**

**Header file**            window.h

**Portability**           TopSpeed DOS/OS2

**Multi-thread**        Module is re-entrant and requires no additional locking for single function calls.

The function `convertcoords` from the TopSpeed window module converts window relative coordinates to absolute screen coordinates.

**win** specifies the window used for the relative coordinates.

**X,Y**                    specify the relative coordinate.

**X0,Y0**                points to the absolute coordinate.

The resulting coordinates are returned in the `X0` and `Y0` arguments.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 16, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;
main()

{
    relcoord y;
    abscoord X, Y;
    int n=0;
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    while(n < 5)
    {
        y=wherey()+1;
        gotoxy(1, y);
        convertcoords(W1, 1, y, &X, &Y);
        cprintf(
            "Output at physical coordinates %d, %d",
            X, Y);
        ++n;
    }
    getch();
    windowclose(W1);
    return(0);
}
```

## **unsigned long coreleft(void);**

---

<b>Header file</b>	alloc.h
<b>Variants</b>	nearcoreleft, farcoreleft
<b>Multi-thread</b>	The value returned is valid only until another thread accesses the heap.

The function coreleft determines the number of bytes left in the heap.

**Note**     **This returns an unsigned in the Small/Medium models and unsigned long in other models.**

### **OS2 Considerations:**

In compact, large, xlarge, and MThread models, coreleft always returns a large value. The value returned by farcoreleft may be inaccurate in a network or a multi-process environment such as OS2.

### **Return value**

The function returns the number of bytes left in the heap.

### **Example**

```
int check_storage()
{
    long space;

    space=coreleft();
    if(space < 100)
        /* are there at least */
        /* 100 bytes in heap */
    {
        fprintf(stderr, "Out Of Storage\n");
        exit(0);
    }
    return(0);
}
```

## double cos(double x);

## A

**Header file** math.h

**See also** sin, sinh, tan, tanh, cosh, matherr.

**Portability** ANSI

**Multi-thread** None.

The function cos returns the cosine of  $x$ , an angle in radians.

### Return value

The function returns  $\cos(x)$ . If  $x$  is greater than  $2^{53}$  a total loss of significance occurs. A TLOSS message is printed to stderr, errno is set to ERANGE, and the value 0 is returned.

Error handling can be altered by installing a different matherr function.

### Example

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    result=cos((double) num);
    /* calculate cosine of input */
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf(
            "cosine of %g is %g\n", num, result);
    return(0);
}
```

---

**double cosh(double x);**

---

**A**

---

**long double coshl(long double x);**

---

**Header file**          math.h**See also**            sin, sinh, tan, tanh, cos.**Portability**        ANSI**Multi-thread**       None.

The function cosh returns the hyperbolic cosine of x, an angle in radians.

**Return value**

cosh returns the function result. If the result is too large, cosh returns HUGE\_VAL and sets errno to ERANGE.

**Example**

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate cosh of input */
    result=cosh((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("cosh of %g is %g\n", num, result);
    return(0);
}
```

## **struct country \* country(int ccode, struct country \*cptr);**

---

**Header file** dos.h

**Portability** DOS version 3.0 and later

**Multi-thread** DOS is not re-entrant

Returns country dependent information (date, time, currency, etc.)

**ccode** specifies the country, and must be a nonnegative value.

**cptr** points to struct containing the country information, defined in dos.h:

```
struct                country{
    chnt                co_date;
    char                co_curr[5];
    char                co_thsep[2];
    char                co_dese[2];
    char                co_dtsep[2];
    char                co_tmsep[2];
    char                co_currstyle;
    char                co_digits;
    char                co_time;
    long                co_case;
    char                co_dasep[2];
    char                co_fill[10];
};
```

The co\_date member can take the following three values:

- |   |                                    |
|---|------------------------------------|
| 0 | U.S. style : month, day, year.     |
| 1 | European style : day, month, year. |
| 2 | Japanese style : year, month, day. |

The co\_currstyle member can take the following values, which determine how the currency symbol is placed with respect to the number.

- |   |  |
|---|--|
| 0 | symbol precedes value, no space between symbol and number. |
|---|--|

- 1 symbol follows value, no space between number and symbol.
- 2 symbol precedes value, space between symbol and number.
- 3 symbol follows value, space between number and symbol.

**Return value**

The function returns the value of cptr if successful; otherwise it returns NULL.



## **int cprintf(const char \*format, ...);**

---

<b>Header file</b>	conio.h
<b>See also</b>	fprintf, printf, sprintf, vprintf
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Not re-entrant if using Turbo C window module.

The function `cstdio` provides formatted output directly to the console.

`format` is a string consisting of ordinary characters, escape sequences and format specifications corresponding to the list of arguments passed to `cstdio` after `format`.

Ordinary characters and escape sequences are output directly in order from left to right. See `printf` for information on how format specifications are processed.

**Note:** `cstdio` does provide linefeed to carriage return/linefeed translation.

`cstdio` will wrap around its output within the currently defined window set by any window module window.

### **Return value**

The function returns the number of characters output.

### **Example**

```
#define _JPI_WIN_ /* select window module */
#include <conio.h>
main()
{
    int x=0;
    int y=0;
    gotoxy(x, y); /* set cursor position */
    cprintf("Top Left Corner, %d, %d", x, y);
    x=79;
    y=24;
    gotoxy(x, y); /* set cursor position */
    cprintf("Bottom Right Corner, %d, %d", x, y);
    return(0);
}
```

## **void cputs(const char \*s);**

---

**Header file**            conio.h

**See also**             putchar, cgets, puts, putc

**Portability**         UNIX/DOS/OS2

**Multi-thread**        Not re-entrant when using Turbo C window module.

The function `cputs` writes a null terminated string `s` to the console.

**Note**        **`cputs` does not automatically append a carriage return/linefeed combination to the end of the output.**

### **Return value**

None.

### **Example**

```
#define _JPI_WIN_ /* select window module */
#include <conio.h>

main()
{
    gotoxy(0, 0); /* set cursor position */
    cputs("Top Hand Corner"); /* output text */
    gotoxy(79, 24);
    cputs("Bottom Right Hand Corner");
    return(0);
}
```

**int creat(const char \*path, int mode);****U****Header file** io.h (types in stat.h)**See also** access, close, \_creat, creatnew, fstat, open, stat**Portability** UNIX/DOS/OS2

The function creat either creates a new file specified by path or opens and truncates an existing one.

If the file does not exist, it is created with the permission setting mode.

If the file exists and its permission setting allows writing, it will be truncated to length 0. Access will be allowed for subsequent reading and writing, depending on the value of mode.

**path** specifies the complete name for the file being opened or created.

**mode** specifies how the file is to be treated. mode can be any of the following, as defined in stat.h:

**S\_IWRITE** Writing allowed for the file.

**S\_IREAD** Reading allowed for the file.

**S\_IWRITE|S\_READ** Reading and writing allowed.

creat applies the current file permission mask (\_fmask) to mode before setting the permission.

Under DOS all files have read permission, so modes S\_IWRITE and S\_IREAD are equivalent.

**Return value**

The function returns the handle of the created or opened file if successful. If an error occurred, a value of -1 is returned and errno is set to one of the following values:

**EACCES** path specifies a read-only file or directory.

**Example** No more handles are available.

**ENOPATH** directory not found.

A call to open with O\_CREAT and O\_TRUNC in the access argument has the same effect as a call to creat, and is preferable for future portability.

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

main()
{
    int fh;

    fh=creat("TEMP. $$$", S_IREAD|S_IWRITE);
    if(fh != -1)
    {
        close(fh);
        printf("TEMP. $$$ created with read and write access\n");
    }
    return(0);
}
```

## **int \_creat(char \*path, int permission);**

---

**Header file**           io.h

**See also**             creat, \_creat\_new

**Portability**         DOS

The function `_creat` creates a new file or truncates an existing file when opening it.

`path`                   specifies the complete name for the file.

`permission`           specifies how the file is to be handled. This argument can have any of the following values:

`FA_RDONLY`           file is read only.

`FA_HIDDEN`           file is hidden.

`FA_SYSTEM`           file is a system file.

`_creat` works like `_creat_new` except that `_creat_new` does not truncate an existing file.

### **Return value**

If successful, the function returns the handle associated with the file; otherwise it returns -1 and `errno` is set to one of the following:

`EEXIST`               file exists

`ENOENT`              path was not found

`EMFILE`              too many open files

`EACCES`              access denied

### **Return value**

The function returns a handle associated with the newly created file.

### **Example**

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

main() {
    int fh;

    fh=_creat("TEMP.***", 0);
    if(fh != -1)
    {
        _close(fh);
        printf("TEMP.*** created or truncated\n");
    }
    return(0);
}
```

---

**int creatnew(const char \*path, int permission);**

---

**U****Header file**        io.h**See also**            creat, \_creat\_new, creattemp, dup, \_open**Portability**        DOS version 3.0 and later

The function creatnew creates a new file.

path                specifies the complete name for the file.

permission        specifies how the file is to be handled, and can be any of:

FA\_RDONLY        file is read only.

FA\_HIDDEN        file is hidden.

FA\_SYSTEM        file is a system file.

creatnew works like creat except that creatnew returns an error if the file exists, and leaves the file untouched.

**Return value**

If successful, the function returns the handle associated with the file; otherwise it returns -1 and errno is set to one of the following values:

EEXIST            file exists

ENOENT           path was not found

EMFILE           too many open files

EACCES           access denied

**Example**

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

main()
{
    int fh;

    fh=creatnew("TEMP. $$$", FA_RDONLY);
    if(fh != -1)
    {
        _close(fh);
        printf(
            "New read-only file TEMP. $$$ created\n");
    }
    else
        printf("TEMP. $$$ already exists\n");
    return(0);
}
```

## **int \_creat\_new(const char \*path, int permission);**

---

**Header file**     io.h

**See also**        creat, creatnew, dup, dup2, open, chmod

**Portability**     DOS version 3.0 and later. MSDOS not re-entrant.

Creates a new file. Works like \_creat except but does not truncate an existing file.

path                specifies the complete name for the file.

permission        says how the file is to be handled. Values can be:

FA\_RDONLY        file is read only.

FA\_HIDDEN        file is hidden.

FA\_SYSTEM        file is a system file.

### **Return value**

If successful, the function returns the handle associated with the file; otherwise it returns -1 and errno is set to one of the following values:

EEXIST            file exists

ENOENT           path was not found

EMFILE            too many open files

EACCES            access denied

### **Example**

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

main()
{
    int fh;
    fh=_creat_new("TEMP.***", FA_RDONLY);
    if(fh != -1)
    {
        _close(fh);
        printf("New read-only TEMP.*** created\n");
    }
    else
        printf("TEMP.*** already exists\n");
    return(0);
}
```

---

**int createmp(const char \*path, int permission);**

---

**U****Header file**           io.h**See also**             close, \_creat, creat, creatnew, dup, open**Portability**         DOS version 3.0 and later. MSDOS not re-entrant.

The function createmp creates a unique file in the directory specified by path. The system will create a unique file name automatically.

path                   specifies the directory in which the file should be created. This must end with a backslash (\).

permission            specifies how the file is to be handled. This argument can have any of the following values:

FA\_RDONLY             file is read only.

FA\_HIDDEN             file is hidden.

FA\_SYSTEM             file is a system file.

A file created with this function is created in either text or binary mode

(depending on the translation mode, as specified by \_fmode).

Files created with createmp are not deleted automatically when the program ends.

**Return value**

If successful, the function returns the handle associated with the file; otherwise the function returns -1 and sets errno to one of the following values:

ENOENT                path was not found

EMFILE                too many open files

EACCES                access denied



**Example**

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

main() {
    int fh;

    fh=createmp("C:\\TSC\\", FA_RDONLY);
    if(fh != -1)
    {
        _close(fh);
        printf("New read-only file created\n");
    }
    else
        printf("error creating file.\n");
    return(0);
}
```

## **int cscanf(const char \*format, ...);**

---

**Header file**            conio.h

**See also**             fscanf, scanf, sscanf, vfscanf

**Portability**         UNIX/DOS/OS2

**Multi-thread**        Not re-entrant when using Turbo C window module.

The function cscanf provides formatted input directly from the console.

**format**                is a string consisting of ordinary characters, escape sequences and format specifications corresponding to the list of arguments passed to cscanf after format.

See scanf for information on how format specifications are processed.

### **Return value**

The function returns the number of fields that were converted and assigned to variables. If no fields were assigned, the function returns 0. If an attempt was made to read at the end of the file, the function returns EOF.

### **Example**

```
#define _JPI_WIN_ /* select window module */
#include <conio.h>

main()
{
    int x=0;
    int y=0;

    clrscr();
    gotoxy(0, 0);
    cscanf("%d %d", &x, &y);
    gotoxy(0, 0);
    cprintf("Cursor at %d, %d", x, y);
    gotoxy(x, y);
    getch();
    return(0);
}
```

---

**char \*ctime(const time\_t \*tt);**

---

**A****Header file**           time.h**See also**             asctime, ftime, gmtime, localtime, time**Portability**         ANSI**Multi-thread**       In a Multi-thread program, ctime will destroy only the result of a previous call from the same thread.

Converts time in seconds, stored as a time\_t to a formatted character string. The tt value would normally be obtained from a call to time which returns the number of seconds elapsed since 00:00:00 GMT, January 1st 1970.

The string produced by ctime comprises 26 characters and has the following format:

Fri Aug 26 18:36:17 1988\n\0

**Note:**     **ctime uses a static buffer for the character string, so a call to ctime will destroy the result of a previous call to asctime or ctime.**

ANSI does not support dates prior to 1970. Such values in the structure tm will produce unpredictable results.

**Return value**

The function returns a pointer to the formatted character string. There is no error

**Return value.****Example**

```
#include <stdio.h>
#include <time.h>
main()
{
    time_t tt;
    char *time_str;
    time(&tt); /* get time in seconds */
    time_str=ctime(&tt); /* convert to string */
    printf(time_str); /* output time string */
    return(0);
}
```

## **void ctrlbrk(int (\*cbhandler)(void));**

---

**Header file**            dos.h

**See also**             getcbkr, signal

**Portability**         DOS.

The function `ctrlbrk` sets the value of the control-break handler. The function accomplishes its task by modifying the interrupt vector for INT 23H to call the function to which `cbhandler` points.

`ctrlbrk` does not call the handler function directly; rather `ctrlbrk` modifies the DOS interrupt handler to do so.

Installing a control break handler with `ctrlbrk` will interfere with the handling of SIGINT and SIGTERM signals.

### **Return value**

There is no explicit return. The handler can use `longjmp` to return to an arbitrary location in the program.

```
short far _cube(  
    short fill,  
    short top,  
    short x1, short y1,  
    short x2, short y2,  
    short depth);
```

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_arc, _ellipse, _rectangle
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_cube` from the graphics module draws a cuboid shape, using the current color. This shape is useful for creating bar charts.

`fill` specifies whether to fill the figure. This argument may be one of the following constants:

`_GFillInterior` fill shape with current fill mask.

`_GBorder` do not fill the shape.

`top` specifies whether a top is to be drawn on the cube. This argument may be one of the following constants

`_GNotop` No top is drawn, allowing cubes to be stacked.

`_GTop` A top is drawn onto the bar, completing the cuboid.

`x1,y1` coordinates of the top left corner of the shape.

`x2,y2` coordinates of the bottom right corner of the shape.

`depth` specifies the two dimensional depth of the shape.

### Return value

The function returns a nonzero value if the shape has been drawn successfully; otherwise the function returns 0.

## Example

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

main()

{
    short x1, y1;
    short x2, y2;
    short x3, y3;
    unsigned char new_mask[8];
    struct videoconfig cv;

    _setvideomode(_ERESCOLOR);
    _getvideoconfig(&cv);
    _clearscreen(_GCLEARSCREEN);

    new_mask[0]=0x88;
    new_mask[1]=0x44;
    new_mask[2]=0x22;
    new_mask[3]=0x11;
    new_mask[4]=0x88;
    new_mask[5]=0x44;
    new_mask[6]=0x22;
    new_mask[7]=0x11;
    _setfillmask(new_mask);
    x1=19;
    y1=10;
    x2=x1;
    y2=321;
    x3=600;
    y3=y2;

    _moveto(x1, y1);
    _lineto(x2, y2);
    _lineto(x3, y3);
    _settextposition(24, 12);
    _outtext("Bar Graphs");
    _setcolor(max(1%cv.numcolors, 1));
    x1=20;
    y1=20;
    x2=59;
    y2=320;
```

```
    _cube(  
        _GFILLINTERIOR, _GTOP, x1, y1, x2, y2, 20);  
    _setcolor(max(2%cv.numcolors, 1));  
    x1=60;  
    y1=120;  
    x2=99;  
    y2=320;  
    _cube(  
        _GFILLINTERIOR, _GTOP, x1, y1, x2, y2, 20);  
    _setcolor(max(3%cv.numcolors, 1));  
    x1=100;  
    y1=80;  
    x2=139;  
    y2=320;  
    _cube(  
        _GFILLINTERIOR, _GTOP, x1, y1, x2, y2, 20);  
    _setcolor(max(4%cv.numcolors, 1));  
    x1=140;  
    y1=200;  
    x2=179;  
    y2=320;  
    _cube(  
        _GFILLINTERIOR, _GTOP, x1, y1, x2, y2, 20);  
    getch();  
    _setvideomode(_DEFAULTMODE);  
    return(0);  
}
```

void cursoroff(void);    W

## **void cursoron(void);**

## **W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls

The functions `cursoroff` and `cursoron` from the TopSpeed window module turn the cursor off and on, respectively, in the current window.

**Note:** the cursor in a particular window is visible only when the cursor is turned on *and* the window is on top.

### **Return value**

None.

### **Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 16, White, Blue,
    TRUE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()
{
    cursoroff(); /* cursor off for full screen */
    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    cprintf("Press a key to hide cursor");
    getch();
    cursoroff();
    getch();
    windowclose(W1);
    cursoron(); /* cursor on for full screen */
    return(0);
}
```



## **void Delay(int T);**

---

<b>Header file</b>	process.h
<b>Portability</b>	Top Speed DOS/OS2
<b>Multi-thread</b>	Process control function.

The function Delay delays the current process for at least T timeslices. A timeslice is approximately 1/18 second. If T is zero, rescheduling takes place without a delay, allowing another process with the same or higher priority to become active. A call such as

Delay(54);                delays the current process for about 3 seconds  
                              (3\*18 time slices).

**Return value**            None.

**Example:**                See example program in , page .

## **void delay(unsigned int tm);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	nosound, sleep, sound
<b>Portability</b>	IBM PC
<b>Multi-thread</b>	None.

The function `delay` generates a delay of `tm` milliseconds. It does *not* cause rescheduling in a multithread process

### **Return value**

None.

### **Example**

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

void wait(int seconds)
{
    printf("Waiting for %d seconds\n", seconds);
    delay(seconds * 1000);
    return;
}
```

**void delline(void);****W**

<b>Header file</b>	window.h
<b>See also</b>	inline
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls

The TopSpeed window module version of function delline deletes the line at the current cursor position. The screen below the line scrolls upward.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>
windef WD1={
    10, 2, 60, 16, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;
main()
{
    int n=1;

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    while(n < 14)
    {
        gotoxy(1, n);
        cputs("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
        ++n;
    }
    gotoxy(2, 4);
    cprintf("Press key to delete line ");
    getch();
    delline();
    getch();
    return(0);
}
```

---

**void delline(void);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	insline, window
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The Turbo C window module version of function delline deletes the line at the current position, and moves all lines below it in the current window one line up.

**Return value**

None.

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#define COLS 40
#define LINES 8

main()
{
    int n=0;
    char block[COLS+2];
    while(n <= COLS)
    {
        block[n]='X';
        ++n;
    }
    block[n]='\0';
    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);n=0;
    while(n < LINES)
    {
        cprintf(block); /* output to new window */
        ++n;
    }
    gotoxy(1, 4);
    cprintf("Press key to delete this line");
    getch();
    delline();
    return(0);
}
```

---

**double difftime(time\_t t1, time\_t t2);**

---

**A****Header file**           time.h**See also**             time.**Portability**         ANSI**Multi-thread**       None.

The function difftime calculates the difference between two times, t1 and t2.

**Return value**

The function returns the elapsed time in seconds, t1 - t2, as a double precision real number.

**Example**

```
#include <time.h>

wait_for(double seconds)
{
    time_t start;
    time_t now;

    time(&start);
    do
    {
        time(&now);
    }while(difftime(now, start) < seconds);
    /* generate delay */
    return;
}
```

**void \_directwrite(relcoord X, relcoord Y, void \*A, unsigned Len);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `_directwrite` from the TopSpeed window module writes directly to the current window at the specified location.

<b>X</b>	horizontal location in the current window.
<b>Y</b>	vertical location in the current window.
<b>A</b>	points to the information to be written. This will generally be a string.

**Len**length (in bytes) of the material to be written.

The function writes directly, beginning at the specified coordinates. There are no checks for special characters, and there is no wrap at the end of a line.

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 16, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()

{
    char *msg="Press key to exit";

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    _directwrite(2, 4, msg, strlen(msg));
    getch();
    return(0);
}
```

## **void disable(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	enable, getvect
<b>Portability</b>	80x86 Family
<b>Multi-thread</b>	Under DOS a thread is locked when interrupts are disabled.

Disables interrupts.

### **Return value**

None.

## **void \_disable(void);**

---

<b>Header file</b>	Dos.h
<b>See also</b>	_enable.
<b>Portability</b>	80x86 Family
<b>Multi-thread</b>	Under DOS a thread is locked when interrupts are disabled.

The function `_disable` disables interrupts with the CLI instruction.

### **Return value**

None.

## **void DisableBreakCheck(void)**

---

<b>Header file</b>	dos.h
<b>See also</b>	EnableBreakCheck
<b>Portability</b>	DOS only

Disables control-break checking. If checking is enabled, a program may be terminated at any point by pressing Ctrl-Break or Ctrl-C. Otherwise it may not.

### **Return value**

None.

---

**short far \_displaycursor(short flag);**

---

**M**

<b>Header file</b>	graph.h
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_displaycursor` from the graphics module specifies whether or not the graphics cursor is to be turned back on when a program exits graphics routines. (The cursor is automatically turned off when the program *enters* a graphics routine.)

<b>flag</b>	specifies whether or not to turn the cursor back on. This parameter can take either of the following values:
<code>_GCURSOROFF</code>	leave the cursor off when the graphic routine finishes.
<code>_GCURSORON</code>	turn the cursor back on when the graphic routine finishes.

**Return value**

The function returns the *previous* value of `flag`. There is no error return.

**Example**

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

main()
{
    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 50);
    _displaycursor(_GCURSORON);
    _outtext("Press key to hide cursor");
    getch();
    _displaycursor(_GCURSOROFF);
    _outtext(" ");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **div\_t div(int num, int den);**

---

**A**

**Header file**            `stdlib.h`

**See also**            `ldiv.`

**Portability**        `ANSI`

**Multi-thread**       `None.`

The function `div` divides `num` by `den`, returning the quotient in the `div_t` structure member `quot` and the remainder in structure member `rem`.

```
typedef struct{
    int quot;
    int rem;
} div_t;
```

If the denominator is zero the program will terminate with an error message.

### **Return value**

The function returns a structure of type `div_t` containing `quotient` and `remainder`.

### **Example**

```
#include <stdlib.h>

int modulus(int x, int y)
{
    div_t result;

    result=div(x, y);
    return(result.rem);
}
```

```

unsigned _dosbeginthread(
    void (* func)(void),
    unsigned st_len,
    unsigned *stck);

```

---

**Header file**            process.h and OS2.h for error definitions.

**See also**              \_dosendthread, \_dosfreestack

**Portability**           OS2 only. Use the JPI process module for portable Multi-thread programs between DOS and OS2.

The function `_dosbeginthread` starts the execution of a thread. It must be used in preference to the OS2 system call `DosCreateThread`.

`func`                    is the address of the thread function.

`st_len`                  is the size (in bytes) of the stack required for the thread. A minimum of 1000 bytes should be specified.

`stck` is the address of an object to hold the stack token. If necessary, the memory allocated for the thread's stack may be freed using `_dosfreestack` when the thread has terminated. (All memory allocated to a process is freed when that process terminates.) The library supports a maximum of 32 threads.

### **Return value**

If the thread has been successfully started, its ID will be returned.

`errno` will be set to `ERROR_INSUFFICIENT_BUFFER` if no memory was available for the stack, and the value 0 will be returned. (The value of `stck` will be undefined in that case.) If no more thread slots were available, `errno` will be set to `ERROR_NO_PROC_SLOTS` and the value 0 will be returned.

### **Example**

```

{
    return(c[thread_number]);
}
struct doserror {
    int exterror;
    unsigned char class;
    unsigned char action;
    unsigned char locus;
};

```

### **Return value**

The function returns the value in the AX register, which is the same as the value in the `exterror` member of the `doserror` structure.

**Example**

```
fh=_open("NOT_HERE. $$$", 2);

if(fh != -1)
{
    _close(fh);
    printf("NOT_HERE. $$$ exists\n");
}
else
{
    MSDOSexterr(&error_buf);
    printf("Error - %d\nClass - %d",
        error_buf.exterror, error_buf.class);
    printf("\nAction - %d\nLocus - %d\n",
        error_buf.action, error_buf.locus);
}
```

```

unsigned _dos_findfirst(
    const char *path,
    unsigned attrib,
    struct find_t *buffer);

```

---

**Header file**            dos.h

**See also**                dos\_findnext

**Portability**            DOS

**Multi-thread**          DOS is not re-entrant

The function `_dos_findfirst` returns information about the first file that matches the criteria set by the `path` and `attrib` parameters. The function uses DOS service 4EH

`path`                    specifies the file name, and may include the  and wildcard characters.

`attrib`                  specifies the file's attributes, and can take any of the following values:

`_A_NORMAL`            The file is normal Æ i.e., it can be read or written.

`_A_RDONLY`            Read only. The file cannot be created, and cannot be used for output.

`_A_HIDDEN`            The file is hidden Æ i.e., cannot be seen in a directory listing.

`_A_SYSTEM`            The file is a system file, and cannot be seen in a directory listing.

`_A_VOLID`             The file is a volume ID Æ i.e., the file must be in the root directory.

`buffer`                  points to a `find_t` structure (defined in `dos.h`) that contains the relevant information about the file.

```

struct find_t {
    char reserved[21];
    char attrib;
    unsigned wr_time;
    unsigned wr_date;
    long size;
    char name[13];
};

```

### Return value

The function returns 0 if successful; otherwise, the function returns a DOS error code, and sets `errno` to `ENONENT` (no matching path).

## Example

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

main() {
    char *path="C:\\*.\\*.*";
    struct find_t find_buf;
    int error;

    error=_dos_findfirst(
        path, _A_NORMAL,
        &find_buf); /* list normal files */
    while(!error)
    {
        printf("File - %s\\n", find_buf.name);
        error=_dos_findnext(&find_buf);
    }
    return(0);
}
```

## void \_dosendthread(void)

---

<b>Header file</b>	process.h
<b>See also</b>	_dosbeginthread.
<b>Portability</b>	OS2 only. Use the JPI process module for portable Multi-thread programs between DOS and OS2.

Terminates the execution of a thread. It must be used in preference to the OS2 system call DosExit or to a simple function return. An attempt to use this function to terminate thread 1 will be ignored.

**Note:** A call to `exit` or `_exit` terminates all threads.

### Return value

None.

### Example

```
#include <process.h>
#include <conio.h>
#include <os2.h>

void thread_function(void); int still_running(unsigned thread_number);
int volatile c[32];

main()
{
    unsigned new_thread, stack_handle;
    new_thread=_dosbeginthread(
        thread_function, 2000, &stack_handle);
    while(still_running(new_thread));
    DosSleep(0); /* allow thread to finish */
    _dosfreestack(stack_handle); /* free memory */
    exit(0);
}

void thread_function(void)
{
    while(!kbhit());
    c[_getTID()]=getch();
    _dosendthread();
}

int still_running(unsigned thread_number)
{
    return(c[thread_number]);
}
```

The function `dosexterr` determines and returns extended information about errors. The function uses interrupt 59H for this purpose.

This information is returned in the `doserror` structure to which `erbuf` points. This structure is defined in `dos.h`.

## **unsigned \_dos\_findnext(struct find\_t \*buffer);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_findfirst
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_findnext` finds the next file name that matches the name and attributes specified in a call to `_dos_findfirst`. The function uses service 4FH for this purpose.

`buffer` points to a `find_t` structure that has already been initialized by the call to `_dos_findfirst`. Information about the next match will be returned in this structure.

### **Return value**

The function returns 0 if successful; otherwise, it returns a DOS error code and sets `errno` to `ENOENT` (no matching path).

### **Example**

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

main()
{
    char *path="C:\\*.\\*";
    struct find_t find_buf;
    int error;

    /* list normal files */
    error=_dos_findfirst(
        path, _A_NORMAL, find_buf);
    while(!error)
    {
        printf("File - %s\\n", find_buf.name);
        error=_dos_findnext(&find_buf);
    }
    return(0);
}
```

## **unsigned \_dos\_freemem(unsigned seg);**

---

**Header file**            doc.h

**See also**             \_dos\_allocmem, \_dos\_setblock, ffree, free, hfree, nfree

**Portability**         DOS

**Multi-thread**        DOS is not re-entrant

Releases a block of memory, using service 49H. This block must have been allocated using \_dos\_allocmem or \_dos\_setblock. Once freed, the memory is no longer accessible to the program.

seg specifies a location returned by a call to functions \_dos\_allocmem or \_dos\_setblock.

### **Return value**

The function returns 0 if successful; otherwise, it returns a DOS error code and sets errno to ENOMEM to indicate a segment value that does not correspond to one returned by the specified functions.

### **Example**

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

main()
{
    char far *buffer;
    unsigned new_seg;
    int error;

    error=_dos_allocmem(100, &new_seg);
    if(error)
    {
        fprintf(
            stderr,
            "Error allocating memory\n");
        return(1);
    }
    buffer=MK_FP(new_seg, 0);
    printf("Memory is at %Fp\n", buffer);
    _dos_freemem(new_seg);
    return(0);
}
```



## **void \_dos\_getdate(struct dosdate\_t \*buffer);**

---

**Header file**            dos.h

**See also**            \_dos\_gettime, \_dos\_setdate, \_dos\_settime,  
                  gmtime, localtime, mktime, \_strdate, \_strtime, time

**Portability**        DOS

**Multi-thread**       DOS is not re-entrant

The function \_dos\_getdate obtains the current date. The function uses service 2AH for this purpose.

**buffer**              points to a dosdate\_t structure in which the date  
                         information is stored.

```
struct dosdate_t {  
    unsigned char day; /* 1-31 */  
    unsigned char month; /* 1-12 */  
    unsigned int year; /* 1980-2099 */  
    unsigned char dayofweek; /* 0-6, 0=Sunday */  
};
```

### **Return value**

There is no **Return value**.

### **Example**

```
#include <dos.h>  
#include <stdio.h>  
  
main()  
{  
    struct dosdate_t date;  
  
    _dos_getdate(&date);  
    printf(  
        "Date is %d-%d-%d\n",  
        date.day, date.month, date.year);  
    return(0);  
}
```

```
unsigned _dos_getdiskfree(  
    unsigned drive,  
    struct diskfree_t *buffer);
```

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_getdrive, _dos_setdrive
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_getdiskfree` provides information about the disk in the specified drive. The function uses service 36H for this purpose.

`drive` specifies the drive, with drive A being 1, drive B being 2, etc. The default drive is 0.

`buffer` points to a `diskfree_t` structure, which contains information about the clusters on the disk and about the total space as well as free space.

```
struct diskfree_t {  
    unsigned total_clusters;  
    unsigned avail_clusters;  
    unsigned sectors_per_cluster;  
    unsigned bytes_per_sector;  
};
```

### Return value

The function returns 0 if successful; otherwise it returns a nonzero value and sets `errno` to `EINVAL` (invalid drive specification).

```
#include <dos.h>  
#include <stdio.h>  
  
main()  
{  
    struct diskfree_t drive;  
    _dos_getdiskfree(0, &drive);  
    printf(  
        "%d clusters free\n",  
        drive.avail_clusters);  
    return(0);  
}
```

## **void \_dos\_getdrive(unsigned \*drive);**

---

**Header file**            dos.h

**See also**             \_dos\_getdiskfree, \_dos\_setdrive

**Portability**         DOS

**Multi-thread**        DOS is not re-entrant

The function \_dos\_getdrive determines the current disk drive. The function uses service 19H for this purpose.

drive                   points to the location in which the drive information is returned. Drive A = 1, drive B = 2, etc.

### **Return value**

None.

### **Example**

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

main()
{
    unsigned drive;

    _dos_getdrive(&drive);
    printf("Current Drive is %c:\n", drive+'A');
    return(0);
}
```

---

**unsigned \_dos\_getfileattr(const char \*path, unsigned \*attrib);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_setfileattr
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_getfileattr` determines information about the file or directory specified by `path`. The function uses service 43H for this purpose.

<code>path</code>	specifies the file or directory of interest.
<code>attrib</code>	points to the location at which information about the attributes is returned. This information is returned in the low-order byte of the <code>target</code> for <code>attrib</code> . The following values can be stored in this location:
<code>_A_NORMAL</code>	The file is normal Æ i.e., it can be read or written.
<code>_A_RDONLY</code>	Read only. The file cannot be created, and cannot be used for output.
<code>_A_HIDDEN</code>	The file is hidden Æ i.e., cannot be seen in a directory listing.
<code>_A_SYSTEM</code>	The file is a system file, and cannot be seen in a directory listing.
<code>_A_VOLID</code>	The file if a volume ID Æ i.e., the file must be in the root directory.
<code>_A_SUBDIR</code>	The file is a SUB-DIRECTORY.
<code>_A_ARCH</code>	The file is an archive.

**Return value**

The function returns 0 if successful; otherwise it returns a DOS error code, and sets `errno` to `ENOENT` (specified file cannot be found).

**Example**

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    unsigned attribute;
    int error;
    if(argc != 2)
        abort();
    error=_dos_getfileattr(argv[1], &attribute);
    if(error)
    {
        printf("File not found\n");
        abort();
    }
    if(!(attribute&_A_RDONLY))
        printf("File %s has write permission\n",
            argv[1]);
    return(0);
}
```

---

**unsigned \_dos\_getftime(int handle, unsigned \*date, unsigned \*time);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_setftime
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_getftime` determines when the file specified by `handle` was last written. Both data and time are determined. The function uses service 57H for this purpose.

<code>handle</code>	specifies the file of interest.
<code>date</code>	points to the date information, which is stored in individual bits of the target value. 0Æ4 Day (1Æ31) 5Æ8 Month (1Æ12) 9Æ15 Year (1980Æ2099)
<code>time</code>	points to the time information, which is stored in individual bits of the target value. 0Æ4 Seconds / 2 (0Æ29) 5Æ8 Minutes (0Æ59) 9Æ15 Hours (0Æ23)

**Return value**

The function returns 0 if successful; otherwise it returns a DOS error code, and sets `errno` to `EBADF` (invalid file handle).

**Example**

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

#define SECONDS(c) ((c)&0x1F)
#define MINUTES(c) (((c)&0x7E0)>>5)
#define HOURS(c) (((c)&0xF800)>>11)
#define DAY(c) ((c)&0x1F)
#define MONTH(c) (((c)&0x1E0)>>5)
#define YEAR(c) (((c)&0xFE00)>>9)

main(int argc, char *argv[])
{
    unsigned date, time;
    int fh, error;

    if(argc != 2)
        abort();
    error=_dos_open(argv[1], 2, &fh);
    if(error)
    {
        printf("File not found\n");
        abort();
    }
    error=_dos_getftime(fh, &date, &time);
    printf("Modification time is %d:2d.%d\n",
        HOURS(time), MINUTES(time), SECONDS(time));
    printf("Modification date is 2d-2d-2d\n",
        DAY(date), MONTH(date), YEAR(date)+80);
    _dos_close(fh);
    return(0);
}
```

## **`void _dos_gettime(struct dostime_t *buffer);`**

---

**Header file**            `dos.h`

**See also**            `_dos_getdate`, `_dos_setdate`, `_dos_settime`

**Portability**        `DOS`

**Multi-thread**       `DOS` is not re-entrant

The function `_dos_gettime` determines the current system time. The function uses interrupt 2CH for this purpose.

`buffer`               points to a `dostime_t` structure containing the relevant time information.

```
struct dostime_t {
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned char hsecond; /* 0-99 */
}
```

### **Return value**

None.

### **Example**

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

main()
{
    struct dostime_t time;

    _dos_gettime(&time);
    printf(
        "Time is %d:%d:%d.%d\n",
        time.hour, time.minute,
        time.second, time.hsecond);
    return(0);
}
```



---

**void ( interrupt far \* \_dos\_getvect(unsigned intnum))();**

---

**Header file**            dos.h

**See also**             \_chain\_intr, \_dos\_setvect

**Portability**         DOS

**Multi-thread**        DOS is not re-entrant

The function `_dos_getvect` returns the value of the interrupt vector specified by `intnum`. The function uses service 35H for this purpose.

**Return value**

The function returns a far pointer to the interrupt handler for the interrupt specified by `intnum` Æ if such a handle exists.

**Example**

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

void interrupt far handler(void);
void rest_of_program(void);

main() {
    void (interrupt far *old_int)(void);
    /* save old vector */
    old_int=_dos_getvect(4);
    /* install new handler */
    _dos_setvect(4, handler);

    rest_of_program();
    /* restore old handler */
    _dos_setvect(4, old_int);
    return(0);
}

void interrupt far handler(void) {
    return;
}

void rest_of_program(void) {
    return;
}
```

## **void \_dos\_keep(unsigned retcode, unsigned paras);**

---

**Header file**            dos.h

**See also**             \_chain\_intr, \_dos\_getvect, \_dos\_setvect

**Portability**         DOS

**Multi-thread**        DOS is not re-entrant

Installs a terminate and stay resident (TSR) program in memory.

retcode                specifies the exist status code for the program.

paras                  specifies the number of paragraphs (16-byte blocks) of memory to be allocated for the resident process.

Exits the calling process, but leaves it in memory. Allocates the specified amount of memory for the calling process (which is now resident), and returns the low-order byte of retcode to the parent of the process just exited. Function \_dos\_keep uses service 31H for this purpose.

### **Return value**

None.

## **unsigned \_dos\_open(const char \*path, unsigned mode, int \*handle);**

---

**Header file** dos.h

**See also** \_dos\_close, \_dos\_read, \_dos\_write,

**Portability** DOS.

**Multi-thread** DOS is not re-entrant

The function \_dos\_open opens the file specified. The file must already exist. The function uses service 3DH for this purpose.

path specifies the file to be opened.

mode specifies the access, sharing, and inheritance properties of the file. These are specified by combining values from the three groups below. In a given call, at most one access and one sharing mode can be selected.

Constant	Mode	Meaning
O_RDONLY	Access	File is read only
O_WRONLY	Access	File is write only
O_RDWR	Access	File is both read and write
SH_COMPAT	Sharing	File is compatible
SH_DENYRW	Sharing	Deny both reading and writing
SH_DENYWR	Sharing	Deny writing only
SH_DENYRD	Sharing	Deny reading only
SH_DENYNONE	Sharing	Deny neither
O_NOINHERIT	Inheritance	File is not inherited by child process

handle points to the buffer in which the file's handle is returned.

### **Return value**

The function returns 0 if successful; otherwise it returns a DOS error code, and sets errno to one of the following values:

Constant	Meaning
EINVAL	Either access mode value is invalid or a sharing mode was specified but file sharing capabilities were not installed.
ENOENT	Path not found.
EMFILE	Too many open file handles.

**EACCESS** Access denied  $\mathcal{A}\mathcal{E}$  because a directory or volume ID was specified or because a read only file was opened for writing.

### Example

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

main() {
    int fh, error;

    error=_dos_open("TEMP.$$$", 2, &fh);
    if(!error)
    {
        _dos_close(fh);
        printf("TEMP.$$$ exists\n");
    }
    return(0);
}
#include <dos.h>
#include <stdio.h>

#define BUFFER_SIZE 512

main()
{
    int fh, error;
    unsigned count;
    char buffer[BUFFER_SIZE];

    error=_dos_open("TEMP.$$$", 2, &fh);
    if(!error)
    {
        error=_dos_read(
            fh,
            (void far *)buffer,
            BUFFER_SIZE,
            &count);
        printf("%u bytes read\n", count);
        _dos_close(fh);
    }
    return(0);
}
```

```
unsigned _dos_setblock(  
unsigned size,  
unsigned seg,  
unsigned *maxsize);
```

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_allocmem
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_setblock` changes the size of a previously allocated segment of memory to a new (generally, larger) value. The memory must have been allocated using `_dos_allocmem`. The function uses 4AH for this purpose.

size	specifies the new block size.
seg	specifies the block whose size is being changed.
maxsize	points to a buffer which specifies the amount actually allocated.

### **Return value**

The function returns 0 if successful; otherwise it returns a DOS error code, and sets `errno` to `ENOMEM`. This indicates that the segment passed was not allocated with `_dos_allocmem`.

## Example

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

#define NEW_SIZE 1000

main()
{
    char far *buffer;
    unsigned new_seg;
    unsigned maxsize;
    int error;

    error=_dos_allocmem(100, &new_seg);
    if(error)
    {
        fprintf(
            stderr,
            "Error allocating memory\n");
        return(1);
    }
    error=_dos_setblock(
        NEW_SIZE,
        new_seg,
        &maxsize);
    if(error)
    {
        printf(
            "Reallocation failed");
        printf(
            "%u byte available\n",
            maxsize);
    }
    else
    {
        buffer=MK_FP(new_seg, 0);
        printf(
            "%u paras of memory are at %Fp\n",
            NEW_SIZE, buffer);
    }
    _dos_freemem(new_seg);
    return(0);
}
```

```
unsigned _dos_read(  
    int handle,  
    void far *buffer,  
    unsigned num,  
    unsigned *numread);
```

---

**Header file**            dos.h

**See also**                \_dos\_close, \_dos\_open, \_dos\_write

**Portability**            DOS

**Multi-thread**          DOS is not re-entrant

The function `_dos_read` reads a specified number of bytes from a specified file, and copies these data to a buffer.

`handle`                   specifies the file to read.

`buffer`                   points to the storage into which the information will be copied after reading.

`num`                       specifies the number of bytes to read.

`numread`                points to a location which contains the actual number of bytes read. This value may be less than the value specified in `num`.

### **Return value**

The function returns 0 if successful; otherwise, it returns a DOS error code and sets `errno` to one of the following values.

<b>Constant</b>	<b>Meaning</b>
-----------------	----------------

EBADF	File handle is invalid.
-------	-------------------------

EACCES	Access denied (e.g., because file is write only).
--------	---

### **Example**

---

**unsigned \_dos\_setdate(const struct dosdate\_t \*buffer);**

---

**Header file**            dos.h

**See also**            \_dos\_gettime, \_dos\_getdate, \_dos\_settime, gmtime, localtime, mktime, \_strdate, \_strtime, time

**Portability**        DOS

**Multi-thread**       DOS is not re-entrant

The function `_dos_setdate` sets the current system date. The function uses service 2BH for this purpose.

`buffer`               points to a `dosdate_t` structure in which the date is stored.

```
struct dosdate_t {
    unsigned char day; /* 1-31 */
    unsigned char month; /* 1-12 */
    unsigned int year; /* 1980-2099 */
    unsigned char dayofweek; /* 0-6, 0=Sunday */
};
```

**Return value**

The function returns 0 if successful; otherwise it returns a nonzero value. In that case, `errno` is set to `EINVAL` to indicate that the specified date was invalid.

**Example**

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

main()
{
    struct dosdate_t date;
    _dos_getdate(&date);
    printf("Date is %d-%d-%d\n",
        date.day, date.month, date.year);
    date.year-=1;
    _dos_setdate(&date);
    printf("Date is now %d-%d-%d\n",
        date.day, date.month, date.year);
    return(0);
}
```



---

**void \_dos\_setdrive(unsigned drive, unsigned \*maxdrives);**

---

**Header file**            dos.h

**See also**            \_dos\_getdiskfree, \_dos\_getdrive

**Portability**        DOS

**Multi-thread**      DOS is not re-entrant

The function `_dos_setdrive` sets the current default drive to a specified drive. The function uses service 0EH for this purpose.

`drive`                specifies which drive is to become the current one. Drive A = 1, drive B = 2, etc.

`maxdrives`          specifies the total number of drives in the configuration.

**Return value**

There is no **Return value**. The function simply fails if an invalid drive number is specified. To check whether the desired changes have been made, use `_dos_getdrive`.

**Example**

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

main(int argc, char *argv[])
{
    unsigned new_drive, max_drive;

    if(argc != 2)
        abort();
    sscanf(argv[1], "%d", &new_drive);
    _dos_setdrive(new_drive, &max_drive);
    printf("%u drives in system\n", max_drive);
    return(0);
}
```

## **unsigned \_dos\_setfileattr(const char \*path, unsigned attr);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_getfileattr
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_setfileattr` sets the attributes of the specified file or directory. The function uses service 43H for this purpose.

path	specifies the file or directory.
attr	specifies the attributes the file is to have. These are represented in the low-order byte of the attr parameter, and can take any of the following values:
_A_NORMAL	The file is normal Æ i.e., it can be read or written.
_A_RDONLY	Read only. The file cannot be created, and cannot be used for output.
_A_HIDDEN	The file is hidden Æ i.e., cannot be seen in a directory listing.
_A_SYSTEM	The file is a system file, and cannot be seen in a directory listing.
_A_VOLID	The file is a volume ID Æ i.e., the file must be in the root directory.
_A_SUBDIR	The file is a SUB-DIRECTORY.
_A_ARCH	The file is an archive.

### **Return value**

The function returns 0 if successful; otherwise it returns a DOS error code and sets `errno` to one of the following:

<b>Constant</b>	<b>Meaning</b>
ENOENT	Path not found.
EACCESS	Access denied Æ because a directory or volume ID was specified .

### **Example**

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

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

    if(argc != 2)
        abort();
    error=_dos_setfileattr(argv[1], _A_NORMAL);
    if(error)    printf("Error\n");
    return(0);
}
```

## **unsigned \_dos\_setftime(int handle, unsigned date, unsigned time);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_getftime
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_setftime` sets the date and time at which a specified file was last modified. The function uses service 57H for this purpose.

**handle** specifies the file.

**date** specifies the date, whose components are contained in individual bits as follows:

<b>bits</b>	<b>Meaning</b>
-------------	----------------

0Æ4	Day (i.e., 1Æ31).
-----	-------------------

5Æ8	Month (i.e., 1Æ12).
-----	---------------------

9Æ15	Year (i.e., 1980Æ2099).
------	-------------------------

**time** specifies the time, whose components are contained in individual bits as follows:

<b>bits</b>	<b>Meaning</b>
-------------	----------------

0Æ4	Seconds / 2 (i.e., 0Æ29).
-----	---------------------------

5Æ8	Minutes (i.e., 0Æ59).
-----	-----------------------

9Æ15	Hours (i.e., 0Æ23).
------	---------------------

### **Return value**

The function returns 0 if successful; otherwise it returns a DOS error code and sets `errno` to `EBADF` to indicate an invalid file handle. The function returns an error in the OS2 DOS compatibility box.

### **Example**

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

#define SECONDS(c) ((c)&0x1F)
#define MINUTES(c) (((c)<<5)&0x7E0)
#define HOURS(c) (((c)<<11)&0xF800)

#define DAY(c) ((c)&0x1F)
#define MONTH(c) (((c)<<5)&0x1E0)
#define YEAR(c) (((c)<<9)&0xFE00)

main(int argc, char *argv[])
{
    unsigned date, time;
    int fh, error;

    if(argc != 2)        abort();
    error=_dos_open(argv[1], 2, &fh);
    if(error)
    {
        printf("File not found\n");
        abort();
    }
    date=YEAR(89-80)|MONTH(1)|DAY(1);
    time=HOURS(12)|MINUTES(30)|SECONDS(0);
    error=_dos_setftime(fh, date, time);
    _dos_close(fh);
    return(0);
}
```

---

**unsigned \_dos\_settime(const struct dostime\_t \*buffer);**

---

**Header file**            dos.h

**See also**            \_dos\_getdate, \_dos\_gettime, \_dos\_setdate, gmtime, localtime, mktime, \_strdate, \_strtime

**Portability**        DOS

**Multi-thread**       DOS is not re-entrant

The function `_dos_settime` sets the current time to a specified value. The function uses service 2DH for this purpose.

`buffer`                points to a `dostime_t` structure that contains the components of the current time.

```
struct dostime_t
{
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned char hsecond; /* 0-99 */
};
```

**Return value**

The function returns 0 if successful; otherwise it returns a nonzero value and sets `errno` to `EINVAL` to indicate that an invalid time was specified.

**Example**

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

main()
{
    struct dostime_t time;

    _dos_gettime(&time);

    printf(
        "Time is %d:%d:%d.%d\n",
        time.hour, time.minute,
        time.second, time.hsecond);
    time.hour+=1;
    _dos_settime(&time);

    printf(
        "Time is now %d:%d:%d.%d\n",
        time.hour, time.minute,
        time.second, time.hsecond);
    return(0);
}
```

---

**void \_dos\_setvect(unsigned intnum, void ( interrupt far \* handler)());**

---

<b>Header file</b>	dos.h
<b>See also</b>	_chain_intr, _dos_getvect, _dos_keep
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_setvect` sets the current value of a specified interrupt vector to a specified function. The function uses service 25H for this purpose.

intnum	specifies the number of the interrupt vector being replaced.
handle	specifies a function that will serve as an interrupt handler. If this is a C function, it must be defined and prototyped with the interrupt keyword or pragma; otherwise, the handler must satisfy the criteria for an interrupt-handling routine.

dostime\_t structure

**Return value** None.

**Example**

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

void interrupt far handler(void);
void rest_of_program(void);

main()
{
    void (interrupt far *old_int)(void);

    old_int=_dos_getvect(4); /* save old vector */
    _dos_setvect(4, handler); /* install new handler */
    rest_of_program();
    _dos_setvect(4, old_int); /* restore old handler */
    return(0);
}

void interrupt far handler(void)
{
    return;
}

void rest_of_program(void)
{
    return;
}
```

```

unsigned _dos_write(
    int handle,
    const void far *buffer,
    unsigned num,
    unsigned *numwrit);

```

---

<b>Header file</b>	dos.h
<b>See also</b>	_dos_close, _dos_open, _dos_read
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dos_write` writes a specified amount of data to a file. The function uses service 25H for this purpose.

<code>handle</code>	specifies the file to which the data are to be written.
<code>buffer</code>	points to the storage from which the data being written will be taken.
<code>num</code>	specifies the number of bytes to write.
<code>numwrit</code>	points to a location that contains the actual number of bytes written. This may be fewer bytes than requested.

### Return value

The function returns 0 if successful; otherwise it returns a DOS error code and sets `errno` to one of the following:

Constant	Meaning
EBADF	File handle is invalid.
EACCES	Access denied (e.g., because file is read only).

### Example

```

#include <dos.h>
#include <stdio.h>
#include <string.h>
main()
{
    int fh, error;
    unsigned count;
    char *msg="WRITE TO FILE";
    error=_dos_open("TEMP.$$$", 2, &fh);
    if(!error)
    {
        error=_dos_write(fh,
            (void far *)msg, strlen(msg), &count);
        printf("%u bytes written\n", count);
        _dos_close(fh);
    }
    return(0);
}

```



## **int dup(int handle);**

## **U**

**Header file**           io.h

**See also**            creat, close, dup2, open.

**Portability**        UNIX/DOS/OS2

**Multi-thread**      Low level I/O functions are not protected by the library.

The low-level I/O function dup causes the file specified by handle to be duplicated. The next available file handle is allocated, and access mode and file pointers are shared between the two handles.

### **Return value**

The function returns the new file handle. If an error occurs, the value -1 is returned and errno is set to one of the following values:

EBADF                Invalid file handle

EMFILE               No more handles available

### **Example**

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

main()

{
    int console;
    FILE *new;

    /* save original stdout handle*/
    console=dup(1);
    new=fopen("TEMP.$$$", "w");
    dup2(fileno(new), 1);          /* stdout now goes to file */
    puts("This is sent to the file TEMP.$$$");
    fflush(stdout);
    fclose(new);
    dup2(console, 1); /* restore stdout */
    puts("stdout restored\n");
    return(0);
}
```

## **int \_dup(int handle);**

---

<b>Header file</b>	io.h
<b>See also</b>	_close, _creat, _open
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dup` duplicates an existing file handle by assigning the same file to a new handle. The function accomplishes this by performing INT 21H function 45H.

`handle` specifies the file to be associated with a second handle. This file must be open when `_dup` is called.

After the call, the two handles refer to the same file pointer, and have the same access mode.

### **Return value**

The function returns a new file handle if successful; otherwise, the function returns -1 and sets `errno` to one of the following:

EBADF	invalid file handle.
EMFILE	too many open files.

### **Example**

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

main()
{
    int fh;
    char *msg="Hello World\n";

    fh=_dup(1); /* duplicate stdout */
    _write(fh, msg, strlen(msg));
    _close(fh);
    return(0);
}
```

## **int dup2(int handle1, handle2);**

**U**

**Header file**           io.h

**See also**             creat, close, dup, open.

**Portability**         UNIX/DOS/OS2

**Multi-thread**        Low level I/O functions are not protected by the library.

The function dup2 forces handle2 to refer to the same file as handle1. If handle2 is already associated with an open file, that file is closed. The file pointer and access mode are shared between the two handles.

### **Return value**

The function returns 0 if successful. If an error occurs, the value -1 is returned and errno is set to one of the following values:

EBADF                 Invalid file handle

EMFILE                No more handles available

### **Example**

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

main()
{
    int console;
    FILE *new;

    /* save original stdout handle */
    console=dup(1);    new=fopen("TEMP.$$$", "w");
    dup2(fileno(new), 1);
    /* stdout now goes to file */
    puts("This is sent to the file TEMP.$$$");
    fflush(stdout);
    fclose(new);
    dup2(console, 1); /* restore stdout */
    puts("stdout restored\n");
    return(0);
}
```

## **int \_dup2(int handle1, handle2);**

---

<b>Header file</b>	io.h
<b>See also</b>	_close, _creat, _open
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant

The function `_dup2` makes `handle2` refer to the same file as `handle1`. the function accomplishes this by performing INT 21H function 46H.

### **Return value**

The function returns 0 if successful; otherwise, the function returns -1 and sets `errno` to one of the following:

EBADF	invalid file handle.
EMFILE	too many open files.

### **Example**

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

main()

{ int fh;
  char *msg="Hello World\n";

  fh=10;
  _dup2(1, 10); /* duplicate stdout */
  _write(fh, msg, strlen(msg));
  _close(fh);
  return(0);
}
```

---

**char \*ecvt(double value, int digits, int \*dec, int \*sign);**

---

**U**

<b>Header file</b>	stdlib.h
<b>See also</b>	fcvt, gcv
<b>Portability</b>	UNIX/DOS/OS2. For future <b>Portability</b> use sprintf.
<b>Multi-thread</b>	This function is not re-entrant.

Function `ecvt` converts a double precision floating point value to a character string.

value	is <code>dostime_t</code> structure the floating point value to be converted.
digits	is the number of digits to be stored.
dec	points to an integer to hold the position of the decimal point. 0 or a negative value indicates that the decimal point lies to the left of the beginning of the string. A positive value indicates that the decimal point lies to the right of the beginning of the string.
sign	points to an integer to hold the sign of the converted number. A value of 0 indicates that the number is positive, while a nonzero value indicates that it is negative.

**Return value**

`ecvt` returns a pointer to the string of digits. There is no error return.

**Note:** The functions `ecvt` and `fcvt` use a statically allocated buffer for the results of each conversion, so a call to either of these functions destroys the result of a previous conversion.

**Example**

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

main()
{
    char *result;
    int precision=12;
    int decimal_point, sign;
    int count, n; /* convert string */
    result=ecvt(
        12.345678, precision,
        &decimal_point, &sign);
    count=strlen(result); /* format result */
    n=0;
    if(sign) /* - sign if negative */
        printf("-");
    while(n < count)
    {
        /* position dec point */
        if(n == decimal_point) putchar('.');
        putchar(result[n++]);
    }
    return(0);
}
```

---

**short far \_ellipse(short fill, short x1, short y1, short x2, short y2);**

---

**M****Header file** graph.h**See also** \_arc, \_lineto, \_pie, \_rectangle, \_setcolor, \_setfillmask**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_ellipse` from the graphics module draws an ellipse (in the current color) within a rectangle whose boundaries are specified. The center of the ellipse is the center of the bounding rectangle.

**fill** specifies whether the ellipse is to be filled. This argument can have either of the following values:

`_GFILLINTERIOR` fill the figure using the current color and fill mask.

`_GBORDER` do not fill the figure.

**x1,y1** coordinates of the upper left point of the rectangle bounding the ellipse.

**x2,y2** coordinates of the lower right point of the rectangle bounding the ellipse.

**Note:** if the arguments for the rectangle specify a point or a line  $\text{Æ}$  that is, if  $x1 = x2$  or  $y1 = y2$ , or both  $\text{Æ}$  then no figure is drawn.

**Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

**Example**

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

main()

{
    struct videoconfig v;
    short X1, Y1, X2, Y2;
    short xoffset, yoffset;
    int shift=0;

    _setvideomode(_MRESNOCOLOR);
    _getvideoconfig(&v);
    xoffset=v.numxpixels/16;
    yoffset=v.numypixels/16;
    X1=xoffset;
    Y1=yoffset;
    X2=v.numxpixels-xoffset;
    Y2=v.numypixels-yoffset;
    while((X1 < X2) && (Y1 < Y2))
    {
        _ellipse(_GBORDER, X1, Y1, X2, Y2);
        X1+=xoffset+shift;
        Y1+=yoffset+shift;
        X2-=xoffset-shift;
        Y2-=yoffset-shift;
        shift+=2;
    }
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **void enable(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	disable, getvect
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	Under DOS, enable will unlock a thread locked by a call to disable.

The function enable enables interrupts.

### **Return value**

None.

## **void \_enable(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_disable.
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	Under DOS, _enable will unlock a thread locked by a call to _disable.

The function \_enable enables interrupts by executing an STI instruction.

### **Return value**

None.

## **void EnableBreakCheck(void)**

---

<b>Header file</b>	dos.h
<b>See also</b>	DisableBreakCheck
<b>Portability</b>	DOS only

Enables control-break checking. If checking is enabled, a program may be terminated at any point by pressing Ctrl-Break or Ctrl-C.

### **Return value**

None.

---

**void far \_enable\_herc(void)**

---

**M****Header file**           graph.h**Portability**           DOS only

Some clone Hercules graphics cards may not be detected automatically at program startup. `_enable_herc` may be called before selecting a graphics video mode to inform the system that a Hercules graphics card is present.

**Return value**

None.

---

**void \_endthread(void)**

---

**Header file**           process.h**Portability**           OS2 only

Identical to `_dosendthread`. Provided for portability.

**Return value**

None.

## int eof(int handle);

## U

**Header file** io.h

**See also** clearerr, feof, ferror, perror

**Portability** UNIX/DOS/OS2

**Multi-thread** Low level I/O functions are not protected by the library.

The function eof determines whether a file *Æ* specified by handle *Æ* has reached the End of File marker.

### Return value

A **Return value** of 1 indicates that the end of the file has been reached. A return value of 0 indicates that it has not. A return value of -1 indicates an error.

If an error does occur, errno will be set to EBADF, invalid file handle.

### Example

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

main()
{
    int fh;
    char *message=
        "This is at offset BUFFER_SIZE";

    fh=open("TEMP.$$$", O_RDWR);
    lseek(fh, BUFFER_SIZE, SEEK_SET);
        /* write message */
        /* if not at end of file */    if(!_eof(fh)
        Write(fh, message, strlen(message));
    close(fh);
    return(0);
}
```

## **int \_eof(int handle);**

---

<b>Header file</b>	io.h
<b>See also</b>	_close, _open, _read, _write
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant

The function `_eof` determines whether the end of the file specified by `handle` has been reached. The function accomplishes this by performing INT 21H function 44H.

### **Return value**

The function returns 1 if the end of file has been reached, and 0 if it has not. If unsuccessful, the function returns -1, and sets `errno` to `EBADF` (invalid file handle).

### **Example**

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

main()
{
    int fh;
    char *msg="OUTPUT TO FILE";

    fh=_open("TEMP.$$$", 2);
    _write(fh, msg, strlen(msg));
    if(_eof(fh))
        printf("At End Of File\n");
    _close(fh);
    return(0);
}
```

```

int execl(const char *path, char *arg0, ... );
int execlp(const char *path, char *arg0, ... );
int execlpe(const char *path, char *arg0, ... );
int execv(const char *path, char *argv[]);
int execve(const char *path, char *argv[], char *env[]);
int execvp(const char *path, char *argv[]);
int execvpe(const char *path, char *argv[], char *env[]);

```

---

**Header file** process.h

**See also** abort, exit, \_exit, atexit, onexit, system and the spawn family of functions.

**Portability** UNIX/DOS/OS2

**Multi-thread** None.

The execX functions load and execute a child process. If the call is successful, the child process occupies the memory previously occupied by the parent process. If sufficient memory is not available, the call will fail.

Each function in this family uses a generic exec function and the letters at the end of the name determine the variation of argument passing:

p	Uses the path environment variable to locate the file to be executed.
l	Lists the command line arguments separately.
v	Passes an array of pointers to command line arguments.
e	Passes an array of pointers to environment arguments.
path	specifies the file to be executed. The following search procedures are used to locate the file:

- If the path does not have a filename extension or does not end with a period, the exec function searches for the file. If it is unsuccessful, the extensions .COM then .EXE are tried.
- If the path has a filename extension, then only that extension is used.
- If the path ends with a period, no extension is used.

The execlp, execlpe, execvp, and execvpe functions use the same procedures, but in all directories specified by the PATH environment variable.

With `execl`, `execle`, `execlp`, and `execlpe`, arguments are passed by giving one or more pointers to character strings which will form the argument list for the child process. The series of pointers must be terminated by a `NULL` pointer. The combined length of the strings specified as arguments must not exceed 128 characters. Terminating null characters are not counted but a space character is automatically inserted between each argument.

With `execv`, `execve`, `execvp`, and `execvpe`, arguments are passed in an array, with the final member being a `NULL` pointer.

At least one argument must be passed to a child process, namely `arg0`. This is usually a copy of the path argument, although a different value will not cause an error. Under DOS 3.0+ and OS2, the path is available as `arg0`.

`execl`, `execlp`, `execv` and `execvp` cause the child process to inherit the environment of the parent process. `execle`, `execlpe`, `execve` and `execvpe` can pass a list of environment variable settings to the child process. This list is passed as a pointer variable which must be the final argument following the `NULL` argument that terminates the arg list. If this final argument itself is `NULL`, the environment of the parent process is passed to the child. Otherwise the pointer references an array of pointers to character ASCIIZ strings, each taking the form:

NAME=environment\_variable

The last element of this array must in turn be a `NULL` pointer.

### Return value

The `exec` family of functions does not return unless an error occurs, in which case the **Return value** is -1, and `errno` will be set to one of the following values:

E2BIG	The argument list exceeded 128 bytes, or the environment variables exceeded 32 Kbytes.
EACCES	The specified file had a sharing or locking violation. (DOS 3.0+ only).
EMFILE	Too many files open. The file must be open to determine whether it is executable.
ENOENT	File or path not found.
ENOMEM	Not enough memory to execute the child process.

**Note:** All open files are inherited by a child process, although the translation mode is not set. The function `setmode` must be used to set the translation mode to the desired mode.

Signal settings revert to their defaults in the child process.

**Example**

```
#include <process.h>
#include <stddef.h>

main()
{
    /* do ts/m sieve using PATH */
    int exit_code;
    exit_code=execlp(
        "TS.EXE", "ts", "/m",
        "sieve", NULL);
    printf("TS.EXE executed\n");
    return(exit_code);
}
```

**void exit(int status);****A**

**Header file**           stdlib.h and process.h

**See also**            abort, \_exit, atexit, onexit, system and the spawn/exec family of functions.

**Portability**        ANSI

**Multi-thread**       A call to exit terminates all threads.

Terminates the calling process. Functions set by atexit and onexit are called. Open streams are then flushed and all open files are closed. All files created by tmpfile are deleted, and \_exit is called to terminate the process. status is passed to the calling process as a return code.

**Return value**

None

**Example**

```
#include <stdlib.h>
#include <stdio.h>
void error_handler(int type){
    if(type == 0)
        printf("Non fatal error no %d \n", errno);
    else{
        printf("Fatal error\n"); exit(errno);
    }
    return;
}
```



## **`void _exit(int status);`**

---

**Header file**            process.h

**See also**             abort, exit, atexit, onexit, system and the spawn/exec family of functions.

**Portability**         UNIX/DOS/OS2

**Multi-thread**        A call to `_exit` terminates all threads.

Terminates a process without calling `atexit` or `onexit` or flushing streams. `status` is passed to the calling process as a return code.

**Return value**

None.

**double exp(double x);****A****void \*\_expand(void \*buffer, size\_t size);**

<b>Header file</b>	alloc.h
<b>Variants</b>	_fexpand, _nexpand
<b>See also</b>	calloc, free, malloc, _msize, realloc
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	Far heap only protected by semaphore.

The function `_expand` attempts to expand or contract the size of a memory block allocated by `malloc` or `calloc` without moving the block's location in memory.

<code>buffer</code>	is a void pointer to a block of memory previously returned by <code>malloc</code> , <code>calloc</code> or <code>realloc</code> .
<code>size</code>	is the new desired size of the block.

**Return value**

The function `_expand` returns `NULL` if the size of the block could not be changed; otherwise it returns the address of the block, which is guaranteed to be the same as the argument `buffer`.

The storage pointed to by the **Return value** is paragraph aligned and suitable for any type of object. The size of a newly allocated item can be checked with the `_msize` function.

**Example**

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

void *reallocate_block(
    void *block,
    size_t new_size,
    int block_type)
{
    if(block_type == 0)
        return(realloc(block, new_size));
    else
        /* block is fixed in memory */
        return(_expand(block, new_size));
}
```

## long double expl(long double x);

---

**Header file**           math.h

**See also**             log.

**Portability**         ANSI

**Multi-thread**       None.

The function `exp` returns the exponential result of its double precision floating point argument, `x`. The function `expl` is the same as `exp` but takes a long double argument and returns a long double value.

### Return value

The function returns  $e^x$ . If the value of `x` is too large, `errno` is set to `ERANGE` and the value `HUGE_VAL` is returned.

### Example

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate exponent of input */
    result=exp((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf(
            "e raised to the power %g is %g\n",
            num, result);
    return(0);
}
```

---

**double fabs(double x);**

---

**A**

---

**long double fabsl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	abs, cabs, labs.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `fabs` returns the absolute value of its double precision floating point argument, `x`. The function `fabsl` is the same as `fabs` but takes a long double argument and returns a long double value.

**Return value**

The function returns `|num|`. Thus, if `num` is positive, the function simply returns `num`; if `num` is negative, the function returns `-num`. There is no error return.

**Example**

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

main()
{
    double an;
    double n=-10.99;
    an=fabs(n);
    printf(
        "Absolute value of %e is %e\n", n, an);
    return(0);
}
```

**void far \*farcalloc(unsigned long num, unsigned long size);**

Far pointer variant of `calloc`. Allocates a huge item from the far heap but returns a far pointer.

---

**unsigned long farcoreleft(void);**

---

Far pointer variant of coreleft

---

**void far \*farrealloc(void far \*buffer, unsigned long size);**

---

Far pointer variant of realloc. Returns huge object but returns far pointer.

---

**void far \*farmalloc(unsigned long num);**

---

Far pointer variant of malloc. Allocates a huge item from far heap, but returns far pointer

---

**void farfree(void far \*buff);**

---

Far pointer variant of free

---

**void far \*\_fcalloc(size\_t size, int nmem);**

---

Far pointer variant of calloc

---

**int fclose(FILE \*st);**

---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	fcloseall, fdopen, fflush, fopen, freopen, tmpfile.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is controlled by a semaphore in Multi-thread programs.

The function `fclose` closes the stream `st`. Any buffer associated with the stream is flushed before closing. If this is a system buffer, it is freed. If the buffer was assigned by the user  $\mathcal{A}$  with its address being supplied to `setbuf` or `setvbuf`  $\mathcal{A}$  the buffer will not automatically be freed.

If the stream was opened by `tmpfile`, the associated file will be deleted.

**Return value**

The function returns 0 if the stream was successfully closed; otherwise it returns EOF.

**Example**

```
#include <stdio.h>

main()
{
    FILE *f;

    f=fopen("TEMP.$$$", "w"); /* open file */
                          /* write to it */
    fputs("This is written to TEMP.$$$", f);
    fclose(f); /* close it */
    return(0);
}
```

## **int fcloseall(void);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	fclose, fdopen, fflush, fopen, freopen, and tmpfile.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to stream is controlled by a semaphore in Multi-thread programs.

The function `fcloseall` closes all open streams except for `stdin`, `stdout`, `stderr`, `stdaux` and `stdprn`, which are flushed.

### **Return value**

The function returns the total number of streams closed.

### **Example**

```
#include <stdio.h>

main()
{ FILE *f1, *f2;

  f1=fopen("TEMP1.$$$", "w");
  f2=fopen("TEMP2.$$$", "w");
  fputs("This is written to TEMP1.$$$", f1);
  fputs("This is written to TEMP2.$$$", f2);
  fcloseall(); /* closes f1 and f2 and */
               /* flushes standard streams */ return(0);
}
```

---

**char \*fcvt(double value, int digits, int \*dec, int \*sign);**

---

**U****Header file**          `stdlib.h`**See also**            `ecvt`, `gcvt`.**Portability**        UNIX/DOS/OS2. Use `sprintf` for future **Portability****Multi-thread**      This function is not re-entrant.

Converts a double precision floating point value to a character string.

`value`                is the floating point value to be converted.`digits`               is the number of digits to be stored.`dec`                  points to an integer to hold the position of the decimal point. A 0 or a negative value means the decimal point lies to the left of the beginning of the string. A positive value means it lies to the right.`sign`                points to an integer to hold the sign of the converted number. A value of 0 indicates the number is positive, while a nonzero value indicates that it is negative.**Return value**       Returns a pointer to the string of digits. There is no error return.

Note:    **The functions `ecvt` and `fcvt` use a statically allocated buffer for the results of each conversion, so a call to either of these functions destroys the result of a previous conversion.**

**Example**



## FILE `*fdopen(int handle, char *type);`

---

<b>Header file</b>	stdio.h
<b>See also</b>	dup, dup2, fclose, fcloseall, fopen, freopen, open.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to stream is controlled by semaphore in Multi-thread programs.

The function `fdopen` associates a stream with a file identified by `handle`. The `type` character string specifies the access mode for the file. The specified `type` must be compatible with the access modes with which the file was originally opened.

“r”	Read only.
“w”	Write only.
“a”	Append (write only at end of file).
“r+”	Read and Write.
“w+”	Read and Write.
“a+”	Read and Append.

In addition to these values, the characters ‘t’ or ‘b’ may be added after the first character of `type` to specify *text* or *binary* translation modes. E.g.,

“r+b”	Read and Write, binary mode.
“wt+”	Read and Write, text mode.

If neither “t” nor “b” are specified, the stream’s translation mode is specified by the variable `_fmode`.

**Note:** If the stream allows both reading and writing, there must be an intervening call to `fseek`, `fsetpos`, or `rewind` between changes in the direction of I/O.

When a file is opened with “a” or “a+” all write operations will take place at the end of the file, regardless of any preceding calls to `fseek`, `fsetpos` or `rewind`.

### Return value

The function returns a pointer to the open stream. If the stream could not be successfully assigned, a NULL pointer is returned.

### Example

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

main()

{
    FILE *f;
    int fh;
    char *file_name="TEMP.$$$";

    fh=open(file_name, O_RDWR);
    f=fdopen(fh, "r+");
        /* mode must be the same */
    fprintf(f,
        "Formatted output file %s", file_name);
    fclose(f);
    return(0);
}
```

---

**double fabs(double x);**

---

**A**

---

**int feof(FILE \*st);**

---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	clearerr, eof, ferror, perror.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is <i>not</i> controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function feof determines whether a stream has reached the end of file. When a stream has reached end of file, all read operations return -1 until the stream is closed or rewind/fseek is called.

If the end of file was caused by encountering the ^Z character in a text file, write operations will also return -1 until rewind/fseek has been called.

### Return value

The routine returns a nonzero value for end of file; otherwise it returns 0.

### Example

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

void strip_high_bit(FILE *o, FILE *n)
{
    int c;

    while(1)
    {
        c=fgetc(o);
        if((feof(o)) || ferror(o))
            break;
        c=toascii(c); /* strips high bit */
                    /* of byte value */
        fputc(c, n);
    }
    return;
}
```

---

**int ferror(FILE \*st);**

---

**A****Header file**           stdio.h**See also**           clearerr, eof, feof, perror.**Portability**       ANSI**Multi-thread**      Access to stream is *not* controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function `ferror` tests a stream's error flag. If an error has occurred, the error flag remains set until the stream is closed, or until a call to `rewind` or `clearerr` is made.

**Return value**

The routine returns a nonzero value for an error; otherwise it returns a 0.

**Example**

```
#include <stdio.h>

restore_stream(FILE *f)
{
    if(ferror(f)) /* check for error */
    {
        clearerr(f); /* clear error */
        printf("Error cleared\n");
    }
    return;
}
```

---

**void far \*\_fexpand(void near \*buffer, size\_t size);**


---

Far pointer variant of `_expand`

---

**int fflush(FILE \*st);**


---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	fclose, fflush.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is controlled by semaphore in Multi-thread programs.

Flushes the stream `st`. If the last operation involving the stream was output, the contents of the buffer are written to the associated file. If the stream's last operation was input, the state of the stream remains unaltered. Calling `fflush` with `st` as `NULL` is the same as a call to `fflush`.

**Note:** `fflush` negates the effect of any preceding call to `ungetc` against `st`.

On an unbuffered stream, the only effect of `fflush` is to negate the `ungetc` call. Buffers are automatically flushed when a process terminates or a stream is closed.

### Return value

If the buffer was successfully flushed, a value of 0 is returned. If an error occurred, EOF is returned.

### Example

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
main()
{
    FILE *f;
    char *file_name="TEMP.$$$";
    f=fopen(file_name, "r+");
    fprintf(f,
        "Formatted output to file %s", file_name);
    fflush(f); /* flushes stream */
    fprintf(f,
        "More formatted output to %s", file_name);
    fclose(f);
    return(0);
}
```

---

**void \_ffree(void far \*buffer);**

---

Far pointer variant of free

---

**int fgetc(FILE \*st);**

---

**A**

**Header file**           stdio.h

**See also**             fputc, fputchar, fgetchar, getc, getchar

**Portability**         ANSI

**Multi-thread**       Access to the stream is *not* controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function fgetc reads an unsigned character (converted to an int) from the current position of stream st, and increments the stream pointer, if any.

**Return value**

The function returns the character read as an integer value. A **Return value** of EOF may indicate an error or end of file condition. The actual condition should be determined by using ferror or feof.

**Example**

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

void set_high_bit(FILE *o, FILE *n)
{
    int c;

    while(1)
    {
        c=fgetc(o); /* read character */
        if((feof(o)) || ferror(o))
            break;
        c|=0x80;
        fputc(c, n);/* write character */
    }
    return;
}
```

## **int fgetchar(void);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	fputc, fputchar, fgetc,getc, getchar
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to the stream is <i>not</i> controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function `fgetchar` reads an unsigned character (converted to an int) from `stdin`. The function is equivalent to `fgetc(stdin)`.

See `fgetc`.

**Note:** `fgetchar` is identical to `getchar` but is a function not a macro.

### **Return value**

The function returns the character read as an integer value. A Return value of EOF may indicate an error or end of file condition. The actual condition should be determined by using `ferror` or `feof`.

### **Example**

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

main()
{
    int c;
    FILE *f;

    f=fopen("TEMP.$$$", "w");
    while(1)
    {
        c=fgetchar(); /* read char from stdin */
        if(feof(stdin))
            break;
        fputc(c, f);
    }
    return(0);
}
```

---

**int fgetpos(FILE \*st, fpos\_t \*fp);**

---

**A****Header file**           stdio.h**See also**             fsetpos.**Portability**         ANSI.**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function `fgetpos` gets the current value of the stream's file position indicator, and stores this value in the object to which `fp` points. This value can be used later to reset the stream's pointer to the saved position.

`st`   specifies the stream.

`fp`   points to a location in which the current file position can be stored.

**Note:** The value stored at `fp` is only suitable for use by `fgetpos` and `fsetpos`, and will not necessarily produce accurate results if passed to any other function, such as `fseek` or `lseek`.

**Return value**

If successful, `fgetpos` returns 0. If an error occurred, a nonzero value is returned and `errno` is set to one of the following values:

**EINVAL**   The stream was invalid.

**EBADF**   The file handle associated with the stream was bad.

**Example**

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



```
main()
{
    FILE *f;
    char *file_name="TEMP.$$$";
    char second_line[128];
    fpos_t end_of_line;

    f=fopen(file_name, "r+");
    fprintf(f,
        "Formatted output to file %s", file_name);
        /* save file position */
    fgetpos(f, &end_of_line);
    fprintf(f,
        "The first word of this sentence also goes
        to stdout");
        /* restore file position */
    fsetpos(f, &end_of_line);
    fscanf(f, "%s", second_line);
    puts(second_line);
    fclose(f);
    return(0);
}
```

---

**char \*fgets(char \*string, int num, FILE \*st);**

---

**A****Header file**           stdio.h**See also**             fputs, gets, puts.**Portability**         ANSI**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function `fgets` reads a string from the stream `st`, and stores it at `string`.

`string`   points to the buffer at which the string read will be stored.

`num`     specifies the maximum length for `string`.

`st`      specifies the stream from which the string will be read.

Characters are read from the current position until any of the following occurs:

- a newline (`\n`) character is encountered,
- an error or end of file occurs or
- the number of characters read is equal to `num-1`.

The string is terminated with a null character, `\0`. If a newline is read, it *is* included in the string

**Note:** If *num* is equal to 1, the string will be a null string

**Return value**

If `fgets` is successful, the function will return `string`. If an error or end of file occurred, `fgets` will return `NULL`.

**Example**

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

main()
{
    FILE *f;
    char *file_name="TEMP.$$$";
    char first_line[128];

    f=fopen(file_name, "r+");
    fprintf(f,
        "The first line of file %s goes to\n", file_name);
    rewind(f);
    /* get maximum of 127 characters */
    fgets(first_line, 127, f);
    puts(first_line);
    fclose(f);
    return(0);
}
```

**int \_fheapchk(void);**

---

Far heap variant of heapchk

**int \_fheapset(int ch);**

---

Far heap variant of heapset

**int \_fheapwalk(struct \_fheapinfo \*entry);**

---

Far heap variant of \_heapwalk

## long filelength(int handle);

U

<b>Header file</b>	io.h
<b>See also</b>	chsize, fileno, stat, fstat
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function filelength returns the length (in bytes) of the file associated with handle.

### Return value

If successful, the file length is returned. If an error occurred, the value -1 is returned and errno is set to EBADF.

### Example

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <stat.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
    int fh;
    long file_size;
    if(argc != 3)
        abort();
    fh=open(argv[1],
        O_RDWR|O_CREAT,
        S_IREAD|S_IWRITE);
    if(fh < 0)
    {
        printf("Could not create %s\n", argv[1]);
        abort();
    }

    /* change file size and */
    /* verify new length */
    sscanf(argv[2], "%ld", &file_size);
    if( (chsize(fh, file_size)) ||
        (filelength(fh) != file_size))
        printf("Error occurred\n");
    close(fh);
    return(0);
}
```

## **int fileno(FILE \*st);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	fdopen, fopen, freopen.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to stream is <i>not</i> controlled by semaphore in Multi-thread programs.

The routine `fileno` returns the file handle associated with stream `st`.

**Note:** If multiple handles have been associated with a stream (e.g., by using `dup` or `dup2`), the handle returned is that assigned when the stream was first opened.

### **Return value**

Returns the file handle. If the stream is not open, the **Return value** is undefined.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <stat.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
    FILE *f;
    long file_size;
    if(argc != 3) abort();
    f=fopen(argv[1], "w");
    if(f == NULL)
    {
        printf("Could not create %s\n", argv[1]);
        abort();
    }
    sscanf(argv[2], "%ld", &file_size);
    if ((chsize(fileno(f), file_size)) ||
        (filelength(fileno(f)) != file_size))
        printf("Error occurred\n");
    fclose(f);
    return(0);
}
```

```
int findfirst(  
    const char *path,  
    struct fblk *buffer,  
    int attrib);
```

---

```
int findnext(struct fblk *buffer);
```

---

<b>Header file</b>	dos.h
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The functions `findfirst` and `findnext` search a directory for files having the specified names and attributes.

**buffer** points to a `fblk` structure, which contains information about the file (such as its name, size, attributes, etc.). The `fblk` structure is defined in `dos.h` as follows:

```
struct fblk {  
    char ff_reserved[21];  
    char ff_attrib;  
    int ff_ftime;  
    int ff_fdate;  
    long ff_fsize;  
    char ff_name[13];  
};
```

**attrib** specifies the attributes of the file(s) being sought. This parameter can have the following values:

<code>FA_RDONLY</code>	file is read only
<code>FA_HIDDEN</code>	hidden file
<code>FA_SYSTEM</code>	system file
<code>FA_LABEL</code>	file name is a volume label
<code>FA_DIREC</code>	directory
<code>FA_ARCH</code>	archive file

**path** specifies the complete name of the file to be sought. This parameter may include wildcard characters.

`findfirst` finds the first file satisfying the search criteria, and returns information about this file in the `buffer` argument.

`findnext` finds subsequent files that satisfy the criteria specified in the call to `findfirst`. `findnext` must be passed the same `ffblk` parameter as `findfirst` was. Information about a matching file will be returned from each call to `findnext`, until there are no more matching files.

### Return value

Both functions return 0 if a matching file is found. When there are no more matches, or if an error occurs, the functions return -1 and `errno` is set to one of the following values:

`ENOENT`      file not found.

`ENMFILE`      no more files.

### Example

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

main()
{
    char *path="C:\\*.\\*";
    struct ffblk find_buf;
    int error;

    /* list normal files */
    error=findfirst(
        path, &find_buf, FA_NORMAL);
    while(!error)
    {
        printf("File - %s\\n", find_buf.ff_name);
        error=findnext(&find_buf);
    }
    return(0);
}
```



---

**short far \_floodfill(short x, short y, short boundary);**

---

**M****Header file** graph.h**See also** \_getcolor, \_getfillmask, \_setfillmask, \_setcliprgn, \_setcolor.**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function \_floodfill from the graphics module fills a specified display area, using the current color and fill mask.

**x,y** coordinates of the starting point.

**boundary** specifies the stopping point for the fill process.

The fill begins at the starting point. If this point lies inside the figure, the interior is filled; if the starting point lies outside the figure, the background is filled.

**Return value**

The function returns a nonzero value if successful; otherwise the function returns 0 if any of the following is true:

- the fill could not be completed.
- the starting point lies on the boundary color.
- the starting point lies outside the clipping region.

**Example**

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

void draw_logo(void);

main()
{
    _setvideomode(_ERESCOLOR);
    draw_logo();
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

```
void draw_logo(void)
{
    _setactivepage(1);
    _setcolor(9);
    _rectangle(
        _GFILLINTERIOR,
        140, 20, 500, 330);
    _setcolor(8);
    _moveto(225, 110);
    _lineto(320, 140);
    _lineto(415, 110);
    _lineto(460, 125);
    _lineto(460, 225);
    _lineto(365, 255);
    _lineto(365, 285);
    _lineto(320, 300);
    _lineto(180, 255);
    _lineto(180, 190);
    _lineto(225, 205);
    _lineto(225, 235);
    _lineto(270, 250);
    _lineto(270, 190);
    _lineto(180, 160);
    _lineto(180, 125);
    _lineto(225, 110);

    _moveto(180, 125);
    _lineto(320, 170);
    _lineto(460, 125);
    _moveto(320, 170);
    _lineto(320, 300);

    _moveto(365, 190);
    _lineto(415, 175);
    _lineto(415, 210);
    _lineto(365, 225);
    _lineto(365, 190);
    _lineto(415, 210);

    _moveto(225, 205);
    _lineto(270, 190);
    _moveto(225, 235);
    _lineto(270, 220);
    _moveto(180, 190);
    _lineto(225, 175);

    _floodfill(250, 130, 8);
    _floodfill(230, 180, 8);
    _floodfill(260, 240, 8);
    _floodfill(380, 210, 8);

    _setcolor(15);
    _floodfill(200, 230, 8);
    _floodfill(250, 220, 8);
    _floodfill(400, 190, 8);
    _floodfill(440, 190, 8);

    _setvisualpage(1);
    return;
}
```

---

**double floor(double x);**

---

**A**

---

**long double floorl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	ceil, fmod.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function floor returns the value of its double precision floating point argument, x, rounded towards zero to the nearest integer. The function floorl is the same as floor but takes a long double argument and returns a long double value.

**Return value**

The function returns the double result of the rounding. There is no error return.

**Example**

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

main()
{
    double x;

    x=floor(21.79);
    printf("floor of 21.79 is %g\n", x);
    return(0);
}
```

## int flushall(void);

## A

<b>Header file</b>	stdio.h
<b>See also</b>	fflush.
<b>Portability</b>	ANSI.
<b>Multi-thread</b>	Access to stream is controlled by semaphore in Multi-thread programs.

The function `flushall` flushes all open streams. If the last operation involving a stream was output, the contents of the buffer are written to the associated file. If the stream's last operation was input, the state of the stream remains unchanged.

**Note:** flushing a stream negates the effect of any preceding call to `ungetc` against the stream.

Buffers are automatically flushed when a process terminates or a stream is closed.

On an unbuffered stream, the only effect of `fflush` is to negate the `ungetc` call.

### Return value

The function returns the number of open streams. There is no error return.

### Example

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

main()
{
    FILE *f;
    char *file_name="TEMP.$$$";

    f=fopen(file_name, "r+");
    fprintf(f,
        "Formatted output to file %s", file_name);
    flushall(); /* flushes all streams */
    fprintf(stdout,
        "More formatted output to stdout");
    fclose(f);
    return(0);
}
```

---

**void far \*\_fmalloc(size\_t size);**

---

Far memory variant of malloc. Allocates far item from far heap, and returns far pointer.

---

**int far \_fmemcmp(const void far \* s1, const void far \* s2, size\_t n);**

---

Far memory variant of memcmp.

---

**void far \* far \_fmemcpy(void far \* dest, void far \* source, size\_t n);**

---

Far memory variant of memcpy.

---

**void far \* far \_fmemccpy(void far \*d, const void far \*s, int c, size\_t n);**

---

Far memory variant of memccpy.

---

**void far \* far \_fmemchr (const void far \* s, int c, size\_t n);**

---

Far memory variant of memchr.

---

**int far \_fmemicmp(const void far \* s1, const void far \* s2, size\_t n);**

---

Far memory variant of memicmp.

---

**void far \* far \_fmemset (void far \* s, int c, size\_t n);**

---

Far memory variant of memset.

---

**double fmod(double x, double y);**

---

**A**

---

**long double fmodl(long double x, long double y);**

---

<b>Header file</b>	math.h
<b>See also</b>	ceil, fabs, floor.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `fmod` calculates the floating point remainder of its two double precision arguments, `x` and `y`. The function `fmodl` is the same as `fmod` but takes long double arguments and returns a long double value.

**Return value**

The function returns the floating point remainder of `x/y`. If `y` is 0, the function returns 0.

**Example**

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

double print_modulus(double x, double y)
{
    double mod;

    mod=fmod(x, y);
    printf("mod of %g/%g is %g\n", x, y, mod);
    return(mod);
}
```

---

**int \_fmsize(void far \*buffer);**

---

Far heap variant of `_msize`

```
void fnmerge(  
    char *path,  
    const char *drive,  
    const char *dir,  
    const char *name,  
    const char *ext);
```

---

<b>Header file</b>	dir.h
<b>See also</b>	fnsplit, _make_path
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	None.

The function `fnmerge` builds a complete path name from component parts.

**path** specifies the location at which the complete path name will be stored. The maximum length for this parameter is determined by `MAXPATH`, which is defined in `dir.h`.

**drive** specifies the drive name, and must end with a colon (:).

**dir** specifies the complete directory, including a trailing backslash (\).

**name** specifies the actual file name, excluding extension.

**ext** specifies the file extension, and must begin with a period (.).

The program must have enough room for the entire file name in `path`.

### Return value

None.

### Example

```
#include <stdio.h>  
#include <dir.h>  
  
main()  
{  
    char full_path[80];  
    fnmerge(full_path,  
        "D:", "\\TSC\\", "TEMP", ".$$$");  
    printf("Path is %s\n", full_path);  
    return(0);  
}
```

```
int fnsplit(  
    const char *path,  
    char *drive,  
    char *dir,  
    char *name,  
    char *ext);
```

---

<b>Header file</b>	dir.h
<b>See also</b>	fnmerge, _split_path
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	None.

The function `fnsplit` breaks a complete path name into its component parts Æ drive, directory, file name, and extension.

**path** specifies the complete file name, which is to be broken into components. The maximum length for this argument is `MAXPATH`, whose value is defined in `dir.h`.

**drive** specifies the drive, which will end with a colon. The maximum length for this argument is `MAXDRIVE`, whose value is defined in `dir.h`.

**dir** specifies the directory, including leading and trailing backslashes. The maximum length for this argument is `MAXDIR`, whose value is defined in `dir.h`.

**name** specifies the file name. The maximum length for this argument is `MAXFILE`, whose value is defined in `dir.h`.

**ext** specifies the extension, including a leading period. The maximum length for this argument is `MAXEXT`, whose value is defined in `dir.h`.

### Return value

The function returns an integer containing flags to indicate whether each component was present in the complete file name. The flags are:

**EXTENSION** specifies whether an extension component was present.

**FILENAME** specifies whether a file name component was present.

**DIRECTORY** specifies whether a directory component was present.

**DRIVE** specifies whether a drive component was present.



**WILDCARDS** specifies whether the file name included wildcards.

### Example

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

main()
{
    char full_path[]="D:\\TSC\\TEMP.$$$";
    char drive[4];
    char dir[20];
    char name[9];
    char ext[5];

    fnsplit(full_path, drive, dir, name, ext);
    printf("Path is %s, %s %s %s %s\n",
        full_path, drive, dir, name, ext);
    return(0);
}
```

---

**FILE \*fopen(const char \*path, const char \*type);**

---

**A****Header file**           stdio.h**See also**            fclose, fcloseall, fdopen, ferror, fileno, freopen, setmode.**Portability**        ANSI**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function `fopen` opens the file specified by `path`, and associates a stream with that file. The character string `type` specifies the access mode for the file, and can be any of the following:

“r” Read only. If the file does not exist or can’t be found, the call to `fopen` will fail.

“w” Write only. If the file exists, it is truncated and its contents are destroyed. If it does not exist, it is created.

“a” Append (i.e., write only at end of file). If the file does not exist, it is created.

“r+” Read and Write. If the file does not exist or can’t be found, the call to `fopen` will fail.

“w+” Read and Write. If the file exists, it is truncated and its contents are destroyed. If it does not exist, it is created.

“a+” Read and Append. If the file does not exist, it is created.

In addition to these values, the characters ‘t’ or ‘b’ may be added after the first character of type `Æ` to specify text or binary translation modes. E.g.,

“r+b” or “rb+”  
Read and Write, binary mode.

“w+t” or “wt+”  
Read and Write, text mode.

If neither ‘t’ nor ‘b’ is specified, the stream’s translation mode is specified by the variable `_fmode`. The default value of `_fmode` is for text mode.

**Note:** If the stream allows both reading and writing, there must be an intervening call to `fseek`, `fsetpos`, or `rewind` between changes in the direction of I/O. If an input stream encounters EOF, then the next operation may be a read operation.

When a file is opened with “a” or “a+”, all write operations will take place at the end of the file, regardless of any preceding calls to fseek, fsetpos or rewind.

When a file is opened, both the end of file and error indicators are clear.

### Return value

The function returns a pointer to the open stream. If the stream could not be successfully opened, a NULL pointer is returned.

### Example:

```
include <stdio.h>

void copy(FILE *old_file, FILE *new_file);

main(int argc, char *argv[])
{
    FILE *old, *new;

    if(argc != 3)
    {
        printf("Usage - COPY oldfile newfile\n");
        return(1);
        /* open to read in text mode */
        old=fopen(argv[1], "rt");
        if(old == NULL)
        {
            printf("Can't open %s\n", argv[1]);
            return(1);
            /* create or truncate for */
            /* writing in text mode */
        }

        new=fopen(argv[2], "wt");
        if(new == NULL)
        {
            printf("Can't open %s\n", argv[2]);
            return(1);
        }
        copy(old, new);
    }
    return(0);
}
```

---

**void \_fpreset(void);**

---

<b>Header file</b>	math.h
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	None.

The function `_fpreset` re-initializes the floating point math package, allowing recovery from floating point errors. This function is redundant since the normal exception handling mechanism clears exceptions.

---

**int fprintf(FILE \*st, const char \*format, ...);**

---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	printf
<b>Portability</b>	ANSI

The function `fprintf` provides formatted output to the stream `st`. Formatting conventions are identical to `printf`.

**Example**

```
#include <stdio.h>

void print_error(char *msg)
{
    /* formatted output to stderr */
    fprintf(stderr, "Error %s", msg);
    return;
}
```

## **int fputc(int c, FILE \*st);**

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	fgetc, fgetchar, fputchar,getc, getchar
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is <i>not</i> controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function fputc writes a character (converted to an unsigned char) to the current position of stream st, and increments the stream pointer, if any.

**Note:** fputc is identical to putc, but is a function not a macro.

### **Return value**

The function returns the character written, as an integer value. A Return value of -1 may indicate an error or end of file condition. To determine which condition holds, use ferror or feof.

### **Example**

```
#include <stdio.h>

int print_message(FILE *f, char *msg)
{
    int n=0;
    int c;

    while((c=msg[n]) != 0)
    {
        fputc(c, f); /* output character */
        ++n;
    }
    return(n);
}
```

## **int fputc(int c);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	fgetc, fgetchar, fputc,getc, getchar
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to stream is <i>not</i> controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function `fputc` writes a character (converted to an unsigned char) to `stdout`. The function is equivalent to `fputc(c, stdout)`.

**See `fputc`.**

### **Return value**

The function returns the character written, as an integer value. A Return value of -1 may indicate an error or end of file condition. To determine which condition holds, use `ferror` or `feof`.

### **Example**

```
#include <stdio.h>

int print_message(FILE *f, char *msg)
{
    int n=0;
    int c;

    while((c=msg[n]) != 0)
    {
        fputc(c, f);
        ++n;
    }
    return(n);
}
```

---

**int fputs(const char \*strng, FILE \*stream);**

---

**A**

<b>Header file</b>	stdio.h
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is <i>not</i> controlled by semaphore in Multi-thread programs. Use Lock if another thread may be accessing the specified stream.

The function fputs writes strng to the specified stream, at the current position in the stream. The null terminator is not copied, nor is a newline copied to the stream.

**Return value**

The function returns the last character written, if successful; otherwise, the function returns EOF.

**Example**

```
#include <stdio.h>

void print_string(char *file, char *msg)
{
    FILE *f;

    f=fopen(file, "r+");
        /* output string to stream f */
    if(fputs(msg, f) == EOF)
        fprintf(stderr, "Error in print_string\n");
    return;
}
```

## **unsigned FP\_OFF(void \*pointer);**

---

**Header file**            dos.h

**See also**            segread, MK\_FP.

**Portability**        8086 Family.

**OS2**                FP\_SEG returns a segment selector.

The FP\_SEG and FP\_OFF macros return segment and offset, respectively, of the argument pointer, which can be any pointer type.

### **Example**

```
#include <dos.h>

void *_mouse_savedriver(void *driver_state)
{
    union REGS r;
    struct SREGS s;

    r.x.ax=0x16;

    #if (defined M_I86SM) | (M_I86MM)
        r.x.dx=(unsigned) driver_state;
        s.es=_DS;
    #else
        /* offset of far pointer */
        r.x.dx=FP_OFF(driver_state);
        /* segment of far pointer */
        s.es=FP_SEG(driver_state);
    #endif

    int86x(0x33, &r, &r, &s);
    return(driver_state);
}
```



```
size_t fread(  
    void *buffer,  
    size_t size,  
    size_t nitems,  
    FILE *stream);
```

---

A

<b>Header file</b>	stdio.h
<b>See also</b>	fopen, fwrite, printf, read
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is controlled by semaphore in Multi-thread programs.

The function `fread` reads a specified number of data elements from a stream.

<b>buffer</b>	points to the block in which the data will be stored.
<b>size</b>	specifies the size (in bytes) of each element being read.
<b>nitems</b>	specifies the number of items to read.
<b>stream</b>	specifies the stream from which the data will be read.

### Return value

The function returns the number of complete *items* read. If this value is less than `nitems`, use `feof` or `ferror` to determine whether the end of file was reached or whether another error occurred.

### Example

```
#include <stdio.h>  
  
main()  
{  
    FILE *f;  
    int n=0;  
    int numbers[10];  
    f=fopen("TEMP.$$$", "rb");  
    fread(numbers, sizeof(int), 10, f);  
    /* read 10 integers */  
    while(n < 10)  
    {  
        printf("Number %d is %d\n", n, numbers[n]);  
        ++n;  
    }  
    fclose(f);  
    return(0);  
}
```

---

**void far \*\_frealloc(void far \*buffer, size\_t size);**

---

Far heap variant of realloc.

---

**void free(void \*buffer);**

---

**A**

<b>Header file</b>	stdlib.h and alloc.h
<b>Variants</b>	_nfree, _ffree, farfree, hfree
<b>See also</b>	calloc, malloc, realloc, strdup
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Far heap only protected by semaphore

The function free de-allocates a block of memory previously allocated with malloc, calloc, or realloc.

**Return value**      None.

**Example**

```
#include <alloc.h>

main()
{
    char *ptr;
    char near *near_ptr;
    char far *far_ptr;
        /* allocate and then free */
        /* block in default heap */
    ptr=malloc(200);
    free(ptr);
        /* allocate and then free */
        /* block in default heap */
    near_ptr=_nmalloc(200);
    _nfree(near_ptr);

    far_ptr=_fmalloc(200);
    _ffree(far_ptr);

    return(0);
}
```

## **unsigned \_freet(size\_t size);**

---

**Header file**            alloc.h

**See also**             calloc, malloc, realloc

**Portability**         Some DOS/OS2

\_freet determines the number times an object of size bytes can be allocated from the near heap by a call to malloc or calloc in small and medium models, or by calls to \_nmalloc or \_ncalloc.

### **Return value**

The function returns a value representing the number of times the specified object can be allocated.

### **Example**

```
#include <alloc.h>

int fill_heap(int size)

{
    unsigned total_blocks;
    int n=0;
        /* number of blocks available */
    total_blocks=_freet(size);
    while(n++ < total_blocks)
        malloc(size); /* allocate them */
    return(total_blocks);
}
```

## **int freemem(unsigned segx);**

---

<b>Header file</b>	dos.h
<b>See also</b>	allocmem
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant. Far heap only protected by semaphore

The function `freemem` returns memory previously allocated by `allocmem` to DOS.

**segx** specifies the segment address of the memory being returned.

The memory must have been allocated by a call to `allocmem`.

### **Return value**

The function returns 0 if successful. In the case of error, the function returns -1 and sets `errno` to `ENOMEM` (not enough memory).

### **Example**

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

main()
{
    char far *buffer;
    unsigned new_seg;
    int error;

    error=allocmem(100, &new_seg);
    if(error)
    {
        fprintf(stderr,
            "Error allocating memory\n");
        return(1);
    }
    buffer=MK_FP(new_seg, 0);
    printf("Memory is at %Fp\n", buffer);
    freemem(new_seg);
    return(0);
}
```

```
FILE *freopen(  
    const char *path,  
    const char *type,  
    FILE *stream);
```

---

**Header file**           stdio.h

**See also**             fclose, fcloseall, fdopen, fileno, fopen, setmode.

**Portability**         ANSI

**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function `freopen` closes the file specified by `stream`, and then reassigns the stream to the file specified by `path`. The character string `type` specifies the access mode for the file, and can be any of the following:

“r” Read only. If the file does not exist or can’t be found, the call to `freopen` will fail.

“w” Write only. If the file exists, it is truncated and its contents are destroyed. If it does not exist, it is created.

“a” Append (i.e., write only at end of file). If the file does not exist, it is created.

“r+” Read and Write. If the file does not exist or can’t be found, the call to `fopen` will fail.

“w+” Read and Write. If the file exists, it is truncated and its contents are destroyed. If it does not exist, it is created.

“a+” Read and Append. If the file does not exist, it is created.

In addition to these values, the characters ‘t’ or ‘b’ may be added after the first character of `type` to specify text or binary translation modes. E.g.,

“r+b” or “rb+”  
Read and Write, binary mode.

“w+t” or “wt+”  
Read and Write, text mode.

If neither ‘t’ nor ‘b’ is specified, the stream’s translation mode is specified by the variable `_fmode`. The default value of `_fmode` is for text mode.

**Note:** If the stream allows both reading and writing, there must be an intervening call to `fseek`, `fsetpos`, or `rewind` between changes in the direction of I/O. If an input stream encounters EOF, then the next operation may be a read operation.

When a file is opened with “a” or “a+”, all write operations will take place at the end of the file, regardless of any preceding calls to `fseek`, `fsetpos` or `rewind`.

When a file is opened, both the end of file and error indicators are clear.

### Return value

The function returns a pointer to the open stream. If the stream could not be successfully opened, a NULL pointer is returned.

### Example

```
#include <stdio.h>

main()
{
    char string[32];
    /* redirect stdin to file */
    freopen("TEMP.$$$", "r", stdin);
    scanf("%32s", string);
    printf("Input from file was %s", string);
    return(0);
}
```

## **double frexp(double val, int \*exptr);**

---

**A**

long double frexpl(long double x, int \*exp);

**Header file**           math.h

**See also**           ldexp, modf

**Portability**       ANSI

**Multi-thread**      None.

The function `frexp` breaks the floating point value `val` into a *mantissa* (which is between 0.5 and 1.0) and an *exponent*. The exponent is stored in the location to which `exptr` points.

### **Return value**

The function returns the mantissa. If `val` is 0, the function returns 0 for the mantissa and the exponent. There is no error return.

### **Example**

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

main()
{
    double mantissa;
    int exponent;

    mantissa=frexp(512, &exponent);
    printf(
        "mantissa is %g, exponent is %d",
        mantissa, exponent);
    return(0);
}
```

**int fscanf(  
FILE \*stream,  
const char \*format,  
[arguments] ...);**

---

**A**

**Header file**           stdio.h

**See also**             cscanf, fprintf, scanf, sscanf

**Portability**         ANSI

**Multi-thread**        Access to stdin is controlled by semaphore in Multi-thread programs.

The function `fscanf` reads data from the current position in the specified stream. The data are read into locations specified by arguments, each of which must be a pointer to an appropriate type.

The types are determined by the contents of the format string, as described in the discussion of `scanf`.

### **Return value**

The function returns the number of fields successfully converted and assigned. If the end of file has been reached, the function returns EOF. If no fields were assigned, the function returns 0.

### **Example**

```
#include <stdio.h>

main()
{
    char string[32];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fscanf(f, "%32s", string);
    printf("Input from file was %s", string);
    fclose(f);
    return(0);
}
```



---

**int fseek(FILE \*stream, long offset, int origin);**

---

**A****Header file**           stdio.h**See also**             ftell, rewind.**Portability**         ANSI**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function `fseek` moves the file pointer associated with the specified stream. The pointer is moved to a location that is offset bytes from the specified origin. The next operation in the stream will take place at the new location.

`stream`   specifies the stream.

`offset`   specifies the number of bytes to move from origin

`origin`   specifies the reference location for the move. This must have one of the following values:

`SEEK_SET` beginning of file.

`SEEK_CUR`   current file position.

`SEEK_END`   end of file.

A call to `fseek` clears the end of file indicator, and negates any prior calls to `ungetc`.

**Note:** It is legal to position the file pointer to any location beyond the beginning of file. Only an attempt to position the pointer before the beginning of file will produce an error.

A call to `fseek` will negate the effect of any preceding call to `ungetc`.

If a file is opened for appending, the next write will occur at the end of file regardless of any call to `fseek` preceding that write.

Because of the translation performed on files opened in text mode, the only reliable operation using `fseek` in this mode is a seek with an offset of 0 relative to any origin or to a value returned by `ftell`.

If a file has read and write permission, the next operation after calling `fseek` may be either input or output.

**Return value**

If successful, `fseek` returns 0. If an error occurred, a nonzero value is returned. On character devices, the effects and Return value of `fseek` are undefined.

### Example

```
#include <stdio.h>

main() {
    char string[32];
    FILE *f1, *f2;

    f1=fopen("TEMP1.$$$", "r");
    f2=fopen("TEMP2.$$$", "r+");
    fscanf(f1, "%32s", string);
        /* seek to end of file */
    fseek(f2, 0L, SEEK_END);
    fputs(string, f2); /* output to file */
    fclose(f1);
    fclose(f2);
    return(0);
}
```

---

**int fsetpos(FILE \*st, const fpos\_t \*fp);**

---

**A****Header file**           stdio.h**See also**             fgetpos.**Portability**         ANSI**Multi-thread**       Access to stream is controlled by semaphore in Multi-thread programs.

The function fsetpos sets the current value of the stream st's file position indicator to the value pointed to by fp.

**Note:** The value stored at fp must have been obtained by a call to fgetpos.

A call to fsetpos negates the effects of any preceding call to ungetc.

If a file is opened for appending, the next write will occur at the end of file, regardless of any call to fsetpos preceding that write.

If a file has read and write permission the next operation after calling fsetpos may be either input or output.

**Return value**

If successful, fsetpos returns 0. If an error occurred, a nonzero value is returned and errno is set to one of the following values:

**EINVAL**   The stream was invalid.

**EBADF** The file handle associated with the stream was bad.

**Example**

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

main()
{
    FILE *f;
    char *file_name="TEMP.$$$";
    char second_line[128];
    fpos_t end_of_line;

    f=fopen(file_name, "r+");
    fprintf(f,
        "Formatted output to file %s", file_name);
        /* save file position */
    fgetpos(f, &end_of_line);

    fprintf(f,
        "The first word of this sentence also goes
        to stdout");
        /* restore file position */
    fsetpos(f, &end_of_line);

    fscanf(f, "%s", second_line);
    puts(second_line);
    fclose(f);
    return(0);
}
```

**int fstat(int handle, struct stat \*buf);****U**

<b>Header file</b>	stat.h
<b>See also</b>	access, chmod, filelength, stat.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Low level I/O functions are not protected by the library.
<b>OS2</b>	The values st_dev and st_rdev have no meaning under OS2.

The function fstat obtains file information associated with handle and stores it in the structure to which buf points. The structure of type stat is defined in stat.h, and contains the following fields:

st\_atime Time of last modification.

st\_ctime Time of last modification.

st\_mtime Time of last modification.

st\_dev Either the drive number of the disk containing the file or the file handle if the file is a device.

st\_rdev Either the drive number of the disk containing the file or the file handle if the file is a device.

st\_mode Bit mask for the file mode information:

S\_IFDIR set if directory.

S\_IFCHR set if device.

S\_IFREG set if ordinary file.

S\_IREAD set if read permission.

S\_IWRITE set if write permission.

st\_link Always 1.

st\_size Size of the file in bytes.

st\_ino No meaning under DOS.

st\_uid No meaning under DOS.

st\_gid No meaning under DOS.

## Return value

The function returns 0 if successful. If an error occurred, the value -1 is returned and `errno` is set to `EBADF`.

## Example

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

main()
{
    int fh;

    struct stat file_status;

    fh=open("TEMP.$$$", O_RDWR); /* open file */
    /* get file status info */
    if(fstat(fh, &file_status) == 0)
    {
        /* print file size and drive */
        printf(
            "File size is %ld bytes\n",
            file_status.st_size);
        printf(
            "File drive is %d\n",
            file_status.st_dev);
    }
    else
        printf("Error\n");
    close(fh);
    return(0);
}
```

---

**char far \* far\_fstpcpy( char far \* dest, const char far \* source);**

---

Far memory variant of stpcpy

---

**char far \* far \_fstrcat(char far \* dest, const char far \* source);**

---

Far memory variant of strcat

---

**char far \* far \_fstrchr(const char far \* s, int c);**

---

Far memory variant of strchr

---

**int far \_fstrcoll(const char far \* s1, const char far \* s2);**

---

Far memory variant of strcoll

---

**int far \_fstrcmp (const char far \* s1, const char far \* s2);**

---

Far memory variant of strcmp

---

**char far \* far \_fstrcpy(char far \* dest, const char far \* source);**

---

Far memory variant of strcpy

---

**size\_t far \_fstrcspn( const char far \* s1, const char far \* s2);**

---

Far memory variant of strcspn

---

**char far \* far \_fstrdup(const char far \* s);**

---

Far memory variant of strdup

---

**int far far \_fstricmp(const char far \* s1, const char far \* s2);**

---

Far memory variant of stricmp

---

**size\_t far\_fstrlen(const char far \* s);**

---

Far memory variant of strlen

---

**char far \* far\_fstrlwr(char far \* s);**

---

Far memory variant of strlwr

---

**char far \* far\_fstrncat(char far \* dst, const char far \* srce, size\_t n);**

---

Far memory variant of strncat

---

**int far\_fstrncmp(const char far \* s1, const char far \* s2, size\_t n);**

---

Far memory variant of strncmp

---

**int far\_fstrnicmp(const char far \* s1, const char far \* s2);**

---

Far memory variant of strnicmp

---

**char far \* far\_fstrnset(char far \* s, int ch, size\_t n);**

---

Far memory variant of strnset

---

**char far \* far\_fstrpbrk(const char far \* s1, const char far \* s2);**

---

Far memory variant of strpbrk

---

**char far \* far\_fstrchr( const char far \*s, int c);**

---

Far memory variant of strchr

---

**char far \* far\_fstrrev (char far \*s);**

---

Far memory variant of strrev



---

**char far \* far \_fstrset(char far \* s, int ch);**

---

Far memory variant of strset

---

**size\_t far \_fstrspn(const char far \*s1, const char far \*s2);**

---

Far memory variant of strspn

---

**char far \* far \_fstrstr(const char far \* s1, const char far \*s2);**

---

Far memory variant of strstr

---

**char far \* far \_fstrtok(char far \*s1, const char far \*s2);**

---

Far memory variant of strtok

---

**char far \* far \_fstrupr(char far \* s);**

---

Far memory variant of strupr

---

**size\_t far \_fstrxfrm( char far \* s1, const char far \* s2, size\_t n);**

---

Far memory variant of strxfrm

## **long ftell(FILE \*st);**

**A**

**Header file**           stdio.h

**See also**            fgetpos, fseek, lseek, tell.

The function returns the position of the file pointer. If an error occurred, the value -1L is returned and `errno` is set to one of the following values:

**EINVAL**   Invalid stream.

**EBADF**    Bad file handle.

### **Example**

```
#include <stdio.h>

main()
{
    FILE *f;
    char *file_name="TEMP.$$$";
    char second_line[128];
    long end_of_line;

    f=fopen(file_name, "r+");
    fprintf(f,
        "Formatted output to file %s", file_name);
    end_of_line=ftell(f); /* save file position */
    fprintf(f,
        "The first word of this sentence also goes
        to stdout");
        /* restore file position */
    fseek(f, end_of_line, SEEK_SET);
    fscanf(f, "%s", second_line);
    puts(second_line);
    fclose(f);
    return(0);
}
```

**void ftime(struct timeb \*timeptr)****M****Header file**           timeb.h**See also**             asctime, ctime, gmtime, localtime, mktime, stime, strftime, time**Portability**         DOS/ OS2

Provided for compatibility with the Microsoft library, ftime gets the current time and stores it in the structure pointed to by timeptr. The timeb structure is defined in time.h:

```
struct timeb {  
    time_t  time;  
    unsigned short millitm;  
    short   timezone;  
    short   dstflag;  
};
```

The fields have the following meanings:

Field	Meaning
dstflag	Nonzero if daylight saving currently in effect in the local time zone.
millitm	Fraction of a second in milliseconds
time	The time in seconds since 00:00:00 GMT on 1 January 1970
timezone	The difference in minutes, moving west, between GMT and local time.

**Return value**

None

---

**int fwrite(const void \*buffer, size\_t size, size\_t num, FILE \*st);**

---

**A****Header file**           stdio.h**See also**             fread.**Portability**         ANSI**Multi-thread**        Access to stream is controlled by semaphore in Multi-thread programs.

The function `fwrite` writes up to `num` blocks, each of `size` bytes, from `buffer` to the stream `st`. The stream pointer `Æ` if there is one `Æ` is incremented by the number of bytes written.

**Note:** If an error occurs, the position of the stream pointer or the state of a partially written item are undefined.

**Return value**

The function returns the number of complete items actually written. If an error or end of file condition occurs, this number will be less than `num`. The functions `ferror` or `feof` should be used to ascertain which condition caused `fwrite` to terminate. If either `num` or `size` is 0, the Return value will be 0 and no bytes are written.

**Example**

```
#include <stdio.h>

main()
{
    FILE *f;
    int n=0;
    int numbers[10];
    f=fopen("TEMP.$$$", "wb");
    while(n < 10)
    {
        if(scanf("%d", &numbers[n]) == -1)
            break;
        ++n;
    }
    /* write 10 integers */
    fwrite(numbers, sizeof(int), n, f);
    fclose(f);
    return(0);
}
```

---

**char \*gcvt(double value, int digits, char \*buf);** **U**

---

---

**char \*gcvtl(long double value, int digits, char \*buf);** **U**

---

**Header file**            stdlib.h**See also**             ecvt, fcvt.**Portability**         UNIX/DOS/OS2. Use sprintf for future **Portability****Multi-thread**       None.

The function gcvt converts a double precision floating point value to a character string. The function gcvtl is the same as gcvt but takes a long double argument.

value    is the floating point value to be converted.

digits   is the number of digits to be stored.

buffer   points to a character array large enough to hold the converted string.

gcvt attempts to convert the value to decimal format. If this is not possible, exponential format will be used.

**Return value**

The function returns a pointer to the string of digits. There is no error return.

**Example**

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

main()
{
    char result[64];
    double d1=3.1415926535;
        /* convert real to string */
    gcvt(d1, 10, result);
    printf("%s\n", result); /* print string */
    return(0);
}
```

## **void geninterrupt(int nr);**

---

**Header file**            dos.h

**See also**              bdos, bdosptr, getvect, int86, int86x, intdos, intdosx, intr

**Portability**          80x86

**Multi-thread**        DOS is not re-entrant

This macro triggers a trap for the interrupt specified by nr. nr must be a constant. The values in the registers after the call depend on what interrupt was called.

**Return value**

None.

## **long far \_getbkcolor(void);**

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_setbkcolor
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getbkcolor` from the graphics module returns the current background color. The default value is 0.

### **Return value**

Returns the current background color. There is no error return.

### **Example**

```
#include <graph.h>
#include <conio.h>
main()
{
    long color;
    _setvideomode(_MRES4COLOR);
    _setbkcolor(_BLUE);
    color=_getbkcolor();
    getch();
    _setbkcolor(color|_RED);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**int getc(FILE \*st);**

---

**A****Header file**           stdio.h**See also**             fputc, fputchar, getchar, fgetc, fgetchar**Portability**         ANSI**Multi-thread**       Access to stream is *not* controlled by semaphore in Multi-thread programs.

The routine `getc` reads a character from the current position of stream `st`, and increments the stream pointer, if any.

**Note:** The routine `getc` is equivalent to `fgetc`, but is a macro instead of a function.

**Return value**

The macro returns the character read as an integer value. A Return value of EOF may indicate an error or end of file condition. To determine which is the case, use `ferror` or `feof`.

**Example**

```
#include <stdio.h>

char *read_string(char *str)
{
    int c;
    int n=0;
    /* read character */
    while((c=getc(stdin)) != EOF)
    {
        if(c == '\n') /* break when CR read */
            break;
        str[n++]=c;
    }
    str[n]='\0';
    return(str);
}
```



## **int getcbrk(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	ctrlbrk, setcbrk
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `getcbrk` returns the current setting for the control-break checking flag. The function accomplishes this by calling INT 33H.

### **Return value**

The function returns 0 if the control-break flag is off (no checking), and 1 if the flag is on.

### **Example**

```
#include <dos.h>

main()
{
    int setting, newsetting;

    setting = getcbrk();
    if ( setting)
    {
        printf ( "Currently checking.\n");
        newsetting = setcbrk ( 0);
        newsetting = getcbrk();
        if ( !newsetting)
        {
            printf ( "Not checking now.\n");
            setting = setcbrk (setting);
            printf ( "original setting restored.\n");
        }
    }
    else
        printf ( "Not checking.\n");
}
```

## **int getch(void);**

---

**Header file**            conio.h

**Portability**            UNIX/DOS/OS2

**Multi-thread**          Not re-entrant when using Turbo C window module.

The function `getch` reads a character from console without echoing. If an extended scan key is pressed, the first call will return 0, and the second call returns the actual key code.

**Note:**    `getch` does not return a code for Ctrl-S. Use DOS function 6.

### **Return value**

The function returns the character read. There is no error **Return value**.

### **Example**

```
#include <conio.h>

char *read_string(char *str)
{
    int c;
    int n=0;

    while(1)    {
        c=getch(); /* read character from console */
        putch(c);
        if(c == '\n')
            break;
        str[n++]=c;
    }
    str[n]='\0';
    return(str);
}
```

## **int getchar(void);**

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	fputc, fputchar, getc, fgetc, fgetchar
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to the stream is <i>not</i> controlled by semaphore. Use Lock if another thread may be accessing the specified stream.

The routine `getc` reads a character from `stdin`. The routine `getchar` is equivalent to `getc(stdin)`. See `getc`.

**Note:** The routine `getchar` is equivalent to `fgetchar`, but is a macro instead of a function.

### **Return value**

The macro returns the character read as an integer value. A Return value of EOF may indicate an error or end of file condition. To determine which is the case, use `ferror` or `feof`.

### **Example**

```
#include <stdio.h>

char *read_string(char *str)

{
    int c;
    int n=0;

    while((c=getchar()) != EOF) {
        if(c == '\n')      break;
        str[n++]=c;
    }
    str[n]='\0';
    return(str);
}
```

## **int getche(void);**

---

**Header file**            conio.h

**Portability**          UNIX/DOS/OS2

**Multi-thread**        Not re-entrant when using Turbo C window module.

The function `getche` reads a character from console, and echoes the character to the screen. If an extended scan key is pressed, the first call will return 0 and the second call returns the actual key code.

The cursor will wrap within any currently defined window.

**Note:** `getche` does not return a code for Ctrl-S. Use DOS function 6.

### **Return value**

The function returns the character read. There is no error **Return value**.

### **Example**

```
#include <conio.h>

char *read_string(char *str)
{
    int c;
    int n=0;

    while(1)
    {
        /* read character from console */
        /* and echo to screen */
        c=getche();
        if(c == '\n')
            break;
        str[n++]=c;
    }
    str[n]='\0';
    return(str);
}
```

## short far \_getcolor(void);

## M

<b>Header file</b>	graph.h
<b>See also</b>	_setcolor
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getcolor` from the graphics module returns the current color used by functions that write to the graphics screen. The default value for color has the highest valid color value.

### Return value

The function returns the previous color value.

### Example

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

main()
{
    short c;
    _setvideomode(_ERESCOLOR);
    _setcolor(1);
    _rectangle(_GFILLINTERIOR, 20, 20, 400, 180);
    _setcolor(4);
    _ellipse(_GBORDER, 20, 20, 400, 180);
    c=_getcolor();
    _setcolor(3);
    _floodfill(100, 100, c);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **int getcurdir(int drive, char \*dir);**

---

**Header file**            dir.h

**See also**             chdir, getcwd, getdisk, mkdir, rmdir

**Portability**         DOS

**Multi-thread**        None.

The function getcurdir returns the current working directory for the specified drive.

drive    specifies the drive. 0 is the default drive;  
1 is drive A, etc.

dir    points to the location at which the directory name will be stored.

The directory name is null terminated, but contains neither the drive specifier nor a leading backslash (\).

### **Return value**

The function returns 0 if successful and -1 if there is an error.

### **Example**

```
#include <dir.h>

main()
{
    int result;
    char dir_str[100];

    result = getcurdir ( 0, dir_str);
    if (!result)
        printf (
            "Current dir on default drive is %s\n",
            dir_str);
    else
        printf (
            "Error getting directory info \n");
}
```

---

**struct xycoord far \_getcurrentposition(void);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_moveto
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

stored in an xycoord structure, which is defined in graph.h:

```
struct xycoord {
    short xcoord;
    short ycoord;
};
```

**Note:** This call does not indicate the position of the text cursor.

**Return value**

The function returns the current position in an xycoord structure. There is no error return.

**Example**

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

main()
{
    struct xycoord cur_pos;

    _setvideomode(_MRESNOCOLOR);
    _moveto(20, 20);
    _lineto(20, 90);
    cur_pos=_getcurrentposition();
    _moveto(cur_pos.xcoord+30, 20);
    /* draw parallel lines */
    _lineto(cur_pos.xcoord+30, 90);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **char \*getcwd(char \*path, int n);**

---

<b>Header file</b>	dir.h
<b>See also</b>	chdir, mkdir, rmdir
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function `getcwd` gets the path name of the current working directory and stores it at `path`.

`n` specifies the maximum length for the path.

If `path` is `NULL`, a buffer of minimum size `n` will be allocated using `malloc`. It is the user's responsibility to free this block when necessary.

### **Return value**

The function returns a pointer to the path string. If an error occurs, the value `NULL` is returned and `errno` is set to one of the following values:

<b>ENOMEM</b>	Insufficient memory to allocate <code>n</code> bytes for path name when <code>path</code> argument was <code>NULL</code> .
<b>ERANGE</b>	Path name is longer than <code>n</code> characters.

### **Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
main()
{
    char *directory;
        /* save current directory */
    directory=getcwd(NULL, 128);
    if(directory == NULL)
        printf(
            "Can't save directory - error %d\n",
            errno);
    rest_of_program();
    if(directory != NULL)
    {
        /* restore directory */
        chdir(directory);
        free(directory);
    }
    return(0);
}
```



## **void getdate(struct date \*dptr);**

---

**Header file**           time.h and dos.h

**See also**             ctime, gettime, setdate, settime

**Portability**          DOS

**Multi-thread**        DOS is not re-entrant

The function getdate returns the current date information in the structure to which dptr points. The date structure is defined in time.h as follows:

```
struct date {  
    int da_year;  
    unsigned char da_day;  
    unsigned char da_mon;  
};
```

### **Return value**

None.

### **Example**

```
#include <time.h>  
  
main()  
{  
    struct date *datptr;  
    datptr =  
        (struct date *)  
        malloc (  
            sizeof ( struct date));  
    getdate ( datptr);  
    printf (  
        "Today's date is: %u / %u / %d\n",  
        datptr->da_mon,  
        datptr->da_day,  
        datptr->da_year);  
}
```

---

**void getdfree(unsigned char drive, struct dfree \*dtable);**

---

<b>Header file</b>	dos.h
<b>See also</b>	date structuregetfat, getfatd
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function getdfree determines the free space available on a disk.

**drive** specifies the drive to be checked, with 0 being the default drive, 1 = drive A, 2 = drive B, etc.

**dtable** points to the dfree structure which will contain information about the storage situation on a disk. The dfree structure is defined in dos.h as follows:

```
struct dfree {
    unsigned df_avail; /* available clusters */
                    /* total number of clusters */
    unsigned df_total;
                    /* number of bytes per sector */
    unsigned df_bsec;
                    /*number of sectors per cluster */
    unsigned df_sclus;
};
```

**Return value**

There is no **Return value**. In the case of error, the df\_sclus member is set to -1.

**Example**

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

main()
{
    struct dfree drive;

    getdfree(0, &drive);
    printf("%d clusters free\n", drive.df_avail);
    return(0);
}
```

## **int getdisk(void);**

---

<b>Header file</b>	dir.h
<b>See also</b>	getcurdir, getcwd, setdisk
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function getdisk determines the current drive number, and returns this value. 0 is drive A, 1 is drive B, etc.

### **Return value**

The function returns a code representing the current drive.

### **Example**

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

main()
{
    int drive;

    drive=getdisk();
    printf("Current drive is %c:\n", drive+'A');
    return(0);
}
```

## **int \_getdrive(void);**

---

<b>Header file</b>	dir.h
<b>See also</b>	getdisk, setdisk
<b>Portability</b>	MSDOS

Returns the current drive: 0 = A:, 1 = B:, etc

## **char far \* getdta(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	setdta
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getdta` returns the location of the disk transfer area (DTA).

### **Return value**

The function returns a far pointer to the location of the DTA.

### **Example**

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

char new_dta[128];

main()
{
    char far * buffer;

    setdta((char far *) new_dta);
    buffer=getdta();
    printf("DTA is now %Fp\n", buffer);
    return(0);
}
```

---

**char \*getenv(const char \*name);**

---

**A****Header file**            stdlib.h**See also**             putenv.**Portability**          ANSI**Multi-thread**        Access to the environment is controlled by semaphore.**OS2**                 The function is case sensitive.

Searches the environment variables for an entry corresponding to name.

**Return value**

Returns a pointer to the environment variable containing the string value of name. If the variable is not defined the **Return value** will be NULL.

**Note:** Do not modify this variable directly. Copy the string using strcpy or strdup.

**Example**

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

FILE *find_header(char *file)
{
    char path[80];
    char *env_var;
    char *pos;
    FILE *f;

    /* find header file */
    /* using INCLUDE variable */
    env_var=getenv("INCLUDE");
    if(env_var == NULL)
        return(NULL);
    strcpy(path, env_var);
    pos=strchr(path, ';');
    if(pos == NULL)
        pos=strchr(path, '\\0');
    *pos++='\\';
    strcpy(pos, file);
    f=fopen(path, "r+");
    return(f);
}
```

---

**void getfat(unsigned char drive, struct fatinfo \*dtable);**

---

<b>Header file</b>	dos.h
<b>See also</b>	getdfree, getfatd
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getfat` gets information about the file allocation table for a specified drive.

`drive` specifies the drive whose FAT information is desired. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc.

`dtable` points to a `fatinfo` structure whose members will contain the desired FAT information. The `struct fatinfo` definition is included in `dos.h`, and is as follows:

```
struct fatinfo
{
    char  fi_sclus; /* # sectors per cluster */
    char  fi_fatid; /* ID byte for FAT */
    int   fi_nclus; /*# of clusters */
    int   fi_bysec; /*# bytes per sector */
};
```

**Return value**

None.

**Example**

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

main()
{
    struct fatinfo fat;

    getfat(0, &fat);
    printf(
        "Bytes per sector = %d\n", fat.fi_bysec);
    return(0);
}
```

## **void getfatd(struct fatinfo \*dtable);**

---

<b>Header file</b>	dos.h
<b>See also</b>	getdfree, getfat
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getfatd` gets information about the file allocation table for the current drive.

`dtable` points to a `fatinfo` structure whose members will contain the desired FAT information. The `struct fatinfo` definition is included in `dos.h`, and is as follows:

```
struct fatinfo {
    char  fi_sclus; /* # sectors per cluster */
    char  fi_fatid; /* ID byte for FAT */
    int   fi_nclus; /* # of clusters */
    int   fi_bysec; /* # bytes per sector */
};
```

### **Return value**

None.

### **Example**

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

main()
{
    struct fatinfo fat;

    getfatd(&fat);
    printf(
        "Bytes per sector = %d\n", fat.fi_bysec);
    return(0);
}
```

---

**unsigned char far \* far \_getfillmask(unsigned char far \*mask\_array);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_floodfill, _setfillmask
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getfillmask` from the graphics module returns a pointer to the contents of the current fill mask and copies this to the mask array

`mask_array` points to an array that specifies the value for an 8x8 collection of bits  $\mathbb{E}$  to contain the fill mask. Each bit in this array can be 1 or 0. If the value is 1, the pixel corresponding to that point will be on; otherwise the pixel will be off.

**Return value**

If no mask is present the function returns `NULL`; otherwise, the function returns a pointer to the fill mask.

**Example**



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

main()
{
    unsigned char new_mask[8];
    unsigned char old_mask[8];
    short px[]={ 0, 200, 180, 120};
    short py[]={ 20, 40, 120, 120};
    int n;

    _setvideomode(_ERESCOLOR);
    _clearscreen(_GCLEARSCREEN);

    new_mask[0]=0x88;
    new_mask[1]=0x44;
    new_mask[2]=0x22;
    new_mask[3]=0x11;
    new_mask[4]=0x88;
    new_mask[5]=0x44;
    new_mask[6]=0x22;
    new_mask[7]=0x11;
    _setfillmask(new_mask);
    _polygon(_GFILLINTERIOR, 4, px, py);
    getch();

    _getfillmask(old_mask);
    n=0;
    while(n < 8)
    {
        old_mask[n]<<=1;
        ++n;
    }
    _setfillmask(old_mask);
    _polygon(_GFILLINTERIOR, 4, px, py);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**int getftime(int handle, struct ftime \*ftptr);**

---

**U****Header file**     io.h**See also**       open, setftime**Portability**    DOS**Multi-thread**   None.

Returns the time and date information for the file associated with the specified handle. The value is returned in the structure to which ftptr points. This ftime structure is defined as follows:

```
struct ftime {
    /* Two second interval */
    unsigned ft_tsec : 5;
    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 */
};
```

**Return value**

The function returns 0 if successful; there is no error return.

**Example**

```
#include <io.h>
#include <fcntl.h>
#include <stat.h>
main()
{
    int fhandle;
    struct ftime *ftptr;
    ftptr =
        (struct ftime *)
        malloc ( sizeof ( struct ftime));
    fhandle = open (
        "FTFILE.$$$", 0_CREAT,
        S_IWRITE|S_IREAD);
    getftime ( fhandle, ftptr);
    printf ( "File time information:\n");
    printf ( "%u:%u o'clock on %u/%u/198%u\n",
        ftptr->ft_hour, ftptr->ft_min,
        ftptr->ft_month, ftptr->ft_day,
        ftptr->ft_year);
}
```

```
void far _getimage(  
    short x1,  
    short y1,  
    short x2,  
    short y2,  
    char far *buffer);
```

---

**M**

**Header file**           graph.h

**See also**             \_getimage, \_imagesize, \_putimage

**Portability**         DOS.

**Multi-thread**        Function is not re-entrant.

The function \_getimage from the graphics module stores a screen image in a specified buffer.

x1,y1                  coordinates of the upper left corner of the rectangle bounding the image.

x2,y2                  coordinates of the lower right corner of the rectangle bounding the image.

buffer                 points to the location in which the image will be stored.

Note:   **The buffer location must be large enough to store the image.  
          To determine the amount of storage required, use \_imagesize.**

**Return value**

None.

**Example**

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

void save_image(char far *buffer, int size);

main()
{
    long size;
    short x1=10, y1=10, x2=40, y2=100;
    char *buffer;

    _setvideomode(_ERESNOCOLOR);
    _ellipse(_GFILLINTERIOR, x1, y1, x2, y2);
    size=_imagesize(x1, y1, x2, y2);
    buffer=malloc((size_t) size);
    _getimage(x1, y1, x2, y2, buffer);
    save_image(buffer, (int) size);
    free(buffer);
    _setvideomode(_DEFAULTMODE);
    return(0);
}

void save_image(char far *image, int size)
{
    FILE *f;

    f=fopen("IMAGE.$$$", "w");
    fwrite(image, sizeof(char), (int) size, f);
    fclose(f);
    return;
}
```

## unsigned char GetInProgramFlag(void)

---

<b>Header file</b>	dos.h
<b>Portability</b>	DOS only
<b>See also</b>	SetInProgramFlag

The InProgramFlag indicates whether a process is executing within its own program code or within the operating system. If a program attempts to re-enter DOS (i.e. when the InProgramFlag is clear) a run-time error will occur.

However it is legal to re-enter DOS from a critical error handler. SetInProgramFlag allows library functions that call DOS to operate. GetInProgramFlag interrogates the InProgramFlag

### Return value

Returns the state of the InProgramFlag.

- 1 in program code
- 0 in DOS code.

---

**unsigned short far \_getlinestyle(void);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getlinestyle
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getlinestyle` from the graphics module returns the mask to be used for drawing lines. Each bit in the 16-bit mask represents a pixel in the line being drawn.

- If a bit is 1, the pixel to which the figure applies is drawn in the current color; if the bit is 0, the pixel is left unchanged.
- Such a template is repeated for the entire line.
- The default mask is all 1's (i.e., 0xFFFF in this case), which represents a solid line.

**Return value**

The function returns the current line style.

**Example**

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

main()
{
    unsigned short mask=0xCCCC;
    unsigned short old_mask;

    _setvideomode(_ERESCOLOR);
    _setcolor(4);
    _setlinestyle(mask);
    _rectangle(_GBORDER, 10, 10, 500, 300);
    getch();
    old_mask=_getlinestyle();
    _setlinestyle(~old_mask);
    _rectangle(_GBORDER, 10, 10, 500, 300);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**struct xycoord far \_getlogcoord(short x, short y);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getphyscoord, _moveto
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getlogcoord` from the graphics module translates the specified physical coordinates into logical coordinates. The logical coordinates are returned in an `xycoord` structure, defined in `graph.h`:

```
struct xycoord {
    short xcoord;
    short ycoord;
};
```

**x** specifies the value for the horizontal physical coordinate.

**y** specifies the value for the vertical physical coordinate.

**Return value**

The function returns the logical coordinates in the `xycoord` structure.

**Example**

```
#include <graph.h>
#include <conio.h>
main()
{
    short X1=25, Y1=10, X2=500, Y2=200;
    struct xycoord tl, br;
    _setvideomode(_ERESNOCOLOR);
    tl=_getlogcoord(X1, Y1);
    br=_getlogcoord(X2, Y2);
    _ellipse(_GFILLINTERIOR, tl.xcoord,
        tl.ycoord, br.xcoord, br.ycoord);
    getch();
    _setlogorg(100, 100);
    _clearscreen(_GCLEARSCREEN);
    tl=_getlogcoord(X1, Y1);
    br=_getlogcoord(X2, Y2);
    _ellipse(_GFILLINTERIOR, tl.xcoord,
        tl.ycoord, br.xcoord, br.ycoord);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**struct xycoord far \_getphyscoord(short x, short y);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getlogcoord
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_getphyscoord` from the graphics module translates the specified logical coordinates into physical coordinates. The physical coordinates are returned in an `xycoord` structure, which is defined in `graph.h`:

```
struct xycoord
{
    short xcoord;
    short ycoord;
};
```

`x` specifies the value for the horizontal logical coordinate.

`y` specifies the value for the vertical logical coordinate.

**Return value**

The function returns the physical coordinates. There is no error return.

**Example**

```
#include <graph.h>
#include <conio.h>
main()
{
    short X1=25, Y1=10, X2=500, Y2=200;
    struct xycoord tl, br;
    _setvideomode(_ERESNOCOLOR);
    _setlogorg(100, 100);
    _ellipse(_GFILLINTERIOR, X1, Y1, X2, Y2);
    getch();
    _clearscreen(_GCLEARSCREEN);
    tl=_getphyscoord(X1, Y1);
    br=_getphyscoord(X2, Y2);
    _setlogorg(0, 0);
    _ellipse(_GFILLINTERIOR, tl.xcoord,
            tl.ycoord, br.xcoord, br.ycoord);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **unsigned getpid(void);**

---

**Header file**            process.h

**Portability**            Some DOS/OS2. Under OS2 A true process ID is returned.

The function `getpid` returns the process ID, which uniquely identifies the calling process.

### **Return value**

The function returns the process ID. Under DOS, this is actually the segment of the Program Segment Prefix.

### **Example**

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

main()
{
    char path[80];
    int proc_id;
    FILE *f;

    proc_id=getpid(); /* print process ID */
    sprintf(path, "TEMP%.4X.$$$", proc_id);
    f=fopen(path, "w");
    fprintf(f, "PROCESS ID FILE\n");
    return(0);
}
```

---

**short far \_getpixel(short x, short y);**

---

**M****Header file** graph.h**See also** \_remapallpalette, \_remappalette, \_selectpalette, \_setpixel, \_setvideomode**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_getpixel` from the graphics module determines the value of the pixel at the logical point corresponding to the horizontal and vertical coordinates specified by `x` and `y`, respectively. The actual color will depend on the palette selection.

**Return value**

The function returns the pixel value if successful; otherwise (e.g., if the point lies outside the clipping region), the function returns -1.

**Example**

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

main()
{
    int x, y, c;
    struct videoconfig v;

    _setvideomode(_ERESCOLOR);
    _clearscreen(_GCLEARSCREEN);
    _getvideoconfig(&v);

    while(!kbhit()) {
        x=random(v.numxpixels-1);
        y=random(v.numypixels-1);
        c=random(v.numcolors-1);
        _setcolor(c);
        if(_getpixel(x, y) == 0)        _setpixel(x, y);
    }
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **unsigned getpsp(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	getenv
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getpsp` gets the program segment prefix location. The function uses service 62H.

This function requires DOS 3.0 or later; to accomplish the same thing in earlier versions of DOS (2.X), use the value of the global variable `_psp`.

### **Return value**

The function returns the segment address of the program segment prefix.

### **Example**

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

main()
{
    unsigned proc_psp;

    proc_psp=getpsp();
    printf("PSP is %X\n", proc_psp);
    return(0);
}
```

**char \*gets(char \*buf);****A**

<b>Header file</b>	stdio.h
<b>See also</b>	fputs, fgets, puts.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is controlled by semaphore in multi-thread programs.

The function `gets` reads a string from the stream `stdin` and stores it at `buf`. Characters are read from the current position until either of the following occurs:

- a newline (`\n`) character is encountered
- an error or end of file occurs

The string is terminated with a null character, `\0`. If a newline is read, it is converted to a null character.

**Return value**

If `gets` is successful, the function will return `buf`. If an error or end of file occurred, `gets` will return `NULL`.

**Example**

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

main()
{
    char line[80];

    printf("Enter String - ");
    gets(line); /* read string from stdin */
    strupr(line); /* convert to upper case */
                /* output new string */
    printf("Upper case version - %s\n", line);
    return(0);
}
```

---

**int gettext(int left, int top, int right, int bottom, void \*buffer);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	puttext
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function gettext from the Turbo C window module copies a block of text (delimited by the specified coordinates) from the screen to a buffer.

left	specifies the leftmost column to be copied.
top	specifies the topmost line to be copied.
right	specifies the rightmost column to be copied.
bottom	specifies the bottom line to be copied.
buffer	points to the location to which the material will be copied.

Both the character and its attribute are copied, so the size of the object to which buffer points must be large enough to accommodate  $(\text{right} - \text{left} + 1) * (\text{bottom} - \text{top} + 1)$  words.

Note: All coordinates are absolute screen coordinates.

**Return value**

The function returns 1 if successful. If any of the coordinates are invalid, the function returns 0.

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define COLS 40
#define LINES 8

#define BUF_SIZE (COLS+1)*(LINES+1)*sizeof(int)

main()
{
    char *win_buffer;
    FILE *f;

    win_buffer=malloc(BUF_SIZE);
    clrscr();

    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    cprintf("This window will be saved to disk");
    gettext(20, 4, 20+COLS, 4+LINES, win_buffer);
    f=fopen("TEMP.$$$", "w");
    if(f == NULL)
        abort();
    fwrite(win_buffer, BUF_SIZE, 1, f);
    fclose(f);
    free(win_buffer);
    return(0);
}
```

---

**short far \_gettextcolor(void);**

---

**M****Header file** graph.h**See also** \_selectpalette, \_settextcolor**Portability** DOS.

Multi-threaded structured Function is not re-entrant.

The function `_gettextcolor` from the graphics module returns the pixel value for the current text color. The default value is the highest legal value of the current palette.

**Return value**

The function returns the pixel value. There is no error return.

**Example**

```
#include <conio.h>
#include <graph.h>
main()
{
    int n;

    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 50);
    while(!kbhit()) {
        n=_gettextcolor();
        if(++n > 15)
            n=1;
        _settextcolor(n);
        _settextposition(0, 0);
        _outtext("changing colors");
    }
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**void gettextinfo(struct text\_info \*r);****T**

<b>Header file</b>	conio.h
<b>See also</b>	gettext
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function gettextinfo from the Turbo C window module copies window data to the structure to which r points. The text\_info structure is defined in conio.h, and contains the following fields:

winleft	left window co-ordinate.
wintop	top window co-ordinate.
winright	right window co-ordinate
winbottom	bottom window co-ordinate.
attribute	current active screen attribute.
normattr	original screen attribute
currmode	current screen mode.
screenheight	screen height
screenwidth	screen width
curx	current cursor x coordinate.
cury	current cursor y co-ordinate.

**Return value**

None

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#define COLS 40
#define LINES 8
main()
{
    struct text_info r;
    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    gettextinfo(&r);
    printf(
        "Window definition: %d, %d, %d, %d",
        r.winleft, r.wintop,
        r.winright, r.winbottom);
    getch();
    return(0);
}
```



---

**struct rccoord far \_gettextposition(void);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_settextposition
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_gettextposition` from the graphics module returns the current text position, relative to an origin of 1,1. This value is returned in an `rccoord` structure, as defined in `graph.h`:

```
struct rccoord
{
    short row;
    short col;
};
```

**Return value**

The function returns the `rccoord`. There is no error return.

**Example**

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

main()
{
    struct rccoord r;

    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 50);
    _displaycursor(_G_CURSORON);
    _settextposition(1, 1);
    _outtext("Press Key To Hide Cursor");
    getch();
    r=_gettextposition();
    _settextposition(r.row, 1);
    _displaycursor(_G_CURSOROFF);
    _outtext("Press Key To Exit  ");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **unsigned \_getTID(void)**

---

**Header file**            process.h

**Portability**            DOS and OS2.

Returns the thread number of the currently executing thread. The main thread always has the number 1.

## **void gettimeofday(struct time \*tptr);**

---

**Header file**            time.h and dos.h

**See also**                getdate, setdate, settime, stime, time

**Portability**            DOS/OS2

**Multi-thread**          DOS is not re-entrant

The function gettimeofday returns the current system time in the structure to which tptr points. The time structure is defined as follows:

```
struct time
{
    unsigned char ti_hour;
    unsigned char ti_min;
    unsigned char ti_sec;
    unsigned char ti_hund;
};
```

### **Return value**

None.

### **Example**

```
#include <time.h>
main()
{
    struct time *timptr;

    timptr =
        (struct time *)
        malloc ( sizeof ( struct time));
    gettimeofday ( timptr);
    printf(
        "The time is: %u:%u:%d\n",
        timptr->ti_hour,
        timptr->ti_min,
        timptr->ti_sec);
}
```

## **`void (interrupt far *getvect(int interruptno))();`**

---

<b>Header file</b>	dos.h
<b>See also</b>	setvect
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getvect` reads the address stored at a specified interrupt vector, and returns that value as a far pointer to an interrupt function.

`interruptno` specifies the interrupt vector whose value is being sought. This parameter can be any value between 0 and 255.

### **Return value**

The function returns the 4-byte location stored in the specified interrupt vector.

### **Example**

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

void interrupt far handler(void)
{
    return;
}

void rest_of_program(void)
{
    return;
}

main()
{
    void (interrupt far *old_int)(void);

    old_int=getvect(4); /* save old vector */
    setvect(4, handler);/* install new handler */
    rest_of_program();
    setvect(4, old_int);/* restore old handler */
    return(0);
}
```

## **int getverify(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	setverify
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `getverify` returns the state of the system verify flag.

### **Return value**

The function returns the current state of the verify flag, which is 0 if verify is off, and 1 if it is on.

### **Example**

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

char *msg[]={“OFF”, “ON”};

main()
{
    int state;

    state=getverify();
    printf(“Verify is %s\n”, msg[state]);
    return(0);
}
```

```
struct videoconfig far * far _getvideoconfig(
    struct videoconfig far *v);
```

---

**M**

**Header file** graph.h

**See also** \_setvideomode

**Portability** DOS.

**Multi-thread** Function is not re-entrant.

The function `_getvideoconfig` from the graphics module copies the current graphics configuration to a structure.

`v` points to a structure that will receive the settings for the current graphics configuration. The `videoconfig` structure is defined in `graph.h`.

```
struct videoconfig {
    /* number of pixels on X axis */
    short numxpixels;
    /* number of pixels on Y axis */
    short numypixels;
    /* number of text cols available */
    short numtextcols;
    /* number of text rows available */
    short numtextrows;
    /* number of actual colors */
    short numcolors;
    /* number of bits per pixel */
    short bitsperpixel;
    /* number of available */
    /* video pages */
    short numvideopages;
    /*current video mode */
    short mode;
    /* active display adapter */
    short adapter;
    /* active display monitor */
    short monitor;
    /* adapter video memory in Kb */
    short memory;
};
```

### Return value

The function returns the pointer to the configuration information. There is no error return.

### Example

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

main()

{
    int page_used=0;
    struct videoconfig v;

    _setvideomode(_MRES16COLOR);
    _getvideoconfig(&v);
    if(v.numvideopages > 1)
        page_used=1;
    _setactivepage(page_used);
        /* draw rectangle on page 1 */
    _rectangle(_GBORDER, 10, 10, 90, 50);
    _setvisualpage(page_used); /* view page 1 */
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **int getw(FILE \*st);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	putw.
<b>Portability</b>	UNIX/DOS/OS2. The size of an integer and the ordering of bytes varies between systems.
<b>Multi-thread</b>	Access to stream is controlled by semaphore in multi-thread programs.

The function `getw` reads a binary value 16-bit integer from the stream `st`, and increments the file pointer by two bytes.

**Note:**    **No particular alignment is assumed.**

### **Return value**

The function returns the value read. A value of -1 may indicate an error; however, since this is a legal integer value, `feof` or `ferror` should be called to verify an error or end of file condition.

### **Example**

```
#include <stdio.h>

main()
{
    FILE *f;
    int num;

    f=fopen("DATAFILE.DAT", "rb");
    num=getw(f); /* read integer */
                /* check for error or EOF */
    if((feof(f)) || (ferror(f)))
        printf("Error reading file\n");
    else
        /* print number */
        printf("Number read was %d\n", num);
    fclose(f);
    return(0);
}
```

**struct tm \*gmtime(const time\_t \*tt);****A**

<b>Header file</b>	time.h
<b>See also</b>	asctime, ctime, time, localtime.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	A call to gmtime only destroys the result of a previous call from the same thread.

Converts the time value stored at tt to a structure, tm. A time\_t is obtained from the function time, and represents the number of seconds elapsed since 00:00:00 January 1st 1970, Greenwich Mean Time. tm represents Greenwich Mean Time, not local time. It contains:

tm_sec	Seconds.
tm_min	Minutes.
tm_hour	Hours (0-24).
tm_mday	Day of Month (0-31).
tm_mon	Month (0-11).
tm_year	Year (Current year minus 1900).
tm_wday	Day of week (Sunday = 0).
tm_yday	Day of year (0 Æ 365).
tm_isdst	Always 0.

**Note:** **gmtime and localtime use a static buffer to hold the result. Each call to one of these functions destroys the result of the last call.**

**Return value**

The function returns a pointer to the structure result. There is no error return value.

**Example**

```
#include <stdio.h>
#include <time.h>
main() {
    time_t tt;
    struct tm *gmt;
    time(&tt); /* get time in seconds */
    gmt=gmtime(&tt); /* convert to structure */
    printf(
        "Hour is %d, Greenwich mean time\n",
        gmt->tm_hour);
    return(0);
}
```



**void gotoxy(relcoord X, relcoord Y);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2.
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function gotoxy from the TopSpeed window module sets the current X,Y position for the cursor in the window currently being used. If X or Y are outside the window frame, they will be clipped.

X	specifies the horizontal coordinate for the location.
Y	specifies the vertical coordinate for the location.

**Return value**

None.

**Example**

```
#define _JPI_WIN_  
#include <conio.h>  
#include <string.h>  
  
windef WD1={10, 2, 60, 22, White, Blue,  
    FALSE, FALSE, FALSE, TRUE,  
    SINGLEFRAME, Red, LightGray};  
wintype W1;  
  
main()  
{  
    int x=1, y=1;  
  
    W1 = windowopen(&WD1) ;  
        /* open window */  
    setttitle(W1, "Window 1", CenterUpperTitle);  
    while(y < 20) {  
        gotoxy(x, y);  
        /* set cursor position */  
        cputs("Hello World");  
        ++y; ++x;  
    }  
    getch();  
    windowclose(W1);  
    return(0);  
}
```

---

**void gotoxy(int x, int y);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	wherex, wherey, window
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `gotoxy` from the Turbo C window module sets the cursor coordinates relative to the current window. If the coordinates are outside the current window, the call is ignored.

**Return value**

None.

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define COLS 40
#define LINES 8

main()
{
    int x=1, y=1;

    clrscr();
    window(20, 4, 20+COLS, 4+LINES);
    /* define new window */
    while(y < LINES) {
        gotoxy(x, y);
        cprintf("printed at %d, %d", x, y);
        ++x;
        ++y;
    }
    getch();
    return(0);
}
```

## **void huge \*halloc(unsigned long size);**

---

**Header file**            alloc.h

**See also**             malloc

**Portability**         Some MSDOS/OS2.

**Multi-thread**       Far heap only protected by semaphore

**Note**                If size is 0, halloc returns NULL.

The function halloc returns a pointer to a huge object of size bytes in a far data segment.

### **Return value**

The function returns a void huge pointer to the allocated space. This space is guaranteed to be suitably aligned for any type of object. If no space is available, a NULL pointer is returned.

### **Related functions**

hfree, \_nmalloc, \_fmalloc

### **Example**

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

void error(char *msg, int error_code);

char huge *allocate(long size)
{
    char huge *hptr;
    hptr = halloc(size);
    /* allocate huge memory block from
       far heap */
    if (hptr == NULL)
        error("Huge memory allocation", errno);
    return(hptr);
}
```

---

```
void _harderr(void (far *handler)(unsigned,unsigned,unsigned far *));
```

---

<b>Header file</b>	dos.h
<b>See also</b>	_chain_intr, dos_getvect, dos_setvect
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_harderr` enables a user-defined function to serve as the interrupt handler for interrupt 24H. This interrupt is called when an I/O request results in a hardware error.

**Note**    **If the handler wishes to re-enter DOS or call a library function that does so, the global variable `InProgramFlag` must be set to 1 with a call to `SetInProgramFlag` with argument 1.**

handler	points to the user-defined function that will serve as the interrupt handler. The <code>_harderr</code> function actually installs a handler that calls the user-defined function, rather than installing the user-defined function directly. The installed handler calls the user-defined function with three parameters: <code>dverror</code> and <code>errorcode</code> , which are of type <code>unsigned</code> , and <code>dvheader</code> , which is a far pointer to <code>unsigned</code> .
dverror	is a device error code. It contains the AX register value that DOS would pass to the INT 24H handler. The high-order bit of this argument will be 0 if the error occurred on a disk device and 1 otherwise. If the high-order bit of <code>dverror</code> is 0, individual bits of the argument's value will provide the following information:

<b>Bit(s)</b>	<b>Meaning</b>
---------------	----------------

15	0 specifies disk error.
14	Not used.
13	If 0, do not allow "Ignore" response.
12	If 0, do not allow "Retry" response.
11	If 0, do not allow "Fail" ("Abort") response.
9Æ10	specify the location of the error:
00	DOS

- 01 File allocation table (FAT)
- 10 Directory
- 11 Data area.
- 8 If 0, read error; if 1, write error.
- 0-7 specify drive number, with drive A = 0, drive B = 1, etc.

If the high-order bit of `dverror` is 1, information in the device header block (to which `dvheader` points) will indicate what kind of device produced the error. Specifically, the attribute word at offset 04 in the device header block will provide this information. If the high-order bit of these two bytes is 0, a bad image of the FAT in memory caused the error; if the bit is 1, a character device caused the error. In that case, the four low-order bits will specify the type of device.

Bit	Meaning
0	Standard input
1	Standard output
2	Null device
3	Clock device

`errorcode` represents the DI register value that DOS passes to the interrupt handler. The low-order byte of this parameter can take the following values:

Value	Meaning
0	Attempt made to write to a write-protected disk.
1	Unit is unknown
2	Drive is not ready
3	Command is unknown
4	CRC error in data
5	Bad drive-request
6	Seek error
7	Media type is unknown
8	Sector was not found
9	No paper in printer
A	Write fault
B	Read fault

## C General failure

`dvheader` is a far pointer to information about the device on which the error occurred. This information may not be changed by the user-defined function.

The user-defined function can use only certain system calls: 01H through 0CH and 59H. The latter call (*Get Extended Error Information*) can be used to get more information about the error.

### Return value

The interrupt handler need not return. If it does, any of the following methods can be used:

- Using the `return` statement.
- Using the `_hardresume` function.
- Using the `_hardretn` function.

The first two return methods return to DOS; the third returns to the application at a point just beyond the I/O request that caused error. See `_hardresume` and `_hardretn` for more information.

## **void \_hardresume(int code);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_harderr, _hardretn
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_hardresume` is used to return from an I/O hardware error, when a user-defined interrupt handler is being used to process an INT 24H.

code specifies the result of the user-defined error handling. This argument must have one of the following values:

<code>_HARDERR_IGNORE</code>	ignore the error
<code>_HARDERR_RETRY</code>	try the operation again
<code>_HARDERR_ABORT</code>	issue an INT 23H ( <i>Control C Handler</i> ) and abort the program.
<code>_HARDERR_FAIL</code>	fail the system call (DOS 3.0 and higher)

The `_hardresume` function should be called only from within the user-defined error handler function. The `_hardresume` function returns to DOS, with an argument (code) specified by the user-defined error handler. See `_harderr` for more information about user-defined interrupt handlers.

### **Return value**

None.

## **void \_hardretn(int code);**

---

<b>Header file</b>	dos.h
<b>See also</b>	_harderr, _hardresume
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_hardretn` is used to return from an I/O hardware error directly to the application program (i.e., bypassing DOS), when a user-defined interrupt handler is being used to process an INT 24H. The application resumes at a point just after the function request that caused the error.

`code` specifies the error code from the user-defined error handling. This should be a DOS (rather than a UNIX) error code. See the DOS Programmer's *Reference* for information about such error codes.

### **Return value**

There is no explicit return value; however, function `_hardretn` may set the AX or AL registers before returning to the application. The change depends on the failing function request.

- If this function request is an INT 24H function with a code greater than or equal to 38H, the carry flag will be set and the AX register will be changed to the value of the code parameter.
- If the code is less than 38H *and* if the function can return an error, the AL register is set to FFH.
- If the code is less than 38H and no error condition can be returned, no error code is returned to the application. In this case, the code parameter is not used.



```
void *hbsearch(  
    const void huge *key,  
    const void huge *base,  
    size_t num,  
    size_t width,  
    int (*compare)(const void *e1, const void *e2))
```

---

huge variant of bsearch

```
int _heapchk(void);
```

---

## **int \_heapset(int fill);**

---

<b>Header file</b>	alloc.h
<b>Variants</b>	_nheapchk/_nheapset, _fheapchk/_fheapset
<b>See also</b>	_heapwalk
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	Far heap only protected by semaphore

The function `_heapchk` checks the integrity of the heap by checking:

- the start of the heap,
- synchronization between the allocated blocks and the free block list, and
- the top of the heap

The function `_heapset` performs these checks and also fills each free block with the value specified in `fill`.

### **Return value**

Both functions return one of the following values, which are defined in `alloc.h`:

<code>_HEAPOK</code>	Heaps appears to be intact.
<code>_HEAPEMPTY</code>	Heap has not been initialized.
<code>_HEAPBADBEGIN</code>	The beginning of the heap or the first header was corrupted.
<code>_HEAPBADNODE</code>	A bad node was found, indicating that heap was damaged.
<code>_HEAPOVERFLOW</code>	The heap exceeds allocated memory.
<code>_HEAPEND</code>	The end of the heap or the last header was corrupted.

### **Example**

```
main()
{
    char *ptr;
    char near *near_ptr;
    char far *far_ptr;

    ptr=malloc(200);
    free(ptr);
    if(_heapchk() != _HEAPOK)
        printf("Error in default heap\n");
    near_ptr=_nmalloc(200);
    _nfree(near_ptr);
    if(_nheapchk() != _HEAPOK)
```

```
        printf("Error in near heap\n");
    far_ptr=_fmalloc(200);
    _ffree(far_ptr);
    if(_fheapchk() != _HEAPOK)
        printf("Error in far heap\n");
    return(0);
}
```

## **int \_heapwalk(struct \_heapinfo \*entry);**

---

<b>Header file</b>	malloc.h
<b>Variants</b>	_nheapwalk, _fheapwalk
<b>See also</b>	_heapchk, _heapset
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	Far heap only protected by semaphore

The function `_heapwalk` can be used to step through the heap, cell by cell. At each step through the heap, the function returns information about the next heap entry in the location to which entry points. The `_heapinfo` structure is defined as follows:

```
struct _heapinfo
{
    int * _pentry;
    size_t _size;
    int _useflag;
};
```

Structure member `_pentry` must be initialized to `NULL` the first time this function is called.

### **Return value**

The function returns one of the following values, defined in `malloc.h`:

<b>Value</b>	<b>Meaning</b>
<code>_HEAPOK</code>	The heap is all right up to the current point.
<code>_HEAPEMPTY</code>	The heap has not been initialized.
<code>_HEAPBADPTR</code>	The <code>_pentry</code> member of the structure to which entry points does not contain a valid pointer into the heap.
<code>_HEAPBADBEGIN</code>	The header information for the heap was invalid or was not found.
<code>_HEAPBADNODE</code>	A bad node was found.
<code>_HEAPEND</code>	The end of the heap was reached.

### **Example**

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

main() {
    struct _heapinfo h;
    int ret;

    h._pentry=NULL;
    while((ret=_heapwalk(&h)) == _HEAPOK)
    {
        printf(
            "Size %8u, address %p",
            h._size, h._pentry);
        if(h._useflag)
            printf(" Allocated\n");
        else
            printf(" Free\n");
    }
    if(ret == _HEAPEND)
        printf("Heap OK\n");
    else
        printf("Heap error %d\n", ret);
    return(0);
}
```

---

**void hfree(void huge \*buffer);**

---

Huge pointer variant of free

---

**size\_t hread(  
    const void huge \*buffer,  
    size\_t size,  
    size\_t n,  
    FILE \*stream);**

---

Huge variant of fread

---

**size\_t hfwrite(  
    const void huge \*buffer,  
    size\_t size,  
    size\_t n,  
    FILE \*stream)**

---

Huge variant of fwrite

## **void hide(wintype win);**

## **W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single call function calls.

The function `hide` from the TopSpeed window module removes the window `win` from the screen and the window stack. The contents of the window are saved in a buffer, so they can be redisplayed later, if necessary.

Any window obscured by `win` will be uncovered as a result of the call to `hide`. Any material written to a hidden window, `win`, is recorded, and will appear when the window is made visible.

### **Return value**

None.

### **Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 60, 8, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

windef WD2={40, 6, 75, 18, Yellow, Red, FALSE, FALSE, FALSE, TRUE,
DOUBLEFRAME, LightGray, Blue;
wintype W2;
main()

{
    clrscr();
    W1 = windowopen(&WD1) ;
    setttitle(W1, "Window 1", CenterUpperTitle);
    W2 = windowopen(&WD2) ;
    setttitle(W2, "Window 2", CenterUpperTitle);
    cprintf("Hello World sent to window 2\n");
    getch();
    hide(W2);
    getch();
    putontop(W2);
    getch();
    windowclose(W1);
    windowclose(W2);
    return(0);
}
```

**void highvideo(void);****T**

<b>Header file</b>	conio.h
<b>See also</b>	lowvideo, normvideo
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `highvideo` from the Turbo C window module sets the active screen attribute to high intensity. The screen attribute is used by the console I/O functions.

**Return value**

None.

**Example**

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

main()
{
    highvideo();
    cprintf("This is displayed with high");
    cprintf(" intensity attributes");
    return(0);
}

char huge * hlfind(
const char * key,
const char huge * base,
unsigned * num,
unsigned width,
int ( * compare)(const void huge * e1, const void huge * e2))
Huge variant of lfind
char huge * hlsearch(
const char * key,
char huge * base,
unsigned * num,
unsigned width,
int ( * compare)(const void huge * e1, const void huge * e2))
Huge variant of lsearch
void huge * hmemccpy(
void huge *s1, const void huge *s2,
char c, size_t num);
Huge pointer variant of memcpy
void huge *hmemchr(const void huge *s1, char c, size_t num);
Huge pointer variant of memchr
int hmemcmp(const void huge *s1, const void huge *s2, size_t num);
Huge pointer variant of memcmp
void huge *hmemcpy(void huge *s1, const void huge *s2, size_t n);
Huge pointer variant of memcpy
int hmemicmp(
const void huge *s1, const void huge *s2,
size_t num);
Huge pointer variant of memicmp
```



```
void huge * hmemset(void huge *s1, char c, size_t num);  
Huge pointer variant of memset  
void huge * hrealloc(void huge *buffer, unsigned long size);  
Huge pointer variant of realloc  
void hqsort(  
void huge * base,  
size_t num,  
size_t width,  
int ( * compare)(const void huge * e1, const void huge * e2))  
Huge variant of qsort  
double hypot(double x, double y);
```

## **long double hypotl(double x, double y);**

---

<b>Header file</b>	math.h
<b>See also</b>	cabs, matherr.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function `hypot` calculates the length of the hypotenuse of its double precision floating point arguments `x` and `y`. The function `hypotl` is the same as `hypot` but takes long double arguments and returns a long double value.

### **Return value**

The function returns the square root of  $x^2 + y^2$ . If an overflow results, the function returns `HUGE_VAL`, and `errno` is set to `ERANGE`.

Error handling can be altered by modifying the `matherr` function.

### **Example**

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

main()
{
    double hyp;

    hyp=hypot(3.0, 4.0);
    printf("Hypotenuse is %g\n", hyp);
    return(0);
}
```

**long far \_imagesize(short x1, short y1, short x2, short y2);****M****Header file** graph.h**See also** \_getimage, \_getvideoconfig, \_putimage**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_imagesize` from the graphics module returns the amount of storage (in bytes) needed to store the image defined by the specified rectangular boundary.

`x1,y1` coordinates of the upper left corner of the bounding rectangle.

`x2,y2` coordinates of the lower right corner of the bounding rectangle.

The size is determined using the following formulas:

$$\begin{aligned} x_{wid} &= |x_1 - x_2| + 1 \\ y_{wid} &= |y_1 - y_2| + 1 \\ size &= 4 + ((wid * bpp) + 7) / 8 * y_{wid} \end{aligned}$$

where  $|x|$  represents the absolute value of  $x$ , and *bpp* (bits per pixel) is the `bitsperpixel` member returned by a call to `_getvideoconfig`. `size` is actually the value derived by casting the result to a long.

**Return value**

The function returns the storage size (in bytes). There is no error return.

**Example**

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

main()
{
    long size;
    short x1=10, y1=10, x2=40, y2=100;
    char *buffer;

    _setvideomode(_ERESNOCOLOR);
    _ellipse(_GFILLINTERIOR, x1, y1, x2, y2);
    size=_imagesize(x1, y1, x2, y2);
    buffer=malloc((size_t) size);
    free(buffer);
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**void info(wintype win, windef \*WD);****T**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single call function calls.

Given the window W, info returns the definition of the window in parameter WD. The function info from the TopSpeed window module gets information about the specified window.

win	specifies the window whose parameters values are to be checked.
WD	points to the data structure that contains information about the specified window. The windef structure is defined in the introduction on page

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 50, 8, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main() {
    windef new_def;
    wintype new_win;

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    /* get window info */
    info(W1, &new_def);
    new_def.X1+=2;
    new_def.X2+=2;
    new_def.Y1+=2;
    new_def.Y2+=2;
    /* open new window */
    new_win = windowopen(&new_def) ;
    setttitle(
        new_win, "New Window",
        CenterUpperTitle);
    getch();
    return(0);
}
```

## **void Init(SIGNAL \*s);**

---

<b>Header file</b>	process.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

Initializes the signal s. That is, the function sets the counter for the SIGNAL to which s points to 0 and its queue to empty. A SIGNAL is defined as a pointer to a record containing counter and queue information.

### **Return value**

None.

### **Example**

See example program in , page .

## **unsigned char inp(int port);**

---

## **unsigned char inportb(int port);**

---

**Header file** conio.h and dos.h

**See also** outp, outportb.

**Portability** 8086 Family

**Multi-thread** None.

The functions `inp` and `inportb` read a byte from the specified port. The value for port can be any integer value from -32768 to 32767.

### **Return value**

The functions return the value read. There is no error return value.

### **Example**

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

unsigned read_status_port()
{
    unsigned byte;
    /* read byte from port 0x3BA */
    byte=inportb(0x3BA);
    return(byte);
}
```

## **unsigned inpw(int port);**

---

## **unsigned inportw(int port);**

---

**Header file**            conio.h and dos.h

**See also**             outpw, outportw

**Portability**         80x86 Family

**Multi-thread**        None.

The functions inpw and inportw read a *word* from the specified port. The value for port can be any integer value from -32768 to 32767.

### **Return value**

The functions return the value read. There is no error return value.

### **Example**

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

unsigned read_status_port()
{
    unsigned word;
    /* read word from port 0x3BA */
    word=inpw(0x3BA);
    return(word);
}
```

**void inline(void);****W**

<b>Header file</b>	window.h
<b>See also</b>	delline
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single call function calls.

The function `inline` from the TopSpeed window module inserts a blank line at the current cursor position. The screen below this line scrolls downward.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 16, White, Blue,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;

main()
{
    int n=1;

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    while(n < 14) {
        gotoxy(1, n);
        cputs("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
        ++n;
    }
    gotoxy(2, 4);
    cprintf("Press key to insert line ");
    getch();
    inline();
    getch();
    return(0);
}
```



---

**void inline(void);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	delline, window
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `inline` from the Turbo C window module inserts a blank line into the text window at the current cursor position. All lines below the new line are scrolled down one line and the bottom line scrolls out of the window.

**Return value**

None.

**Example**

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

#define COLS 40
#define LINES 8

main()
{
    int n=0;
    char block[COLS+2];

    while(n <= COLS) {
        block[n]='X';
        ++n;
    }
    block[n]='\0';
    clrscr();
    window(20, 4, 20+COLS, 4+LINES);
    /* define new window */
    n=0;
    while(n < LINES) {
        cprintf(block); /* output to new window */
        ++n;
    }
    gotoxy(1, 4);
    cprintf("Press key to insert a line");
    getch();
    inline();
    return(0);
}
```

```
int int86(
    int intnum,
    const union REGS *inregs,
    union REGS *outregs);
```

---

**Header file**            dos.h

**See also**              bdos, intdos, intdosx, int86x

**Portability**          80x86 family.

**Multi-thread**        DOS BIOS not re-entrant

The function `int86` executes a specified interrupt.

`intnum`                  specifies the desired interrupt.

`inregs`                 points to the register values passed to the interrupt.

`outregs`                points to the register values returned from the interrupt.  
The union `REGS` has the following members:

```
struct WORDREGS {
    unsigned int
        ax, bx, cx, dx, si, di, cflag, flags;
struct BYTEREGS {
    unsigned char
        al, ah, bl, bh, cl, ch, dl, dh;
};
union REGS
{
    struct WORDREGS x;
    struct BYTEREGS h;
};
```

Before calling `int86`, the values required by the desired interrupt must be stored in the location to which `inregs` points.

**Note:**    **int86 may not be used for interrupts 25H and 26H. Use `absread` and `abswrite` instead**

**Return value**    The function returns its result as the contents of the `AX` register after the desired interrupt has returned. If the `cflag` member of `outregs` is nonzero (i.e., if an error has occurred), the error code is stored in `_doserrno`.

### Example

```
#include <dos.h>
void _ms_setdouble(int threshold) /* set mouse speed threshold */
{
    union REGS r;
    r.x.ax=0x13;
    r.x.dx=threshold;
    int86(0x33, &r, &r);
    return;
}
```

```
int int86x(
    int intnum,
    const union REGS *inregs,
    union REGS *outregs,
    struct SREGS *segregs);
```

---

**Header file**            dos.h

**See also**              bdos, FP\_SEG, intdos, intdosx, int86x, segread

**Portability**          80x86 family.

**Multi-thread**        DOS BIOS not re-entrant.

The function `int86x` executes a specified interrupt. This interrupt may work with large model data segments or far pointers.

`intnum`                specifies the desired interrupt.

`inregs`                points to the register values passed to the interrupt.

`outregs`               points to the register values returned from the interrupt.

`segregs` points to the structure containing the segment-register values required by the interrupt.

The union `REGS` has the following members:

```
union REGS {
    unsigned int /* WORDREGS member */
        ax, bx, cx, dx, si, di, cflag, flags;
    unsigned char /* BYTEREGS member */
        al, ah, bl, bh, cl, ch, dl, dh;
};
```

The struct `SREGS` has the following members:

```
struct SREGS {
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
};
```

Before calling `int86x`, the values required by the desired interrupt must be stored in the location to which `inregs` points. Similarly, the DS and ES registers of `segregs` must be initialized.

When the interrupt returns, the `int86x` function updates the ES and DS registers in `segregs` and restores DS.

**Note:**    `int86x` may not be used for interrupts 25H and 26H. Use `absread` and `abswrite` instead

**Return value**

The function returns its result as the contents of the AX register after the desired interrupt has returned. If the cflag member of `outregs` is nonzero (i.e., if an error has occurred), the error code is stored in `_doserrno`.

### Example

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

int unlink_file(char far *file) {
    union REGS r;
    struct SREGS s;
    int ret;

    r.h.ah=0x41; /* delete file */
    r.x.dx=FP_OFF(file);
    s.ds=FP_SEG(file);
    ret=int86x(0x21, &r, &r, &s);
    if(r.x.cflag) {
        fprintf(stderr, "Error %d\n", ret);
        return(-1);
    }
    return(0);
}
```

## **int intdos(const union REGS \*inregs, union REGS \*outregs);**

---

<b>Header file</b>	dos.h
<b>See also</b>	bdos, intdosx
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `intdos` calls the function dispatcher (INT 21H) to execute the system call specified in the `inregs` parameter.

<code>inregs</code>	points to the register values passed to the function dispatcher.
<code>outregs</code>	points to the register values returned from the function dispatcher.

The union `REGS` has the following members:

```
union REGS {
    unsigned int /* WORDREGS member */
        ax, bx, cx, dx, si, di, cflag, flags;
    unsigned char /* BYTEREGS member */
        al, ah, bl, bh, cl, ch, dl, dh;
};
```

Before calling `intdos`, the values required to indicate the desired system call (as well as any parameter values this system call needs) are stored in the location to which `inregs` points.

**Return value** The function returns its result in the AX register after the desired interrupt has returned. If the `cflag` member of `outregs` is nonzero (i.e., if an error has occurred), the error code is stored in `_doserrno`.

### **Example**

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

main(int argc, char *argv[])
{
    union REGS r;
    int ret;

    if(argc != 2)
        abort();
    r.h.ah=0x41;
    r.x.dx=(unsigned)(argv[1]);
    ret=intdos(&r, &r);
    if(r.x.cflag) {
        fprintf(stderr, "Error %d\n", ret);
        return(-1);
    }
    return(0);
}
```

```
int intdosx(
    const union REGS *inregs,
    union REGS *outregs,
    struct SREGS sregs);
```

---

<b>Header file</b>	dos.h
<b>See also</b>	bdos, FP_SEG, intdos, segread
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `intdosx` calls the function dispatcher (INT 21H) to execute the system call specified in the `inregs` parameter. This interrupt can be used with large model data segments or far pointers.

<code>intnum</code>	specifies the desired interrupt.
<code>inregs</code>	points to the register values passed to the interrupt.
<code>outregs</code>	points to the register values returned from the interrupt.
<code>sregs</code>	points to the structure containing the segment-register values required by the interrupt.

The union `REGS` has the following members:

```
union REGS {
    unsigned int /* WORDREGS member */
        ax, bx, cx, dx, si, di, cflag, flags;
    unsigned char /* BYTEREGS member */
        al, ah, bl, bh, cl, ch, dl, dh;
};
```

The struct `SREGS` has the following members:

```
struct SREGS {
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
};
```

Before calling `intdosx`, the values required by the desired interrupt must be stored in the location to which `inregs` points. Similarly, the DS and ES registers of `sregs` must be initialized. When the interrupt returns, the `intdosx` function updates the ES and DS registers in `sregs` and restores DS.

### Return value

The function returns its result in the AX register after the desired interrupt has returned. If the `cflag` member of `outregs` is nonzero (i.e., if an error has occurred), the error code is stored in `_doserrno`.

### Example

```
#endif
ret=intdosx(&r, &r, &s);/* delete file */
if(r.x.cflag) {
    fprintf(stderr, "Error %d\n", ret);
    return(-1);
}
return(0);
}
```

## **void intr(int intno, struct REGPACK \*preg);**

---

<b>Header file</b>	dos.h
<b>See also</b>	geninterrupt, int86, int86x, intdos, intdosx
<b>Portability</b>	80x86 family
<b>Multi-thread</b>	DOS is not re-entrant

The function `intr` provides a means of executing software interrupts. The function generates an 8086 interrupt when called.

**Note:** `intr` may not be used for interrupts 25H and 26H. Use `absread` and `abswrite` instead

<code>intno</code>	specifies the interrupt being requested.
<code>preg</code>	points to a structure that will contain the values needed by the interrupt when <code>intr</code> is called, and that will contain the resulting values from the interrupt after <code>intr</code> returns.

The `REGPACK` structure is defined in `dos.h` as follows:

```
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**

None.

### **Example**

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

char *msg="Hello World\n\r";

main()
{
    struct REGPACK r;

    r.r_ax=0x4000;
    r.r_bx=1;
    r.r_cx=strlen(msg);
    r.r_ds=FP_SEG((void far *) msg);
    r.r_dx=FP_OFF((void far *) msg);
    intr(0x21, &r);
    printf(
        "%d bytes written to stdout\n",
        r.r_ax);
    return(0);
}
```



## **int ioctl(int handle, int action [, void \*argd, int argc] );**

**U**

**Header file**           io.h

**Portability**           UNIX/DOS/OS2. The exact operation depends on the operating system and hardware.

Interfaces to DOS interrupt 44H, which controls I/O devices. The result depends on the value of action, which can be any of the following:

- 0           get device information.
- 1           set device information to the values specified in argd.
- 2           read argc bytes into the location to which argd points.
- 3           write argc bytes from the location to which argd points.
- 4           same as 2 except that handle is interpreted as a device number, with 0 = default drive, 1 = drive A, etc.
- 5           same as 3 except that handle is interpreted as a device number, with 0 = default drive, 1 = drive A, etc.
- 6           get the input status
- 7           get the output status
- 8           test whether the drive is removable (DOS 3.0 and later only)
- 11          set count for sharing conflict retries (DOS 3.0 or later only).

See the MS-DOS *Programmer's Reference Manual* for more information about INT 44H.

Because this function provides direct access to device drivers, the exact behavior of this function will vary from machine to machine

RetREGPACK structureREREGPACK structure

### **Return value**

depends on the value of *action*:

- 0 or 1           the returned value represents the device information.
- 2, 3, 4, or 5    it represents the number of bytes actually transferred.
- 6 or 7           it represents the device status.

In case of error, the function returns -1, and sets errno to one of:

EINVAL	invalid argument
EBADF	invalid file number.
EINVDAT	invalid data

<code>int isalnum(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isalpha(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isascii(int c);</code>	
----------------------------------	--

<code>int iscntrl(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isdigit(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isgraph(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int islower(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isprint(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int ispunct(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isspace(int c);</code>	<b>A</b>
----------------------------------	----------

<code>int isupper(int c);</code>	<b>A</b>
----------------------------------	----------

**int isxdigit(int c);****A**

<b>Header file</b>	ctype.h
<b>See also</b>	tolower, toupper.
<b>Portability</b>	ANSI, except isascii
<b>Multi-thread</b>	None.

The ctype routines listed here test the integer *c*, returning a nonzero value if the integer satisfies the test condition and zero if it does not.

isascii	produces a meaningful result for all integers. The remaining routines produce a defined result only for integer values corresponding to the ASCII character set plus the IBM extended character set.
isalnum	tests whether <i>c</i> is an alphanumeric, i.e., whether it has any of the following values: 'A' Æ 'Z,' 'a' Æ 'z,' '0' Æ '9.'
isalpha	tests whether <i>c</i> is an alphabetic character, i.e., whether it has any of the following values: 'A' Æ 'Z,' 'a' Æ 'z.'
isascii	tests whether <i>c</i> is an ASCII character, i.e., whether it has any of the following values: 0x0 Æ 0x7F.
isctrl	tests whether <i>c</i> is a control character, i.e., whether it has any of the following values: 0x0 Æ 0x1F or 0x7F.
isdigit	tests whether <i>c</i> is a decimal digit i.e., whether it has any of the following values: '0' Æ '9.'
isgraph	tests whether <i>c</i> is a printable character (not including space), i.e., whether it has any of the following values: 0x21 Æ 0x7E.
islower	tests whether <i>c</i> is lower case, i.e., whether it has any of the following values: 'a' Æ 'z.'
isprint	tests whether <i>c</i> is a printable character, i.e., whether it has any of the following values: 0x20 Æ 0x7E.
ispunct	tests whether <i>c</i> is a punctuation character, i.e., whether it has any of the following values: 0x21 Æ 0x2F, 0x3A Æ 0x40, 0x5B Æ 0x60, 0x7B Æ 0x7E.
isspace	tests whether <i>c</i> is a space character, i.e., whether it has any of the following values: 0x9 Æ 0xD and 0x20.
isupper	tests whether <i>c</i> is upper case, i.e., whether it has any of the following values: 'A' Æ 'Z.'
isxdigit	tests whether <i>c</i> is a hexadecimal digit, i.e., whether it has any of the following values: 'A' Æ 'F,' 'a' Æ 'f,' '0'

Æ ‘9.’

**Note:** If the ANSI switch is used, the ctype routines are implemented as functions unless the macro `_CT_MTF` is defined.

### Return value

Each function returns a nonzero value if the tested character is within the specified category and 0 if it is not.

### Example

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

main()
{
    int c;

    printf("Alphanumeric characters : ");
    c=0;
    while(c < 0x80)
    {
        if(isalnum(c))
            printf("%c ", c);
        ++c;
    }
    printf("\n");
    printf("Punctuation characters : ");
    c=0;
    while(c < 0x80) {
        if(ispunct(c))
            printf("%c ", c);
        ++c;
    }
    return(0);
}
```

## **int isatty(int handle);**

**U**

<b>Header file</b>	io.h
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `isatty` checks whether the file associated with `handle` is a character device (i.e., a terminal, console, printer or serial port.)

### **Return value**

The function returns a nonzero value if the file is a character device and 0 if it is not.

### **Example**

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

#define FILE 1
#define DEVICE 0

int check_stdout()
{
    int status=DEVICE;
    /* is stdout a device? */
    if(!isatty(fileno(stdout)))
    {
        fprintf(stderr, "stdout is redirected\n");          /* no -
        redirected */
        status=FILE;
    }
    return(status);
}
```

## **char \*itoa(int num, char \*s, int radix);**

---

<b>Header file</b>	stdlib.h
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function itoa converts the integer num to a null terminated ASCII string and stores the result at s.

num	specifies the number to be converted.
s	points to the buffer in which the string version of the converted number will be stored. This can be up to 18 bytes long (including the null terminator).
radix	specifies the base of the value to be converted. This must be within the range 2Æ36.

If the radix is 10 and the value is negative, the first character in the string will be the minus sign.

### **Return value**

The function returns a pointer to s. There is no error return value.

### **Example**

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

main()
{
    int number;
    char buffer[20];
    char *binary_result;

    printf("Enter decimal number ");
    scanf("%d", &number);
    /* convert number to string */
    binary_result=itoa(number, buffer, 2);
    printf(
        "\nBinary equivalent: %s\n",
        binary_result);
    return(0);
}
```

## int kbhit(void);

---

<b>Header file</b>	conio.h
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function kbhit checks the keyboard to determine whether there is a key press waiting. The key press remains in the buffer after the call to kbhit.

**Return value** The function returns a nonzero value if there is a keystroke waiting, and 0 if there is not.

### Example

```
#define _JPI_WIN_  
#include <conio.h>  
#include <process.h>  
  
wintype W4;  
SIGNAL DS;  
int _isbw;  
void P4()  
{  
    unsigned i;  
    use(W4) ;  
    do  
    {  
        for(i = 1; i <= BUFFER_SIZE; i++)  
        {  
            printf("This is a test of ");  
            printf("9 time-sliced processes, ");  
            printf("and also of window ");  
            printf("writing, moving, ");  
            printf("rearranging and resizing.");  
            W4->WDef.Foreground = White ;  
            if( _isbw )  
                W4->WDef.Background = Black ;  
            printf(" Press any key to ");  
            printf("terminate, P to freeze. ");  
            W4->WDef.Foreground = Black ;  
            if(_isbw )  
                W4->WDef.Background = LightGray;
```

```
                /* if a key is hit break*/
                /* out of loops */
                if(kbhit())
                    goto out;
            }
        }
        while(1);
out:
    SEND(DS);
    use(_fullscreen) ;
    snapshot();
    putontop(_fullscreen) ;
    gotoxy(1, SCREENDEPTH-1);
    clrscr();
    cursoron();
    return;
}
```



## **void keep(unsigned char status, unsigned size);**

---

<b>Header file</b>	dos.h
<b>See also</b>	abort, exit
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function keep exits the current program, and leaves it resident in memory. The function uses interrupt 31H for this purpose.

status                      specifies the exit status for the program.

size specifies the number of paragraphs of memory space to allocate for the program.

### **Return value**

None.

## **long labs(long num);**

**A**

```
long double labsl(long double x);
```

**Header file**            `stdlib.h`

**See also**            `abs`, `cabs`, `fabs`.

**Portability**        `ANSI`

**Multi-thread**      `None`.

The function `labs` returns the absolute value of its long integer argument, `num`.

### **Return value**

The function returns `|num|`. Thus, if `num` is positive, the function simply returns `num`; if `num` is negative, the function returns `-num`. There is no error return value.

### **Example**

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

main() {
    unsigned long an;
    long n=-10;

    an=labs(n);
    printf(
        "Absolute value of %ld is %lu\n", n, an);
    return(0);
}
```

## double ldexp(double x, int exp);

**A**

```
long double ldexpl(long double x, int exp);
```

**Header file**           math.h

**See also**            frexp, modf.

**Portability**        ANSI

**Multi-thread**       None.

The function `ldexp` calculates the value of its double precision floating point argument `x`, multiplied by  $(2 \text{ raised to the power } \text{exp})$ . `ldexpl` is the same as `ldexp`, but takes a long double argument.

`x`                    specifies the number being raised to a power.

`exp`                  specifies the power of 2 to which `x` is being raised.

### Return value

The function returns  $x * 2^{\text{exp}}$ . If an overflow occurs the function returns `HUGE_VAL`, depending on the sign of `x`, and sets `errno` to `ERANGE`.

### Example

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

main()
{
    double result, val;
    int exponent;

    result=ldexp(val, exponent);
    printf(
        "%g * 2^%d = %g", val, exponent, result);
    return(0);
}
```

---

**struct ldiv\_t ldiv(long num, long den);**

---

**A****Header file**            stdlib.h**See also**             div.**Portability**          ANSI**Multi-thread**        None.

The functions ldiv divides num by den, storing the quotient in structure member quot and the remainder in structure member rem. The struct ldiv\_t is defined in stdlib.h:

```
typedef struct
    long quot;
    long rem;
} ldiv_t;
```

If the denominator is zero, the program will terminate with an error message.

**Return value**

The function returns a structure of type ldiv\_t, which contains members for quotient and remainder.

**Example**

```
#include <stdlib.h>

long modulus(long x, long y)

{
    ldiv_t result;

    result=ldiv(x, y);
    return(result.rem);
}
```

```
char *lfind(  
    const char *key, const char *base,  
    unsigned *num, unsigned width,  
    int (*compare)(const void *e1,  
    const void *e2));
```

---

```
char *lsearch(  
    const char *key, const char *base,  
    unsigned *num, unsigned width,  
    int (*compare)(const void *e1,  
    const void *e2));
```

---

<b>Header file</b>	search.h
<b>See also</b>	bsearch.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	Access to static data by these functions is not protected by the library.

The functions `lfind` and `lsearch` perform a linear search for the value `key` in the array `base`.

<code>key</code>	specifies the key being sought in the array.
<code>base</code>	specifies the array being searched.
<code>num</code>	specifies the number of elements in <code>base</code> .
<code>width</code>	specifies the number of bytes in each element of <code>base</code> .
<code>compare</code>	is a pointer to a user-supplied function that compares two array members ( <code>e1</code> and <code>e2</code> ) and returns an integer corresponding to their relative values:
<code>e1 != e2</code>	return nonzero
<code>e1 == e2</code>	return 0

If the key value is not found, `lsearch` appends the value while `lfind` does not.

Note:

**Neither `lfind` nor `lsearch` requires the array to be sorted.**

**Return value**

If the value is found, both functions return a pointer to the array element that matches that value. If the value is not found, `lfind` returns `NULL`, while `lsearch` returns a pointer to the newly added member.

### Example

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

#define NUMBER_OF_ELEMENTS BUFFER_SIZE

int compare(void *n1, void *n2);

extern *data[];

main()
{
    double key=3.142;
    char *result;

    result=bsearch(
        &key, data,
        NUMBER_OF_ELEMENTS,
        sizeof(double),
        compare);
    if(result)
        printf("Key Found\n");
    return(0);
}

int compare(void *n1, void *n2)
{
    return(
        (int)
        ((*((double *)n1)) - *((double *)n2)));
}
```

---

**short far \_lineto(short x, short y);**

---

**M****Header file** graph.h**See also** \_getcurrentposition, \_setlinestyle, \_moveto**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_lineto` from the graphics module draws a line from the current graphics position to the logical point specified by the arguments. The line is drawn using the current color and line style.

`x` specifies the horizontal coordinate of the target point.

`y` specifies the vertical coordinate of the target point.

The function updates the current position to the end point of the line.

**Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

**Example**

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

main()
{
    _setvideomode(_MRESNOCOLOR);
    _moveto(20, 20);
    _lineto(20, 90);
    _lineto(80, 90);
    _lineto(80, 20);
    _lineto(20, 20); /* draw rectangle */
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**struct lconv \*localeconv(void)****A**

<b>Header file</b>	locale.h
<b>See also</b>	setlocale.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Locale static data are not protected by semaphore. Use Lock if another thread may call locale functions.

The function `localeconv` returns a pointer to the structure specifying the current locale. This structure is defined in `locale.h`:

```
struct lconv {
    char *decimal_point;
    char *thousands_sep;
    char *grouping;
    char *int_curr_symbol;
    char *currency_symbol;
    char *mon_decimal_point;
    char *mon_thousands_sep;
    char *mon_grouping;
    char *positive_sign;
    char *negative_sign;
    char int_frac_digits;
    char frac_digits;
    char p_cs_precedes;
    char p_sep_by_space;
    char n_cs_precedes;
    char n_sep_by_space;
    char p_sign_posn;
    char n_sign_posn;
};
```

If a char structure member has the value `CHAR_MAX`, this indicates that the value is not available in the current locale

If a char \* member points to "", this also indicates that the value is not available in the current locale.

**Return value** The function returns the value of the struct lconv associated with the current locale.

**Example**

```
#include <stdio.h>
#include <locale.h>
#include <limits.h>
main()
{
    struct lconv *current_locale;
    char *msg;

    current_locale=localeconv();
    if(current_locale->p_sign_posn == CHAR_MAX)
        printf("p_sign_posn not available ");
        printf("in this locale\n");
    return(0);
}
```



---

**struct tm \*localtime(const time\_t \*tt);**

---

**A**

<b>Header file</b>	time.h
<b>See also</b>	asctime, ctime, time, gmtime.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	A call to localtime only destroys the result of a previous call from the same thread.

The function localtime converts time value stored at tt to a structure. The value tt would be obtained from a call to the function time, and represents the number of seconds elapsed since 00:00:00 January 1st 1970, Greenwich Mean Time. The structure pointed to represents local time adjusted for timezone and any daylight saving.

The structure tm is defined in time.h and contains the following fields:

tm_sec	Seconds (0-59).
tm_min	Minutes (0-59)
tm_hour	Hours (0-23).
tm_mday	Day of Month (1-31).
tm_mon	Month (0-11).
tm_year	Year (Current year minus 1900).
tm_wday	Day of week(0-6 Sunday = 0).
tm_yday	Day of year(0-365).tm_isdst
Nonzero	if daylight saving.

The function localtime uses the global variables timezone and daylight to calculate the local time.

timezone must be set to the signed number of seconds difference to GMT. If daylight saving is in effect, daylight must be nonzero.

**Note:** The functions gmtime and localtime use a statically allocated buffer to hold the result. Each call to one of these functions destroys the result of a preceding call.

**Return value**

The function returns a pointer to the structure result. There is no error return value.

**Example**

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

main()

{
    time_t tt;
    struct tm *lct;

    time(&tt); /* get time in seconds */
    lct=gmtime(&tt); /* convert to structure */
    printf(
        "Hour is %d, local time\n",
        lct->tm_hour);
    return(0);
}
```

## **void Lock(void);**

---

<b>Header file</b>	process.h
<b>See also</b>	Unlock
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

The function Lock prevents the current process from being descheduled by means of time-slicing, until a call to Unlock. This is useful when a process is accessing data which is shared among several processes. Calls to Lock may be nested, but must always be paired with calls to UnLock.

### **Return value**

None.

## **int lock(int handle, long offset, long length);**

---

**T**

<b>Header file</b>	io.h
<b>See also</b>	unlock
<b>Portability</b>	DOS and OS2. Provided for Turbo C compatibility

The function lock, provided for compatibility with Turbo C, sets file-sharing locks via the DOS 3.x file-sharing mechanism. A lock can be placed on non-overlapping regions of any file. A program trying to read or write into a locked region will retry the operation three times. If all fail the call fails with error.

**Note:** To avoid error all locks must be removed before a file is closed. A program must release all locks before completing. If this is not done the result is undefined.

handle	low-level handle of the file to lock
offset	starting point of the region to lock
length	length of the region to lock

### **Return value**

If successful returns 0. If unsuccessful after three tries returns -1.

## **int locking(int handle, int mode, long numbytes);**

**U**

**Header file**           locking.h, io.h

**Portability**           DOS 3.X and above. Under OS2 Neither SHARE.COM nor SHARE.EXE is required.

The function `locking` locks or unlocks a specified number of bytes from the current position of the file pointer in the file specified by `handle`. Locking a section of a file prevents access to those bytes by other processes. Unlocking a file allows access to a previously locked section of a file.

`handle`                specifies the file in which the locking or unlocking is to occur.

`numbytes`             specifies the number of bytes to be locked or unlocked.

`mode`                 specifies the action to be performed, and can take the following values, which are defined in `locking.h`:

`LK_LOCK`             Locks the specified bytes. If the bytes cannot be locked, it tries again after 1 second. Up to 10 attempts will be made. If the bytes still cannot be locked after 10 tries, the function will fail.

`LK_RLCK`            Same as `LK_LOCK`.

`LK_NBLCK`           Locks the specified bytes. If the bytes cannot be locked, the function fails.

`LK_NBRLCK`          Same as `LK_NBLCK`.

`LK_UNLCK`           Unlocks the specified bytes.

**Note:**   **More than one region of a file may be locked, but overlapping regions may not be locked.**

When unlocking a file, the section of file to be unlocked must be a section that was previously locked.

Adjacent locked sections of a file are not merged and must be unlocked separately.

All locks should be removed before closing a file or terminating a process.

### **Return value**

The function returns 0 if successful. If the function failed, the value -1 is returned and `errno` is set to one of the following values:

EACCES	Locking violation. File already locked or unlocked.
EBADF	Invalid file handle.
EDEADLOCK	File could not be locked after 10 attempts (for mode values LK_LOCK and LK_RLCK only).
EINVAL	An invalid mode argument was passed to the function.

**Example**

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

int lock_file(FILE *f, int byte_count)
{
    int ret;

    if(_osmajor < 3) {
        fprintf(
            stderr,
            "Locking not available under v2.%d",
            _osminor);
        return(-1);
    }
    ret=locking(fileno(f), LK_LOCK, byte_count);
    return(ret);
}
```

---

**double log(double x);**

---

**A**

---

**long double logl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	log10, exp, matherr, pow.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `log` returns the natural logarithm of its double precision floating point argument `x`. `logl` is the same as `log` but takes a long double argument.

**Return value**

The function returns the logarithm result. If `x <= 0`, the function prints a DOMAIN error to `stderr`, returns `-HUGE_VAL` and sets `errno` to `ERANGE`.

Error handling can be altered by assigning a new handler to the `matherr` variable.

**Example**

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

main() {
    double result;
    double x=2.718;

    result=log(x);
    printf("Log of %g is %.2g\n", x, result);
    return(0);
}
```

---

**double log10(double x);**

---

**A**

---

**long double log10l(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	log, exp, matherr, pow.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function log10 returns the base 10 logarithm of its double precision floating point argument x. log10l is the same as log10 but takes a long double argument.

**Return value**

The function returns the logarithm result. If  $x \leq 0$ , the function prints a DOMAIN error to stderr, returns -HUGE\_VAL and sets errno to ERANGE.

Error handling can be altered by assigning a new handler function to the matherr variable.

**Example**

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

main() {
    double result;
    double x=10;

    result=log10(x);
    printf("Log10 of %g is %.2g\n", x, result);
    return(0);
}
```

---

**void longjmp(jmp\_buf env, int retval);**

---

**A**

<b>Header file</b>	setjmp.h
<b>See also</b>	setjmp.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.
<b>Initialization</b>	setjmp().

The longjmp function restores the processor environment previously saved in env by setjmp. These functions provide a mechanism for executing inter-function gotos, and are usually used to pass control to error recovery code.

A call to setjmp causes the current processor environment to be saved in env. A following call to longjmp restores the saved environment and causes execution to resume at a point immediately after the corresponding setjmp call. Execution continues with retval as the return value from setjmp.

As long as longjmp is called before the function calling setjmp returns, all variables local to the routine will have the same value as when setjmp was called. However, register variables may not be restored. Use the volatile keyword to ensure that local variables are properly restored.

retval must be nonzero. If retval is passed a zero, the value 1 will be substituted.

**jmp\_buf is defined in setjmp.h.**

```
.i.jmp_buf structure;
typedef struct {
    unsigned j_sp; /* saved 80X86 registers */
    unsigned j_ss;
    unsigned j_flag;
    unsigned j_cs;
    unsigned j_ip;
    unsigned j_bp;
    unsigned j_di;
    unsigned j_es;
    unsigned j_si;
    unsigned j_ds;
} jmp_buf[1];
```

**Return value**

There is no return value.

**Example**



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

void func(void);
jmp_buf error_env;

main() {

    /* initialize jump environment */
    if(setjmp(error_env) != 0)
        abort(); /* arrives here from longjmp */
    func();
    printf("Program terminated without error\n");
    return(0);
}

void func(void)
{
    printf("Press any key to end program. ");
    printf("'e' will cause error\n");
    if(getch() == 'e')
        /* jump to abort call */
        longjmp(error_env, 1);
    return;
}
```

## **void lowvideo(void);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	highvideo, normvideo
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `lowvideo` from the Turbo C window module sets the active screen attribute to low intensity. The screen attribute is used by the console I/O functions.

### **Return value**

None.

### **Example**

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

main()
{
    highvideo();
    cprintf("This is displayed with ");
    cprintf("high intensity attributes\n");
    lowvideo();
    cprintf("This is displayed with ");
    cprintf("low intensity attributes");
    return(0);
}
```

## **unsigned long \_lrotl(long val, int count);**

---

## **unsigned long \_lrotr(long val, int count);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	_rotl, _rotr.
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	None.

The functions `_lrotl` and `_lrotr` rotate the long integer argument `val` left or right, respectively, by `count` bits.

### **Return value**

The rotated value is returned. There is no error return value.

### **Example**

```
unsigned long _lrotr(long val, int count);

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

main()
{
    long number;
    char buffer[40];
    char *binary_result;

    printf("Enter long decimal number ");
    scanf("%ld", &number);
    do
    {
        /* print binary */
        binary_result=ltoa(number, buffer, 2);
        printf(
            "\nBinary equivalent: %32s\n",
            binary_result);
        /* rotate number */
        number=_lrotl(number, 1);
    }
    while(getch() != 'q');
    return(0);
}
```

**long lseek(int handle, long offset, int origin);****U****Header file** io.h**See also** tell.**Portability** UNIX/DOS/OS2**Multi-thread** Low level I/O functions are not protected by the library.

The function lseek positions the file pointer associated with handle at a position that is offset bytes from origin. The next operation on the file takes place at the new position.

handle specifies the handle for the file or device in which the positioning will take place.

offset specifies the number of bytes the new position is displaced from origin.

origin specifies the reference location from which an offset will be computed. This argument may have the following values, which are defined in stdio.h:

SEEK\_SET Beginning of file.

SEEK\_CUR Current position.

SEEK\_END End of File.

**Note:** If a file is opened with O\_APPEND, all write operations will occur at the end of the file.

If handle refers to a character device, the return value is undefined.

**Return value**

The function returns the offset from the beginning of the file (in bytes). If an error occurred, the value -1 is returned and errno is set to one of the following values:

EBADF Invalid file handle.

EINVAL Invalid origin or offset is before the beginning of the file.

**Example**

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
main()
{
    int fh;
    char *msg="THIS WAS WRITTEN AT END OF FILE";

    fh=open("TEMP.$$$", O_RDWR);
    if(fh < 0)
        abort();
        /* seek to end of file */
    lseek(fh, 0L, SEEK_END);
    write(fh, msg, strlen(msg));
    close(fh);
    return(0);
}
```

## **char \*ltoa(long num, char \*s, int radix);**

---

**Header file**     stdlib.h

**See also**        itoa, ultoa.

**Portability**     UNIX/DOS/OS2

**Multi-thread**   None.

The function ltoa converts the long integer num to a null terminated ASCII string and stores the result at s.

num                specifies the number to be converted.

s                   points to the buffer in which the string version of the converted number will be stored. This can be up to 34 bytes long (including the null terminator).

radix              specifies the base of the value to be converted. This must be within the range 2Æ36.

If the radix is 10 and the value is negative, the first character in the string will be the minus sign.

### **Return value**

The function returns the a pointer to s There is no error return value.

### **Example**

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

main()
{
    long number;
    char buffer[40];
    char *binary_result;

    printf("Enter long decimal number ");
    scanf("%ld", &number);
    binary_result=ltoa(number, buffer, 2);
    printf(
        "\nBinary equivalent: %s\n",
        binary_result);
    return(0);
}
```

```
void _makepath(  
    char *path,  
    const char *drive,  
    const char *dir,  
    const char *name,  
    const char *ext);
```

---

**Header file**     dir.h

**See also**        \_splitpath

**Portability**     Some DOS/OS2

The function `_makepath` creates a single path name from individual components (drive, path, file name, and file extension).

path	points to buffer that will contain the complete path name. This buffer must be large enough to store the complete name. The value of the constant <code>_MAX_PATH</code> (defined in <code>stdlib.h</code> ) specifies the longest path that can be handled.
drive	specifies the letter of the desired drive (A, B, C, etc.). The argument may, but need not, include a colon. If this argument is a null string, neither drive letter nor colon will appear in the final path string.
dir	specifies the directory path <i>Æ</i> excluding the actual file name and the drive designator. The argument may, but need not, include a trailing slash, and may use either forward slashes (/) or backslashes (\) or both. A trailing slash is inserted automatically if not specified. If this argument is an empty string, no slash is inserted in the final path string.
name	specifies the actual file name, <i>excluding</i> extensions.
ext	specifies the extension for the actual file name. The argument may, but need not, include a leading period. If this argument is a null string, no period is inserted in the final path string.

While there are no size constraints on the individual components, the final path name must be at most `_MAX_PATH` characters.

### Return value

None.

**Example**

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

main()
{
    char full_path[80];

    _makepath(
        full_path, "D:",
        "\\TSC\\", "TEMP", "$$$");
    printf("Path is %s\n", full_path);
    return(0);
}
```



**void \*malloc(size\_t size);****A**

**Header file**     stdlib.h and alloc.h

**Variants**       \_nmalloc, \_fmalloc, farmalloc, halloc.

**See also**        calloc, free, realloc.

**Portability**     ANSI

**Multi-thread**   Far heap only protected by semaphore.

Allocates a block of at least size bytes from the heap.

**Note:**     **If size is 0, malloc returns NULL.**

In small and medium models, the object to which the return value points is word aligned and is in the default data segment; in other models, the object is paragraph aligned, and is not in the default data segment.

**Return value**

Returns a void pointer to the allocated space. If no space was available, a NULL pointer will be returned and errno is set to ENOMEM.

**Example**

```
#include <alloc.h>

main()
{
    char *ptr;
    char near *near_ptr;
    char far *far_ptr;
        /* allocate from default heap */
    ptr=malloc(200);
    free(ptr);
        /* allocate from near heap */
    near_ptr=_nmalloc(200);
    _nfree(near_ptr);
        /* allocate from far heap */
    far_ptr=_fmalloc(200);
    _ffree(far_ptr);
    return(0);
}
```

## **int (\*matherr)(struct exception \*buf);**

---

**Header file**     math.h

**Portability**     UNIX/DOS/OS2

**Multi-thread**   None

The function `matherr` processes errors generated by functions in the `math` module. It can be reassigned by the user. Whenever an error occurs, the function concerned will call `matherr`, passing a pointer to an exception type structure defined in `math.h`: exception structure

```
struct exception{
    int      type;
    char     *name;
    double   arg1, arg2, retval;
};
```

All the mathematical functions that use `matherr` do so in a standard way and respond to the return values which it supplies.

**type**                specifies the type of error and will be one of the following constants:

**DOMAIN**            parameter not in valid domain

**SING**                function is in calculable at this place

**OVERFLOW**        result exceeds representable range

**UNDERFLOW**      result is too close to zero to be represented

**TLOSS**            total loss of precision, e.g. computing  $\sin(2^{65})$

**PLOSS**            partial precision loss

**name**                is a pointer to a null terminated string containing the name of the function that caused the error.

**arg1**                aexception structure  
**arg2**                the arguments that caused the error.

**retval**              is the return value for the function that caused the error.

### **Return value**

If `matherr` returns 0, `errno` is set and the function will return its default error return value.

If `matherr` returns a nonzero value, no action is taken by the function causing the error and that function returns `retval`.

`matherr` is actually a function pointer, and a new error handling function may be defined and assigned to `matherr`, as shown in the example below.

**Example**

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

int new_matherr(struct exception *buf)
{
    printf(
        "Function %s caused error %d",
        buf->name, buf->type);
    buf->retval=1.0;
    return(1);
}

void f()
{
    matherr=new_matherr;
    return;
}
```

## **type max(type val1, type val2);**

---

**Header file**     stdlib.h

**See also**       min.

**Portability**    UNIX/DOS/OS2

**Multi-thread**   None.

The max macro returns the larger of the two values val1 and val2. Both arguments must be of the same type.

### **Return value**

The function returns the larger argument.

### **Example**

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

main()
{
    int num1, num2;

    printf("Enter two numbers ");
    scanf("%d %d", &num1, &num2);
    printf("Largest is %d\n", max(num1, num2));
    return(0);
}
```

---

**int mblen(const char \*s, size\_t n);**

---

**A****Header file**     stdlib.h**See also**        mbtowc**Portability**     ANSI**Multi-thread**    None.

The function `mblen` determines the number of bytes comprising the multibyte character to which `s` points.

`s`                   points to a multibyte character.

`n`                   specifies the maximum number of bytes in a multibyte character.

**Return value**

If `s` points to the null character, `mblen` returns 0, otherwise `mblen` returns 1 in this implementation.

**Example**

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

#define MAX_LEN 2

main()
{
    const char s='\ ' ;
    int mbyte_len;

    mbyte_len=mblen(&s, MAX_LEN);
    if(mbyte_len == 0)
        printf("Null character\n");
    else
        printf(
            "Length of character is %d\n",
            mbyte_len);
    return(0);
}
```

---

**size\_t mbstowcs(wchar\_t \*pwcs, const char \*s, size\_t n);**

---

**Header file**     stdlib.h

**See also**       mbtowc

**Portability**    ANSI

**Multi-thread**   None.

The function `mbstowcs` converts a sequence of multibyte characters pointed to by `s`, and stores them as codes at the address to which `pwcs` points. In this implementation, the function is equivalent to a call to `strcpy`.

`n`                specifies the maximum number of characters to convert.

`s`                points to the buffer in which the multibyte character sequence is stored.

`pwcs`            points to a buffer in which the codes associated with the multibyte characters can be stored.

**Return value**

`mbstowcs` returns the number of characters copied.

**Example**

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

#define MAX_LEN 20

main()
{
    const char *mbchars="Hello World";
    wchar_t buffer[MAX_LEN];
    int number_modified;

    number_modified=mbstowcs(
        buffer, mbchars, MAX_LEN);
    printf(
        "%d characters converted in \"%s\"\n",
        number_modified, mbchars);
    return(0);
}
```

---

**int mbtowc(wchar\_t \*pwc, const char \*s, size\_t n);**

---

**A****Header file**     `stdlib.h`**See also**        `mbstowcs`**Portability**    `ANSI`

The function `mbtowc` converts up to `n` bytes comprising the multibyte character pointed to by `s` to a code representing that character, and stores this code at `pwc`, provided that `pwc` is not a `NULL` pointer.

`n`                    specifies the maximum number of characters to convert.

`s`                    points to the buffer in which the multibyte character sequence is stored.

`pwc`                points to a buffer in which the code associated with the multibyte character can be stored.

**Return value**

If `s` points to the null character, 0 is returned; otherwise `mbtowc` returns 1 in this implementation.

**Example**

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

#define MAX_LEN 1

main()
{
    const char mbchar='H';
    wchar_t buffer;
    mbtowc(&buffer, &mbchar, MAX_LEN);
    return(0);
}
```

## **size\_t \_memavl(void);**

---

**Header file**     alloc.h

**See also**        malloc, \_nmalloc

**Portability**     Some DOS/OS2

**Multi-thread**   Far heap only protected by semaphore.

The function `_memavl` determines the number of bytes available for allocation in the near heap.

### **Return value**

The function return the number of available bytes.

### **Example**

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

void near *get_all_heap(int *size)

{
    void near *buffer;
    size_t mem_left;

    mem_left=_memavl(); /* get size of block */
                        /* try to allocate it */
    buffer=_nmalloc(mem_left);
    if(buffer == NULL)
        /* probably wasn't one block */
        return(NULL);
    *size=mem_left;
    return(buffer);
}
```



---

**void \*memcpy(void \*dest, const void \*source, int c, size\_t num);**

---

**Header file** mem.h and string.h**See also** hmemcpy, hmemchr, hmemcmp, hmemcpy, hmemset, memchr, memcmp, memcpy, memicmp, memset**Portability** UNIX/DOS/OS2**Multi-thread** None.

The function memcpy copies bytes from source to dest. The function copies all bytes up to and including the first occurrence of character c or until a maximum number of bytes have been copied Æ whichever comes first.

**dest** specifies the target location for the bytes being copied.

**source** specifies the location from which the bytes are being copied.

**c** specifies the character whose occurrence will signal the end of the block to be copied.

**num** specifies the maximum number of bytes to copy.

**Return value**

If the character c is copied, the function returns a pointer to the byte immediately after c in dest. If c is not copied it returns NULL.

**Example**

```
#include <stdio.h>
#include <string.h>
#define BUFFER_SIZE 200
main()
{
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(char), BUFFER_SIZE, f);
    memcpy(buffer2, buffer1, '\n', BUFFER_SIZE);
    fclose(f);
    return(0);
}
```

---

**void \*memchr(const void \*buf, int c, size\_t num);**

---

**A**

**Header file** mem.h and string.h

**See also** hmemccpy, hmemchr, hmemcmp, hmemcpy, hmemset, memccpy, memcmp, memcpy, memicmp, memset

**Portability** ANSI

**Multi-thread** None.

The function memchr searches for character c in the first num bytes of buf. The search stops when c is found or when num bytes have been searched.

buf specifies the location to be searched.

c specifies the character begin sought.

num specifies the maximum number of characters to search.

**Return value**

If c was found, the function returns a pointer to the first occurrence. If c was not found, the function returns NULL.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    char buffer1[BUFFER_SIZE];
    char *check;
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(char), BUFFER_SIZE, f);
    check = memchr(buffer1, '\n', BUFFER_SIZE);
    if (check) printf ( "found\n");
    else      printf ( "not found\n");
    fclose(f);
    return(0);
}
```

---

**int memcmp(const void \*s1, const void \*s2, size\_t num);**

---

**A****Header file** mem.h and string.h**See also** hmemccpy, hmemchr, hmemcmp, hmemcpy, hmemset, memccpy, memchr, memcpy, memicmp, memset**Portability** ANSI**Multi-thread** None.

The function memcmp compares the first num bytes of s1 and s2.

**s1** specifies the first block of memory being compared.

**s2** specifies the second block of memory being compared.

**num** specifies the number of bytes to compare.

**Return value**

The function returns an integer indicating the relationship of s1 and s2:

```
s1 < s2 return < 0
s1 = s2 return = 0
s1 > s2 return > 0
```

**Example**

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

#define BUFFER_SIZE 512

main(int argc, char *argv[])
{
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE];
    FILE *f1, *f2;
    int size1, size2, comp_size;
    int c;
```

```
if(argc != 3)
{
    printf("Usage - COMPARE file1 file2\n");
    abort();
}
f1=fopen(argv[1], "rb");
f2=fopen(argv[2], "rb");
do
{
    size1=fread(buffer1, sizeof(char),
        BUFFER_SIZE, f1);
    size2=fread(buffer2, sizeof(char),
        BUFFER_SIZE, f2);
    if(size1 != size2)
    {
        printf("Files are different sizes, ");
        printf("press C to continue");
        printf(" or any key to exit\n");
        c=getch();
        if(tolower(c) != 'c')
            return(0);
        comp_size=min(size1, size2);
    }
    else
        comp_size=size1;
    if(memcmp(buffer1, buffer2, comp_size))
    {
        printf("Files are not equal\n");
        return(0);
    }
}
while(comp_size == BUFFER_SIZE);

printf("Files are equal\n");
fclose(f1);
fclose(f2);
return(0);
}
```

---

**void \*memcpy(void \*dest, const void \*source, size\_t num);**

---

**A****Header file** mem.h and string.h**See also** hmemcpy, hmemchr, hmemcmp, hmemcpy, hmemset, memccpy, memchr, memcmp, memicmp, memset**Portability** ANSI**Multi-thread** None.

The function `memcpy` copies `num` bytes from `source` to `dest`. `memcpy` does not ensure that overlapping regions of memory are correctly copied. If this is necessary use `memmove`.

`dest` specifies the target location for the bytes being copied.

`source` specifies the location from which the bytes are being copied.

`num` specifies the number of bytes to copy.

**Return value**

The function returns a pointer to `dest`.

**Example**

```
#define BUFFER_SIZE 200

main()
{
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(char), BUFFER_SIZE, f);
    memcpy(buffer2, buffer1, BUFFER_SIZE);
    fclose(f);
    return(0);
}
```

---

**int memicmp(const void \*s1, const void \*s2, size\_t num);**

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemccpy, hmemchr, hmemcmp, hmemcpy, hmemset, memccpy, memchr, memcmp, memset
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function memicmp compares the first num bytes of s1 and s2 without regard for the case of letters. I.e., 'A' is considered equal to 'a'.

s1	specifies the first block of memory being compared.
s2	specifies the second block of memory being compared.
num	specifies the number of bytes to compare.

**Return value** an integer indicating the relationship of s1 and s2:

```
s1 < s2 return < 0  
s1 = s2 return = 0  
s1 > s2 return > 0
```

**Example**

```
#include <stdio.h>  
#include <string.h>  
  
#define BUFFER_SIZE 200  
  
main()  
{  
    char buffer1[]="Hello World, mixed case";  
    char buffer2[]="hello world, lower case";  
    int  similar, identical;  
  
    identical=memcmp(buffer2, buffer1, 12);  
    similar=memcmp(buffer2, buffer1, 12);  
    if(identical == 0)  
        printf("Strings are identical\n");  
    if(similar == 0){  
        printf("Strings are the same ");  
        printf("disregarding case\n");  
    }else  
        printf("Strings are different\n");  
    return(0);  
}
```

## **size\_t \_memmax(void);**

---

**Header file**            alloc.h

**See also**             malloc, \_msize

**Portability**         Some DOS/OS2

**Multi-thread**        Far heap only protected by semaphore.

The function `_memmax` returns the size (in bytes) of the largest object that can be allocated from the near heap.

### **Return value**

The function returns the number of contiguous bytes that could be allocated. If there is no more available storage in the near heap, the function returns 0.

### **Example**

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

void near *get_large_block(int *size)
{
    void near *buffer;
    size_t available;
    /* get size of largest block */
    available=_memmax();
    buffer=_nmalloc(available); /* allocate it */
    if(buffer == NULL)
        abort();
    *size=available; /* set to block size */
    return(buffer); /* return block address */
}
```



```
void *memmove(void *dest, const void *source,  
              size_t num);
```

---

**A**

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemcpy, memcpy, memmove, memwmove
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `memmove` copies `num` bytes from `source` to `dest`. If some regions of memory are overlapping, `memmove` ensures that these regions are copied before being overwritten.

<code>dest</code>	specifies the target location for the bytes being copied.
<code>source</code>	specifies the location from which the bytes are being copied.
<code>num</code>	specifies the number of bytes to copy.

**Return value**

The function returns a pointer to `dest`.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE*2];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(char), BUFFER_SIZE, f);
    memmove(&buffer2[50], buffer1, BUFFER_SIZE);
    fclose(f);
    return(0);
}
```

---

**void \*memset(void \*s, int c, size\_t num);**

---

**A**

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemccpy, hmemchr, hmemcmp, hmemcpy, hmemicmp, memccpy, memchr, memcmp, memcpy, memicmp, memwcpy, memwmove, memwset
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `memset` sets the first `num` bytes of `s` to a specified character.

<code>s</code>	points to the location being initialized.
<code>c</code>	specifies the character to which the buffer will be set.
<code>num</code>	specifies the number of bytes to set.

**Return value**

The function returns a pointer to `s`.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    char buffer1[]="Hello World";
    char buffer2[]="Hello World";
    /* will return 0 */
    memcmp(buffer2, buffer1, 12);
    memset(buffer2, 'A', 12);
    /* will now return < 0 */
    memcmp(buffer2, buffer1, 12);
    return(0);
}
```

---

**void \*memcpy(void \*dest, const void \*source, size\_t num);**

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemcpy, hmemcpy, memcpy, memcpy, memmove, memmove
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `memcpy` copies `num` *words* from `source` to `dest`. `memcpy` does not ensure that overlapping regions of memory are correctly copied. If this is necessary use `memmove`.

<code>dest</code>	specifies the target location for the bytes being copied.
<code>source</code>	specifies the location from which the bytes are being copied.
<code>num</code>	specifies the number of words to copy.

**Return value**

The function returns a pointer to `dest`.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    int buffer1[BUFFER_SIZE];
    int buffer2[BUFFER_SIZE];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(int), BUFFER_SIZE, f);
    memcpy(buffer2, buffer1, BUFFER_SIZE);
    fclose(f);
    return(0);
}
```

---

**void \*memwmove(void \*dest, const void \*source, size\_t num);**

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemccpy, hmemcpy, memccpy, memcpy, memmove, memwcpy
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `memwmove` copies `num` *words* from `source` to `dest`. If some regions of memory are overlapping, `memwmove` ensures that these regions are copied before being overwritten.

<code>dest</code>	specifies the target location for the bytes being copied.
<code>source</code>	specifies the location from which the bytes are being copied.
<code>num</code>	specifies the number of words to copy.

**Return value**

The function returns a pointer to `dest`.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    int buffer1[BUFFER_SIZE];
    int buffer2[BUFFER_SIZE*2];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(int), BUFFER_SIZE, f);
    memwmove(&buffer2[50], buffer1, BUFFER_SIZE);
    fclose(f);
    return(0);
}
```

## **void \*memset(void \*s, int c, size\_t num);**

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemccpy, hmemchr, hmemcmp, hmemcpy, hmemicmp, memccpy, memchr, memcmp, memcpy, memicmp, memwcpy, memwmove, memset
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `memset` sets the first `num` *words* of `s` to integer `c`.

<code>s</code>	points to the location being initialized.
<code>c</code>	specifies the character to which the buffer will be set.
<code>num</code>	specifies the number of words to set.

### **Return value**

The function returns a pointer to `s`.

### **Example**

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

#define BUFFER_SIZE 200

main()
{
    int buffer1[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    int buffer2[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    /* will return 0 */
    memcmp(buffer2, buffer1, 20);
    memset(buffer2, 1, 10);
    /* will now return < 0 */
    memcmp(buffer2, buffer1, 20);
    return(0);
}
```

## **type min(type val1, type val2);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	max.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The min macro returns the smaller of the two values val1 and val2. Both arguments must be of the same type.

### **Return value**

The function returns the smaller argument.

### **Example**

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

main()
{
    int num1, num2;

    printf("Enter two numbers ");
    scanf("%d %d", &num1, &num2);
    printf("Smallest is %d\n", min(num1, num2));
    return(0);
}
```

## **int mkdir(const char \*path);**

---

<b>Header file</b>	dir.h
<b>See also</b>	chdir, rmdir.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function mkdir creates a new directory with the argument path. Only the last component of path can be a new directory.

### **Return value**

The function returns 0 if the new directory was created. If an error occurred, the value -1 is returned and errno is set to one of the following values:

EACCES	Path referred to existing file, directory or device.
ENOENT	Path name not found.

### **Example**

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

main(int argc, char *argv[])
{
    if(argc != 2) {
        printf("USAGE - NEWDIR directory\n");
        abort();
    }
    if(mkdir(argv[1])) {
        printf(
            "couldn't create directory %s\n",
            argv[1]);
        abort();
    }
    chdir(argv[1]);
    return(0);
}
```

---

**char \*mktemp(char \*base);**

---

**U**

<b>Header file</b>	io.h
<b>See also</b>	tmpfile, tmpnam
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

Creates a unique filename from base, which has the following form:

```
path\baseXXXX.XXX
```

where the ‘X’ are the place-holders for the part of the filename supplied by mktemp. Subsequent calls to mktemp cause the function to check whether the previously returned name has been used. If it has, a new filename will be returned. TMP\_MAX (which is defined in stdio.h) defines the maximum number of temporary file names that may be created.

**Note:**    **The function does not create or open the file, and the file will *not* be automatically deleted on program termination.**

**Return value**

The function returns a pointer to the modified template. If the function failed, the value NULL will be returned.

**Example**

```
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
main()
{
    char *file;
    FILE *f;

    file=mktemp("c:\\tempfiles\\tempXXXX.XXX");
    if(file != NULL) {
        f=fopen(file, "w");
        fprintf(f,
            "This won't be here for long!\n");
        fclose(f);
        unlink(file);
    }
    return(0);
}
```



---

**time\_t mktime(struct tm \*timeptr);**

---

**A**

<b>Header file</b>	time.h
<b>See also</b>	asctime, gmtime, localtime, time
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function mktime converts the local time (stored in the structure timeptr) into the format that would have been returned by a direct call to the function time. The function may be used to perform arithmetic on the values contained in the structure.

**Return value**

The function returns the encoded time. If the calendar time could not be represented, the value (time\_t)-1 is returned.

**Example**

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

time_t when_is_tomorrow()
{
    time_t tomorrow, now;
    struct tm *temp;

    time(&now); /* time in seconds */
    temp=localtime(&now);
    ++temp->tm_mday; /* this time tomorrow */
    /* tomorrow = time in seconds */
    tomorrow=mktime(temp);
    return(tomorrow);
}
```

## **void far \* MK\_FP(unsigned seg, unsigned off);**

---

**Header file**            dos.h

**See also**             FP\_OFF, FP\_SEG.

**Portability**         8086 family.

The macro MK\_FP constructs a far pointer using the value seg as the segment, or selector, and off as the offset.

### **Return value**

The function returns the far pointer value.

### **Example**

```
#include <dos.h>

#define SCREENWIDTH 160
#define SCREEN_BUFFER_SEG 0xB800

int write_char(int row, int col, int c)
{
    unsigned off;
    char far *ptr;

    off=row*SCREENWIDTH+col;
        /* construct far pointer */
    ptr=MK_FP(SCREEN_BUFFER_SEG, off);
    *ptr=c;
    return;
}
```

---

**double modf(double x, double \*intptr);**

---

**A**

---

**long double modfl(long double x, long double \*ipart);**

---

<b>Header file</b>	math.h
<b>See also</b>	frexp, ldexp
<b>Portability</b>	ANSI
<b>multi-thread</b>	None.

The function `modf` breaks its double precision floating point argument `x` into fractional and integer parts. The integer part is stored as a double precision floating point number at `intptr`. The function returns the fractional part. `modfl` is the same as `modf` but takes a long double argument.

<code>x</code>	specifies the value to be broken into components.
<code>intptr</code>	points to the buffer at which the integer part will be stored.

### Return value

The function returns the signed fractional part of `x`. There is no error return value.

### Example

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

main()
{
    double x=2.71828;
    double integer_part, fract_part;

    fract_part=modf(x, &integer_part);
    printf(
        "%g splits into portions %g and %g\n",
        x, integer_part, fract_part);
    return(0);
}
```

```
void movedata(
    unsigned sseg,
    unsigned soff,
    unsigned dseg,
    unsigned doff,
    size_t num);
```

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	FP_OFF, FP_SEG, memcpy, memmove, memwcpy, memwmove, segread
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `movedata` copies `num` bytes from the source (which is a pointer composed of segment `sseg` and offset `soff`) to the destination (which is a pointer composed of segment `dseg` and offset `doff`). If some regions of memory are overlapping, `movedata` ensures that these regions are copied before being overwritten.

<code>sseg</code>	specifies the segment value for the source block.
<code>soff</code>	specifies the offset value for the source block.
<code>dseg</code>	specifies the segment value for the destination block.
<code>doff</code>	specifies the offset value for the destination block.
<code>num</code>	specifies the number of bytes to copy.

### Return value

None.

### Example

```
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include <string.h>
#define SCREEN_SIZE 80*25*sizeof(unsigned)
#define SCREEN_SEG 0xB800
unsigned far *save_screen()
{
    unsigned far *buffer;
    buffer=_fmalloc(SCREEN_SIZE);
    if(buffer == NULL)
        return(NULL);
    movedata(
        SCREEN_SEG, 0, FP_SEG(buffer),
        FP_OFF(buffer), SCREEN_SIZE);
    return(buffer);
}
```

```
int movetext(  
    int left, int top,  
    int right, int bottom,  
    int newleft, int newtop);
```

---

T

<b>Header file</b>	conio.h
<b>See also</b>	gettext, puttext
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `movetext` from the Turbo C window module copies the contents of the screen area defined by the specified coordinates to a new position whose upper left corner is defined by `newleft` and `newtop`.

<code>left</code>	specifies the leftmost column to be copied.
<code>top</code>	specifies the topmost line to be copied.
<code>right</code>	specifies the rightmost column to be copied.
<code>bottom</code>	specifies the bottom line to be copied.
<code>newleft</code>	specifies the leftmost column of the new location.
<code>newtop</code>	specifies the top line of the new location.

Note: **coordinates are absolute screen coordinates.**

### Return value

The function returns 1 if the operation succeeded. If any of the coordinates were invalid, the return value is 0.

### Example

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

#define COLS 40
#define LINES 4

main()

{
    int n=0;
    char block[COLS+2];

    while(n <= COLS) {
        block[n]='X';
        ++n;
    }
    block[n]='\0';
    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    n=0;
    while(n < LINES) {
        cprintf(block); /* output to new window */
        ++n;
    }
    gotoxy(1, 1);
    cprintf("Press key to move text");
    getch();
    movetext(20, 4, 20+COLS, 4+LINES, 30, 10);
    return(0);
}
```

---

**struct xycoord far \_moveto(short x, short y);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_lineto
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_moveto` from the graphics module moves the current graphics output position to the logical point specified by the parameters.

<code>x</code>	specifies the horizontal coordinate of the target point.
<code>y</code>	specifies the vertical coordinate of the target point.

**Return value**

The function returns the logical coordinates of the *previous* position. This information is returned in an `xycoord` structure, which is defined in `graph.h`.

```
struct xycoord {
    short xcoord;
    short ycoord;
};
```

**Example**

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

main()
{
    _setvideomode(_MRESNOCOLOR);
    _moveto(20, 20);
    _lineto(20, 90);
    _moveto(50, 20);
    _lineto(50, 90); /* draw parallel lines */
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**void movmem(const void \*src, void \*dest, unsigned length);**

---

<b>Header file</b>	mem.h and string.h
<b>See also</b>	hmemccpy, hmemcpy, memccpy, memcpy, memmove, memwmove
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `movmem` copies `length` bytes from `src` to `dest`. If some regions of memory are overlapping, `movmem` ensures that these regions are copied before being overwritten.

<code>src</code>	points to the location being copied.
<code>dest</code>	points to the location to which the block is being copied.
<code>length</code>	specifies the number of bytes to copy.

**Return value**

None.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    char buffer1[BUFFER_SIZE];
    char buffer2[BUFFER_SIZE*2];
    FILE *f;

    f=fopen("TEMP.$$$", "r+");
    fread(buffer1, sizeof(char), BUFFER_SIZE, f);
    movmem(buffer1, &buffer2[50], BUFFER_SIZE);
    fclose(f);
    return(0);
}
```



---

**int \_msize(void \*buffer);**

---

<b>Header file</b>	alloc.h
<b>Variants</b>	_fmsize, _nmsize
<b>See also</b>	calloc, _expand, _fmalloc, _malloc, _nmalloc, realloc
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	Far heap only protected by semaphore.

The function `_msize` returns the size (in bytes) of the memory block allocated from the heap by the functions `calloc`, `malloc` or `realloc`. `buffer` points to this memory block.

**Return value**

The function returns the size (in bytes) as an unsigned integer.

**Example**

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

#define BLOCK_REQUEST 199 main()

{
    void *buffer;
    int block_size;

    buffer=malloc(BLOCK_REQUEST);
    block_size=_msize(buffer);
    /* rounding will have occurred */
    printf(
        "Request %d causes block ",
        BLOCK_REQUEST, );
    printf(
        "of size %d to be allocated\n",
        block_size);
    return(0);
}
```

## **void \_ms\_cursor(int mode);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Reveals or hides the mouse cursor, depending on the value of mode.

mode

specifies whether the cursor is to be displayed or hidden. This argument can take on either of the following two values.

<code>_MS_HIDE</code>	hide cursor
<code>_MS_SHOW</code>	show cursor

Multiple nested calls can be made to `_ms_cursor` with the value of `_MS_HIDE` to hide the cursor; however, a corresponding number of calls must be made with mode equal to `_MS_SHOW` to reveal the cursor.

**Return value** None.

### **Example**

```
#include <mouse.h>
#include <conio.h>
#include <limits.h>
main(){
    _ms_reset();
    _ms_cursor(_MS_SHOW);
    getch();
    _ms_cursor(_MS_HIDE);
    return(0);
}
```

## **unsigned \_ms\_driversize(void);**

---

<b>Header file</b>	mouse.h
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Returns the number of bytes required to store the current state of the mouse driver.

### **Example**

See `_ms_savedriver`.

## **void \_ms\_getmotion(struct \_ms\_motion \*mp);**

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getmotion` copies the number of mickeys moved horizontally and vertically since the last call to `_ms_motion`.

**mp** points to an `_ms_motion` structure which contains information about the mouse movement. This structure is defined in `mouse.h`

```
struct _ms_motion {  
    /* number of mickeys moved vertically */  
    int vert;  
    /* number of mickeys moved horizontally*/  
    int horiz; };
```

### **Return value**

The function returns the number of bytes required to store the current state of the mouse driver.

### **Example**

```
#define _JPI_WIN_  
#include <stdlib.h>  
#include <mouse.h>  
#include <conio.h>  
  
main()  
{  
    struct _ms_motion mp;  
  
    _ms_reset();  
    _ms_setposition(0, 0);  
    _ms_cursor(_MS_SHOW);  
    clrscr();  
    while(!kbhit())  
    {  
        _ms_getmotion(&mp);  
        gotoxy(10, 10);  
        cprintf(  
            "Rows moved %4d Cols moved %4d",  
            mp.vert, mp.horiz);  
        delay(200);  
    }  
    getch();  
    clrscr();  
    return(0);  
}
```

## **int \_ms\_getpage(void);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getpage` returns the display page on which the cursor appears.

### **Return value**

The function returns the display page.

### **Example**

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

main()
{
    int current_page;

    current_page=_ms_getpage();
    printf("Cursor is on page %d\n",
        current_page);
    return(0);
}
```

## **void \_ms\_getpress(int button, struct \_ms\_data \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getpress` reads the mouse position and button press information for the specified button and copies the data to the `_ms_data` structure to which `mp` points.

<code>button</code>	specifies which button has been pressed. Takes the following values:
<code>_MS_MIDDLE</code>	middle button.
<code>_MS_RIGHT</code>	right button.
<code>_MS_LEFT</code>	left button.
<code>mp</code>	points to a <code>_ms_data</code> structure which will contain the information about the mouse's status. This structure is defined in <code>mouse.h</code> :

```
struct _ms_data {
    /* non-zero if left button pressed */
    int left_pressed;
    /* non-zero if middle button pressed */
    int middle_pressed;
    /* non-zero if right button pressed */
    int right_pressed;
    /* number of presses on specified */
    /* button since last call */
    int actions;
    /* cursor row at last press */
    int row;
    /* cursor column at last press */
    int col;
};
```

**Return value** None.

### **Example**

```
#define _JPI_WIN_
#include <mouse.h>
#include <conio.h>
```

```
main()
{
    struct _ms_data mp;
    _ms_reset();
    _ms_setposition(0, 0);
    clrscr();
    cursoroff();
    while(!kbhit())
    {
        _ms_getpress(_MS_RIGHT, &mp);
        gotoxy(10, 10);
        cprintf("Right button pressed");
        cprintf(
            "  %d times at Row %4d Col %4d",
            mp.actions, mp.row, mp.col);

        if(mp.left_pressed){
            gotoxy(10, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(10, 8);
            putchar(' ');
        }
        if(mp.middle_pressed){
            gotoxy(14, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(14, 8);
            putchar(' ');
        }
        if(mp.right_pressed){
            gotoxy(18, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(18, 8);
            putchar(' ');
        }
    }
    getch();
    clrscr();
    cursoron();
    return(0);
}
```

## **void \_ms\_getrelease(int button, struct \_ms\_data \*mp);**

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getrelease` reads the mouse position and button release information for the specified button, and copies the data to the structure to which `mp` points.

`button` specifies which button has been released. This argument can take the following values:

<code>_MS_MIDDLE</code>	middle button.
<code>_MS_RIGHT</code>	right button.
<code>_MS_LEFT</code>	left button.

`mp` points to a `_ms_data` structure which will contain the information about the mouse's status. This structure is defined in `mouse.h`:

```
struct _ms_data {
    /* non-zero if left button pressed */
    int left_pressed;
    /* non-zero if middle button pressed */
    int middle_pressed;
    /* non-zero if right button pressed */
    int right_pressed;
    /* number of presses on specified */
    /* button since last call */
    int actions;
    /* cursor row at last press */
    int row;
    /* cursor column at last press */
    int col;
};
```

**Return value** None.

### **Example**

```
#define _JPI_WIN_
#include <mouse.h>
#include <conio.h>
```

```
main(){
    struct _ms_data mp;
    _ms_reset();
    _ms_setposition(0, 0);
    clrscr();
    cursoroff();
    while(!kbhit()){
        _ms_getrelease(_MS_LEFT, &mp);
        gotoxy(10, 10);
        cprintf(
            "Right button released %d ",
            cprintf("times at: Row %d Col %4d",
                mp.actions, mp.row, mp.col));
        if(mp.left_pressed){
            gotoxy(10, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(10, 8);
            putchar(' ');
        }
        if(mp.middle_pressed){
            gotoxy(14, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(14, 8);
            putchar(' ');
        }
        if(mp.right_pressed){
            gotoxy(18, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(18, 8);
            putchar(' ');
        }
    }
    getch();
    clrscr();
    cursoron();
    return(0);
}
```



---

**void \_ms\_getsensitivity(struct \_ms\_sense \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getsensitivity` copies the current values for speed and double speed threshold to the structure to which `mp` points.

`mp` points to an `_ms_sense` structure which contains information about the mouse speed. This structure is defined in `mouse.h`

```
struct _ms_sense {
    int h_speed; /* horizontal speed */
    int v_speed; /* vertical speed */
    int threshold; /* double speed threshold */
};
```

**Return value**

None.

**Example**

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

main()
{
    struct _ms_sense mp;
        /* get default values */
    _ms_getsensitivity(&mp);
    printf("Vertical speed: %d, ", mp.v_speed);
    printf("Horizontal speed %d, ", mp.h_speed);
    printf("threshold %d\n", mp.threshold);
    return(0);
}
```

## **void \_ms\_getstatus(struct \_ms\_data \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_getstatus` reads the mouse position and button status, and copies the data to the `_ms_data` structure to which `mp` points. This structure is defined in `mouse.h`.

```
struct _ms_data {  
    /* non-zero if left button pressed */  
    int left_pressed;  
    /* non-zero if middle button pressed */  
    int middle_pressed;  
    /* non-zero if right button pressed */  
    int right_pressed;  
    /* number of presses on specified */  
    /* button since last call */  
    int actions;  
    /* cursor row at last press */  
    int row;  
    /* cursor column at last press */  
    int col;  
};
```

**Return value** None.

### **Example**

```
#define _JPI_WIN_  
#include <mouse.h>  
#include <conio.h>
```

```
main()
{
    struct _ms_data mp;
    _ms_reset();
    _ms_setposition(0, 0);
    clrscr();
    cursoroff();
    while(!kbhit())
    {
        _ms_getstatus(&mp);
        gotoxy(10, 10);
        printf("Row %4d Col %4d", mp.row, mp.col);
        if(mp.left_pressed)
        {
            gotoxy(10, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(10, 8);
            putchar(' ');
        }
        if(mp.middle_pressed)
        {
            gotoxy(14, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(14, 8);
            putchar(' ');
        }
        if(mp.right_pressed)
        {
            gotoxy(18, 8);
            putchar('ý');
        }
        else
        {
            gotoxy(18, 8);
            putchar(' ');
        }
    }
    getch();
    clrscr();
    cursoron();
    return(0);
}
```

## **void \_ms\_lightpen(int mode);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_lightpen` turns light pen emulation on or off, depending on the value of `mode`. To turn the light pen off, use `_MS_OFF`; to turn the light pen on, use `_MS_ON`.

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()

{
    _ms_reset();
    _ms_lightpen(_MS_ON);
    return(0);
}
```

## **int \_ms\_reset(void);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Resets the mouse driver and reads the driver status.

### **Return value**

If the driver or hardware are not installed, the value INT\_MAX is returned.  
If the driver is installed, returns the number of mouse buttons:

-1	Two buttons.
0	Other than two buttons.
3	Mouse Systems Mouse.

### **Example**

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

main() {
    int mouse_status;
    char *msg;

    mouse_status=_ms_reset();
    switch(mouse_status) {
        case -1 :
            msg="2 button mouse installed";
            break;
        case 0 :
            msg="3? button mouse installed";
            break;
        case 3 :
            msg="Mouse Systems mouse installed";
            break;
        case INT_MAX :
            msg="Mouse driver NOT installed";
            break;
    }
    printf("%s\n", msg);
    return(0);
}
```

---

**void \_ms\_restoredriver(void \*buffer);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Function `_ms_restoredriver` restores the mouse driver state, stored in the memory location to which `buffer` points through a previous call to `_ms_savedriver`.

**Example**

See `_ms_savedriver`.

---

**void \_ms\_savedriver(void \*buffer);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Function `_ms_savedriver` stores the current state of the mouse driver in the memory location to which `buffer` points. The buffer size should be determined by a call to `_ms_driversize`.

**Return value** None.

**Example**

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

main() {
    unsigned char *mouse_buffer;
    unsigned mouse_buffer_size;
    mouse_buffer_size=_ms_driversize();
    mouse_buffer=malloc(mouse_buffer_size);
    if(mouse_buffer)
    {
        _ms_savedriver(mouse_buffer);
        rest_of_program();
        _ms_restoredriver(mouse_buffer);
    }
    return(0);
}
```

## **void \_ms\_setdouble(int threshold);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setdouble` defines the double speed threshold in mickeys per second. The double speed threshold is the mouse speed where the on-screen motion is doubled. The default value is 64 mickeys per second.

The argument `threshold` is the speed in mickeys per second. A value of 0 selects the default.

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()
{
    _ms_setdouble(128); /* set to new value */
    _ms_cursor(_MS_SHOW);
    _ms_setdouble(64); /* set to default */
    return(0);
}
```

## **void \_ms\_setgraphcursor(struct \_ms\_graphcur \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

Defines a graphics cursor specified by the structure to which mp points.

mp points to an \_ms\_graphcur structure which contains information about the mouse graphics cursor. This structure is defined in mouse.h.

```
struct _ms_graphcur {
    int row;
    int col;
    unsigned int screen_mask[16];
    unsigned int cursor_mask[16];
};
```

The row member specifies the cursor hot-spot row (16 to -16); the col member specifies the cursor hot-spot column (16 to -16). The screen\_mask member is an array of 16 pixel rows, defining a mask that will be *ANDed* with the screen at the current cursor position. The cursor\_mask member is an array of 16 pixel rows, defining a mask that will be *ORed* with the screen at the current cursor position.

**Return value** None

### **Example**

```
#include <stdio.h>
#include <mouse.h>
/* hot spot row and column */
struct _ms_graphcur mp={ 0, 0,
/* screen mask */
{ 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
/* cursor mask */
{ 0xFFFF, 0x8001, 0x8001, 0x8001,
0x8001, 0x8001, 0x8001, 0x8001,
0x8001, 0x8001, 0x8001, 0x8001,
0x8001, 0x8001, 0x8001, 0xFFFF
};
main() {
    _ms_setgraphcursor(&mp);
    return(0);
}
```



## **void \_ms\_setinterrupt(struct \_ms\_interrupt \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setinterrupt` sets the call mask to the value of structure `mp` member `mask`, and installs the function address in structure `mp` member `func` to service mouse hardware interrupts.

```
struct _ms_interrupt {  
    void (far * func)(void);  
    unsigned mask;  
};
```

### **Return value**

None.

### **Example**

```
#include <mouse.h>  
  
extern void far handler(void);  
  
void set_mouse_interrupt(void)  
{  
    struct _ms_interrupt mi;  
  
    mi.mask=8; /* mask for right button press */  
    mi.func=handler;  
    _ms_setinterrupt(&mi);  
    return;  
}
```

## **void \_ms\_setmickeys(int vert, int horiz);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setmickeys` defines the mickey per pixel ratio, both vertically and horizontally. The default values are 8 horizontally and 16 vertically.

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()

{
    _ms_setmickeys(32, 32);
    _ms_cursor(_MS_SHOW);
    return(0);
}
```

## **void \_ms\_setpage(int page);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setpage` sets the display page on which the cursor appears to the value specified by `page`.

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()

{
    _ms_setpage(1);
    return(0);
}
```

## **void \_ms\_setposition(int row, int col);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setposition` sets the position of the mouse cursor. If the coordinates exceed the cell size, they are truncated.

To ensure that the mouse cursor's position is updated on the screen, turn off the cursor before setting the position and restore the cursor in its new position.

### **Return value**

None.

### **Example**

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

main()
{
    _ms_reset();
    _ms_cursor(_MS_SHOW);
    getch();
    _ms_cursor(_MS_HIDE);
    _ms_setposition(0, 0);
    _ms_cursor(_MS_SHOW);
    getch();
    _ms_cursor(_MS_HIDE);
    return(0);
}
```

## **void \_ms\_setrange(struct \_ms\_range \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setrange` defines the horizontal and vertical cursor range as specified by the structure to which `mp` points.

`mp` points to an `_ms_range` structure which contains information about the range of values in the horizontal and vertical positions for the mouse. This structure is defined in `mouse.h`

```
struct _ms_range {
    int max_col; /* maximum horizontal range */
    int min_col; /* minimum horizontal range */
    int max_row; /* maximum vertical range */
    int min_row; /* minimum vertical range */
};
```

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()
{
    struct _ms_range mp;

    _ms_reset();
    _ms_setposition(0, 0);
    mp.max_col=100; /* set cursor range */
    mp.min_col=50;
    mp.max_row=80;
    mp.min_row=10;
    _ms_setrange(&mp);
    return(0);
}
```

---

**void \_ms\_setsensitivity(struct \_ms\_sense \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_setsensitivity` sets the current values for speed and double speed threshold to those in the structure to which `mp` points.

`mp` points to an `_ms_sense` structure which contains information about the mouse speed. This structure is defined in `mouse.h`

```
struct _ms_sense {
    int h_speed; /* horizontal speed */
    int v_speed; /* vertical speed */
    int threshold; /* double speed threshold */
};
```

**Return value**

None.

**Example**

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

main()
{
    struct _ms_sense mp;
        /* get default values */
    _ms_getsensitivity(&mp);
    printf("Vertical speed: %d, ", mp.v_speed);
    printf("Horizontal speed %d, ", mp.h_speed);
    printf("threshold %d\n", mp.threshold);
    mp.v_speed=128;
    mp.h_speed=256;
    mp.threshold=1024;
    _ms_setsensitivity(&mp); /* set new values */
    _ms_getsensitivity(&mp); /* get new values */
    printf("Vertical speed: %d, ", mp.v_speed);
    printf("Horizontal speed %d, ", mp.h_speed);
    printf("threshold %d\n", mp.threshold);
    return(0);
}
```

---

**void \_ms\_settextcursor(struct \_ms\_textcur \*mp);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed MSDOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_settextcursor` defines a text mode cursor as specified by the contents of the structure to which `mp` points.

`mp` points to an `_ms_textcur` structure which contains information about the text mode cursor. This structure is defined in `mouse.h`

```
struct _ms_textcur {
    int type;
    union _ms_type c;
};
```

The structure member `type` is initialized to the type of cursor required. If the cursor type is specified by software, the constant `_MS_SOFT` is used; if the cursor type is specified by hardware, the constant `_MS_HARD` must be used.

If a hardware cursor is required, union member `h` is initialized with the starting and ending scan lines for the cursor. If a software cursor is required, union member `s` is initialized with the desired screen and cursor masks. The character and attribute data at the current screen position will be *ANDed* with the screen mask and then *XORed* with the cursor mask. These data structures are defined in `mouse.h`:

```
struct _hardware {
    int start_scan;
    int end_scan;
};
struct _software {
    unsigned screen_mask;
    unsigned cursor_mask;
};
union _ms_type {
    struct _hardware h;
    struct _software s;
};
```

**Return value**

None.

**Example**

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

main()
{
    struct _ms_textcur mp;

    _ms_reset();
    _ms_setposition(0, 0);
    /* make hardware block cursor */
    mp.type=_MS_HARD;
    mp.c.h.start_scan=1;
    mp.c.h.end_scan=8;
    _ms_settextcursor(&mp);
    _ms_cursor(_MS_SHOW);
    getch();
    return(0);
}
```



## **`void _ms_swapinterrupt(struct _ms_interrupt *mp);`**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_swapinterrupt` sets the call mask to the value of structure `mp` member `mask`, and installs the function address in structure `mp` member `func`, to service mouse hardware interrupts. The structure members are then loaded with the previous values for the mask and interrupt handler.

```
struct _ms_interrupt {  
    void (far * func)(void);  
    unsigned mask;  
};
```

### **Return value**

None.

### **Example**

```
#include <mouse.h>  
  
extern void far handler(void);  
  
unsigned swap_mouse_interrupt(void)  
{  
    struct _ms_interrupt mi;  
  
    mi.mask=8; /* mask for right button press */  
    mi.func=handler;  
    _ms_setinterrupt(&mi);  
    return(mi.mask); /* return old mask setting */  
}
```

## **void \_ms\_updatescreen(int x1, int y1, int x2, int y2);**

---

<b>Header file</b>	mouse.h
<b>See also</b>	all _ms_* functions.
<b>Portability</b>	TopSpeed DOS
<b>Multi-thread</b>	Not re-entrant

The function `_ms_updatescreen` defines a region of the screen for updating. this region is bounded by the rectangle with upper left coordinates `x1, y1` and lower right coordinates `x2, y2`.

When the screen is updated, the mouse cursor is hidden and must be explicitly restored with a call to `_ms_cursor`.

### **Return value**

None.

### **Example**

```
#include <mouse.h>

main()
{
    _ms_updatescreen(10, 10, 400, 90);
    _ms_cursor(_MS_SHOW);
    return(0);
}

void near *_ncalloc(size_t sizes, size_t nmem);
Near heap version of calloc.
unsigned nearcoreleft(void);
Near heap version of coreleft.
void near *_nexpand(void near *buffer, size_t size);
Near heap version of _expand.
void _nfree(void near *buffer);
Near heap version of free.
int _nheapchk(void);
Near heap version of heapchk.
int _nheapset(int ch);
Near heap version of heapset.
int _nheapwalk(struct _nheapinfo *entry);
Near heap version of heapwalk.
void near *_nmalloc(size_t size);
Near heap version of malloc.
int _nmsize(void near *buffer);
Near heap version of msize.
```

**void normvideo(void);****T**

<b>Header file</b>	conio.h
<b>See also</b>	highvideo, lowvideo
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function normvideo from the Turbo C window module sets the active screen attribute to the default value. The screen attribute is used by the console I/O functions.

**Return value**

None.

**Example**

```
#define _CLIP_WIN_  
#include <conio.h>  
  
main()  
{  
    highvideo();  
    cprintf("This is displayed with ");  
    cprintf("high intensity attributes\n");  
    normvideo();  
    cprintf("This is displayed with ");  
    cprintf("normal intensity attributes");  
    return(0);  
}
```

## **void nosound(void);**

---

<b>Header file</b>	dos.h
<b>See also</b>	delay, sound
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	None.

Turns the hardware speaker off. The sound function turns it on again.

### **Return value**

None.

### **Example**

```
#include <dos.h>
#define INC 500
main()
{
    int freq;
    for (freq = 500; freq <= 5000; freq += INC)
    {
        sound ( freq);
        delay ( 100);
        nosound ();
        delay ( 100);
    }
}
```

## **void Notify(SIGNAL s);**

---

<b>Header file</b>	process.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

Causes a task waiting on signal *s* to be scheduled when *possible*, for example, at the next time-slice. If no process is waiting for *s*, the call has no effect. This call does not cause rescheduling, so an interrupt handler can safely use it to notify another process of an event's occurrence.

### **Return value**

None.

### **Example**

**See example program in , page .**

```
void near * _nrealloc(void near *buffer, size_t size);  
Near pointer variant of realloc.  
char near * _nstrdup(const char far *s);  
Memory variant of strdup. Required to produce near heap copy of far string
```

**int obscuredat(wintype win, relcoord X, relcoord Y);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `obscuredat` from the TopSpeed window module returns a nonzero value if the window is obscured at the position specified by the *relative* coordinates, X and Y.

<b>win</b>	specified the window being examined.
<b>X</b>	specifies the horizontal coordinate for the location.
<b>Y</b>	specifies the vertical coordinate for the location.

**Return value** The function returns a nonzero value if the window is obscured at the specified point, and zero otherwise.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 8, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;

windef WD2={40, 6, 75, 18, Yellow, Red,
  FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
  LightGray, Blue};
wintype W2;

main()

{
  W1 = windowopen(&WD1) ; /* open window */
  setttitle(W1, "Window 1", CenterUpperTitle);
  W2 = windowopen(&WD2) ; /* open window */
  setttitle(W2, "Window 2", CenterUpperTitle);
  cprintf("Hello World sent to window 2\n");
  getch();
  /* put on top if under another window */
  if(obscuredat(W1, 0, 0))
    putontop(W1);
  use(W1);
  cprintf("Window 1 being used\n");
  getch();
  putontop(W2);
  getch();
  windowclose(W1);
  windowclose(W2);
  putontop(_fullscreen);
  return(0);
}
```

---

**void (\* onexit(void (\* func)(void)))();**

---

**Header file**            stdlib.h**See also**                exit, \_exit, atexit.**Portability**           DOS/OS2. onexit does not conform to the ANSI standard, but has been provided for compatibility with other compilers. If portability is required use atexit.**Multi-thread**          Access to the function stack is controlled by semaphore in multi-thread programs. The actual calling order of functions in a multi-thread program will be undefined.

The function onexit places the function pointer func onto a stack to be called from the function exit when the process terminates. Functions are called on a Last in First *out* basis, and cannot take parameters or return any value.

The maximum number of functions accommodated by the onexit stack is 32. Any further calls to onexit will return an error.

**Return value**

onexit returns a pointer to the function if the function was successfully placed on the stack. If the stack was full, a NULL pointer will be returned and errno set to ENOMEM.

**Example**

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

void func1(void);
void func2(void);
void func3(void);

main() {
    onexit(func1); /* push functions onto exit stack */
    onexit(func2);
    if(onexit(func3) != func3)
        printf("exit stack full");
    printf("main\n");
    return(0);
}

void func1(void)
{
    printf("func1\n");
    return;
}
void func2(void)
{
    printf("func2\n");
    return;
}
void func3(void)
{
    printf("func3\n");
    return;
}

/* Output is :
   main
   func3
   func2
   func1 */
}
```



## **int open(const char \*path, int access [, permission]);**

**U**

**Header file** io.h plus types in fcntl.h and stat.h

**See also** access, chmod, close, creat, dup, dup2, fopen, sopen, umask

**Portability** UNIX/DOS/OS2.

**Multi-thread** Low-level I/O is not protected by semaphore.

**OS2** The default share mode is SH\_DENYNO.

The function open opens a file specified by path and prepares the file for reading and/or writing, as specified with access.

access is an integer composed of one or more of the following constants defined in fcntl.h:

Value	Meaning
O_APPEND	Write only at end of file.
O_BINARY	Open file in untranslated mode.
O_CREAT	Create file if it does not exist. Has no effect if path exists.
O_EXCL	Return error if file exists. This is only valid with O_CREAT.
O_RDONLY	Open file for reading only.
O_RDWR	Open for reading and writing.
O_TEXT	Open in translated mode. (See I/O module for description of translated mode.)
O_TRUNC	Open and truncate existing file to zero length. Creates nonexistent file. Existing file must have write permission.
O_WRONLY	Open file for writing only.

If more than one constant is to be specified they should be joined with a bitwise OR operator (|).

Either O\_RDONLY, O\_RDWR or O\_WRONLY must be given; there is no default.

If neither O\_TEXT or O\_BINARY are given, the translation mode defaults to that in \_fmode.

permission is required only if O\_CREAT is specified. If the file exists, the value is ignored. The file permission settings are only set when the file is closed for the first time.

ENOENT

Pathname not found.

**Example**

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stat.h>
#include <string.h>

main()
{
    int fh;
    char *msg="THIS WAS WRITTEN TO FILE";

    fh=open(
        "TEMP.***",
        O_RDWR|O_CREAT|O_TRUNC,
        S_IREAD|S_IWRITE);
        /* check file was created */
        /* or truncated */
    if(fh < 0)
        abort();
    write(fh, msg, strlen(msg));
    close(fh);
    return(0);
}
```

## **int \_open(const char \*path, int access);**

---

<b>Header file</b>	io.h
<b>See also</b>	_read
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_open` opens a file in a mode specified by `access`.

<code>path</code>	specifies the file being opened.
<code>access</code>	specifies the mode in which the file can be used. This parameter can have the following values, which are defined in <code>fcntl.h</code> :

<b>Value</b>	<b>Meaning</b>
<code>O_RDONLY</code>	the file can only be read.
<code>O_WRONLY</code>	the file can only be written.
<code>O_RDWR</code>	the file can be read and written.
<code>O_NOINHERIT</code>	the file cannot be passed to child programs. (DOS 3.0 and later only.)
<code>O_DENYALL</code>	only the current handle can be used to access the file. (DOS 3.0 and later only.)
<code>O_DENYWRITE</code>	the file can only be read if opened by another function/process. (DOS 3.0 and later only.)
<code>O_DENYREAD</code>	the file can only be written if opened by another function/process. (DOS 3.0 and later only.)

---

**void outp(int port, unsigned char ch);**

---

---

**void outportb(int port, unsigned char ch);**

---

**Header file** conio.h and dos.h

**See also** inp, inportb.

**Portability** 8086 Family

**Multi-thread** None.

The functions outp and outportb write the byte ch to the specified port. The value for port can be any integer value from -32768 to 32767.

### **Return value**

None

### **Example**

```
#include <dos.h>

void _resetEGA(void)
{
    outportb( 0x3CE, 2); /* write byte value 2 */
                          /* to port 0x3CE */
    outportb( 0x3CF, 0xF);
    outportb( 0x3CE, 1);
    outportb( 0x3CF, 0);
    outportb( 0x3CE, 2);
    outportb( 0x3CF, 0);
    outportb( 0x3CE, 3);
    outportb( 0x3CF, 0);
    outportb( 0x3CE, 8);
    outportb( 0x3CF, 0xFF);
    outportb( 0x3CE, 7);
    outportb( 0x3CF, 0xF);
    outportb( 0x3CE, 5);
    outportb( 0x3CF, 0);
    return;
}
```

---

**void outpw(int port, unsigned ch);**

---

---

**void outportw(int port, unsigned ch);**

---

**Header file** conio.h and dos.h**See also** inpw, inportw**Portability** 80x86 Family**Multi-thread** None.

The functions outpw and outportw write the word ch to the specified port. The value for port can be any integer value from -32768 to 32767.

**Return value**

None

**Example**

```
#include <dos.h>

void _resetEGA(void)
{
    outpw( 0x3CE, 2);/* write word value 2 */
                      /* to port 0x3CE */
    outpw( 0x3CF, 0xF);
    outpw( 0x3CE, 1);
    outpw( 0x3CF, 0);
    outpw( 0x3CE, 2);
    outpw( 0x3CF, 0);
    outpw( 0x3CE, 3);
    outpw( 0x3CF, 0);
    outpw( 0x3CE, 8);
    outpw( 0x3CF, 0xFF);
    outpw( 0x3CE, 7);
    outpw( 0x3CF, 0xF);
    outpw( 0x3CE, 5);
    outpw( 0x3CF, 0);
    return;
}
```

---

**void far \_outtext(const char far \*str);**

---

**M****Header file** graph.h**See also** \_setactivepage, \_settextposition**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_outtext` from the graphics module outputs the specified string to the graphics screen. The text output begins at the current text position on the screen and is not formatted (in contrast to functions such as `printf`).

The function updates the current text position.

`str` specifies the null-terminated string to be output.

This is not the only method of outputting to the graphics screen. For example, functions such as `cprintf` also can be used.

**Return value**

None.

**Example**

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

main()

{
    _setvideomode(_ERESCOLOR);
    _outtext("Hello World\n");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **int palettcolor(void);**

---

**W**

**Header file**            window.h

**Portability**            TopSpeed DOS/OS2

**Multi-thread**          Module is re-entrant and requires no additional locking for single function calls.

The function `palettcolor` from the TopSpeed window module returns the palette color set being used in the current window.

### **Return value**

The function returns the current palette color set.

### **Example**

See example program in , page .

## int palettectorused(wintype win, int pc);

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `palettectorused` from the TopSpeed window module checks whether the specified color has been used in the specified window.

### Return value

The function returns a nonzero value if the color is in use anywhere in the specified window. The function returns 0 if the window is not a palette window.

### Example

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={
    10, 2, 50, 8, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

PaletteDef pal;

main()

{
    clrscr();
    pal[NORMALPALETTECOLOR].Fore=Red;
    pal[NORMALPALETTECOLOR].Back=Blue;
    pal[FRAMEPALETTECOLOR].Fore=Green;
    pal[FRAMEPALETTECOLOR].Back=Brown;
    pal[2].Fore=LightGray;
    pal[2].Back=Red;
    pal[3].Fore=Yellow;
    pal[3].Back=Blue;
    /* open window */
    W1 = paletteopen(&WD1, pal) ;
    setttitle(W1, "Window 1", CenterUpperTitle);
    cprintf(
        "Current palette color is %d\n",
        palettectorused());
    getch();
    setpalettecolor(2);
    cprintf("Palette ");
    if(palettectorused(W1, 2))
        cprintf("color 2 used");
    getch();
    return(0);
}
```



**wintype paletteopen(windef \*WD, PaletteDef Pal);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `paletteopen` from the TopSpeed window module creates a new palette window. This window and its palette information are specified by `WD` and `Pal`.

`WD` points to the window whose palette information is to be modified.

`Pal` the palette information that will be used for the window

The window is cleared to the colors given `bypal[NORMALPALETTECOLOR]` and the frame is drawn in the colors given by `Pal[FRAMEPALETTECOLOR]`. Finally, the window is put on view on top of any existing windows. The current palette color is given by `NORMALPALETTECOLOR`.

**Return value**

The function returns a window handle.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 50, 8, White, Brown,
FALSE, FALSE, FALSE, TRUE,SINGLEFRAME,
Red, LightGray};
wintype W1;
PaletteDef pal;

main() {
    clrscr();
        /* initialize palette */
    pal[NORMALPALETTECOLOR].Fore=Red;
    pal[NORMALPALETTECOLOR].Back=Blue;
    pal[FRAMEPALETTECOLOR].Fore=Green;
    pal[FRAMEPALETTECOLOR].Back=Brown;
        /* open window */
    W1 = paletteopen(&WD1, pal) ;
    setttitle(W1, "Window 1", CenterUpperTitle);
    getch();
    return(0);
}
```

## **char \* parsfnm(const char \*cmdline, struct fcb \*fcb, int opt);**

---

<b>Header file</b>	dos.h
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `parsfnm` parses a string to get a file name, and stores this information in a file control block (FCB).

<code>cmdline</code>	specifies a string which contains the file name to be parsed.
<code>fcb</code>	points to an fcb structure, which will be used to store information about the file. Such a structure is defined in <code>dos.h</code> as follows:

```
struct fcb {
    char fcb_drive; /* 0 = default, 1 = A, 2 = B */
    char fcb_name[8]; /* File name */
    char fcb_ext[3]; /* File extension */
    short fcb_curblk; /* Current block number */
    /* Logical record size in bytes */
    short fcb_recsize;
    long fcb_filsize; /* File size in bytes */
    /* Date file was last written */
    short fcb_date;
    char fcb_resv[10]; /* Reserved for DOS */
    /* Current record in block */
    char fcb_currec;
    long fcb_random; /* Random record number */
};
```

<code>opt</code>	specifies the value of AL in the DOS system call 29H, which is used for parsing file names.
------------------	---

### **Return value**

If successful, the function returns a pointer to the byte *after* the file name in the `cmdline` string. In case of an error, the function returns NULL.

### **Example**

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

char *msg="Hello World\n\r";

main()
{
    struct fcb fbuf;

    parsfnm("ctest.c", &fbuf, 1);
    return(0);
}
```

## **int peek(unsigned segment, unsigned offset);**

---

<b>Header file</b>	dos.h
<b>See also</b>	harderr, peekb, poke
<b>Portability</b>	80x86 family.
<b>Multi-thread</b>	None.

The peek routine returns the *word* stored at the memory location specified by segment:offset.

By default, the function version of peek is called. A macro version is also available. To use the macro version, define the macro `_PTM` before including dos.h.

### **Return value**

The function returns the value stored at the specified location.

## **char peekb(unsigned segment, unsigned offset);**

---

<b>Header file</b>	dos.h
<b>See also</b>	peek, poke
<b>Portability</b>	80x86 family.
<b>Multi-thread</b>	None.

The peekb routine returns the *byte* stored at the memory location specified by segment:offset.

By default, the function version of peekb is called. A macro version is also available. To use the macro version, define the macro `_PTM` before including dos.h.

### **Return value**

The function returns the byte value stored at the specified location.

---

**void perror(const char \*message);**

---

**A****Header file**           stdio.h**See also**           clearerr, ferror, strerror, \_strerror.**Portability**       ANSI**Multi-thread**      None.

The function `perror` prints an error message to `stderr`. The argument, `message`, is printed first, followed by a colon. This is followed by the system defined error message for the last error that occurred, plus a newline character. If `message` is a `NULL` pointer or points to a null string, the function only prints the system error message.

The error number is stored in `errno` which is declared in `errno.h`.

To ensure that the correct results are obtained, `perror` should be called immediately after an error has occurred Æ since any subsequent error will overwrite `errno`.

**Return value**

None.

**Example**

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

main()
{
    int fh;

    fh=open("TEMP.$$$", O_RDWR|O_BINARY);
    if(fh < 0)
    {
        /* print error message */
        perror("Error opening file");
        abort();
    }
    close(fh);
    return(0);
}
```

```

short far _pie(
    short fill,
    short x1, short y1,
    short x2, short y2,
    short x3, short y3,
    short x4, short y4);

```

---

**M**

**Header file** graph.h

**See also** \_arc, \_ellipse, \_getcolor, \_lineto, \_pie, \_rectangle, \_setcolor, \_setfillmask

**Portability** DOS.

**Multi-thread** Function is not re-entrant.

Draws a wedge-shaped figure, using the current color. The figure is drawn within a bounding rectangle specified by the parameters.

**fill** specifies how the wedge is to be drawn. This parameter can have either of the following two values:

**\_GFILLINTERIOR**

fill the figure using the current color and fill mask.

**\_GBORDER**

do not fill the figure.

**x1,y1**

coordinates of the upper left point of the rectangle bounding the pie wedge.

**x2,y2fcb**

coordinates of the lower right point of the rectangle bounding the pie wedge.

**x3,y3**

coordinates of a vector used to determine the starting point for the arc.

**x4,y4**

coordinates of a vector used to determine the ending point for the arc.

When the arc is drawn, it begins at the point at which the arc intersects the vector specified by x3, y3, and ends at the point at which the arc intersects the vector specified by x4, y4. When the arc is drawn, it is drawn in a counterclockwise direction to the ending point.

**Note:** Coordinates for the points specified are for logical points.

**Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

**Example**

```
#include <graph.h>
#include <conio.h>
main(){
    struct videoconfig v;
    short X1, Y1, X2, Y2, X3, Y3, X4, Y4;
    struct xycoord arc_array[2][4];
    int n;
    _setvideomode(_MRESNOCOLOR);
    _getvideoconfig(&v);

    X1=40;
    Y1=20;
    X2=v.numpixels-40;
    Y2=v.numpixels-20;
    arc_array[0][0].xcoord=v.numpixels-1;
    arc_array[0][0].ycoord=0;
    arc_array[1][0].xcoord=v.numpixels-1;
    arc_array[1][0].ycoord=v.numpixels-1;

    arc_array[0][1].xcoord=v.numpixels-1;
    arc_array[0][1].ycoord=v.numpixels-1;
    arc_array[1][1].xcoord=0;
    arc_array[1][1].ycoord=v.numpixels-1;

    arc_array[0][2].xcoord=0;
    arc_array[0][2].ycoord=v.numpixels-1;
    arc_array[1][2].xcoord=0;
    arc_array[1][2].ycoord=0;
    arc_array[0][3].xcoord=0;
    arc_array[0][3].ycoord=0;
    arc_array[1][3].xcoord=v.numpixels-1;
    arc_array[1][3].ycoord=0;
    n=0;

    while(n < 4) {
        X3=arc_array[0][n].xcoord;
        Y3=arc_array[0][n].ycoord;
        X4=arc_array[1][n].xcoord;
        Y4=arc_array[1][n].ycoord;
        _pie(_GBORDER,
            X1, Y1, X2, Y2, X3, Y3, X4, Y4);
        getch();
        _pie(_G_FILLINTERIOR,
            X1, Y1, X2, Y2, X3, Y3, X4, Y4);
        getch();
        _clearscreen(_GCLEARSCREEN);
        ++n;
    } _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **void poke(unsigned segment, unsigned offset, int val);**

---

<b>Header file</b>	dos.h
<b>See also</b>	harderr, peek, pokeb
<b>Portability</b>	80x86 family.
<b>Multi-thread</b>	None.

The poke routine stores the word-value val at the memory location specified by segment:offset.

By default, the function version of poke is called. A macro version is also available. To use the macro version, define the macro `_PTM` before including dos.h.

### **Return value**

None.

## **void pokeb(unsigned segment, unsigned offset, char val);**

---

<b>Header file</b>	dos.h
<b>See also</b>	peekb, poke
<b>Portability</b>	80x86 family.
<b>Multi-thread</b>	None.

The pokeb routine stores the byte-value val at the memory location specified by segment:offset.

By default, the function version of pokeb is called. A macro version is also available. To use the macro version, define the macro `_PTM` before including dos.h.

### **Return value**

None.

```
double poly(
    double indep,
    int degree,
    double coeff[]);
```

---

```
long double polyl(
    long double indep,
    int degree,
    long double coeff[]);
```

**Header file**            math.h

**Portability**            UNIX/DOS/OS2

**Multi-thread**          None.

The function `poly` generates a polynomial of the specified degree.

`indep`                    value for which the polynomial will be computed.

`degree`                  degree of the polynomial  $\mathcal{A}$  that is, the highest power to which `indep` will be raised.

`coeff`                   stores the coefficients for the polynomial.

For example, for `degree = 3`, a polynomial such as the following would be generated:

```
coeff[3]indep3 + coeff[2]indep2 +
coeff[1]indep + coeff[0]
```

### Return value

The function returns the value resulting when the polynomial is evaluated.

### Example

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

double coeff[6] = { 1.0, 1.2, 1.44, 1.728, 2.074, 2.488;

main() {
    double indep, result;
    int degree;

    printf ("Degree (highest power) ");
    printf ("of polynomial? ");
    scanf ( "%d", &degree);
    printf ( "Value for variable? ");
    scanf ( "%lf", &indep);
    printf (
        "degree = %d; variable = %lf\n",
        degree, indep);
    result = poly ( indep, degree, coeff);
    printf ( "Result = %lf\n", result);
}
```



```
short far _polygon(  
    short fill,  
    short n,  
    short far px[],  
    short far py[]);
```

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_arc, _cube, _ellipse, _rectangle
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_polygon` from the graphics module draws a polygon, as defined by the coordinates of its corners. The function uses the current color.

<code>fill</code>	specifies whether to fill the figure. This argument may be one of the following constants:
<code>_GFillInterior</code>	fill shape with current fill mask.
<code>_GBorder</code>	do not fill the shape.
<code>n</code>	specifies the number of edges in the polygon.
<code>px</code>	specifies an array of x coordinates for the corners of the polygon.
<code>py</code>	specifies an array of y coordinates for the corners of the polygon.

**Return value** The function returns a nonzero value if the shape has been drawn successfully; otherwise the function returns 0.

### Example

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

main()

{
    unsigned char new_mask[8];
    short px[]={ 0, 200, 180, 120};
    short py[]={ 20, 40, 120, 120};

    _setvideomode(_ERESCOLOR);
    _clearscreen(_GCLEARSCREEN);

    new_mask[0]=0x88;
    new_mask[1]=0x44;
    new_mask[2]=0x22;
    new_mask[3]=0x11;
    new_mask[4]=0x88;
    new_mask[5]=0x44;
    new_mask[6]=0x22;
    new_mask[7]=0x11;
    _setfillmask(new_mask);
    _polygon(_GFILLINTERIOR, 4, px, py);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**double pow(double x, double y);**

---

**A**

---

**long double powl(long double x, long double y)**

---

<b>Header file</b>	math.h
<b>See also</b>	exp, log, pow10, sqrt.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function pow raises its double precision floating point argument x to the power y. The function powl is the same as pow but takes a long double argument.

**Return value**

The function returns  $x^y$ . If x is negative and y is not integral, or if x = 0 and y <= 0 errno is set to EDOM and 0 is returned.

**Example**

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

main()
{
    double x=3.14;
    int y=1;
    double power_of_pi;

    while(y < 11.0)
    {
        power_of_pi=pow(x, (double) y);
        printf("pi raised to the power");
        printf("%2d is %20.10g\n", y, power_of_pi);
        ++y;
    }
    return(0);
}
```

## **double pow10(int exp);**

---

<b>Header file</b>	math.h
<b>See also</b>	exp, log, pow, sqrt
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function pow10 returns 10 raised to the power p.

### **Return value**

This function returns the value of  $10^p$ .

### **Example**

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

main()
{
    int y=1;
    double power_of_10;

    while(y < 11.0)
    {
        power_of_10=pow10((double) y);
        printf("10 raised to the power ");
        printf("%2d is %20.10g\n", y, power_of_10);
        ++y;
    }
    return(0);
}
```

## **int printf(const char \*format[, argument] ...);**

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	cprintf, fprintf, sprintf, vprintf, vfprintf, vsprintf.
<b>Portability</b>	ANSI. Prefixes F and N are not part of the ANSI definition for printf, but are extensions provided for compatibility with other compilers.
<b>Multi-thread</b>	Access to stdout is controlled by semaphore in multi-thread program

The function printf formats and prints a series of arguments to stdout.

**format** is a string consisting of ordinary characters, escape sequences and format specifications corresponding to the list of arguments passed to printf after format.

Ordinary characters and escape sequences are output directly in order from left to right.

When the first format specification is encountered the first argument is converted and output according to the format specification. The second format specification causes the second argument to be output, continuing until there are no more format specifications in the format string. If there are more arguments than format specifications the surplus arguments will be ignored. If there are fewer arguments than format specifications the results are undefined.

A format specification has the following structure:

```
%[flags][width][.precision][{F|N|h|l|L}]type
```

<b>flags</b>	These control justification, printing of signs, blanks, decimal points and radix prefixes. More than one flag can appear in a format specification.
<b>0</b>	For d, i, o, u, x, X, e, E, f, g & G conversions, leading zeros are used to pad the number to its field width. If both the 0 and - flags are used, the 0 flag is ignored. For d, i, o, u, x, X conversions: if a precision is specified, the 0 flag is ignored.
<b>-</b>	Left justify the result within field width. Default is right justify.
<b>+</b>	Prefix output with sign if output is signed type. Default is sign for negative values only.
<b>' '</b>	(space) Add leading space if value is signed and positive. Default is no space. If both space and + flags appear the space flag will be ignored.

#	<p>When used with o, x, X type prefix any nonzero output with 0, 0x or 0X respectively.</p> <p>When used with e, E, f type output always contains decimal point. Default: a decimal point only appears if digits follow it.</p> <p>When used with g or G type output always contains a decimal point and trailing zeros are not truncated.</p>
width	Minimum number of characters output. If the number of characters in the converted output value is less than the specified width output is padded, on the left or right, depending on the justification chosen. Blanks are used to pad the output, unless width is prefixed with a zero, in which case zeros are used.
Note	<b>Regardless of the value of width, the output will never be truncated (subject to the precision specification).</b>

width may be any combination of decimal digits or the character ‘\*,’ in which case the width parameter is supplied by the next int argument in the parameter list (i.e., the argument preceding the one to be formatted). A 0 leading the width specifier will be treated as a 0 flag.

precision	The number of characters to be printed; significant digits or decimal places:
-----------	---

When used with d, i, o, u, x, X, precision specifies the minimum number of digits to be output. If the number of digits in an argument is less than precision, the output is padded on the left with zeros. The value is not truncated if the number of digits exceeds precision. Default precision is 1. If value = 0 and precision is 0, no character is output.

When used with e, E or f, precision specifies the number of digits to be output after the decimal point. The last digit is rounded. Default precision is 6. If precision is 0 or the period appears without any number, no decimal point is printed.

```

settitle(W1, "Window 1", CenterUpperTitle);
W2 = windowopen(&WD2) ; /* open window */
settitle(W2, "Window 2", CenterUpperTitle);
cprintf("Hello World sent to window 2\n");
getch();
putbeneath(W2, W1);
getch();
putontop(W2);
getch();
windowclose(W1);
windowclose(W2);
return(0);
}

```

```
#include <conio.h>

char *read_string(char *str)
{
    int c;
    int n=0;

    while(1)
    {
        c=getch(); /* read character from */
                /* console without echo */
        putchar(c); /* output character to screen */
        if(c == '\n')
            break;
        str[n++]=c;
    }
    str[n]='\0';
    return(str);
}
```

## int putchar(int ch);

---

**A**

**Header file**           stdio.h

**See also**            getc, fputc, fgetc, fputc, putc.

**Portability**       ANSI

**Multi-thread**      Access to stream is *not* controlled by semaphore in multi-thread program.

The routine `putchar` puts a character `ch` on `stdout`. The routine is equivalent to `putc(ch, stdout)`. See `putc`.

**Note:**

`putchar` is identical to `fputc` but is a macro, not a function.

**Return value**

The macro returns the character written as an integer value. A return value of EOF may indicate an error or end of file condition. To determine which is the case, use `ferror` or `feof`.

**Example**

```
#include <stdio.h>

int print_to_stdout(char *msg)
{
    int n=0;
    int c;

    while((c=msg[n]) != 0)
    {
        putchar(c);
        ++n;
    }
    return(n);
}
```



## **int putenv(const char \*env);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	getenv
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	Access to environment is controlled by semaphore in multi-thread programs.

The function `putenv` adds a new environment variable or modifies the value of an existing variable.

`env` is a pointer to a string with the form:

```
variablename=string
```

where `variablename` is the name of the environment variable and `string` is its value. If `variablename` is already in the environment its value is replaced by `string`, otherwise the new variable is added to the environment.

**Note:** The function affects the environment local to a process. When that process terminates the environment will revert back to that of the parent process. However an altered environment can be passed to a child process created by `spawn` or `exec`.

### **Return value**

The function returns 0 if successful, or -1 if an error occurred.

### **Example**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int set_new_path(char *path)
{
    int ret;
    char new_path[128]="PATH=";
        /* make new path variable */
    strcat(new_path, path);
        /* insert into environment */
    if((ret=putenv(new_path)) == -1)
        printf("Environment full\n");
    return(ret);
}
```

```
void far _putimage(  
    short x, short y,  
    const char far *buffer,  
    short action);
```

---

**M**

Header file `graph.h`  
See also `_getimage`, `_imagesize`  
Portability DOS.  
Multi-thread Function is not re-entrant.

The function `_putimage` from the graphics module displays an image stored at a specified location. The image is displayed at a location whose upper left corner is specified. The manner in which the image is displayed and the effect the display has on any images already on the screen depend on the value of the action parameter.

**x,y** coordinates of the upper left corner of the screen area in which the image will be displayed.

**buffer** points to the location at which the image to be displayed is stored.

**action** specifies the manner in which the image is to be displayed. This parameter can have the following values:

**\_GAND** transfer the image over any image already on the screen. The resulting image is the logical *AND* of the two images. Portions of the original image may be erased, depending on the result of *ANDing* the pixels.

**\_GOR** superimpose the image over any image already on the screen. The original image is not erased.

**\_GPRESET** transfer a “negative” of the image to the screen. Each point has the inverse of the color attribute found when the image was stored using `_getimage`.

**\_GPSET** transfer the image to the screen, point-by-point. Each point has the same color attribute as when the image was stored using `_getimage`.

**\_GPXOR** This mode makes animation possible in two color modes. If an image is stored with itself, it will be erased and can be redrawn at another location, using modes `_GPSET` or `_GPOR`.

### Return value

None.

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

main()
{
    long size;
    short x1=10, y1=10, x2=40, y2=100;
    char *buffer;

    _setvideomode(_MRESNOCOLOR);
    _ellipse(_GFILLINTERIOR, x1, y1, x2, y2);
    size=_imagesize(x1, y1, x2, y2);
    buffer=malloc((size_t) size);
    _getimage(x1, y1, x2, y2, buffer);
    getch();
    /* erase old image */
    _putimage(x1, y1, buffer, _GXOR);
    /* put new image */
    _putimage(x1+10, y1+10, buffer, _GPSET);
    getch();
    free(buffer);
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**void putontop(wintype win);****W**

<b>Header file</b>	window.h
<b>See also</b>	putbeneath
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `putontop` from the TopSpeed window module puts the specified window (`win`) on the top of the window stack, ensuring that `win` is fully visible. If this results in other windows becoming obscured, then a buffer is allocated for each of these windows.

All subsequent output will appear within this window  $\text{\AA}$  except output redirected with `use`.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD={10, 2, 60, 8, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;
main()

{
  clrscr();
  W1 = windowopen(&WD) ;/* open window */
  setttitle(W1, "New Window Title",
    CenterUpperTitle);
  use(_fullscreen);
  gotoxy(0, 8);
  cprintf("Hello World sent to full screen\n");
  getch();
  putontop(_fullscreen);
  getch();
  putontop(W1);
  getch();
  windowclose(W1);
  return(0);
}
```

## **int puts(const char \*s);**

---

**A**

**Header file**           stdio.h

**See also**             gets.

**Portability**         ANSI

**Multi-thread**       Access to stream is controlled by semaphore in multi-thread programs.

The function puts copies a string (s) to stdout, replacing the string's terminating null character with a newline character (\n).

### **Return value**

The function returns 0 if successful, or EOF if an error occurred.

### **Example**

```
#include <stdio.h>

void print_string(char *msg)
{
    if(puts(msg) == EOF)/* output string */
        fprintf(stderr, "Error in print_string\n");
    return;
}
```

---

**int puttext(int left, int top, int right, int bottom, const void \*source);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	gettext
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `puttext` from the Turbo C window module copies a block of text from `buffer`. The text is copied to a location on the screen, delimited by the specified coordinates. Both the character and its attribute are copied.

**left** specifies the leftmost column of the target site.

**top** specifies the topmost line of the target site.

**right** specifies the rightmost column of the target site.

**bottom** specifies the bottom line of the target site.

**source** points to the location from which the material will be copied.

**Note:** All coordinates are absolute screen coordinates.

**Return value**

The function returns 1 if successful. If any of the coordinates are invalid, the function returns 0.

**Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define COLS 40
#define LINES 8
#define BUF_SIZE (COLS+1)*(LINES+1)*sizeof(int)
main() {
    char *win_buffer;
    FILE *f;
    win_buffer=malloc(BUF_SIZE);
    f=fopen("TEMP.$$$", "r");
    if(f == NULL)
        abort();
    fread(win_buffer, BUF_SIZE, 1, f);
    fclose(f);
    clrscr();

    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    puttext(20, 4, 20+COLS, 4+LINES, win_buffer);
    getch();
    free(win_buffer);
    return(0);
}
```

## **int putw(int w, FILE \*st);**

---

<b>Header file</b>	stdio.h
<b>See also</b>	getw.
<b>Portability</b>	Some DOS/OS2 The size of an integer and the ordering of bytes varies between systems.
<b>Multi-thread</b>	Access to the stream is controlled by semaphore in multi-thread programs.

The function `putw` writes the binary value, 16-bit integer `w` to the stream `st`, and increments the file pointer by two bytes.

**Note:** No particular alignment is assumed.

`st` specifies the stream to which the value is being written.

`w` specifies the value to write.

### **Return value**

The function returns the value written. A value of -1 may indicate an error, but since this is a legal integer value, `feof` or `ferror` should be called to verify an error or end of file condition.

### **Example**

```
#include <stdio.h>

main()
{
    FILE *f;
    int num;

    f=fopen("DATAFILE.DAT", "wb");
    scanf("%d", &num); /* get number from stdin */
    putw(num, f); /* write to file */
    if(ferror(f)) /* check for error */
        printf("Error writing file\n");
    fclose(f);
    return(0);
}
```

```
void qsort(  
    const void *base, size_t num, size_t width,  
    int (*compare)(const void *e1, const void *e2));
```

---

**A**

**Header file**      search.h and stdlib.h

**See also**          bsearch, lsearch.

**Portability**      ANSI

**Multi-thread**      Protection for static data accessed by this function is not provided by the library.

The function `qsort` is an implementation of the Quicksort algorithm for sorting an array.

**base**      is the array being sorted.

**num**      is the number of elements in the array.

**width**      represents the number of bytes required to store an element.

**compare**      is a pointer to a user supplied function that compares array elements (e.g., `e1` and `e2`) and returns an integer value indicating their relationship:

if `e1 < e2`, return `< 0`

if `e1 = e2`, return `= 0`

if `e1 > e2`, return `> 0`

The array is ordered by increasing value. Reversing the sense of less than and greater than in `compare` will cause the array to be ordered by decreasing value.

The original array is overwritten with the sorted array.

**Return value**

None.

**Example**



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

#define NUMBER_OF_ELEMENTS 6

int compare(void *n1, void *n2);

double data[]={1.1, 3.3, 2.2, 4.4, 0.0, 5.5};

main()
{
    int n;

    printf("Before sort - \n");
    n=0;
        /* list elements */
    while(n < NUMBER_OF_ELEMENTS)
    {
        printf("Element %d: %g\n", n, data[n]);
        ++n;
    }
        /* sort elements */
    qsort(
        data,
        NUMBER_OF_ELEMENTS,
        sizeof(double),
        compare);
    printf("After sort - \n");
    n=0;
        /* list sorted elements */
    while(n < NUMBER_OF_ELEMENTS)
    {
        printf("Element %d: %g\n", n, data[n]);
        ++n;
    }
    return(0);
}

int compare(void *n1, void *n2)
{
    return(
        (int)
        (((double *)n1)) - (((double *)n2)));
}
```

## **int raise(int sig);**

**A**

<b>Header file</b>	signal.h
<b>See also</b>	abort, signal
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Signal tables not protected by semaphore.
<b>OS2</b>	Only thread 1 of a multi-thread process may send or receive a signal.

The function `raise` sends a signal, `sig` to the current process. The default action for that signal will be taken unless a new action was defined previously by a call to `signal`. After a call to `raise` the signal action is reset to its default.

<b>Signal</b>	<b>Action</b>
---------------	---------------

<b>SIGABRT</b>	Abnormal termination. Default action is to terminate process with exit code of 3.
----------------	---

<b>SIGFPE</b>	Floating point unit error. Default action is to terminate calling process.
---------------	--

<b>SIGILL</b>	Illegal instruction. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to terminate process.
---------------	--

<b>SIGINT</b>	Ctrl-C interrupt. Default is to issue INT 23H.
---------------	--

<b>SIGSEGV</b>	Illegal storage access. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to terminate process.
----------------	---

<b>SIGTERM</b>	Termination request. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to ignore the signal.
----------------	--

### **Return value**

The function `raise` may not return. A return value of 0 indicates successful execution of a returning signal. If a value of -1 is returned, an error occurred.

Under OS2 the following signal values are also available

<b>Signal</b>	<b>Action</b>
---------------	---------------

<b>SIGUSR1</b>	Process flag 1
----------------	----------------

**SIGUSR2** Process flag 2**SIGUSR3** Process flag 3**Example**

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

main()
{
    /* set SIGABRT to ignore signal */
    if(signal(SIGABRT, SIG_IGN) == SIG_ERR)
        printf("Couldn't set SIGABRT");
    raise(SIGABRT); /* raise signal */
    printf("Program reaches here ");
    printf("if SIGABRT set to SIG_IGN");
    return(0);
}
```

---

**int rand(void);****A**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	srand, randomize.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	A sequence of random numbers can only be guaranteed if no other thread calls rand.

The function rand returns a pseudo-random integer in the range 0 - RAND\_MAX. A preceding call to srand can be used to set the random starting point.

If srand is used to initialize a sequence of random numbers, that sequence will be repeated if srand is called again with the same seed.

**Return value**

The function returns a pseudo-random number. There is no error return value.

**Example**

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

main()
{
    int count;

    count=0;
    printf("First series of random numbers\n");
    while(count++ < 10) {
        printf("%d\n", rand());
    }
    srand(1);
    printf("The next series should ");
    printf("be the same\n");
    count=0;
    while(count++ < 10) {
        printf("%d\n", rand());
    }
    return(0);
}
```

## **int randbrd(struct fcb \*fcb, int rcnt);**

---

**Header file**            dos.h

**See also**            getdta, randbwr, setdta

**Portability**        fcb structurety   Some DOS/OS2

**Multi-thread**      DOS is not re-entrant

The function randbrd does a random block read on a file. The function uses DOS system call 27H for this purpose.

### **fcb**

points to a fcb structure that contains the file control block information about the file. The fcb structure is defined in dos.h as

```
struct fcb {
    char fcb_drive; /*0 = default, 1 = A, 2 = B */
    char fcb_name[8]; /* File name */
    char fcb_ext[3]; /* File extension */
    short fcb_curblk; /* Current block number */
    short fcb_recsz; /* Logical record size */
                /* in bytes */
    long fcb_filsize; /* File size in bytes */
    short fcb_date; /* Date file last written */
    char fcb_resv[10]; /* Reserved for DOS */
    char fcb_currec; /* Current record in block */
    long fcb_random; /* Random record number */
};
```

### **rcnt**

specifies the number of records to read.

The function reads the specified number of records, beginning at the record specified in the fcb\_random member of the file control block. Records are read into memory at the disk transfer area (DTA).

After the read, the random record member of the file control block structure will be incremented by the number of records read.

### **Return value**

The function returns one of the following values:

- 0    all records were read.
- 1    the end of file was reached, and the last record read was complete.
- 2    no records were read, because DTA is too small to accommodate the specified number of records.

3 the end of file was reached, but the last record read was incomplete.

### Example

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

void error_handler(int code, char *msg);

struct fcb fbuf;

rand_read(int count)
{
    int ret;

    ret=randbrd(&fbuf, count);
    if(ret) error_handler(ret, "rand_read");
    return(0);
}
```

## **int randbwr(struct fcb \*fcb, int rcnt);**

---

<b>Header file</b>	dos.h
<b>See also</b>	randbrd
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function `randbwr` writes one or more records from the disk transfer area (DTA) to a file. The function uses DOS system call 28H for this purpose.

### **fcb**

points to a `fcb` structure that contains the file control block information about the file. The `fcb` structure is defined in `dos.h` as

```
struct fcb {
    char fcb_drive; /*0 = default, 1 = A, 2 = B */
    char fcb_name[8]; /* File name */
    char fcb_ext[3]; /* File extension */
    short fcb_curblk; /* Current block number */
    short fcb_recsz; /* Logical record size */
                    /* in bytes */
    long fcb_filsize; /* File size in bytes */
    short fcb_date; /* Date file last written */
    char fcb_resv[10]; /* Reserved for DOS */
    char fcb_currec; /* Current record in block */
    long fcb_random; /* Random record number */
};
```

**rcnt** specifies the number of records to write.

The function writes the specified number of records. If this value is 0, the file will be truncated to the length specified by the random record member of the file control block. After the function call, the random record member of the file control block structure will be incremented by the number of records written.

### **Return value**

The function returns one of the following values:

- 0 all records were written.
- 1 disk full, so no records are written.
- 2 DTA is too small to accommodate the specified number of records, so as many records as possible are written.

### **Example**

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

void error_handler(int code, char *msg);

struct fcb fbuf;

rand_write(int count)
{
    int ret;

    ret=randbwr(&fbuf, count);
    if(ret)
        error_handler(ret, "rand_write");
    return(0);
}
```



## **int random(int num)**

---

**Header file**            stdlib.h

**Portability**           DOS and OS2.

Returns a random number in the range 0 to num - 1.

## **void randomize(void);**

---

**Header file**            stdlib.h

**See also**               srand, rand.

**Portability**           Some DOS/OS2

**Multi-thread**          None.

The function randomize initializes the random number generator by passing a value derived from the system time to srand.

### **Return value**

None.

### **Example**

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

main()
{
    int count;

    count=0;
    printf("First series of random numbers\n");
    while(count++ < 10) {
        printf("%d\n", rand());
    }
    randomize();
    printf("The next series should ");
    printf("NOT be the same\n");
    count=0;
    while(count++ < 10) {
        printf("%d\n", rand());
    }
    return(0);
}
```

```
void _rdbufferln(  
    wintype win,  
    relcoord X, relcoord Y,  
    void *Dest, unsigned Len);
```

---

**W**

**Header file**            window.h

**See also**             \_wrbufferln

**Portability**         TopSpeed DOS/OS2

**Multi-thread**        Module is re-entrant and requires no additional locking  
for single function calls.

The function `_rdbufferln` from the TopSpeed window module accesses the window directly, reading one line at a time. The function reads a specified number of character/attribute words from the window, beginning at a specified location.

**win** specifies the window.

**X,Y**     specifies the location of the current position.

**Dest**    points to the location at which the material read is to be stored.

**Len**     specifies the number of words to read.

**Return value**

None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

#define BUFFER_LENGTH 18

windef WD1={10, 2, 50, 8, White, Brown,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;

main()
{
    int buffer[BUFFER_LENGTH];
    int n=0;

    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    gotoxy(1, 1);
    cprintf("Press Key to blink");
    getch();

    /* get line of text */
    _rdbufferln(W1, 1, 1, buffer, BUFFER_LENGTH);    while(n <
    BUFFER_LENGTH)
    {
        buffer[n]=0x8000; /* set blink bits */
        ++n;
    }

    /* write line of text */
    _wrbufferln(W1, 1, 1, buffer, BUFFER_LENGTH);
    getch();
    return(0);
}
```

---

**int read(int handle, void \*buf, unsigned num);**

---

**U****Header file**        io.h**See also**            close, creat, lseek, open, write.**Portability**        UNIX/DOS/OS2**Multi-thread**       No protection for low level I/O functions is provided by the library.

The function `read` reads a specified number of bytes into `buf` from the file associated with `handle`. The read operation begins at the current file position (if any) and continues to the end of file or until `num` bytes have been read. After the read the file position indicator is pointing to the next byte to be read.

**handle** specifies the handle for the file from which the information is to be read.

**buf** points to the location into which the information will be read.

**num**    specifies the number of bytes to read.

**Return value**

The function returns the number of bytes actually read. This number may be fewer than the number of bytes requested if the read encountered the end of file, or if the file was opened in text mode.

A return value of 0 indicates that an attempt to read at end of file was made. A return value of -1 indicates that an error may have occurred, although a successful read of 65535 bytes would return the integer value -1. In this case `errno` should be used to indicate an error. In the event of an error, `errno` is set to the following value:

**EBADF**    File handle invalid.

**Note:** Text mode translation causes carriage return / linefeed pairs to be replaced by a single linefeed character. Only the linefeed character is counted in the return value. The `^Z` character is treated as an end-of-file indicator.

If the file associated with `handle` is a device, output is merely terminated. If it is a file, the end of file indicator is set, and the file must be closed and reopened to clear it.

**Example**

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

#define BUFFER_SIZE 1024

main()

{
    int fh;
    int num_read;
    char *buffer[BUFFER_SIZE];

    fh=open("TEMP.$$$", O_RDWR|O_BINARY);
    if(fh < 0) {
        perror(NULL);
        abort();
    }
    num_read=read(fh, buffer, BUFFER_SIZE);
    if(num_read != BUFFER_SIZE)
    {
        if(!eof(fh))
            perror(NULL);
        else
            printf("Reached EOF\n");
    }
    close(fh);
    return(0);
}
```

## **int \_read(int handle, void \*buf, unsigned num);**

---

<b>Header file</b>	io.h
<b>See also</b>	_open, _write
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_read` reads a specified number of bytes from a file. The file can be a disk file or a device. The function accomplishes this by performing INT 21H function 3FH.

**handle** specifies the handle for the file to be read. This handle must be from a call to `_creat`, `_open`, `_dup`, or `_dup2`.

**buf** points to the location in which the information read will be stored.

**num** specifies the number of bytes to read from the file.

Text mode translation is not performed if the file being read has been opened in text mode.

If the file is a device, the bytes are read directly; if the file is a disk file, the function reads from the current file position, and increments the current position pointer by the number of bytes read.

### **Return value**

The function returns the number of bytes actually read. This number may be fewer than the number of bytes requested if the read encountered the end of file, or if the file was opened in text mode.

A return value of 0 indicates that an attempt to read at end of file was made. A return value of -1 indicates that an error may have occurred, although a successful read of 65535 bytes would return the integer value -1. In this case `errno` should be used to indicate an error. In the event of an error, `errno` is set to the following value:

**EACCES** access denied (e.g., because the file is a directory).

**EBADF** invalid handle.

### **Example**

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

#define BUFFER_SIZE 512

main()
{
    int fh;
    char msg_buffer[BUFFER_SIZE];
    int ret;

    fh=_open("TEMP.$$$", 2);
    ret=_read(fh, msg_buffer, BUFFER_SIZE);
    if(ret != BUFFER_SIZE)
        fprintf(stderr,
            "Reached end file on read\n");
    _close(fh);
    return(0);
}
```

---

**void \*realloc(void \*buffer, size\_t size);**

---

**A**

<b>Header file</b>	stdlib.h and alloc.h
<b>Variants</b>	realloc, _nrealloc, _frealloc, farrealloc, hrealloc.
<b>See also</b>	calloc, free, malloc.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Far heap only protected by semaphore

The function `realloc` changes the size of a memory block (`buffer`) previously allocated from the heap by a call to `calloc`, `malloc` or `realloc`. `size` is the new number of bytes requested.

**buffer** points to a block of storage on the heap.

**size** specifies the number of bytes the new memory block should have.

If `buffer` is `NULL`, the call is equivalent to a call to `malloc` with `size` as the argument.

If `size` is zero, the call is equivalent to a call to `free` with `buffer` as the argument.

In large and compact models, the object pointed to by the return value of `realloc` is paragraph aligned, and is not in the default data segment. In small and medium models the object is word aligned and *is* in the default data segment. The contents of the original block are guaranteed to be preserved up to the size of the original block.

**Return value**

The function returns a void pointer to the allocated space. If no space was available, a `NULL` pointer will be returned and `errno` set to `ENOMEM`. The original block may have been moved, so the return value will not necessarily be equal to the argument `buffer`.

**Example**



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

main()
{
    char *ptr;
    char near *near_ptr;
    char far *far_ptr;
    ptr=malloc(200);
        /* reallocate from default heap */
    ptr=realloc(ptr, 1000);
    if(ptr == NULL)
        perror("realloc failed");
    else
        realloc(ptr, 0);
        /* reallocate from near heap */
    near_ptr=_nrealloc(NULL, 200);
    _nfree(near_ptr);
        /* reallocate from far heap */
    far_ptr=_fmalloc(200);
    far_ptr=_frealloc(far_ptr, 10000);
    if(far_ptr == NULL)
        perror("_frealloc failed");
    else
        _ffree(far_ptr);
    return(0);
}
```

---

**short far \_rectangle(short fill, short x1, short y1, short x2, short y2);**

---

**M****Header file**           graph.h**See also**             \_getcolor, \_getlinestyle, \_setcolor, \_setlinestyle**Portability**         DOS.**Multi-thread**       Function is not re-entrant.

The function `_rectangle` from the graphics module draws a rectangle bounded by the specified upper left and lower right points. The function uses the current line style and color.

**fill** specifies how the rectangle is to be drawn. This argument can have either of the following values:

**\_GFillInterior**

fill the figure using the current color and fill mask.

**\_GBorder**   do not fill the figure.**x1,y1**   coordinates of the upper left point of the rectangle.**x2,y2**   coordinates of the lower right point of the rectangle.**Note:**

Coordinates for the points specified are for logical points.

**Return value**

The function returns a nonzero value if successful; otherwise it returns 0.

**Example**

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

main()
{
    struct videoconfig v;
    short X1, Y1, X2, Y2;
    short xoffset, yoffset;

    _setvideomode(_MRESNOCOLOR);
    _getvideoconfig(&v);
    xoffset=v.numxpixels/16;
    yoffset=v.numypixels/16;
    X1=xoffset;
    Y1=yoffset;
    X2=v.numxpixels-xoffset;
    Y2=v.numypixels-yoffset;
    while((X1 < X2) && (Y1 < Y2))
    {
        _rectangle(_GBORDER, X1, Y1, X2, Y2);
        X1+=xoffset;
        Y1+=yoffset;
        X2-=xoffset;
        Y2-=yoffset;
    }
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**short far \_remapallpalette(long far \*color\_array);****M****Header file** graph.h**See also** \_remapallpalette, \_selectpalette, \_setvideomode**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function \_remapallpalette from the graphics module remaps all pixels values to colors. The colors are specified in the color\_array.

The number of colors that can be assigned depends on the number supported in the current video mode. E.g., a 16-color mode can map all the colors in the following default color array, whereas an 8-color mode can map only colors 0 through 7.

Number	Color	Number	Color
0	Black	8	Dark gray
1	Blue	9	Light blue
2	Green	10	Light green
3	Cyan	11	Light cyan
4	Red	12	Light red
5	Magenta	13	Light magenta
6	Brown	14	Yellow
7	White	15	Bright white

A 4-color mode will use a palette for the EGA and a two-color mode will support only black and white (i.e., colors 0 and 7).

This function requires an EGA or VGA adapter.

**Return value**

The function returns the *previous* color number of pixel. If there is an error (e.g., if a monitor other than EGA or VGA is being used), the function returns -1.

**Example**

```
void setup256mode(void)
{
    unsigned short i=0;
    unsigned short red, green, blue;
    long color;
    long colors[256];

    _setvideomode(_MRES256COLOR);
    while(i < 256) {
        red  =(i&0xE0)>>2;
        green=(i&0x1C)<<1;
        blue =(i&0x03)<<4;
        color= (
            ((long)blue)<<16) |
            (((long)green)<<8) |
            (red);
        colors[i]=color;
        ++i;
    }
    _remapallpalette(colors);
    return;
}
```

## long far \_remappalette(short pixel, long col);

**M**

**Header file** graph.h

**See also** \_remapallpalette, \_selectpalette, \_setvideomode

**Portability** DOS.

**Multi-thread** Function is not re-entrant.

The function \_remappalette from the graphics module remaps a pixel value to a color.

**pixel** specifies the pixel value being reassigned to a color.

**col** specifies the color to which the pixel value is being assigned. This value must refer to a color supported by the video mode.

Certain colors have been given predefined names:

Name	Number	Name	Number
_BLACK	0	_GRAY	8
_BLUE	1	_LIGHTBLUE	9
_GREEN	2	_LIGHTGREEN	10
_CYAN	3	_LIGHTCYAN	11
_RED	4	_LIGHTRED	12
_MAGENTA	5	_LIGHTMAGENTA	13
_BROWN	6	_LIGHTYELLOW	14
_WHITE	7	_BRIGHTWHITE	15

This function requires an EGA or VGA adapter.

### Return value

The function returns the *previous* color number of pixel. If there is an error (e.g, if a monitor other than EGA or VGA is being used), the function returns -1.

### Example

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

#define _JPI_WIN_
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <conio.h>

#pragma warn(wall => on)

void draw_logo(short cols[]);

long colors[]={
    _BLACK, _BLUE, _GREEN, _CYAN, _RED,
    _MAGENTA, _BROWN, _WHITE, _GRAY, _LIGHTBLUE,
    _LIGHTGREEN, _LIGHTCYAN, _LIGHTRED,
    _LIGHTMAGENTA, _LIGHTYELLOW, _BRIGHTWHITE };

main()
{
    int n;
    short c[4]={9, 8, 15, 6};

    _setvideomode(_ERESCOLOR);
    _setactivepage(1);
    _clearscreen(_GCLEARSCREEN);
    draw_logo(c);
    _settextposition(3, 22);
    _setvisualpage(1);
    n=1;
    while(!kbhit())
    {
        _remappalette(6, colors[(6+n)%16]);
        _remappalette(8, colors[(8+n)%16]);
        _remappalette(9, colors[(9+n)%16]);
        _remappalette(15, colors[(15+n)%16]);
        delay(100);
        ++n;
    }
    _setvideomode(_DEFAULTMODE);
    return;
}
```

```
void draw_logo(short cols[])
{
    _setcolor(cols[0]);
    _rectangle(_GFillInterior,
        140, 20, 500, 330);
    _setcolor(cols[1]);
    _moveto(225, 110);
    _lineto(320, 140);
    _lineto(415, 110);
    _lineto(460, 125);
    _lineto(460, 225);
    _lineto(365, 255);
    _lineto(365, 285);
    _lineto(320, 300);
    _lineto(180, 255);
    _lineto(180, 190);
    _lineto(225, 205);
    _lineto(225, 235);
    _lineto(270, 250);
    _lineto(270, 190);
    _lineto(180, 160);
    _lineto(180, 125);
    _lineto(225, 110);

    _moveto(180, 125);
    _lineto(320, 170);
    _lineto(460, 125);
    _moveto(320, 170);
    _lineto(320, 300);

    _moveto(365, 190);
    _lineto(415, 175);
    _lineto(415, 210);
    _lineto(365, 225);
    _lineto(365, 190);
    _lineto(415, 210);

    _moveto(225, 205);
    _lineto(270, 190);
    _moveto(225, 235);
    _lineto(270, 220);
    _moveto(180, 190);
    _lineto(225, 175);

    _floodfill(250, 130, cols[1]);
    _floodfill(230, 180, cols[1]);
    _floodfill(260, 240, cols[1]);
    _floodfill(380, 210, cols[1]);

    _setcolor(cols[2]);
    _floodfill(200, 230, cols[1]);
    _floodfill(250, 220, cols[1]);
    _floodfill(400, 190, cols[1]);
    _floodfill(440, 190, cols[1]);

    return;
}
```



## **int remove(const char \*path);**

## **A**

<b>Header file</b>	stdio.h
<b>See also</b>	close, unlink.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function remove deletes the file specified by path.

### **Return value**

The function returns 0 if the file was successfully deleted. If an error occurred, the value -1 is returned and errno is set to one of the following values:

<b>EACCES</b>	path specified a read-only file.
<b>ENOENT</b>	File or pathname not found.

### **Example**

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

main(int argc, char *argv[])
{
    if(argc != 2) {
        printf("Usage - DELETE filename\n");
        exit(0);
    }
    if(remove(argv[1]) != 0)/* delete file */
        printf("Error deleting file %s\n",
            argv[1]);/* delete failed */
    return(0);
}
```

---

**int rename(const char \*oldname, const char \*newname);**

---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	remove, unlink.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `rename` changes the name of the file or directory specified by `oldname` to `newname`. `oldname` must be the name of an existing file or directory; `newname` must be the name of a new file or directory. Files may be moved from one directory to another, but not between devices. Directories may not be moved, only renamed.

**Return value**

The function returns 0 if the file was successfully renamed. If an error occurred, the value -1 is returned and `errno` is set to one of the following values:

<b>EACCES</b>	Path specified by <code>newname</code> could not be created.
<b>ENOENT</b>	File or pathname not found.
<b>ENOTSAM</b>	Not same device.

**Example**

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

main(int argc, char *argv[])
{
    if(argc != 3)
    {
        printf("Usage - RENAME oldname newname\n");
        exit(0);
    }
    if(rename(argv[1], argv[2]) != 0)
        printf(
            "Error renaming file %s\n",
            argv[1]);
    return(0);
}
```

---

**void rewind(FILE \*st);**

---

**A****Header file**           stdio.h**See also**             fseek, ftell.**Portability**         ANSI**Multi-thread**       Access to stream is controlled by semaphore in multi-thread programs.

The function `rewind` sets the file pointer associated with `st` to the beginning of the file, and clears the error and end of file indicators.

**Return value**

None.

**Example**

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

main()
{
    FILE *f;
    char string[80];

    f=fopen("TEMP.$$$", "r");
    fgets(string, 80, f);
    printf("The first string read ");
    printf("is %s\n",string);
    rewind(f);
    printf("The second string read ");
    printf("is also %s\n",string);
    fclose(f);
    return(0);
}
```

## **int rmdir(const char \*path);**

---

<b>Header file</b>	dir.h
<b>See also</b>	chdir, mkdir.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.
<b>OS2</b>	The function will return an error if the specified directory is the current working directory of another session.

The function `rmdir` removes the directory specified by `path`. The directory must be empty and not the current or root directory.

### **Return value**

The function returns 0 if directory was removed. If an error occurred, the value -1 is returned and `errno` is set to one of the following values:

<b>EACCES</b>	path referred to file, root directory or current directory.
<b>ENOENT</b>	Pathname not found.

### **Example**

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

main(int argc, char *argv[])
{
    if(argc != 2) {
        printf("Usage - RMDIR directory\n");
        exit(0);
    }
    if(rmdir(argv[1]) != 0)
        printf(
            "Error %d removing directory %s\n",
            errno, argv[1]);
    return(0);
}
```

## **int rmtmp(void);**

---

**Header file**           stdio.h

**See also**             tmpfile, tmpnam.

**Portability**         UNIX/DOS/OS2

The function rmtmp removes all the files created by tmpfile in the current directory.

### **Return value**

The function returns the number of files closed and deleted.

### **Example**

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

main()
{
    FILE *f;

    f=tmpfile();
    if(f != NULL)
        fprintf(f,
            "This won't be here for long\n");
    rmtmp();
    return(0);
}
unsigned _rotl(unsigned val, int count);
```

## **unsigned \_rotr(unsigned val, int count);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	_lrotr, _lrotr.
<b>Portability</b>	8086 Family
<b>Multi-thread</b>	None.

The functions \_rotr and \_lrotr rotate the unsigned integer argument (val) left or right, respectively, by count bits.

**val**        the value to be rotated.

**count**       the number of bits by which val is to be rotated.

### **Return value**

The rotated value is returned. There is no error return.

### **Example**

```
include <stdio.h>
#include <stdlib.h>
#include <conio.h>
main()
{
    unsigned int number;
    char buffer[40];
    char *binary_result;
    printf("Enter decimal number ");
    scanf("%lu", &number);
    do{
        binary_result=ltoa(number, buffer, 2);
        printf(
            "\nBinary equivalent: %32s\n",
            binary_result);
        number=_rotr(number, 1);
        while(getch() != 'q');
        return(0);
    }
```

## **void \*sbrk(int incr);**

---

<b>Header file</b>	alloc.h
<b>Multi-thread</b>	None.

The function sbrk performs no function in this library, so it returns with a value (void \*) -1.

## **int scanf(const char \*format, [arguments]...);**

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	cscanf, fscanf, sscanf, vscanf, vfscanf, vsscanf.
<b>Portability</b>	ANSI. Prefixes F and N are not part of the ANSI definition for scanf but are extensions provided for compatibility with other compilers.
<b>Multi-thread</b>	Access to stdin is controlled by semaphore in multi-thread programs.

The function scanf reads characters from stdin and stores the converted data in the locations given by arguments.

### **format**

is a string consisting of ordinary characters, escape sequences and format specifications corresponding to the list of arguments passed to scanf after the format argument. format can contain the following objects:

1. *Whitespace characters* — as specified by the isspace function: scanf will read but not store any whitespace characters in the input up to the next non-whitespace character. One whitespace character matches any number of whitespace characters in the input.
2. *Non-whitespace characters*  $\mathcal{A}$  (not including '%'). scanf will read, but not store, a matching character. If the character does not match, scanf terminates.
3. *Format specifications*  $\mathcal{A}$  a format specification, beginning with the '%' character causes scanf to read a character, convert it into the specified type and store the result at the location specified by the next argument in the list. An input field is defined as all convertible characters up to the first white space character, or the number of characters specified by the field width.

When the first format specification is encountered, the first argument is converted according to the format specification and stored. The second format specification causes the second argument to be scanned, continuing until there are no more format specifications in the format string. If there are more arguments than format specifications, the surplus arguments will be ignored. If there are fewer arguments than format specifications the results are undefined

A format specification has the following structure:

```
%[*][width][{F|N|h|l|}]{type}
```

\* The '\*' character suppresses assignment of the converted data. The field is scanned but not stored.

**width**

Maximum number of characters to be scanned for the current argument. width may be any (non-zero) combination of decimal digits

**prefix**

The following prefixes are allowed:

**F** causes any pointer argument to scanf to be treated as a far pointer.

causes any pointer argument to scanf to be treated as a near pointer.

**h and l** determine the size of the argument expected:

**h** is used as a prefix with the types d, i, o, u, x or X to specify that the argument is a pointer to short int.

**l** is used as a prefix with the types d, i, o, u, x or X to specify that the argument is a pointer to long int. With e f g it specifies a double.

**L** specifies a long double with e, f and g.

**type**

data type to be input. *Always* expects a *pointer* to the argument; usually with arguments other than strings this means that the argument requires and ‘address of’ (&) operator. The type specifier can have the following values.

**d** int, signed decimal (base 10) integer.

**i** int, signed integer: base determined by input format thus:

**prefix 0** expects octal (base 8) integer;

**prefix 0x** expects hexadecimal (base 16) integer.

**all esle:** expects decimal (base 10) integer.

**u** int, unsigned decimal (base 10) integer.

**o** int, unsigned octal (base 8) integer.

**x** int, unsigned hexadecimal (base 16) integer using the lowercase digits ‘abcdef’ or the uppercase digits ‘ABCDEF’.

**e, f, g**

float, signed floating point value. Input is expected to have the form



[ - ]XX.XXXXXX[e[+-]]XXX

An optional sign followed by decimal digits, containing an optional decimal point, followed by an optional signed exponent in the form

[ {E|e} ][+-]XXX

**c** sequence of characters. White space characters are read and stored.

**s** string. Characters are read and stored until the first white space character is read. A terminating null is automatically appended.

To read strings not delimited by white space characters, a set of characters in brackets ([]) can be substituted for the scanlist. The corresponding input field is read up to the first character that is not included in the bracketed set. If the first character in the set is '^' the effect is reversed and the input field is read up to the first character that *is* included in the set. If the scanlist begins either with [] or [^], the right bracket is considered to be a member of the scanlist and the next right bracket terminates the list.

**n** pointer to int. The number of characters input is stored in the integer to which this argument points. No input is consumed.

**p** either a far or a near pointer to void, depending on the memory model being used. A far pointer is read in the form SSSS:OOOO, where SSSS is the segment and OOOO is the offset in uppercase hexadecimal digits. A near pointer is read only in the form OOOO and expects a pointer to a pointer.

**%** matches a single '%' character. No input consumed.

**Note** scanf scans the input stream character by character. If scanf stops reading a field for any reason, the next input field is considered to start at the next unread character. Any character that conflicts with the expected input is considered unread.

### Return value

The function scanf returns the number of fields converted and assigned. This does not include fields where assignment was suppressed by the '\*' character in the format string.

A return value of 0 indicates that no fields were assigned.

A return value of -1 indicates that an attempt was made to read past the end of the file before any conversion was made. (In the case of sscanf and vsscanf, an attempt to read past the end of the string.)

### Example

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

main()

{
    int n1;
    long l1;
    double d1=0.0;
    char s1[64];
    int fields;

    fields=scanf("%d %ld %lg", &n1, &l1, &d1);
    printf(
        "%d fields scanned: %d, %ld, %g\n",
        fields, n1, l1, d1);
    printf("Is that OK? ");
    scanf("%s", s1);
    if(stricmp(s1, "YES"))
        printf("Sorry");
    return(0);
}
```

---

**void \_searchenv(const char \*path, const char \*env, char \*buf);**

---

**Header file**            stdlib.h**See also**                putenv, getenv.**Portability**            Some DOS/OS2**Multi-thread**          Access to environment is controlled by semaphore in multi-thread programs.

Function **\_searchenv** searches for the file specified in **path** in the domain of the environment variable, **env**. The function searches first in the current working directory, then in directories specified in the environment variable, if necessary

**path**    specifies the file being sought.

**env** specifies the environment variable whose value will guide the search.

**buf** points to a buffer to contain the complete pathname of the file. If the file is found, the complete pathname will be constructed and stored in **path**. If the file was not found, **path** will contain a null terminated string.

**Return value**

None.

**Example**

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

main(int argc, char *argv[]) {
    char full_name[14];
    char full_path[80];

    if(argc != 2)
    {
        printf("Usage - FIND name ");
        printf("(no extension)\n");
        abort();
    }

    /* construct program name */
    strcpy(full_name, argv[1]);
    /* search using path variable */
    strcat(full_name, ".EXE");
    _searchenv(full_name, "PATH", full_path);
    if(full_path[0] == '\0') {
        strcpy(full_name, argv[1]);
        strcat(full_name, ".COM");
        _searchenv(full_name, "PATH", full_path);
    }
    if(full_path[0] != '\0')
        printf(full_path);
    else
        printf("%s.??? not found\n", argv[1]);
    return(0);
}
```

## **char \*searchpath(const char \*path);**

---

**Header file**            dir.h

**See also**             exec, spawn, system

**Portability**         DOS/OS2

**Multi-thread**       Environment variables are protected by semaphore.

The function searchpath searches for the specified file in the directories included in the DOS PATH environment variable.

**path**       specifies the name of the file being sought.

The function searches the current directory first. If the file is not found, the function searches directories in the PATH variable until the file is found or until there are no more path directories to search.

If the file is found, the function returns a string with the complete file name.

### **Return value**

The function returns the complete file name string if successful; otherwise the function returns a null string.

### **Example**

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

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

    if(argc != 2)
        abort();
    full_path=searchpath(argv[1]);
    if(full_path)
        printf("Path = %s\n", full_path);
    else
        printf("%s not found\n", argv[1]);
    return(0);
}
```

## **void segread(struct SREGS \*segs);**

---

<b>Header file</b>	dos.h
<b>See also</b>	FP_SEG
<b>Portability</b>	8086 family.
<b>Multi-thread</b>	None.

The function `segread` copies the contents of the segment registers to the structure of type `SREGS` pointed to by `segs`. This structure is defined in `dos.h`:

```
struct SREGS{
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
};
```

### **Note:**

Under OS2 the values returned in the `SREGS` structure are selectors.

### **Return value**

None.

### **Example**

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

main()
{
    struct SREGS sr;

    segread(&sr); /* read segment registers */
    printf(
        "ES = %X\nCS = %X\nSS = %X\nDS = %X\n",
        sr.es, sr.cs, sr.ss, sr.ds);
    return(0);
}
```

---

**short far \_selectpalette(short num);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getvideoconfig, _setvideomode
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_selectpalette` from the graphics module selects the specified color palette.

**num** specifies the palette to be used. The range of values this parameter can have depends on the video adapter and on the video mode.

A *palette* consists of a background color (which can be selected) and three other colors (which are set).

Only the MRES4COLOR and MRESNOCOLOR video modes can be used with this function.

For an EGA adapter in MRES4COLOR mode, the following four palettes are available:

<b>Palette</b>	<b>Colors 1, 2, and 3</b>
0	Green, Red, and Brown
1	Cyan, Magenta, and Light gray
2	Light green, Light red, and Yellow
3	Light cyan, Light magenta, and White

For an EGA adapter in MRESNOCOLOR mode, the following palettes are available:

<b>Palette</b>	<b>Colors 1, 2, and 3</b>
0	Green, Red, and Brown
1	Light green, Light red, and Yellow
2	Light cyan, Light red, and Yellow
3	Same as palette 1.

For a CGA adapter in MRESNOCOLOR mode, the following palettes are available:

**Palette            Colors 1, 2, and 3**

0	Blue, Red, and Light gray
1	Light blue, Light red, and White

**Return value**

The function returns the value of the *previous* palette. There is no error return.

**Example**

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

main()

{
    _setvideomode(_MRES4COLOR);
    _selectpalette(2);
    _ellipse(_GFILLINTERIOR, 10, 10, 200, 125);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **void SEND(SIGNAL s);**

---

<b>Header file</b>	process.h
<b>See also</b>	WAIT
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

A call to function SEND with s as argument will cause the first process waiting for s to become active. If no processes are waiting, the call will be queued.

The SEND operation works by incrementing the counter associated with s. If the counter is less than or equal to zero, then at least one process is waiting for s, and the first one in the queue will become *ready* for execution. This process will also start execution if its priority is greater than or equal to the priority of the current process.

### **Return value**

None.

### **Example**

See example program in , page .

---

**short far \_setactivepage(short page);**

---

**M****Header file** graph.h**See also** \_gettextcolor, \_gettextposition, \_outtext, \_setttextcolor, \_setttextposition, \_setttextwindow, \_setvideomode, \_setvisualpage, \_wrapon**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_setactivepage` from the graphics module specifies the graphics page where graphics output will be written.

**page** specifies the current active page. The default page is 0.

**Note:**

This function works only on configurations with enough memory to support multiple graphics pages. E.g., the CGA can support multiple video pages only in text mode; the EGA and VGA can have up to 256K of memory for multiple video pages in graphics mode.

**Return value**

The function returns the number of the *previous* active page if successful; otherwise the function returns a negative value.

**Example**

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

main()
{
    _setvideomode(_MRES16COLOR);
    _setactivepage(1);
    /* draw rectangle on page 1 */
    _rectangle(_GBORDER, 10, 10, 90, 50);
    _setvisualpage(1); /* view page 1 */
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **long far \_setbkcolor(long col);**

---

**M**

**Header file**           graph.h

**See also**             \_getbkcolor, \_remappalette, \_selectpalette

**Portability**         DOS.

**Multi-thread**       Function is not re-entrant.

The function \_setbkcolor from the graphics module sets the current background color to a specified value.

**col** specifies the color to be used for the background.

### **Return value**

The function returns the pixel value of the *old* background color. There is no error return.

### **Example**

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

main()
{
    _setvideomode(_MRES4COLOR);
    _setbkcolor(_BLUE);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **int setblock(unsigned segx, unsigned newsize);**

---

<b>Header file</b>	dos.h
<b>See also</b>	allocmem
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function setblock changes the size of a previously allocated block.

segx	specifies the segment address of the block. This value must be from a call to allocmem.
newsize	specifies the desired size (in paragraphs).

### **Return value**

The function returns -1 if successful; otherwise it returns the largest available block size (in paragraphs) and sets \_doserrno.

### **Example**

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

#define NEW_SIZE 1000

main()
{
    char far *buffer;
    unsigned new_seg;
    unsigned maxsize;
    int error;

    error=allocmem(100, &new_seg);
    if(error >= 0)
    {
        fprintf(stderr,
            "Error allocating memory\n");
        return(1);
    }
    maxsize=setblock(new_seg, NEW_SIZE);
    if((int) maxsize >= 0)
        printf("Reallocation failed, ");
    printf("%u byte available\n", maxsize);
    else
    {
        buffer=MK_FP(new_seg, 0);
        printf(
            "%u paras of memory are at %Fp\n",
            NEW_SIZE, buffer);
    }
    freemem(new_seg);
    return(0);
}
```

---

**void setbuf(FILE \*st, char \*buffer);**

---

**A**

<b>Header file</b>	stdio.h
<b>See also</b>	setvbuf.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Access to stream is controlled by semaphore in multi-thread programs.

Allows the user to control buffering for stream st. The argument st must refer to an open stream before it has been read or written.

If the argument buffer is NULL, the stream will be unbuffered. If buffer is not NULL, it must point to a character array of at least BUFSIZ, defined in stdio.h. This buffer will be used for the stream in place of the system allocated buffer.

A call to setbuf is the equivalent to a call to setvbuf with the value of \_IOFBF for mode and BUFSIZ for size. stderr and stderr are unbuffered by default and may be assigned a buffer using this function.

**Return value**

None.

**Example**

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

void read_small_block(FILE *f);
void read_large_block(FILE *f);
main()
{
    FILE *f;
    char user_buffer[BUFSIZ];

    f=fopen("TEMP.$$$", "r");
    if(f != NULL)
    {
        setbuf(f, NULL); /* no buffering */
        read_small_block(f);
        /* buffered with users buffer */
        setbuf(f, user_buffer);
        read_large_block(f);
        fclose(f);
    }
    return(0);
}
```

## **int setcbrk(int val);**

---

<b>Header file</b>	dos.h
<b>See also</b>	getcbrk
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant

The function `setcbrk` sets the value of the control-break flag. The function uses DOS system call 33H for this purpose.

`val` specifies the action to take, and can be either 0 or 1.

- If `val` is 0, checking is turned off, except for checking during I/O involving the console, printer or a communications device.
- If `val` is 1, checking is turned on, so that a check occurs at every system call.

### **Return value**

The function returns the value passed in `val`.

### **Example**

```
#include <dos.h>

main() {
    int setting, newsetting;

    setting =getcbrk();
    if ( setting)
    {
        printf ( "Currently checking.\n");
        newsetting = setcbrk ( 0);
        newsetting =getcbrk();
        if ( !newsetting)
        {
            printf ( "Not checking now.\n");
            setting = setcbrk (setting);
            printf ( "original setting");
            printf(" restored.\n");
        }
    }
    else
        printf ( "Not checking.\n");
}
```

---

**void far \_setcliprgn(short x1, short y1, short x2, short y2);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_setviewport, _settextwindow
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setcliprgn` from the graphics module sets the boundaries for graphics calls to the specified area of the screen. Only output that will fit within the upper left and lower right boundaries will be visible. This area is known as the *clipping region*.

<code>x1,y1</code>	coordinates of the upper left corner of the clipping region.
<code>x2,y2</code>	coordinates of the lower right corner of the clipping region.

**Note:** this function affects only graphics output.

**Return value** None

**Example**

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

main()
{
    short X1=25, Y1=10, X2=500, Y2=200;

    _setvideomode(_ERESNOCOLOR);
    _setcliprgn(10, 5, 400, 120);
    _rectangle(_GBORDER, 10, 5, 400, 120);
    _ellipse(_GFILLINTERIOR, X1, Y1, X2, Y2);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**short far \_setcolor(short col);**

---

**M****Header file** graph.h**See also** \_arc, \_ellipse, \_floodfill, \_getcolor, \_lineto, \_pie, \_rectangle, \_selectpalette, \_setpixel**Portability** DOS.**Multi-thread** Function is not re-entrant.

The function `_setcolor` from the graphics module sets the current color to the value specified in `col`.

The default color is the color with the highest number in the palette.

`col` values are always within range, because these values are masked.

**Return value**

The function returns the previous color value.

**Example**

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

main()
{
    _setvideomode(_ERESCOLOR);
    _setcolor(1);
    _rectangle(_GFILLINTERIOR, 20, 20, 400, 180);
    _setcolor(4);
    _ellipse(_GBORDER, 20, 20, 400, 180);
    _setcolor(3);
    _floodfill(100, 100, 4);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **void setdate(const struct date \*dptr);**

---

<b>Header file</b>	time.h and dos.h
<b>See also</b>	getdate, gettime, settime
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

Changes the system date to the values stored in the structure to which dptr points. The date structure is defined as follows:

```
struct date {  
    int da_year;  
    unsigned char da_day;  
    unsigned char da_mon;  
};
```

**Return value** None.

### **Example**

```
#include <time.h>  
main()  
{  
    struct date *datptr1, *datptr2;  
    datptr1 =  
        (struct date *)  
        malloc ( sizeof ( struct date));  
    getdate ( datptr1);  
    printf ( "Today's date is: %u / %u / %d\n",  
        datptr1->da_mon, datptr1->da_day,  
        datptr1->da_year);  
    datptr2 =  
        (struct date *)  
        malloc ( sizeof ( struct date));  
    datptr2->da_year = datptr1->da_year + 1;  
    datptr2->da_mon = datptr1->da_mon;  
    datptr2->da_day = datptr1->da_day;  
  
    setdate ( datptr2);  
    printf ( "Revised date is: %u / %u / %d\n",  
        datptr2->da_mon, datptr2->da_day,  
        datptr2->da_year);  
    setdate ( datptr1);  
    printf ( "Restored date is: %u / %u / %d\n",  
        datptr1->da_mon, datptr1->da_day,  
        datptr1->da_year);  
}
```

## **int setdisk(int drive);**

---

<b>Header file</b>	dir.h
<b>See also</b>	getdisk
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function setdisk sets the current drive to the one specified in drive. Drive A is 0, drive B is 1, etc.

### **Return value**

The function returns the total number of drives.

### **Example**

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

main(int argc, char *argv[])
{
    unsigned new_drive, max_drive;

    if(argc != 2)
        abort();
    sscanf(argv[1], "%d", &new_drive);
    max_drive=setdisk(new_drive);
    printf("%u drives in system\n", max_drive);
    return(0);
}
```

## **void setdta(char far \*dta);**

---

**Header file**        dos.h

**See also**         getdta

**Portability**      Some DOS/OS2

**Multi-thread**    DOS is not re-entrant

The function setdta sets the disk transfer area (DTA) to the value specified by dta.

dta                    specifies the new location of the DTA.

**Return value**    None.

### **Example**

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

char new_dta[128];

main()
{
    char far * buffer;

    setdta((char far *) new_dta);
    buffer=getdta();
    printf("DTA is now %Fp\n", buffer);
    return(0);
}
```

---

**void far \_setfillmask(const unsigned char far \*mask\_array);**

---

**M**

**Header file** graph.h  
**See also** \_floodfill, \_getfillmask  
**Portability** DOS.  
**Multi-thread** Function is not re-entrant.

The function `_setfillmask` from the graphics module sets the current fill mask to the specified values.

`mask_array` specifies the value for an 8x8 array of bits. Each bit in this array can be 1 or 0. If the value is 1, the pixel corresponding to that point will be on (in the current fill color); otherwise the pixel will be off.

**Return value** None.

### Example

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

main()
{
    unsigned char new_mask[8];
    short px[]={ 0, 200, 180, 120};
    short py[]={ 20, 40, 120, 120};

    _setvideomode(_ERESCOLOR);
    _clearscreen(_GCLEARSCREEN);

    new_mask[0]=0x88;
    new_mask[1]=0x44;
    new_mask[2]=0x22;
    new_mask[3]=0x11;
    new_mask[4]=0x88;
    new_mask[5]=0x44;
    new_mask[6]=0x22;
    new_mask[7]=0x11;
    _setfillmask(new_mask);
    _polygon(_GFILLINTERIOR, 4, px, py);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **void setframe(wintype win, framestr Frame, Color Fore, Color Back);**

---

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `setframe` from the TopSpeed window module changes the frame around the specified window, and redisplay the title if required.

<code>win</code>	specifies the window whose frame is being changed.
<code>Frame</code>	specifies the new frame.
<code>Fore</code>	specifies the foreground color for the frame.
<code>Back</code>	specifies the background color for the frame.

The constants `SINGLEFRAME` and `DOUBLEFRAME` are defined in `window.h`.

**Return value** None.

### **Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD={10, 2, 60, 8, White, Blue,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;
main()

{
clrscr();
W1 = windowopen(&WD) ;/* open window */
settitle(W1, "New Window Title",
CenterUpperTitle);
/* output to current window */
cprintf("Hello World\n");
getch();
setframe(W1, DOUBLEFRAME, Green, Red);
getch();
windowclose(W1); /* close window */
return(0);
}
```

---

**int setftime(int handle, struct ftime \*ftptr);**

---

**U**

<b>Header file</b>	io.h
<b>See also</b>	getftime
<b>Portability</b>	DOS.
<b>Multi-thread</b>	DOS is not re-entrant

The function `setftime` sets the time information for the file associated with `handle`. The time is set to the values stored in the structure to which `ftp` points. The `ftime` structure is defined as follows:

```
struct ftime {
    unsigned ft_tsec  : 5; /* 2 second interval */
    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 */
};
```

**Return value**

The function returns 0 if successful; otherwise, the function returns -1 and sets `errno` to one of the following values:

<b>EINVFNC</b>	function number was invalid.
<b>EBADF</b>	file number was bad.

The function returns an error in the OS2 DOS compatibility box.

**Example**

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

```

main()
{
    int fhandle, result;
    struct ftime *ftptr;

    ftptr =
        (struct ftime *)
        malloc ( sizeof ( struct ftime));
    fhandle = open (
        "FTFILE.$$$",
        O_CREAT,
        S_IWRITE|S_IREAD);
    getftime ( fhandle, ftptr);
    printf ( "File time information:\n");
    printf (
        "%u:%u o'clock on %u/%u/198%u\n",
        ftptr->ft_hour, ftptr->ft_min,
        ftptr->ft_month, ftptr->ft_day,
        ftptr->ft_year);
    ftptr->ft_year += 1;
    result = setftime ( fhandle, ftptr);
    if (!result)
        printf ( "Success.\n");
    else
    {
        printf ( "Error!\n");
        exit();
    }
    getftime ( fhandle, ftptr);
    printf ( "Revised file time information:\n");
    printf (
        "%u:%u o'clock on %u/%u/198%u\n",
        ftptr->ft_hour, ftptr->ft_min,
        ftptr->ft_month, ftptr->ft_day,
        ftptr->ft_year);
    ftptr->ft_year -= 1;
    result = setftime ( fhandle, ftptr);
    if (!result)
        printf ( "Success.\n");
    else
    {
        printf ( "Error!\n");
        exit();
    }
}

```

## **void SetInProgramFlag(unsigned char State)**

---

**Header file**        dos.h

**Portability**        DOS only

**See also**            GetInProgramFlag

The InProgramFlag indicates whether a process is executing within its own program code or within the operating system. If a program attempts to re-enter DOS (i.e. when the InProgramFlag is clear) a run-time error will occur for certain functions.

However it is legal to re-enter DOS from a critical error handler. In this case a call to SetInProgramFlag with the argument 1 will allow library functions that call DOS to operate.

**Return Value**    None



---

**int setjmp(jmp\_buf env);**

---

**A****Header file**            setjmp.h**See also**             longjmp.**Portability**         ANSI**Multi-thread**        None.

The setjmp function saves the processor environment in env so that it can be subsequently used by longjmp. These functions provide a mechanism for executing inter-function gotos, usually used to pass control to error recovery code.

A call to setjmp causes the current processor environment to be saved in env. A subsequent call to longjmp restores the saved environment and causes execution to resume at a point immediately after the corresponding setjmp call. Execution continues with retval as the return value from setjmp.

As long as longjmp is called before the function calling setjmp returns, all variables local to the routine will have the same values when setjmp was called. Register variables may not, however, be restored.

jmp\_buf is defined in setjmp.h as follows:

```
typedef struct{
    unsigned j_sp; /* saved 80X86 registers */
    unsigned j_ss;
    unsigned j_flag;
    unsigned j_cs;
    unsigned j_ip;
    unsigned j_bp;
    unsigned j_di;
    unsigned j_es;
    unsigned j_si;
    unsigned j_ds;
} jmp_buf[1];
```

**Return value**

The function returns 0 after saving the processor environment. If setjmp returns after a call to longjmp, it returns the value argument of longjmp, which is guaranteed to be nonzero.

**Example**

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

void func(void);
jmp_buf error_env;

main() {
    /* initialize longjmp */
    /* environment */
    if(setjmp(error_env) != 0)
        abort(); /* arrives here from longjmp */
    func();
    printf("Program terminated without error\n");
    return(0);
}

void func(void) {
    printf("Press any key to end program. ");
    printf("'e' will cause error\n");
    if(getch() == 'e')
        longjmp(error_env, 1);
    return;
}
```

---

**void far \_setlinestyle(unsigned short mask);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getlinestyle
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setlinestyle` from the graphics module selects the mask to be used for drawing lines. Each bit in the 16-bit mask represents a pixel in the line being drawn.

- If a bit is 1, the pixel to which the figure applies is drawn in the current color; if the bit is 0, the line is left unchanged. Such a template is repeated for the entire line.
- The default mask is all 1's (i.e., 0xFFFF in this case), which produces a solid line.

**Return value** None.

**Example**

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

main()
{
    unsigned short mask=0xCCCC;

    _setvideomode(_ERESCOLOR);
    _setcolor(4);
    _setlinestyle(mask);
    _rectangle(_GBORDER, 10, 10, 500, 300);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

**char \*setlocale(int category, const char \*locale);****A**

<b>Header file</b>	locale.h
<b>See also</b>	localeconv.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	Local is process-wide, and static data are not protected by semaphore. The function setlocale may be used to set or query the program's current locale or a portion.
category	specifies the action requested. This can take the following values:
<b>Category</b>	<b>Action</b>
LC_ALL	specifies the entire locale.
LC_COLLATE	specifies the behavior of functions strcoll and trxfm..
LC_CTYPE	specifies the behavior of the character and multibyte functions.
LC_MONETARY	specifies the formatting information returned by localeconv.
LC_NUMERIC	specifies the decimal point character for the formatted I/O functions and string conversion functions.
LC_TIME	specifies the behavior of the strftime function.
locale	points to a specification for the locale

A value of "C" for locale specifies the minimal C environment; "" specifies the native environment. In the TopSpeed implementation, these environments are the same. At program startup the "C" environment is selected.

**Return value** If a valid string is given for the locale argument, the area of the locale specified by category is changed to that value. If an error occurs, a NULL value is returned.

If the value of locale is NULL, the current setting for that portion of the locale is returned, and the locale remains unchanged.

**Example**

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

main() {
    setlocale(LC_TIME, "C");
    return(0);
}
```

---

**struct xycoord far \_setlogorg(short x, short y);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getlogorg
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The \_setlogorg from the graphics module function moves the logical origin  $\mathbb{A}$  which has coordinates (0,0)  $\mathbb{A}$  to a specified physical point.

x	specifies the horizontal coordinate for the new origin.
y	specifies the vertical coordinate for the new origin.

**Return value**

The function returns the *physical* coordinates of the previous logical origin. The struct xycoord in which this information is returned is defined in graph.h.

```
struct xycoord {
    short xcoord;
    short ycoord;
};
```

**Example**

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

main() {
    _setvideomode(_ERESNOCOLOR);
    _ellipse(_GFILLINTERIOR, 25, 10, 500, 200);
    getch();
    _setlogorg(100, 100);
    _clearscreen(_GCLEARSCREEN);
    _ellipse(_GFILLINTERIOR, 25, 10, 500, 200);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**void setmem(void \*dest, unsigned length, char value);**

---

**Header file** mem.h and string.h

**See also** memset

**Portability** Some DOS/OS2

**Multi-thread** None.

The function setmem sets the first length bytes of dest to value. dest points to the location being assigned the value.

**length** specifies the number of bytes to assign the specified value.

**value** specifies the value being stored in the memory block.

**Return value** None.

**Example**

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

#define BUFFER_SIZE 200

main()
{
    char buffer1[]="Hello World";
    char buffer2[]="Hello World";
    /* will return 0 */
    memcmp(buffer2, buffer1, 12);
    setmem(buffer2, 12, 'A');
    /* will now return < 0 */
    memcmp(buffer2, buffer1, 12);
    return(0);
}
```

## **int setmode(int handle, unsigned mode);**

**U**

<b>Header file</b>	io.h and fcntl.h
<b>See also</b>	creat, fopen, open.
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	Access to stream is controlled by semaphore in multi-thread program.

The function setmode sets the file translation mode of the file associated with handle to mode. The value of mode must be one of the following:

handle	specifies the handle for the file whose mode is being set.
O_TEXT	Text mode - Carriage return/Linefeed pairs are translated to a single Line feed on Input. Linefeed characters are translated to a Carriage return/Linefeed pair on output. The ^Z character is treated as an end of file indicator.
O_BINARY	Binary mode - None of the above translations are performed and ^Z is treated as an ordinary character.

### **Return value**

The function returns the previous translation mode. If an error occurred, a value of -1 is returned and errno is set to one of the following values:

EBADF	Invalid file handle.
EINVAL	mode argument invalid.

### **Example**

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

#define WRITE_SIZE 100

extern char binary_data[];
extern char text_data[];
int func(char *file)
{
    int fh;
    /* opened in binary mode */
    fh=open(file, O_RDWR);
    write(fh, binary_data, WRITE_SIZE);
    setmode(fh, O_TEXT); /* change to text mode */
    write(fh, text_data, WRITE_SIZE);
    close(fh);
    return(0);
}
```

**void setpalette(wintype win, PaletteDef Pal);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function setpalette from the TopSpeed window module changes the palette of the specified window, and redisplay the colors.

**win** specifies the window whose palette is being changed.

**Pal** specifies the new palette information.

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>
windef WD1={10, 2, 50, 8, White, Brown,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;
PaletteDef pal;
main() {
    clrscr();
        /* initialize palette */
    pal[NORMALPALETTECOLOR].Fore=Red;
    pal[NORMALPALETTECOLOR].Back=Blue;
    pal[FRAMEPALETTECOLOR].Fore=Green;
    pal[FRAMEPALETTECOLOR].Back=Brown;
        /* open window */
    W1 = paletteopen(&WD1, pal) ;
    setttitle(W1, "Window 1", CenterUpperTitle);
    getch();
        /* change palette */
    pal[NORMALPALETTECOLOR].Fore=LightGray;
    pal[NORMALPALETTECOLOR].Back=Red;
    pal[FRAMEPALETTECOLOR].Fore=Yellow;
    pal[FRAMEPALETTECOLOR].Back=Blue;
    setpalette(W1, pal);/* set new palette */
    getch();
    return(0);
}
```



## void setpalettec color (int c);

## W

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

Sets the palette color set to be used in the current window to the color set specified by c. Any subsequent output will now appear in the colors specified by entry c.

**Return value** None.

### Example

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 50, 8, White, Brown,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;

PaletteDef pal;

main() {
clrscr();
pal[NORMALPALETTECOLOR].Fore=Red;
pal[NORMALPALETTECOLOR].Back=Blue;
pal[FRAMEPALETTECOLOR].Fore=Green;
pal[FRAMEPALETTECOLOR].Back=Brown;
pal[2].Fore=LightGray;
pal[2].Back=Red;
pal[3].Fore=Yellow;
pal[3].Back=Blue;
/* open window */
W1 = paletteopen(&WD1, pal) ;
settitle(W1, "Window 1", CenterUpperTitle);
cprintf("Current palette color is %d\n",
palettecolor());
getch();
setpalettec color(2);
cprintf("Current palette color is %d\n",
palettecolor());
getch();
return(0);
}
```

---

**short far \_setpixel(short x, short y);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_getpixel
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setpixel` from the graphics module sets the pixel at the logical point whose horizontal and vertical coordinates correspond to the arguments `x` and `y`, respectively.

**Return value**

The function returns the *previous* value for the pixel of interest. If there is an error (e.g., if the specified point is outside the clipping region), the function returns -1.

**Example**

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

main()
{
    int x, y, c;
    struct videoconfig v;

    _setvideomode(_ERESCOLOR);
    _clearscreen(_GCLEARSCREEN);
    _getvideoconfig(&v);

    while(!kbhit())
    {
        x=random(v.numxpixels-1);
        y=random(v.numypixels-1);
        c=random(v.numcolors-1);
        _setcolor(c);
        _setpixel(x, y);
    }
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**short far \_settextcolor(short col);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_gettextcolor
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_settextcolor` from the graphics module sets the current color for text output to the value specified by `col`. The value of `col` must be between 0 and 31 in text color mode. Values 0 through 15 have the following meaning:

Value	Color	Value	Color
0	Black	8	Dark gray
1	Blue	9	Light blue
2	Green	10	Light green
3	Cyan	11	Light cyan
4	Red	12	Light red
5	Magenta	13	Light magenta
6	Brown	14	Yellow
7	White	15	Bright white

Values between 16 and 31 specify the same colors as in the previous list, except that the text blinks. Thus, a value of 12 specifies light red, and a value of 28 (12+16) specifies light red with blinking text.

**Note:** **text colors are not restricted to the current palette.**

The default text color is the highest valid pixel value.

**Return value** the function returns the pixel value of the *previous* text color. There is no error return.

**Example**

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

main() {
    int n;

    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 50);
    n=1;
    while(n < 16)
    {
        _settextcolor(n);
        _settextposition(0, 0);
        _outtext("Press key to change color");
        getch();
        ++n;
    }
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**struct rccoord far \_settextposition(short row, short col);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_gettextposition, _outtext
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_settextposition` from the graphics module moves the current text position to the specified location in the text window. The location is specified in terms of the logical origin, which has coordinates 1,1. Subsequent text output will begin at the new location. The location is specified in row and column (rather than pixel) coordinates.

`row` specifies the vertical coordinate for the text position.  
`col` specifies the horizontal coordinate for the text position.

**Return value**

The function returns the *previous* text position. this result is returned in an `rccoord` structure, as defined in `graph.h`:

```
struct rccoord {
    short row;
    short col;
};
```

**Example**

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

main() {
    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 50);
    _displaycursor(_GCURSORON);
    _settextposition(1, 1);
    _outtext("Press Key To Hide Cursor");
    getch();
    _settextposition(1, 1);
    _displaycursor(_GCURSOROFF);
    _outtext("Press Key To Exit    ");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**void far \_settextwindow(short x1, short y1, short x2, short y2);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_outtext
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_settextwindow` from the graphics module specifies a window in which all text output to the graphics screen is displayed. This window's boundaries are specified in row and column (rather than pixel) coordinates. The logical origin of a text window has coordinates 1,1.

`x1,y1` coordinates of the upper left corner of the window.  
`x2,y2` coordinates of the lower right corner of the window.

The text is output beginning at the top of the window. When the window is full, the uppermost line of text scrolls out.

**Return value**

None.

**Example**

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

main()
{
    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 70);
    _outtext("Hello World\n");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

## **void settime(const struct time \*tpr);**

---

<b>Header file</b>	time.h and dos.h
<b>See also</b>	ctime, getdate, gettime, setdate, time
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function settime sets the system time information to the values stored in the structure to which tpr points. The time structure is defined as follows:

```
struct time {  
    unsigned char ti_hour;  
    unsigned char ti_min;  
    unsigned char ti_sec;  
    unsigned char ti_hund;  
};
```

**Return value** None.

### **Example**

```
#include <time.h>  
#include <stdlib.h>  
  
main()  
{  
    struct time *timpr, *newtime;  
  
    timpr =  
        (struct time *)  
        malloc ( sizeof ( struct time));  
    newtime =  
        (struct time *)  
        malloc ( sizeof ( struct time));  
    gettime ( timpr);  
    gettime ( newtime);  
    printf ( "The time is: %u:%u:%d\n",  
        timpr->ti_hour, timpr->ti_min,  
        timpr->ti_sec);  
    if (newtime->ti_hour < 22)  
        newtime->ti_hour += 2;  
    else  
    if (newtime->ti_hour >= 2)  
        newtime->ti_hour -= 2;  
    settime ( newtime);  
    gettime ( newtime);  
    printf ( "The new time is: %u:%u:%d\n",  
        newtime->ti_hour, newtime->ti_min,  
        newtime->ti_sec);  
    newtime->ti_hour = timpr->ti_hour;  
    settime ( newtime);  
    printf ( "Time reset.\n");  
}
```

**void settitle(wintype win, char \* NewTitle, TitleMode Mode );****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

Updates the window title in the specified window frame.

<b>win</b>	specifies the window.
<b>NewTitle</b>	specifies the new title to be used in the window.
<b>Mode</b>	specifies the position of the title.

The TitleMode type is defined in window.h:

```
typedef enum {
    NoTitle,
    LeftUpperTitle,   CenterUpperTitle,
    RightUpperTitle,  LeftLowerTitle,
    CenterLowerTitle, RightLowerTitle
} TitleMode;
```

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD={10, 2, 60, 8, White, Blue,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;
main()
{
    clrscr();
    W1 = windowopen(&WD) ;/* open window */
        /* output to current window */
    cprintf("Hello World\n");
    getch();
    settitle(W1, "New Window Title",
        CenterUpperTitle);
    getch(); /* close window */
    windowclose(W1);
    return(0);
}
```



---

**int setvbuf(FILE \*st, char \*buffer, int type, size\_t size);**

---

**A****Header file**      stdio.h**See also**          setbuf.**Portability**      ANSI**Multi-thread**    Access to stream is controlled by semaphore in multi-thread programs.

The function setvbuf allows the user to control buffering for stream st. The argument stream must refer to an open stream before it has been read or written.

st                    represents the stream whose buffering will be controlled.

buffer              points to the array that will be the buffer. If buffer is NULL, a system buffer will be allocated which will be freed when the stream is closed.

type                specifies the buffering mode and can be one of the following values, defined in stdio.h:

  \_IOFBF            Full buffering.

  \_IOLBF            Line buffering. The buffer is flushed whenever a line-feed character is output.

  \_IONBF            No buffer is used, regardless of the value of buffer or size.

size                specifies the size of the buffer to be used. size may be an integer value, where  $0 < \text{size} < 32768$ .

stderr and stderraux are unbuffered by default and may be assigned a buffer using this function.

**Return value**    The function returns 0 if successful, or nonzero if an error occurred.

**Example**

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

void read_vlarge_block(FILE *f);

main() {
    FILE *f;
    char user_buffer[BUFSIZ*10];

    f=fopen("TEMP.$$$", "r");
    if(f != NULL)
    {
        /* allocate very large buffer */
        setvbuf(f, user_buffer, _IOFBF, BUFSIZ*10);
        read_vlarge_block(f);
        fclose(f);
    }
    return(0);
}
```

## **void setvect(int interruptno, void (interrupt far \*isr) ());**

---

<b>Header file</b>	dos.h
<b>See also</b>	getvect
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

Function setvect sets a specified interrupt vector to point to a new interrupt function.

interruptno	specifies the interrupt vector whose value is being changed.
isr	points to the new interrupt function that will be executed if the interrupt vector is invoked. This function must be declared as an <i>interrupt</i> routine.

**Return value** None.

### **Example**

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

void interrupt far handler(void);
void rest_of_program(void);

main()
{
    void (interrupt far *old_int)(void);

    old_int=getvect(4); /* save old vector */
    setvect(4, handler);/* install new handler */
    rest_of_program();
    setvect(4, old_int);/* restore old handler */
    return(0);
}

void interrupt far handler(void)
{
    return;
}

void rest_of_program(void)
{
    return;
}
```

## **void setverify(int value);**

---

<b>Header file</b>	dos.h
<b>See also</b>	getverify
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant

The function setverify sets the value of the verify flag.

value specifies the new setting for the verify flag. This can be 0 for off or 1 for on.

**Return value** None.

### **Example**

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

main()
{
    setverify(0);
    printf("Verify is now off\n");
    return(0);
}
```

**short far \_setvideomode(short mode);****M**

<b>Header file</b>	graph.h
<b>See also</b>	_getvideoconfig
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setvideomode` from the graphics module selects a screen mode suitable for a particular configuration. The mode parameter can take the following values.

<code>_DEFAULTMODE</code>	specifies the default video mode for the given hardware configuration. When called with this mode, the function restores the default video setting.
<code>_TEXTBW40</code>	40x25 character monochrome text mode with 16 gray shades; used for CGA monitor.
<code>_TEXTC40</code>	40x25 character color text mode with 16 colors; used for CGA adapter.
<code>_TEXTBW80</code>	80x25 character monochrome text mode with 16 gray shades; used for CGA adapter.
<code>_TEXTC80</code>	80x25 character color text mode with 16 colors; used for CGA adapter.
<code>_TEXTMONO</code>	80x25 character color text mode with one color; used for monochrome adapter.
<code>_MRESNOCOLOR</code>	320x200 pixel monochrome graphics mode with four gray shades; used for CGA adapter.
<code>_MRES4COLOR</code>	320x200 pixel color graphics mode with four colors; used for CGA adapter.
<code>_MRES16COLOR</code>	320x200 pixel color graphics mode with 16 colors; used for EGA adapter.
<code>_MRES256COLOR</code>	320x200 pixel color graphics mode with 256 colors; used for VGA adapter.
<code>_HRESBW 640x200</code>	pixel color graphics mode with two shades of gray; used for CGA adapter.
<code>_HRES16COLOR</code>	640x200 pixel color graphics mode with 16 colors; used for EGA adapter.
<code>_ERESNOCOLOR</code>	640x350 pixel monochrome graphics mode with one color;

	used for EGA adapter.
<code>_ERESCOLOR</code>	640x350 pixel color graphics mode with 64 colors; used for EGA adapter.
<code>_VRES2COLOR</code>	640x480 pixel monochrome graphics mode with two colors; used for VGA adapter.
<code>_VRES16COLOR</code>	640x480 pixel color graphics mode with 16 colors; used for VGA adapter.
<code>_HERCMONO</code>	hercules graphic 2 colors graphics mode, 720 by 348.

**Return value** The function returns a nonzero value if successful. If the mode selected is not supported by the hardware, the function returns 0.

### Example

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

main()
{
    _setvideomode(_MRES4COLOR);
    _outtext("_MRES4COLOR selected");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**void far \_setviewport(short x1, short y1, short x2, short y2);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_setcliprgn, _settextwindow
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setviewport` from the graphics module sets the boundaries for graphics calls to a specified area of the screen (called a *clipping region*), and then makes the upper left corner of this region the logical origin. Only output that will fit within the clipping region will be visible.

x1,y1	physical location of the upper left corner of the clipping region.
x2,y2	physical location for the lower right corner of the clipping region.

**Note:** this function affects only graphics output.

**Return value** None.

**Example**

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

main()
{
    short X1=25, Y1=10, X2=500, Y2=200;

    _setvideomode(_ERESNOCOLOR);
    _setviewport(10, 5, 400, 120);
    _rectangle(_GBORDER, 10, 5, 390, 115);
    _ellipse(_GFILLINTERIOR, X1, Y1, X2, Y2);
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

---

**short far \_setvisualpage(short page);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_setactivepage, _setvideomode
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_setvisualpage` from the graphics module selects the current visual page Æ i.e., the one displayed.

`page` specifies the current visual page. The default page is 0.

**Note:** This function works only on configurations with an EGA adapter and with enough memory to support multiple graphics pages.

**Return value**

The function returns the number of the *previous* visual page if successful; otherwise the function returns a negative value.

**Example**

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

main()
{
    _setvideomode(_MRES16COLOR);
    _setactivepage(1);
    /* draw rectangle on page 1 */
    _rectangle(_GBORDER, 10, 10, 90, 50);
    _setvisualpage(1); /* view page 1 */
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```



## **void (\* signal(int sig, void (\*func()))(int);**

**A**

<b>Header file</b>	signal.h
<b>See also</b>	abort, exec, exit, spawn.
<b>Portability</b>	ANSI. There are some OS2-specific actions: see below.
<b>Multi-thread</b>	Signal tables are not protected by semaphore.

The function `signal` sets the signal handler for signal `sig` to the value `func`. The argument `sig` may one of the following constants, defined in `signal.h`:

<b>Signal</b>	<b>Action</b>
SIGABRT	Abnormal termination. Default action is to terminate process with exit code of 3.
SIGFPE	Floating point unit error. Default action is to terminate calling process.
SIGILL	Illegal instruction. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to terminate process.
SIGINT	Ctrl C interrupt. Default is to issue <code>int23H</code> .
SIGSEGV	Illegal storage access. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to terminate process.
SIGTERM	Termination request. This signal is not issued by DOS but is supported for ANSI compatibility. Default action is to terminate the process.

The argument `func` must one of the following constants or the function address of a user defined signal handler.

<b>Signal</b>	<b>Action</b>
SIG_DFL	The default action is taken. The process terminates. Open files are closed but buffers are not flushed.
SIG_IGN	The signal is ignored.

**Note:** **Signal settings are not passed to a child process created by `exec` or `spawn`.**

The signal handler is reset to the default value on being raised or generated by a system event.

**Return value**

The function returns the previous value of the signal handler. If an error occurred, a value of `SIG_ERR` is returned, and `errno` is set to `EINVAL`, indicating an invalid value for `sig`.

OS2 Considerations :      Only thread 1 of a multi-thread process may send or receive a signal.

Under OS2 the following signal values are also available

Signal	Action
<code>SIGUSR1</code>	Process flag 1
<code>SIGUSR2</code>	Process flag 2
<code>SIGUSR3</code>	Process flag 3

Under OS2 also the `func` argument may be one of these two extra manifest constants:

Signal	Action
<code>SIG_ACK</code>	Acknowledge receipt of signal. This must be sent to reset a user defined signal. After a signal has been sent all further signals of that type are ignored until it is reset.
<code>SIG_ERR</code>	The signal is ignored and, if the signal was <code>SIGUSR1</code> , <code>SIGUSR2</code> , or <code>SIGUSR3</code> the calling process receives an error.

### Example

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

void func(int sig);

main()
{
    if(signal(SIGINT, func) == SIG_ERR) /* func now handles SIGINT */
        printf("Couldn't set SIGINT");
    if(getch() == 's')
        raise(SIGINT); /* will be handled by func */
    printf("Normal termination\n");
    return(0);
}

void func(int sig)
{
    printf("Program reaches here if ");
    printf("SIGINT raised or Ctrl C pressed");
    printf("- Signal received was %d", sig);
    exit(0);
}
```

---

**double sin(double x);**

---

**A**

---

**long double sinl(long double x);**

---

**Header file**           math.h**See also**            cos, cosh, tan, tanh, sinh, matherr.**Portability**        ANSI**Multi-thread**       None.

The function sin returns the sine of x, an angle in radians.

**Return value**

The function returns  $\sin(x)$ . If x is greater than  $2^{53}$ , a total loss of significance occurs. A TLOSS message is printed to stderr, errno is set to ERANGE and the value 0 is returned.

Error handling can be altered by installing a new matherr function.

**Example**

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate sine of input */
    result=sin((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("sin of %g is %g\n", num, result);
    return(0);
}
```

---

**double sinh(double x);**

---

**A**

---

**long double sinhl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	cos, cosh, tan, tanh, sin.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `sinh` returns the hyperbolic sine of `x`, an angle in radians.

**Return value**

The function returns *sinh*(`x`). If `x` is greater than  $2^{53}$ , a total loss of significance occurs. A TLOSS message is printed to `stderr`, `errno` is set to `ERANGE` and the value 0 is returned.

Error handling can be altered by assigning a new handler function to the `matherr` variable.

**Example**

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate sinh of input */
    result=sinh((double) num);
    if(errno)
        printf(
            "Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("sinh of %g is %g\n", num, result);
    return(0);
}
```

## **void sleep(unsigned nrsec);**

---

**Header file**        dos.h

**See also**            delay

**Portability**        DOS/OS2.

The function sleep suspends the current program for a specified number of seconds. The accuracy is limited by the hardware and the operating system.

**Return value**    None.

**void snapshot(void);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `snapshot` from the TopSpeed window module updates the window buffer from the screen. This works only with non-palette windows.

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 8, White, Blue,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;

windef WD2={40, 6, 75, 18, Yellow, Red,
FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
LightGray, Blue};
wintype W2;

main()
{
    snapshot(); /* save full screen */
    W1 = windowopen(&WD1) ; /* open window */
    settitle(W1, "Window 1", CenterUpperTitle);
    W2 = windowopen(&WD2) ; /* open window */
    settitle(W2, "Window 2", CenterUpperTitle);
    cprintf("Hello World sent to window 2\n");
    getch();
    hide(W2);
    getch();
    putontop(W2);
    getch();
    windowclose(W1);
    windowclose(W2);
    /* restore original full screen */
    putontop(_fullscreen);
    return(0);
}
```

## **int sopen(const char \*path, int access, int shflag, ...);**

---

**Header file** io.h plus types in fcntl.h and stat.h

**See also** close, creat, fopen, open, umask

**Portability** DOS/OS2

**Multi-thread** None.

**OS2, DOS 4.x** Neither SHARE.COM nor SHARE.EXE is required.

The function sopen opens a file specified by path and prepares the file for reading and/or writing and sharing, as specified with access and shflag.

path specifies the file name.

access is an integer composed of one or more of the following constants, which are defined in fcntl.h:

O\_APPEND Write only at end of file.

O\_BINARY Open file in untranslated mode.

O\_CREAT create file if it does not exist. This constant has no effect if path exists.

O\_EXCL Return error if file exists. This constant is valid only with O\_CREAT.

O\_RDONLY Open file for reading only.

O\_RDWR Open for reading and writing.

O\_TEXT Open in translated mode. (See I/O module for description of translated mode.)

O\_TRUNC Open and truncate existing file to zero length. If the file does not exist, it is created. Existing files must have write permission.

O\_WRONLY Open file for writing only.

If more than one constant is to be specified the constants should be joined with a bitwise OR operator (|).

Either O\_RDONLY, O\_RDWR or O\_WRONLY must be given; there is no default.

If neither O\_TEXT nor O\_BINARY is given, translation mode defaults to that in \_fmode.

shflag specifies the sharing mode, and may be one of the following constants defined in share.h:

SH_COMPAT	Sets compatibility mode.
SH_DENYRW	Deny read and write access.
SH_DENYWR	Deny write access.
SH_DENYRD	Deny read access.
SH_DENYNO	Permit read and write access.

**Note:** If neither **SHARE.COM** nor **SHARE.EXE** is installed, or if the process is running under DOS v2, the value of **shflag** is ignored.

Permission is required only if **O\_CREAT** is specified. If the file exists, the value is ignored. The file permission settings are only set the first time the file is closed.

Permission may one of the three following constants:

S_IWRITE	Writing permitted.
S_IREAD	Reading permitted.
S_IWRITE S_IREAD	Reading and writing permitted.

If write permission is given, under DOS read permission is implied.

**sopen** applies the (optional) permission argument to the file permission mask before creating the file.

**Return value** **sopen** returns a file handle for the opened or created file. If the function failed, a value of -1 is returned and **errno** is set to one of the following:

EACCES	Path was a directory or write permission not granted.
EEXIST	File exists. <b>O_CREAT</b> or <b>O_EXCL</b> specified.
EMFILE	No more handles available.
ENOENT	Pathname not found.

### Example

```

if(fh < 0)
    abort();
write(fh, msg, strlen(msg));
close(fh);
return(0);
}

```



## **void sound(unsigned freq);**

---

**Header file**        dos.h

**See also**            delay, nosound

**Portability**        PC and compatibles.

The function sound turns the computer's speaker on. The speaker emits a sound at the frequency (in cycles per second, or Hertz) specified by freq. The sound continues until a call to nosound turns the speaker off.

**Return value**    None.

### **Example**

```
#include <dos.h>

#define INCREMENT 500

main()
{
    int freq;
    for ( freq = 500;
        freq <= 5000;
        freq += INCREMENT)
    {
        sound ( freq);
        delay ( 100);
        nosound ();
        delay ( 100);
    }
}
```

```

int spawnl(int mode, const char *path, char *arg0, ... );
int spawnle(int mode, const char *path, char *arg0, ...);
int spawnlp(int mode, const char *path, char *arg0, ... );
int spawnlpe(int mode, const char *path, char *arg0,...);
int spawnv(int mode, const char *path, char *argv[]);
int spawnve(int mode, const char *path, char *argv[], char *env[]);
int spawnvp(int mode, const char *path, char *argv[]);
int spawnvpe(int mode, const char *path, char *argv[], char *env[]);

```

---

<b>Header file</b>	process.h
<b>See also</b>	abort, exit, _exit, atexit, onexit, system and the exec family of functions.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	DOS is not re-entrant
<b>OS2</b>	P_OVERLAY is valid under OS2. In this case the return value is the process ID of the child.

The function spawn loads and executes a child process. If the call is successful, the child process occupies the memory previously occupied by the parent process. If sufficient memory is not available, the call will fail.

mode	specifies the action of the parent process while the child process executes, and may be one of the following constants defined in process.h:
P_WAIT	Parent waits until child process terminates before resuming execution.
P_NOWAIT	Parent continues to execute while child process executes.
P_OVERLAY	Child process overlays parent destroying it. (This is the same as a call to exec.)

Only P\_WAIT and P\_OVERLAY are legal under DOS.

path	specifies the file to be executed. The following search procedures are used:
------	--

- If the path does not have a file-name extension or does not end with a period, the spawn function searches for the file. If the search is unsuccessful, the extensions .COM then .EXE are tried.
- If the path has a file-name extension, then only that extension is used.
- If the path ends with a period, no extension is used.

Each function in this family uses a generic spawn function and the letters at the end determine the variation of argument passing:

- p Uses the path environment variable to locate the file to be executed.
- l Lists the command line arguments separately.
- v Passes an array of pointers to command line arguments.
- e Passes an array of pointers to environment arguments.

The `spawnlp`, `spawnlpe`, `spawnvp`, and `spawnvpe` functions use the same procedures, but in all directories specified by the `PATH` environment variable.

With `spawnl`, `spawnle`, `spawnlp`, and `spawnlpe`, arguments are passed by giving one or more pointers to character strings which will form the argument list for the child process. The series of pointers must be terminated by a `NULL` pointer.

The combined length of the strings specified as arguments must not exceed 128 characters. Terminating null characters are not counted but a space character is automatically inserted between each pair of arguments.

With `spawnv`, `spawnve`, `spawnvp`, and `spawnvpe`, arguments are passed in an array, with the final member being a `NULL` pointer.

At least one argument must be passed to a child process, i.e., `arg0`. This is usually a copy of the path argument although a different value will not cause an error. Under DOS 3.0+ the path is available as `arg0`.

`spawnl`, `spawnlp`, `spawnv` and `spawnvp` cause the child process to inherit the environment of the parent process. `spawnle`, `spawnlpe`, `spawnve` and `spawnvpe` can pass a list of environment variable settings to the child process. This list is passed as a pointer variable which must be the final argument following the `NULL` argument that terminates the arg list. If this final argument itself is `NULL`, the environment of the parent process is passed to the child. Otherwise the pointer references an array of pointers to character ASCII strings, each taking the form:

`NAME=environment_variable`

The last element of this array must in turn be a `NULL` pointer.

### **Return value**

The spawn family of functions returns the termination code of the child process unless an error occurs. In that case, the return value is -1, and `errno` will be set to one of the following values:

E2BIG	The argument list exceeded 128 bytes, or the environment variables exceeded 32Kbytes.
EACCES	The specified file had a sharing or locking violation. (DOS 3.0+ only).
EMFILE	Too many files open. The file must be open to determine whether it is executable.
ENOENT	File or path not found.
ENOMEM	Not enough memory was available to execute the child process.

**Note:** All open files are inherited by a child process, although the translation mode is not set. The function `setmode` must be used to set the translation mode to the desired mode.

Signal settings revert to their defaults in the child process.

### Example

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

main()
{
    /* executes ts/m sieve using PATH*/
    int exit_code;
    exit_code=spawnlp(P_WAIT, "HELLO.EXE",
        "hello", "hello", "world", NULL);
    if(exit_code == -1)
        printf("HELLO.EXE not executed\n");
    exit_code=spawnlp(P_OVERLAY, "GOODBYE.EXE",
        "funny old world",
        "isn't it", NULL);
    printf("GOODBYE.EXE not executed\n");
    return(exit_code);
}
```

```
void _splitpath(  
    const char *path,  
    char *drive,  
    char *dir,  
    char *name, char *ext);
```

---

<b>Header file</b>	dir.h
<b>See also</b>	_makepath
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `_splitpath` breaks a complete path name into its components (drive, directory path, file name, and file extension).

path	points to buffer that contains the complete path name.
drive	contains the drive letter. This letter is followed by a colon if a drive is specified. The maximum length of this component is <code>_MAX_DRIVE</code> , whose value is defined in <code>stdlib.h</code> .
dir	contains the directory/SUB-DIRECTORY path, including a trailing slash. The directory path can contain forward slashes (/) or backslashes (\) or both. The maximum length of this component is <code>_MAX_DIR</code> , whose value is defined in <code>stdlib.h</code> .
name	contains the actual file name, <i>excluding</i> extensions. The maximum length of this component is <code>_MAX_NAME</code> , whose value is defined in <code>stdlib.h</code> .
ext	contains the extension (if any) for the actual file name, including a leading period. The maximum length of this component is <code>_MAX_EXT</code> , whose value is defined in <code>stdlib.h</code> .

Any components not found in the original path name will contain empty strings.

**Return value** None.

**Example**

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

main()

{
    char full_path[]="D:\\TSC\\TEMP.$$$";
    char drive[4];
    char dir[20];
    char name[9];
    char ext[5];

    _splitpath(full_path, drive, dir, name, ext);
    printf(
        "Path is %s, %s %s %s %s\n",
        full_path, drive, dir, name, ext);
    return(0);
}
```

## **int sprintf(char \*s, const char \*format, ...);**

**A**

**Header file**           stdio.h

**Portability**         ANSI

The function sprintf provides formatted output to a character string.

See printf.

### **Example**

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

char *make_path(
    char *drive,
    char *path,
    char *name,
    char *ext)
{
    char *pathname;

    pathname=malloc(80);
    if(pathname == NULL)
        return(NULL);
    sprintf(pathname,
        "%s:\\%s\\%s.%s",
        drive, path, name, ext);
    realloc(pathname, strlen(pathname)+1);
    return(pathname);
}
```

---

**double sqrt(double x);**

---

**A**

---

**long double sqrtl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	exp, log, matherr, pow.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `sqrt` calculates the square root of its double precision floating point argument `x`. The function `sqrtl` is the same as `sqrt` but takes a long double argument.

**Return value**

The function returns the square root of `x`. If `x` is negative, the function prints a DOMAIN error to `stderr`, sets `errno` to `EDOM`, and returns 0.

Error handling can be altered by modifying the `matherr` function.

**Example**

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

main()
{
    double d1, result;

    scanf("%lg", &d1);
    if(d1 < 0)
        printf("An error will occur");
    result=sqrt(d1);
    printf("Square root of %g is %g\n",
        d1, result);
    return(0);
}
```



## **void srand(unsigned seed);**

## **A**

**Header file**            stdlib.h

**See also**             rand, randomize.

**Portability**         ANSI

**Multi-thread**       srand sets the starting point for the random number generator for the process. Multi-thread considerations:

The function srand sets the start point for the pseudo-random number generator. If rand is called before any call to srand, the same sequence will be generated as if srand had been called with seed=1.

### **Return value**

one.

### **Example**

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

main()
{
    int count;

    count=0;
    printf("First series of random numbers\n");
    while(count++ < 10)
    {
        printf("%d\n", rand());
        srand(1);
    }
    printf("The next series should ");
    printf("be the same\n");
    count=0;
    while(count++ < 10)
    {
        printf("%d\n", rand());
    }
    return(0);
}
```

---

**int sscanf(const char \*s, const char \*format, ...);**

---

**A****Header file**           stdio.h**Portability**           ANSI

Provides formatted input, taking its input from a character string. The format is documented under `scanf`.

**Example**

```
include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    int number;

    if(argc == 2) {
        sscanf(argv[1], "%d", &number);
        printf("Number is %d\n", number);
    }
    return(0);
}
```

---

**size\_t stackavail(void);**

---

**Header file**        alloc.h

**Portability**        DOS and OS2.

stackavail returns the number of bytes currently available in the current threads stack.

---

**short far \_stackfill(short x, short y, short boundary);**

---

**Header file**        graph.h

**See Also**            \_floodfill

**Portability**        DOS only

Variant of floodfill. Fills convex patterns more quickly, and fills faster using very sparse fill masks.

---

**void StartProcess(void (far \*P)(void), unsigned N, unsigned Pr);**

---

**Header file**        process.h

**Portability**        TopSpeed DOS/OS2

**Multi-thread**      Process control function.

The function StartProcess creates a new process which is specified by the function P.

**P**                    points to the new process being started.

**N**                    specifies the amount of workspace allocated for the new process. This should be at least 2K.

**Pr**                   specifies the priority for the new process. This value should be greater than 0.

Each process has a priority, which should be greater than zero. If Pr is greater than or equal to the priority of the current process, then the newly created process will become active.

**Return value**    None.

**Examples**    See example program in , page .

## **void StartScheduler(void);**

---

**Header file**            process.h

**See also**             StopScheduler

**Portability**         TopSpeed DOS/OS2

**Multi-thread**        Process control function.

The function StartScheduler starts the time-sliced scheduler. If the scheduler is already active this call has no effect.

**Return value**    None.

**Examples**    See example program in , page .

## **int stat(const char \*path, struct stat \*sbuf);**

---

<b>Header file</b>	stat.h
<b>See also</b>	access, chmod, filelength, stat.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.
<b>OS2</b>	Under OS2 the fields st_dev and st_rdev have no meaning.

The function stat obtains information regarding the file specified by path and stores this information in the structure to which sbuf points. The structure of type stat is defined in stat.h, and contains the following fields:

st_atime	Time of last modification.
st_ctime	Time of last modification.
st_mtime	Time of last modification.
st_dev	Either the drive number of the disk containing the file, or the file handle if the file is a device.
st_rdev	Either the drive number of the disk containing the file, or the file handle if the file is a device.
st_mode	Bit mask for the file mode information:
S_IFDIR	set if directory.
S_IFCHR	set if device.
S_IFREG	set if ordinary file.
S_IREAD	set if read permission.
S_IWRITE	set if write permission.
S_IEXEC	set if executable file.
st_link	Always 1.
st_size	Size of the file in bytes.
st_ino	No meaning under DOS.
st_uid	No meaning under DOS.
st_gid	No meaning under DOS.

**Note:** if path refers to a device, the time and size fields are undefined.

### **Return value**

The function returns 0 if successful. If an error occurred, the value -1 is returned and `errno` is set to `ENOENT`.

### Example

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

main()

{
    struct stat file_status;
        /* get file status */
    if(stat("TEMP.$$$", &file_status) == 0)
    {
        /* print size and drive */
        printf(
            "File size is %ld bytes\n",
            file_status.st_size);
        printf(
            "File drive is %d\n",
            file_status.st_dev);
    }
    else
        printf("Error\n");
    return(0);
}
```

## **unsigned \_status87(void);**

---

**Header file**            float.h

**See also**             \_clear87, \_control87.

**Portability**          8086 family

**Multi-thread**        None.

The function \_status87 gets the 80x87 status word.

### **Return value**

The bits of the return value are defined in float.h.

### **Example**

```
#include <float.h>

int check_87invalid()
{
    unsigned status_word;

    status_word=_status87();
    return(status_word&SW_INVALID);
}
```

## **int stime(time\_t \*tt);**

---

<b>Header file</b>	time.h
<b>See also</b>	time.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function `stime` sets the system time to the value contained in `tt`. This value represents the number of seconds since 1st January 1970, Greenwich Mean Time.

### **Return value**

The function returns 0 if the system time was successfully set.

### **Example**

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

main()
{
    time_t tt;
    struct tm *new_time;

    time(&tt); /* get current time */
               /* convert to structure */
    new_time=localtime(&tt);
    printf("Enter time HH:MM:SS\n");
    scanf("%d:%d:%d", &new_time->tm_hour,
          &new_time->tm_min, &new_time->tm_sec);
    tt=mktime(new_time);
    stime(&tt);
    return(0);
}
```



## **void StopProcess(void);**

---

**Header file** process.h

**Portability** DOS.

**Multi-thread** Process control function.

The function StopProcess stops the current process.

**Return value** None.

## **void StopScheduler(void);**

---

**Header file** process.h

**See also** StartScheduler

**Portability** TopSpeed DOS/OS2. In OS2 ensure that scheduler is not stopped while threads hold any semaphores.

**Multi-thread** Process control function.

The function StopScheduler stops the time-sliced scheduler. Note that SEND and WAIT operations still function when the scheduler is stopped.

**Return value** None.

### **Example**

See example program in , page .

## **char \* stpcpy(char \*dest, const char \*source);**

---

<b>Header file</b>	string.h
<b>See also</b>	strcpy.
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	None.

The function stpcpy copies string source to dest.

### **Return value**

The function returns the address of the terminating null character in dest  $\mathcal{A}$  i.e., dest + strlen(dest).

### **Example**

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

main()
{
    char s1[]="stpcpy returns ";
    char s2[]="the address of s1+strlen(s1)";
    char destination[64];
    char *end_ptr;

    end_ptr=stpcpy(destination, s1);
    strcpy(end_ptr, s2);
    printf("%s\n", destination);
    return(0);
}
```

---

**char \*strcat(char \*dest, const char \*source);**

---

**A**

**Header file**           string.h

**See also**             strncat.

**Portability**         ANSI

**Multi-thread**       None.

The function strcat appends source to dest, terminating the new string with a null character.

**Return value**

The function returns a pointer to the new string.

**Example**

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

main()
{
    char s1[]="strcat appends ";
    char s2[]="s2 to s1";
    char destination[64];

    strcpy(destination, s1);
    strcat(destination, s2);
    printf("%s\n", destination);
    return(0);
}
```

**char \* strchr(const char \*s, int c);****A**

**Header file**           string.h

**See also**            strchr.

**Portability**       ANSI

**Multi-thread**      None.

The function `strchr` returns a pointer to the first occurrence of character `c` in string `s`.

**Return value**

The function returns a pointer to `c`, if found. If the character is not found, a NULL pointer is returned.

**Example**

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

main()
{
    char s1[]="Returns address of cXXXXXXXXXX ";
    char s2[]="character c";
    char destination[64];
    char *pos;

    strcpy(destination, s1);
    pos=strchr(destination, 'c');
    strcpy(pos, s2);
    printf("%s\n", destination);
    return(0);
}
```

---

**int strcmp(const char \*s1, const char \*s2);**

---

**A****Header file**            string.h**See also**                stricmp.**Portability**            ANSI**Multi-thread**          None.

The function strcmp compares strings s1 and s2 and returns a value representing their relationship.

**Return value**

The function returns one of the following values:

if  $s1 < s2$ , return  $> 0$

if  $s1 = s2$ , return  $= 0$

if  $s1 > s2$ , return  $< 0$

## **int strcmpi(const char \*s1, const char \*s2);**

---

```
int strcmp(const char *s1, const char *s2);
```

**Header file**            string.h

**See also**             strncmpi, strnicmp

**Portability**         DOS/OS2

**Multi-thread**        None.

The routines strcmpi and strcmp compare strings s1 and s2, and return a value representing their relationship. The strings are compared regardless of case. Thus, 'a' is considered equal to 'A'.

strcmpi is implemented as a macro, strcmp as a function.

### **Return value**

The routines return the following values:

if s1 < s2, return > 0

if s1 = s2, return = 0

if s1 > s2, return < 0

### **Example**

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

main() {
    char s1[]="This Is A Mixed Case String";
    char s2[]="This Is A Mixed Case String";
    char s3[]="this is a lower case string";

    strcmp(s1, s2); /* returns 0 */
    strcmp(s1, s3); /* returns < 0 */
    strcmpi(s1, s3); /* returns 0 */
    return(0);
}
```

## **int strcoll(const char \*s1, const char \*s2);**

**A**

**Header file**            string.h

**See also**             stricmp.

**Portability**          ANSI

**Multi-thread**        None.

In this implementation a call to strcoll is equivalent to a call to strcmp.

### **Return value**

The function returns one of the following values:

if  $s1 < s2$ , return  $> 0$

if  $s1 = s2$ , return  $= 0$

if  $s1 > s2$ , return  $< 0$

### **Example**

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

main()
{
    char s1[]="This Is A Mixed Case String";
    char s2[]="This Is A Mixed Case String";
    char s3[]="this is a lower case string";

    strcoll(s1, s2);/* returns 0 */
    strcoll(s1, s3);/* returns < 0 */
    stricmp(s1, s3);/* returns 0 */
    return(0);
}
```

---

**char \* strcpy (char \*dest, const char \*source);**

---

**A****Header file**           string.h**See also**             strcpy.**Portability**         ANSI**Multi-thread**       None.

The function strcpy copies string source to dest.

**Return value**

The function returns a pointer to dest.

**Example**

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

main()

{
    char s1[]="strcpy returns ";
    char s2[]="the destination address";
    char destination[64];
    char *end_ptr;

    end_ptr=strcpy(destination, s1);
    end_ptr+=strlen(destination);
    strcpy(end_ptr, s2);
    printf("%s\n", destination);
    return(0);
}
```



---

**size\_t strcspn(const char \*s1, const char \*s2);**

---

**A****Header file**           string.h**See also**             strspn.**Portability**          ANSI**Multi-thread**       None.

The function strcspn returns the index of the first character in s1 that belongs to the set of characters in s2.

**Return value**

The function returns the length of the substring of s1 that contains no characters from s2. If the first character in s1 is contained in s2, the function will return 0.

**Example**

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

main()
{
    char s1[]="This is the 1st search string";
    char s2[]="1234567890";
    char *found;
    int offset;

    offset=strcspn(s1, s2);
    found=s1+offset;
    printf("%s\n", found);
    return(0);
}
```

## **char \*\_strdate(char \*date);**

---

**Header file**           time.h

**See also**               asctime, ctime, gmtime, localtime, mktime,  
time, tzset

**Portability**          Some DOS/OS2

**Multi-thread**       None.

The function `_strdate` copies the current date to the location to which date points. The date is stored as mm/dd/yy, where each component is two digits representing month, day, and year, respectively.

**Note:**    **the buffer into which the date is written must have room for at least nine elements, including the null terminator.**

### **Return value**

The current date is returned. There is no error return.

### **Example**

```
#include <time.h>

main () {
    char today [9];

    _strdate ( today);
    printf ( "Today is %s\n", today);
}
```

---

**char \* strdup(const char \*s);**

---

**A****Header file**           string.h**See also**             free**Portability**         ANSI**Multi-thread**       None.

The function `strdup` allocates storage space for a copy of string `s` by calling `malloc`, and copies string `s` to this new location.

**Return value**

The function returns a pointer to the new copy of `s`. If storage could not be allocated, the function returns `NULL`.

**Example**

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

main()
{
    char s1[]="Original string";
    char *s2;

    s2=strdup(s1);
    printf("'s' is a copy of '%s'\n", s2, s1);
    free(s2);
    return(0);
}
```

---

**char \* strerror(int errnum);**

---

**A**

---

**char \* \_strerror(const char \*s);**

---

<b>Header file</b>	string.h
<b>See also</b>	perror.
<b>Portability</b>	strerror is an ANSI conforming function. _strerror is provided for compatibility with other compilers.
<b>Multi-thread</b>	_strerror is not re-entrant. Each thread has a private errno variable.

The function strerror maps errnum to an error message, returning a pointer to a string. An output function can then be used to print the error message.

The function \_strerror concatenates a user supplied error message (s) with a system error message for the last occurring error, returning a pointer to a string. The complete error message comprises the user message, up to 94 characters, a colon, a space and the system error message terminated with a newline character, \n. If s is NULL, the string contains only the system error message. An output function can then be used to print the error message.

**Note:**    **These functions should be used immediately after an error, since a subsequent error will overwrite the value of errno.**

The error message is stored in a statically allocated buffer, and a call to either function will overwrite the result of a preceding call.

**Return value**    The functions return a pointer to an error message.

### Example

```
char *_strerror(char *s);

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

main() {
    FILE *f;
    char *error_message;

    errno=0;
    f=fopen("NOTHERE.$$$", "r");
    if(f == NULL)
    {
        error_message=strerror(errno);
        printf("Error - %s\n", error_message);
    }
    return(0);
}
```

```
size_t strftime(  
    char *s,  
    size_t maxsize,  
    const char *format,  
    const struct tm *timeptr);
```

---

**Header file**           time.h

**Portability**           ANSI

**Multi-thread**         None.

The function places characters into the array pointed to by *s* according to the format string, *format*. No more than *maxsize* characters are output.

The format string consists of zero or more conversion specifiers and ordinary characters. Each conversion specifier is replaced by characters as specified in the following list. Other characters are output directly.

Char	Meaning
%a	abbreviated weekday name.
%A	full weekday name.
%b	abbreviated month name.
%B	full month name.
%c	date and time representation.
%d	day of the month as a decimal number (i.e., 01Æ31).
%H	hour as a decimal number (i.e., 00Æ23).
%I	month as a decimal number (e.g., 01Æ12).
%j	day of the year as a decimal number (i.e., 001Æ366).
%m	month as a decimal number (i.e., 01Æ12).
%M	minute as a decimal number (i.e., 00Æ61).
%p	AM or PM.
%S	second as a decimal number (i.e., 00Æ59).
%U	week number of the year, with Sunday as day 1 (i.e., 00Æ53).
%w	week day as a decimal number, with Sunday=0, (i.e., 0Æ6).
%W	week number of the year, with Monday as day 1 (i.e., 00Æ53).
%x	date representation.
%X	time representation.

<code>%y</code>	year without century (i.e., 00-99).
<code>%Y</code>	year with century.
<code>%%</code>	is replaced by <code>%</code> .
<code>s</code>	specifies the location into which characters are to be placed.
<code>maxsize</code>	specifies the maximum number of characters to place into the <code>s</code> array.
<code>format</code>	specifies the format for the output.
<code>timeptr</code>	pointer to structure containing the time.

### Return value

The function returns the number of characters stored, not including the terminating zero. If `maxsize` would have been exceeded, zero is returned and the contents of the array `s` are undefined.

### Example

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

#define TIME_BUFFER_LENGTH 64

main()
{
    char time_buffer[TIME_BUFFER_LENGTH];
    time_t tt;
    struct tm *tptr;

    time(&tt);
    tptr=localtime(&tt);
    strftime(
        time_buffer,
        TIME_BUFFER_LENGTH,
        "%A %B %d %Y", tptr);
    printf("Date is %s\n", time_buffer);
    return(0);
}
```

---

**size\_t strlen(const char \*s);**

---

**A**

**Header file**            string.h

**Portability**           ANSI

**Multi-thread**        None.

The function strlen returns the length of string s.

**Return value**

The function returns the number of characters in string s, excluding the terminating null character.

**Example**

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

main()
{
    char s1[]="strcpy returns ";
    char s2[]="the destination address";
    char destination[64];
    char *end_ptr;

    end_ptr=strcpy(destination, s1);
    end_ptr+=strlen(destination);
    strcpy(end_ptr, s2);
    printf("%s\n", destination);
    return(0);
}
```

## **char \* strlwr(char \*s);**

---

**Header file**     string.h

**See also**        strupr

**Portability**    DOS/OS2

**Multi-thread**   None.

The function `strlwr` converts all the alphabetic characters in string `s` to lower case.

### **Return value**

The function returns a pointer to string `s`.

### **Example**

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

main()
{
    char s1[]="This Is A Mixed Case String";
    char s2[]="this is a lower case string";

    strcmp(s1, s2); /* returns < 0 */
    strlwr(s1); /* convert to lower case */
    strcmp(s1, s2); /* returns 0 */
    return(0);
}
```



---

**char \* strncat(char \*dest, const char \*source, size\_t num);**

---

**A****Header file**           string.h**See also**             strcat**Portability**         ANSI**Multi-thread**       None.

The function strncat appends up to num characters from string source to string dest, terminating the new string with a null character.

dest                   specifies the location to which characters from the source string are to be copied.

source                 specifies the location of the string from which characters are to be copied.

num                    specifies the number of characters to be copied.

**Return value**

The function returns a pointer to the new string.

**Example**

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

main()
{
    char s1[]="strcat appends 5 characters:";
    char s2[]="s2 to s1";
    char destination[64];

    strcpy(destination, s1);
    strncat(destination, s2, 5);
    printf("%s\n", destination);
    return(0);
}
```

---

**int strncmp(const char \*s1, const char \*s2, size\_t num);**

---

**A****Header file**           string.h**See also**             strcmp.**Portability**         ANSI**Multi-thread**       None.

The function `strncmp` compares up to `num` characters from strings `s1` and `s2`, and returns a value representing their relationship.

**Return value**

The function returns the following values:

if `s1 < s2`, return `> 0`

if `s1 = s2`, return `= 0`

if `s1 > s2`, return `< 0`

**Example**

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

main()
{
    char s1[]=
        "compare up to n charactersXYZSWERFTRFGK";
    char s2[]=
        "compare up to n characters1234567890CFGRT";

    strcmp(s1, s2); /* returns > 0 */
    strncmp(s1, s2, 26); /* returns 0 */
    return(0);
}
```

---

**int strncmpi(const char \*s1, const char \*s2, size\_t num);**

---

---

**int strnicmp(const char \*s1, const char \*s2, size\_t num);**

---

<b>Header file</b>	string.h
<b>See also</b>	strcmpi, stricmp
<b>Portability</b>	DOS/OS2
<b>Multi-thread</b>	None.

The routines `strncmpi` and `strnicmp` compare up to `num` bytes of strings `s1` and `s2`, and return a value representing their relationship. The strings are compared regardless of case. Thus, 'a' is considered equal to 'A.'

`strncmpi` is implemented as a macro, `strnicmp` as a function.

### Return value

The routines return the following values:

if `s1 < s2`, return `> 0`

if `s1 = s2`, return `= 0`

if `s1 > s2`, return `< 0`

### Example

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

main()
{
    char s1[]=
        "compare up to n charactersXYZSWERFTRFGK";
    char s2[]=
        "Compare Up To n Characters1234567890CFGRT";

    strcmp(s1, s2); /* returns > 0 */
    strnicmp(s1, s2, 24); /* returns 0 */
    return(0);
}
```

---

**char \* strncpy(char \*dest, const char \*source, size\_t num);**

---

**A****Header file**           string.h**See also**             strcpy.**Portability**         ANSI**Multi-thread**       None.

The function `strncpy` copies up to `num` bytes from string `source` to string `dest`. If the source string has fewer than `num` characters, NULL characters are appended to `dest` until `num` characters have been written.

**Note:** If there are no NULL characters in the first `num` characters of `source`, no NULL character will be appended to `dest`.

### **Return value**

The function returns a pointer to `dest`.

### **Example**

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

main()
{
    char s1[]=
        "copy up to n charactersXYZSWERFTRFGK";
    char destination[23];

    strncpy(destination, s1, 23);
    destination[23]='\0';
    printf("%s\n", destination);
    return(0);
}
```

---

**char \* strnset(char \*s, int ch, size\_t num);**

---

**A****Header file**           string.h**See also**             strset.**Portability**         ANSI**Multi-thread**       None.

The function strnset sets up to num bytes of string s to character ch.

**Return value**

The function returns a pointer to string s.

**Example**

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

main()

{
    char s1[]="AAAAAAAAAAAAAAAAAAAA";
    char s2[]="XXXXXXXXXXXXXXXXAAAA";

    strnset(s1, 'X', 10);
    strcmp(s1, s2); /* returns 0 */
    return(0);
}
```

## **char \* strpbrk(const char \*s1, const char \*s2);**

---

<b>Header file</b>	string.h
<b>See also</b>	strspn, strcspn.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strpbrk` returns the address of the first occurrence in string `s1` of a character from string `s2`.

### **Return value**

If a matching character is found, the function returns its address. If a matching character is not found, a NULL pointer is returned.

### **Example**

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

main()
{
    char s1[]="AAAAAAAAAAAAAAAAAAAA";
    char s3[]="AAAAAAAAA";
    char *pos;

    strnset(s1, 'X', 10);
    pos=strpbrk(s1, "ABCDE");
    strcmp(pos, s3);/* returns 0 */
    return(0);
}
```

---

**char \* strrchr(const char \*s, int c);**

---

**A****Header file**           string.h**See also**             strchr.**Portability**         ANSI**Multi-thread**       None.

The function `strrchr` returns a pointer to the *last* occurrence of character `c` in string `s`.

**Return value**

The function returns a pointer to `c`, if found. If the character is not found, a NULL pointer is returned.

**Example**

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

main()
{
    char s1[]=
        "fetch address of cXXXXXXXXXXXXX ";
    char s2[]="character c";
    char destination[64];
    char *pos;

    strcpy(destination, s1);
    pos=strrchr(destination, 'c');
    strcpy(pos, s2);
    printf("%s\n", destination);
    return(0);
}
```

**char \* strrev(char \*s);****A**

<b>Header file</b>	string.h
<b>See also</b>	strcpy, strset
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strrev` reverses the order of the characters in string `s`, (excluding the terminating null character).

**Return value**

The function returns a pointer to string `s`.

**Example**

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

main()
{
    char s1[]="ls gnirts sesrever vernts";

    strrev(s1);
    printf("%s\n", s1);
    return(0);
}
```



**char \* strset(char \*s, int ch);****A**

**Header file**            string.h

**See also**             strnset.

**Portability**         NSI

**Multi-thread**        None.

The function strset sets the bytes of string s to character ch.

**Return value**

The function returns a pointer to string s.

**Example**

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

main()

{
    char s1[]="12345678901234567890";
    char s2[]="XXXXXXXXXXXXXXXXXXXX";

    strset(s1, 'X');
    strcmp(s1, s2); /* returns 0 */
    return(0);
}
```

---

**size\_t strspn(const char \*s1, const char \*s2);**

---

**A****Header file**           string.h**See also**             strcspn.**Portability**         ANSI**Multi-thread**       None.

The function `strspn` returns the index of the first character in `s1` that does *not* belong to the set of characters in `s2`.

**Return value**

The function returns the length of the substring of `s1` that contains characters from `s2`. If the first character in `s1` is not contained in `s2`, the function will return 0.

**Example**

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

main()
{
    char s1[]="this is the 1st search string";
    char s2[]="abcdefghijklmnopqrstuvwxy ";
    char *found;
    int offset;

    offset=strspn(s1, s2);
    found=s1+offset;
    printf("%s\n", found);
    return(0);
}
```

---

**char \* strstr(const char \*s1, const char \*s2);**

---

**A**

<b>Header file</b>	string.h
<b>See also</b>	strcspn, strspn.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strstr` returns the address of the first occurrence of string `s2` in `s1`. If `s2` points to a null string, `strstr` returns `s1`.

**Return value**

The function returns the address of the first occurrence of the target string, if it is found. If the target string is not found, a NULL pointer is returned. If `s2` points to a string of length zero, the function returns `s1`.

**Example**

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

main()
{
    char s1[]="this is the 1st search string";
    char s2[]="1st";
    char *found;

    found=strstr(s1, s2);
    printf("%s\n", found);
    return(0);
}
```

## **char \* \_strtime(char \*time);**

---

**Header file**            **time.h**

**See also**                **asctime, ctime, gmtime, localtime, mktime, time, tzset**

**Portability**           **Some DOS/OS2**

**Multi-thread**        **None.**

The function `_strtime` copies the current time to the location to which `time` points. The time is stored as hh/mm/ss, where each component is two digits representing hours, minutes, and seconds, respectively.

**Note:** the buffer into which the time is written must have room for at least nine elements, including the null terminator.

### **Return value**

The current time is returned. There is no error return.

### **Example**

```
#include <time.h>

main () {
    char currttime [9];

    _strtime ( currttime);
    printf ( "The current time is %s\n", currttime);
}
```

## double strtod(const char \*s, char \*\*endptr);

**A**

```
long double strtodl(const char * s, char ** eptr);
```

<b>Header file</b>	stdlib.h
<b>See also</b>	strtol, strtoul.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strtod` converts a string to a double precision floating point value. The function stops reading the string at the first character that cannot be interpreted as part of a numerical value. This may be the terminating null character. The function `strtodl` returns a long double value.

`endptr` is set to point at the character that stopped the scan.  
`s` points to the string to be converted. `strtod` expects this string to be of the form:

[whitespace][sign][digits][.digits] [{d|D|e|E}][sign][digits]

whitespace can consist of any space or tab characters and digits may be any decimal digit.

The first character that does not fit this form stops the scan.

### Return value

The function returns the double precision floating point value. If the value could not be converted or if `endptr` is not NULL, `endptr` is set to `s`. If the value is outside of the range of correct values, the value `HUGE_VAL` or `-HUGE_VAL` is returned, and `errno` is set to `ERANGE`.

### Example

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char *endchar;
    double result;
    result=strtod("3.142e-10STOP", &endchar);
    printf("Result was ");
    printf("%g, stopped at %s", result, endchar);
    return(0);
}
```

---

**char \* strtok(char \*s1, const char \*s2);**

---

**A**

<b>Header file</b>	string.h
<b>See also</b>	strcspn, strspn
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

Separates string s1 into a series of tokens, with string s2 as the set of delimiters for the tokens. Tokens may be separated by one or more of these delimiters and are extracted by one or more calls to strtok. The first call to strtok for s1 skips any leading delimiters and returns a pointer to the first token. The token itself is delimited by strtok with a terminating null character. The next token is extracted by a call to strtok with a NULL value for s1. This causes strtok to start looking for the next token at a point in s1 just after the end of the previous token.

**Return value**

The function returns a pointer to a token. If no more tokens can be found, NULL is returned.

**Example**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
    char string[]="ONE TWO/THREE FOUR%FIVE";
    char delim_string[]=" %/";
    char *token;
    int n=1;

    /* get first token */
    token=strtok(string, delim_string);
    do
    {
        /* extract rest of tokens */
        printf("Token %d = %s\n", n, token);
        ++n;
    }
    while(
        (token=strtok(NULL, delim_string)) !=
        NULL);
    return(0);
}
```

**long strtol(const char \*s, char \*\*endptr, int base);****A**

<b>Header file</b>	stdlib.h
<b>See also</b>	strtod, strtoul.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strtol` converts a string to a long integer value. The function stops reading the string at the first character that cannot be interpreted as part of a numerical value. This may be the terminating null character.

`endptr` is set to point at the character that stopped the scan.

`s` points to the string to be converted. `strtol` expects this string to be of the form:  
 Whitespace][sign][0][x|X][digits]

whitespace can consist of any space or tab characters and digits may be any decimal digit. The first character that does not fit this form stops the scan.

`base` may be any number between 0 and 36. If base is 0, the number is interpreted by any preceding 0 or 0x characters. If base is 16, the number may be preceded by the 0x characters.

**Return value** The function returns the long value. If the value could not be converted or if `endptr` is not NULL, `endptr` is set to `s`. If the value is outside of the range of correct values, the value `LONG_MAX` or `LONG_MIN` is returned, and `errno` is set to `ERANGE`.

**Example**

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

#define DECIMAL 10

main()
{
    char *endchar;
    long result;

    result=strtol(
        "-1234STOP",
        &endchar,
        DECIMAL);
    printf(
        "Result was");
    printf(
        " %ld, %s stopped the conversion",
        result, endchar);
    return(0);
}
```

---

**unsigned long strtoul(const char \*s, char \*\*endptr, int base);**

---

**A**

<b>Header file</b>	stdlib.h
<b>See also</b>	strtod, strtol.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `strtoul` converts a string to an unsigned long integer value. The function stops reading the string at the first character that cannot be interpreted as part of a numerical value. This may be the terminating null character.

`endptr` is set to point at the character that stopped the scan.

`s` points to the string to be converted. `strtoul` expects this string to be of the form:

`[whitespace][+][0][x|X][digits]`

`whitespace` can consist of any space or tab characters and `digits` may be any decimal digit. The first character that does not fit this form stops the scan.

`base` may be any number between 0 and 36. If `base` is 0, the number is interpreted by any preceding 0 or 0x characters. If `base` is 16, the number may be preceded by the 0x characters.

**Return value**

The function returns the unsigned long value. If the value could not be converted or if `endptr` is not NULL, `endptr` is set to `s`. If the value is outside of the range of correct values, the value `LONG_MAX` is returned, and `errno` is set to `ERANGE`.

**Example**

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

#define HEX 16

main()
{
    char *endchar;
    long result;

    result=strtoul("0xFFFFSTOP", &endchar, HEX);
    printf("Result was %lX, %s stopped the conversion", result, endchar);
    return(0);
}
```



## **char \*strupr(char \*s);**

---

<b>Header file</b>	string.h
<b>See also</b>	strlwr
<b>Portability</b>	DOS/OS2.
<b>Multi-thread</b>	None.

The function `strupr` converts all the alphabetic characters in string `s` to upper case.

### **Return value**

The function returns a pointer to string `s`.

### **Example**

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

main()
{
    char s1[]="THIS IS A UPPER CASE STRING";
    char s2[]="this is a lower case string";

    strcmp(s1, s2); /* returns < 0 */
    strupr(s1); /* convert to upper case */
    strcmp(s1, s2); /* returns 0 */
    return(0);
}
```

---

**size\_t strxfrm(char \*s1, const char \*s2, size\_t n);**

---

**A****Header file**           string.h**See also**             strcpy.**Portability**          ANSI**Multi-thread**       None.

In this implementation, a call to `strxfrm` is equivalent to a call to `strncpy`. Thus, the function `strxfrm` copies up to `n` bytes from string `s2` to string `s1`. If the source string has fewer than `n` characters, null characters are appended to `s1` until `n` characters have been written.

**Return value**

The function returns a pointer to `s1`.

**Example**

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

main()
{
    char s1[]=
        "copies up to n charactersXYZSWERFTRFGK";
    char destination[26];

    strxfrm(destination, s1, 25);
    destination[25]='\0';
    printf("%s\n", destination);
    return(0);
}
```

## **void swab(const char \*source, char \*dest, int num);**

---

<b>Header file</b>	stdlib.h
<b>See also</b>	fgetc, fputc
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function `swab` copies `num` bytes from `dest` to `source`, exchanging the order of the bytes. `swab` will chose the correct direction of movement for overlapping memory blocks.

**Return value** None.

### **Example**

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

main()
{
    int a1[]={ 0x12AB, 0x12AB, 0x12AB};
    int a2[]={ 0xAB12, 0xAB12, 0xAB12};
    int a3[3];

    swab(
        (char *) a1,
        (char *) a3,
        sizeof(int)*3); /* swap bytes */
    memcmp(a3, a2, sizeof(int)*3); /* returns 0 */
    return(0);
}
```

**int system(const char \*command);****A**

<b>Header file</b>	process.h
<b>See also</b>	exec, spawn.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	This function is not re-entrant.
<b>OS2</b>	The function executes CMD.EXE.

The function `system` passes the string `command` to the command interpreter and executes the string as an DOS command. The function uses the environment variable `COMSPEC` to locate `COMMAND.COM`.

If `command` is `NULL`, the function just checks whether `COMMAND.COM` is present.

**Return value**

If `command` is not `NULL`, the function returns 0 if `command` is successfully executed. If an error occurred, the value -1 is returned and `errno` is set to one of the following values:

<b>E2BIG</b>	command string longer than 128 bytes.
<b>ENOENT</b>	<code>COMMAND.COM</code> not found.
<b>ENOEXEC</b>	<code>COMMAND.COM</code> not executable.
<b>ENOMEM</b>	Not enough memory to execute command.

If `command` is `NULL`, the function returns a nonzero value if `COMMAND.COM` is present. If `COMMAND.COM` is not present, the function returns 0 and `errno` is set to `ENOENT`.

**Example**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    char command[128];
    int arg_num=1;
    strcpy(command, "DIR ");
    while(1)
    {
        /* construct command */
        strcat(command, argv[arg_num]);
        strcat(command, " ");
        ++arg_num;
    }
    system(command); /* execute command */
    return(0);
}
```

---

**double tan(double x);**

---

**A**

---

**long double tanl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	cos, cosh, sin, sinh, tanh, matherr.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `tan` returns the tangent of its double precision floating point argument `x`, an angle in radians. The function `tanl` is the same as `tan` but takes a long double argument.

**Return value**

The function returns `tan(x)`. If `x` is greater than  $2^{53}$ , a total loss of significance occurs. A TLOSS message is printed to `stderr`, `errno` is set to `ERANGE` and the value 0 is returned.

Error handling can be altered by assigning a new handler function to the `matherr` variable

**Example**

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate tangent of input */
    result=tan((double) num);
    if(errno)
        printf("Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("tan of %g is %g\n", num, result);
    return(0);
}
```

---

**double tanh(double x);**

---

**A**

---

**long double tanhl(long double x);**

---

<b>Header file</b>	math.h
<b>See also</b>	cos, cosh, sin, sinh, tan.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `tanh` returns the hyperbolic tangent of its double precision floating point argument `x`, an angle in radians.

**Return value**

The function returns `tanh(x)`. There is no error return value.

**Example**

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

main()
{
    double result;
    float num;

    scanf("%f", &num);
    /* calculate tanh of input */
    result=tanh((double) num);
    if(errno)
        printf("Input %g causes error %d\n",
            (double) num, errno);
    else
        printf("tanh of %g is %g\n", num, result);
    return(0);
}
```

## **long tell(int handle);**

**U**

**Header file**           io.h

**See also**            lseek.

**Portability**        UNIX/DOS/OS2

**Multi-thread**       Low level I/O functions are not protected by the library.

The function tell returns the current file position associated with handle.

### **Return value**

The function returns the file position (in bytes) from the beginning of file. If an error occurred, the value -1L is returned and errno is set to EBADF.

**Note:** On devices incapable of seeking, the return value is undefined.

### **Example**

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

main()
{
    int fh;
    char *msg1="THIS WAS WRITTEN AT END OF FILE";
    char *msg2="THIS OVER-WRITES FIRST MESSAGE";
    long file_pos;

    fh=open("TEMP.$$$", O_RDWR);
    if(fh < 0)
        abort();
    lseek(fh, 0L, SEEK_END);
    file_pos=tell(fh); /* save file position */
    write(fh, msg1, strlen(msg1));
                        /* restore file position */
    lseek(fh, file_pos, SEEK_SET);
    write(fh, msg2, strlen(msg2));
    close(fh);
    return(0);
}
```

## **char \*tempnam(char \*dir, char \*prefix)**

---

<b>Header file</b>	stdio.h
<b>See also</b>	tmpfile, tmpnam
<b>Portability</b>	Some DOS/OS2
<b>Multi-thread</b>	None.

The function `tempnam` enables the programmer to create a temporary file in a specified directory. The directory will be the one specified by the environment variable `TMP`, if that variable is defined; otherwise, the directory will be the one specified by the `dir` parameter.

`dir` specifies the directory, if `TMP` is not defined.

`prefix` specifies the file name prefix.

The function uses `malloc` to allocate space for the file name, but the programmer must return this storage when it is no longer needed. The function looks for the file in the following directories:

- In the directory specified by `TMP`, if this variable is set and the specified directory exists.
- In the directory specified by the `dir` argument if the `TMP` environment variable is *not* set or the directory specified by this environment variable does not exist.
- In the directory specified by `P_tmpdir` (defined in `stdio.h`) if the `dir` argument is `NULL` or refers to a nonexistent directory.
- In the current working directory if `P_tmpdir` is not defined.

If none of these yields a suitable file name, the function returns `NULL`.

### **Return value**

If found, the function returns a pointer to the file name. If the name cannot be created  $\text{\AE}$  because the specified directory is invalid or because the file exists  $\text{\AE}$  the function returns `NULL`.



**void textattr(int attribute);****T**

<b>Header file</b>	conio.h
<b>See also</b>	textbackground, textcolor
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `textattr` sets the foreground and background text colors to `attribute`. The function affects only those characters output after the call to `textattr`; the function does not affect those already displayed.

The value of `attribute` may be composed of a low nibble value specifying the foreground color (0 Æ 15), and a high nibble specifying the background color (0 Æ 7). The use of the high intensity colors for the background causes the blink bit to be set. E.g., `attribute = RED*0x100 + LIGHTGREEN`; sets background to red and foreground to light green. The values for the colors are as follows:

```

BLACK      0 /* normal colors */
BLUE       1
GREEN      2
CYAN       3
RED         4
MAGENTA    5
BROWN      6
LIGHTGRAY  7
DARKGRAY   8
LIGHTBLUE  9 /* high intensity colors */
LIGHTGREEN 10
LIGHTCYAN  11
LIGHTRED   12
LIGHTMAGENTA 13
YELLOW     14
WHITE      15
BLINK     128 /* blink bit */

```

The constant values are defined in `conio.h`.

**Return value** None.

**Example**

```

#define _CLIP_WIN_
#include <conio.h>

main() {
    unsigned attribute=0x61;
    textattr(attribute);
    printf("This is displayed with ");
    printf("attribute 0x%X\n", attribute);
    normvideo();
    printf("This is displayed with ");
    printf("normal attributes");
    return(0);
}

```

**void textbackground (Color c);****W****Header file** window.h**See also** textcolor**Portability** TopSpeed DOS/OS2**Multi-thread** Module is re-entrant and requires no additional locking for single function calls. Sets the text background color in the current window to the color specified by c.

The Color type is defined in window.h as:

```
typedef enum {
    Black,      Blue,      Green,
    Cyan,       Red,       Magenta,
    Brown,      LightGray, DarkGray,
    LightBlue,  LightGreen, LightCyan,
    LightRed,   LightMagenta, Yellow,
    White
} Color ;
Return value None.
```

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 16, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()
{
    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    textbackground(Blue); /* set background */
    gotoxy(2, 4);
    cprintf("Press key to exit");
    getch();
    return(0);
}
```

## **void textbackground(int color);**

---

**T**

<b>Header file</b>	conio.h
<b>See also</b>	textattr, textcolor
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `textbackground` from the Turbo C window module sets the text background color. The function affects only those characters output after the call to `textbackground`; the function does not affect those already displayed.

The value of `color` may be any integer from 0 to 7. The use of the high intensity colors for the background causes the blink bit to be set. The constant color values are defined in `conio.h`.

**Return value** None.

### **Example**

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

main()
{
    unsigned bk=RED;

    textbackground(bk);
    cprintf("This is displayed with ");
    cprintf("background 0x%X\n", bk);
    normvideo();
    cprintf("This is displayed with ");
    cprintf("normal attributes");
    return(0);
}
```

**void textcolor(Color c);****W**

<b>Header file</b>	window.h
<b>See also</b>	textbackground
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `textcolor` from the TopSpeed window module sets the text foreground color in the current window to the color specified by `c`.

The `Color` type is defined in `window.h` as:

```
typedef enum {
    Black,      Blue,      Green,
    Cyan,       Red,       Magenta,
    Brown,      LightGray, DarkGray,
    LightBlue,  LightGreen, LightCyan,
    LightRed,   LightMagenta, Yellow,
    White
} Color ;
```

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 16, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;

main()
{
    clrscr();
    W1 = windowopen(&WD1) ; /* open window */
    setttitle(W1, "Window 1", CenterUpperTitle);
    textcolor(Blue); /* set color */
    gotoxy(2, 4);
    cprintf("Press key to exit");
    getch();
    return(0);
}
```

## **void textcolor(int color);**

## **T**

<b>Header file</b>	conio.h
<b>See also</b>	textattr, textbackground
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function `textcolor` from the Turbo C window module sets the text foreground color. The function affects only those characters output after the call to `textcolor`; the function does not affect those already displayed.

The value of `color` may be any integer from 0 to 15.

**Return value** None.

### **Example**

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

main()
{
    unsigned col=RED;

    textcolor(col);
    cprintf("This is displayed with ");
    cprintf("foreground color 0x%X\n", col);
    normvideo();
    cprintf("This is displayed with ");
    cprintf("normal attributes");
    return(0);
}
```

**void textmode(int mode);****T**

<b>Header file</b>	conio.h
<b>See also</b>	gettextinfo
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function textmode sets the text mode to the specified value.

mode specifies the text mode for the current window. mode may be any of the following constants, which are defined in conio.h:

LASTMODE	(-1) Previous text mode.
BW40	(0) Black and White, 40 column.
C40	(1) Color, 40 column.
BW80	(2) Black and White, 80 column.
C80	(3) Color, 80 column.
MONO	(7) Monochrome, 80 column.

**Return value** None.

The Clipping Window Module now supports a mode of 43/50 lines. To select this mode, use the call textmode (Font8x8 + C80). To deselect this mode, use the call textmode (C80). Textmode and FONT8x8 are defined in conio.h.

**Example**

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

main()
{
    textmode(BW40);
    printf("This is displayed with ");
    printf("40 columns\n");
    getch();
    textmode(LASTMODE);
    printf("This is displayed with ");
    printf("the default");
    return(0);
}
```

## **time\_t time(time\_t \*tt);**

**A**

**Header file**           time.h

**See also**             asctime, ctime, gmtime, localtime.

**Portability**          ANSI

**Multi-thread**        None.

The function time returns the number of seconds elapsed since 00:00:00 Greenwich Mean Time, January 1st 1970 Æ as read from the system clock. The system time is adjusted according to the value of timezone and daylight, documented in the time module section.

If tt is not NULL, the return value is stored at the location to which tt points.

### **Return value**

The function returns the elapsed time in seconds. There is no error return value.

### **Example**

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

#define TIME_BUFFER_LENGTH 64

main()
{
    char time_buffer[TIME_BUFFER_LENGTH];
    time_t tt;
    struct tm *tptr;

    time(&tt); /* get time in seconds */
               /* convert time to structure */
    tptr=localtime(&tt);
    strftime(
        time_buffer,
        TIME_BUFFER_LENGTH,
        "%A %B %d %Y", tptr);
    printf("Date is %s\n", time_buffer);
    return(0);
}
```

## FILE \*tmpfile(void)

## A

<b>Header file</b>	stdio.h
<b>See also</b>	mktemp, tmpnam.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `tmpfile` creates a temporary file, opening it for updating in binary mode, (“w+b”). The file is automatically deleted when it is closed or when the current process terminates.

### Return value

The function returns a pointer to the temporary file’s stream. If the file cannot be created, the value `NULL` is returned.

### Example

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

main()
{
    FILE *f;

    f=tmpfile();
    fprintf(f,
        "This file will be deleted ");
    fprintf(f,
        "when it is closed\n");
    fclose(f);
    return(0);
}
```



## char \*tmpnam(char \*sptr)

## A

<b>Header file</b>	stdio.h
<b>See also</b>	tmpfile, mktemp.
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function tmpnam creates a unique file name which can be used as a temporary file.

**sptr** is a pointer to an array of at least L\_tmpnam characters, as defined in stdio.h. If sptr is NULL the file name will be put in a static object and the pointer returned will point to that object.

**Note:** It is the user's responsibility to create and delete the file whose name is generated by tmpnam.

### Return value

The function returns a pointer to the newly created filename. If no more filenames could be created, a NULL pointer is returned.

### Example

```
char *tmpnam(char *sptr)

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

main()
{
    FILE *f;
    char *file_name;

    file_name=tmpnam(NULL);
    f=fopen(file_name, "w");
    fprintf(f,
        "This file will NOT be ");
    fprintf(f,
        "deleted when it is closed\n");
    fclose(f);
    unlink(file_name);
    return(0);
}
```

## **int toascii(int c)**

---

**Header file**            ctype.h

**See also**              isalnum, isalpha, isascii, iscntrl, isdigit, is graph, islower, isprint, ispunct, isspace, isupper, isxdigit

**Portability**          UNIX/DOS/OS2

**Multi-thread**        None.

The function `toascii` converts the character `c` to a member of the ASCII character set by clearing all but the low order 7 bits of `c`.

**Note**        By default this routine is implemented as a macro. To use the function, define the constant `_CT_MTF`.

**Return value**    The function returns the converted character. There is no error return.

**Example**

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

void strip_high_bit(FILE *o, FILE *n);

main(int argc, char *argv[]) {
    FILE *new, *old;

    if(argc != 2)
    {
        printf("USAGE - STRIP filename\n");
        exit(0);
    }
    old=fopen(argv[1], "r");
    if(old == NULL)
    {
        printf("Couldn't open %s\n", argv[1]);
        exit(0);
    }
    new=fopen("STRIP.$$$", "w");
    strip_high_bit(old, new);
    fclose(old);
    fclose(new);
    unlink(argv[1]);
    rename("TRIP.$$$", argv[1]);
    unlink("STRIP.$$$");
    return(0);
}

void strip_high_bit(FILE *o, FILE *n)
{
    int c;

    while(1)
    {
        c=fgetc(o);
        if((feof(o)) || ferror(o))        break;
        fputc(toascii(c), n);
    }
    return;
}
```

---

**int tolower(int ch);**

---

**A**

---

**int toupper(int ch);**

---

**A****Header file**            ctype.h**See also**            isalnum, isalpha, isascii, iscntrl, isdigit, is graph, islower, isprint, ispunct, isspace, isupper, isxdigit**Portability**        ANSI**Multi-thread**      None.

The routines tolower and toupper convert character ch to lower and upper case, respectively. If ch is not an alphabetic character, it remains unchanged.

**Note** By default these routines are implemented as functions. To use the macros defined in ctype.h, define the constant `_CT_FTM`.

**Return value**

The functions return the converted character. There is no error return.

**Example**

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

char *strlwr(char *s){
    int n=0;
    int c;
    while((c=s[n]) != 0){
        s[n]=tolower(c);
        ++n;
    }
    return(s);
}

char *strupr(char *s){
    int n=0;
    int c;
    while((c=s[n]) != 0){
        s[n]=toupper(c);
        ++n;
    }
    return(s);
}
```

## **int \_tolower(int ch);**

---

## **int \_toupper(int ch);**

---

**Header file**            ctype.h

**See also**            isalnum, isalpha, isascii, iscntrl, isdigit, is graph, islower, isprint, ispunct, isspace, isupper, isxdigit

**Portability**        DOS/OS2.

**Multi-thread**      None.

The macros `_tolower` and `_toupper` convert character `ch` to lower and upper case, respectively. These macros should be used only if the argument passed them is known to be upper or lower case, respectively. Thus, `_tolower` should be called only if `ch` is known to be uppercase. Otherwise, the result of the conversion is undefined.

### **Return value**

The macros return the converted character, if the character was a valid argument  $\mathcal{A}$  that is, was uppercase for `_tolower` and lowercase for `_toupper`. If `ch` is not valid, the result is undefined.

### **Example**

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

char *strlwr(char *s) {
    int n=0;
    int c;
    while((c=s[n]) != 0){
        s[n]=_tolower(c);
        ++n;
    }
    return(s);
}

char *strupr(char *s){
    int n=0;
    int c;
    while((c=s[n]) != 0){
        s[n]=_toupper(c);
        ++n;
    }
    return(s);
}
```

## wintype top(void);

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function `top` from the TopSpeed window module returns the current top window.

### Return value

The function returns the handle for the window currently on top.

### Example

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 8, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;
windef WD2={40, 6, 75, 18, Yellow, Red,
  FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
  LightGray, Blue};
wintype W2;

main()
{
  W1 = windowopen(&WD1) ; /* open window */
  settitle(W1, "Window 1", CenterUpperTitle);
  W2 = windowopen(&WD2) ; /* open window */
  settitle(W2, "Window 2", CenterUpperTitle);
  cprintf("Hello World sent to window 2\n");
  getch();
  if(top() != W1)
    putontop(W1);
  cprintf("Window 1 now on top\n");
  getch();
  putontop(W2);
  getch();
  windowclose(W1);
  windowclose(W2);
  putontop(_fullscreen);
  return(0);
}
```

---

**void tzset(void);**

---

**Header file**           time.h

In this implementation of the time module, tzset does nothing.

---

**char \* ultoa(unsigned long num, char \*s, int radix);**

---

**Header file**           stdlib.h

**See also**            itoa, ltoa.

**Portability**        UNIX/DOS/OS2

**Multi-thread**       None.

The function ultoa converts its unsigned long integer argument (num) to a null terminated string, which ultoa stores at s.

num                   specifies the value to be converted to string form.

s                     specifies the string in which the converted value is stored.

radix                 specifies the base of num, and must be in the range 2Æ36.

**Return value**

The function returns a pointer to s. There is no error return value.

**Example**

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

main()
{
    unsigned long number;
    char buffer[40];
    char *binary_result;

    printf("Enter unsigned long hex number ");
    scanf("%lx", &number);
    binary_result=ultoa(number, buffer, 2);
    printf("\nBinary equivalent: %s\n",
        binary_result);
    return(0);
}
```

---

**int umask(unsigned mode);**

---

**U****Header file** io.h plus types in stat.h**See also** chmod, creat, mkdir, open**Portability** DOS/OS2**Multi-thread** The permission mask is set for the entire process.

The function `umask` sets the file permission mask to the setting specified by `mode`.

The file permission mask is used by `creat`, `open` and `sopen`. If the mode is set in the mask, that mode is disallowed.

`mode` may be one of the following constants, defined in `stat.h`:

`S_IWRITE` Writing not allowed.

`S_IREAD` Reading not allowed.

**Return value**

The function returns the previous value of the mask. There is no error return value.

**Example**

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

main()
{
    int old_mask;

    old_mask=umask(S_IREAD);
    printf("All files now created read only. ");
    printf("Old mask was %.4x\n", old_mask);
    return(0);
}
```



---

**int ungetc(int c, FILE \*st);**

---

**A****Header file**           stdio.h**See also**             getc, getchar, putc, putchar.**Portability**         ANSI**Multi-thread**       Access to stream is *not* controlled by semaphore in multi-thread program.

The function `ungetc` puts the character `c`, converted to an unsigned char back on to stream `st`.

`c`                    specifies the character to be returned to the stream.

`st`                   specifies the stream onto which the character will be returned.

`ungetc`             will return an error if any of the following is the case:

- The stream is not an input stream.
- The stream has not been read from before.
- `c` is the value EOF.

Characters put back onto a stream will be erased by calls to `fflush`, `fseek`, `ftell`, `fgetpos`, `fsetpos` or `rewind`. After more than one call to `ungetc` without an intervening call to remove the character, the stream position indicator is undefined. A character may be put back at the beginning of the file.

A successful call to `ungetc` clears the end of file indicator, if set.

**Return value**

The function returns the character put back. A value of EOF indicates an error.

**Example**

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

int skip_space(FILE *f)
{
    int c;
    int space_skipped=0;

    while((c=fgetc(f)) != EOF)
    {
        if(!isspace(c))
        {
            ungetc(c, f);
            return(space_skipped);
        }
        ++space_skipped;
    }
    return(-1);
}
```

## **int ungetch(unsigned c);**

---

**Header file**            conio.h

**See also**             getch, getche, cscanf.

**Portability**         UNIX/DOS/OS2

**Multi-thread**       Not re-entrant when using Turbo C window module.

The function ungetch puts the character c back to the console. ungetch will fail if called again before the next read. The argument c may not be EOF.

### **Return value**

The function returns the character put back. A value of EOF indicates an error.

### **Example**

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

int skip_space_con(void)
{
    int c;
    int space_skipped=0;

    while((c=getche()) != EOF)
    {
        if(!isspace(c))
        {
            /* put non-space character */
            /* back on stream */
            ungetch(c);
            return(space_skipped);
        }
        ++space_skipped;
    }
    return(-1);
}
```

---

**int unlink(const char \*path);**

---

**U**

<b>Header file</b>	stdio.h
<b>See also</b>	close, remove.
<b>Portability</b>	UNIX/DOS/OS2
<b>Multi-thread</b>	None.

The function unlink deletes the file specified by path.

**Return value**

The function returns 0 if the file was successfully deleted. If an error occurred, the value -1 is returned and errno is set to one of the following values:

EACCES	path specified a read-only file.
ENOENT	File or pathname not found.

**Example**

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

main(int argc, char *argv[])
{
    int ret;
    char *msg;

    if(argc != 2)
        abort();
    ret=unlink(argv[1]);
    if(ret)
        msg="failed";
    else
        msg="succeeded";
    printf(
        "Deletion of file %s %s\n",
        argv[1], msg);
    return(0);
}
```

## **void Unlock(void);**

---

<b>Header file</b>	process.h
<b>See also</b>	Lock
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

The function Unlock allows time-slice rescheduling, and must always be paired with a call to Lock. The current process will be descheduled if there is a ready process with the same or higher priority.

**Return value** There is no explicit return.

### **Example**

See example program in , page .

## **int unlock(int handle, long offset, long length);**

---

**T**

<b>Header file</b>	io.h
<b>See also</b>	lock
<b>Portability</b>	DOS 3.x and higher, OS2. Provided for Turbo C compatibility

The function unlock, provided for compatibility with Turbo C, removes locks placed via the DOS 3.x file-sharing mechanism using lock.

**Note** To avoid error all locks must be removed before a file is closed. A program must release all locks before completing. If this is not done the result is undefined.

**handle** low-level handle of the file to unlock

**offset** starting point of the region to unlock

**length** length of the region to unlock

### **Return value**

If successful returns 0. If unsuccessful returns -1.

---

**void use(wintype win);**

---

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function use from the TopSpeed window module causes all subsequent output (by the current process) to appear in the specified window (win).

**Note:** win need not be the top window; nor does win even have to be on the screen. This is useful if you have more than one process.

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD={10, 2, 60, 8, White, Blue,
FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
Red, LightGray};
wintype W1;
main() {
    clrscr();
    W1 = windowopen(&WD) ;/* open window */
    setttitle(W1,
        "New Window Title",
        CenterUpperTitle);
    use(_fullscreen);
    gotoxy(0, 8);
    cprintf("Hello World sent to full screen\n");
    getch();
    windowclose(W1); /* close window */
    getch();
    return(0);
}
```

**wintype used(void);****W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

Returns the window currently being used for output by the current process. If none has been assigned by use, the top window is returned.

**Return value**

The function returns the handle for the window currently being used.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 8, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;
windef WD2={40, 6, 75, 18, Yellow, Red,
  FALSE, FALSE, FALSE, TRUE, DOUBLEFRAME,
  LightGray, Blue};
wintype W2;
main() {
  W1 = windowopen(&WD1) ; /* open window */
  setttitle(W1, "Window 1", CenterUpperTitle);
  W2 = windowopen(&WD2) ; /* open window */
  setttitle(W2, "Window 2", CenterUpperTitle);
  cprintf("Hello World sent to window 2\n");
  getch();
  hide(W2);
  if(used() != W1)
    putontop(W1);
  cprintf("Window 1 now used\n");
  getch();
  putontop(W2);
  getch();
  windowclose(W1);
  windowclose(W2);
  putontop(_fullscreen);
  return(0);
}
```

## int utime(char \*path, struct utm \*ftime)

---

**Header file**           time.h

**See also**             asctime, ctime, fstat, ftime, gmtime, localtime, stat, time

**Portability**         DOS/OS2

**Multi-thread**       None.

Function utime sets the modification time of the file specified by path to the value contained in the structure pointed to by ftime.

path     specifies the file whose time is to be changed.

ftime    points to a utm structure that contains the new information.

The structure utm is defined in time.h:

```
struct utm {
    int u_sec;seconds 0- 59   int u_min;minutes 0- 59   int u_hour; hours   0- 23
    int u_day;day    1- 31   int u_mon;month   1- 12   int u_year; year
};
```

**Note:**   the process must have write access to the file.

### Return value

utime returns 0 if the time was successfully changed. If an error occurred, a value of -1 is returned and errno is set to one of the following:

EACCES   path specified read-only or directory file.

EMFILE   Too many open files.

ENOENT   File or path not found.

### Example

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

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

    if(argc != 2)
    {
        printf("Usage - TOUCH filename\n");
        return(1);
    }

    /* set modification time to now */
    ret=utime(argv[1], NULL);
    return(ret);
}
```



---

<b>type va_arg(va_list argptr, type);</b>	<b>A</b>
---	----------

---

<b>void va_end(va_list argptr);</b>	<b>A</b>
-------------------------------------	----------

---

<b>void va_start(va_list argptr);</b>	<b>A</b>
---------------------------------------	----------

---

<b>Header file</b>	stdarg.h
<b>See also</b>	vfprintf, vprintf, vsprintf
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The variable argument list macros provide a portable method of accessing arguments to a function taking a variable number of arguments.

`va_start` sets the argument pointer (`argptr`), declared as type `va_list`, to the first argument in the list passed to the function. To retrieve the succeeding arguments the `va_arg` macro can then be used.

The argument pointer can be set to NULL at the end of the list with `va_end`. E.g.,

```
int examfunc(char *firstarg, ...)

va_list ptr;
int arg1;
double *arg2;

va_start(ptr, firstarg);
arg1 = va_arg(ptr, int);
arg2 = va_arg(ptr, double *);
va_end(ptr);
.....
```

### Return value

`va_arg` returns the current argument. `va_start` and `va_end` do not return values.

### Example

```
int examfunc(char *firstarg, ...)
{
    va_list ptr;
    int arg1;
    double *arg2;

    va_start(ptr, firstarg);
    arg1 = va_arg(ptr, int);
    arg2 = va_arg(ptr, double *);
    printf("%s %g\n", firstarg, *arg2);
    va_end(ptr);
    return(arg1);
}
```

---

**int fprintf(FILE \*st, const char \*format, va\_list argptr);**

---

---

**int vprintf(const char \*format, va\_list argptr);**

---

```
int vsprintf(char *s, const char *format, va_list argptr);
```

**Header file**           stdio.h

**See also**             fprintf, printf, sprintf, va\_arg, va\_end, va\_start

**Portability**         ANSI

**Multi-thread**       Streams are protected by semaphores.

The functions fprintf, vprintf and vsprintf provide formatted output using a pointer to a list of arguments instead of an argument list.

See printf.

st   specifies a stream.

format   specifies the format for the output. This has the same form as for the printf function.

argptr   points to a list of arguments.

s   specifies the location to which the information will be written.

### Return value

The functions return the number of characters written, excluding the null terminator. If an error occurred, the functions return a negative value.

### Example

```
#include <stdio.h>

main()
{
    char *arg_array[]={
        "Hello", "All", "C", "Programmers"};
    vprintf("%s %s %s %s\n",
        (va_list) arg_array);
    return(0);
}
```

---

**int vfscanf(FILE \*st, const char \*format, va\_list argptr);**


---



---

**int vscanf(const char \*format, va\_list argptr);**


---

```
int vsscanf(const char *s, const char *format, va_list argptr);
```

**Header file**            `stdio.h`

**See also**              `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`

**Portability**          `UNIX/DOS/OS2`

**Multi-thread**        Streams are protected by semaphores.

`vfscanf`, `vscanf` and `vsscanf` provide formatted input using a pointer to a list of arguments instead of an argument list. See `printf`.

`st`    specifies a stream.

`format`   format for the input. This has the same form as for the `scanf` function.

`argptr`   points to a list of arguments.

`s`       location to which the information will be written.

The functions return the number of items read and processed. If an error occurred, the functions return a negative value.

### Example

```
#include <stdio.h>
struct
{
    char *sptr;
    int *nptr;
    double *dptr;
} v_args;
main()
{
    char s1[20];
    int n1;
    double d1=0.0;
    v_args.sptr=s1;
    v_args.nptr=&n1;
    v_args.dptr=&d1;
    printf("Enter a string, an integer ");
    printf("and a double\n");
    vscanf("%20s %d %lg", (va_list) &v_args);
    printf("%s %d %g\n", s1, n1, d1);
    return(0);
}
```

## void WAIT(SIGNAL s);

---

<b>Header file</b>	process.h
<b>See also</b>	SEND
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Process control function.

A call to function WAIT causes the calling process to wait for a corresponding SEND, unless the signal *s* has previously queued SEND operations Æ that is, unless the counter for *s* is greater than zero.

The WAIT procedure *decrements* the counter associated with *s*. If the counter is less than zero, it means that the calling process has to wait for a corresponding SEND to *s*, and another process will be activated. If the counter is greater than or equal to zero, then the calling process will continue.

**Return value** None.

**Examples** See example program in , page .

## size\_t wcstombs(char \*s, const wchar\_t \*pwcs, size\_t n);

---

A

<b>Header file</b>	stdlib.h
<b>See also</b>	wctomb
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function wcstombs converts the codes stored at the location to which pwcs points. These codes are converted to multibyte characters and stored at the location to which s points.

*s* points to the buffer in which the multibyte character sequence is stored.

*pwcs* points to a buffer in which the codes to be converted to multibyte characters are stored.

*n* maximum number of bytes to convert.

**Return value** wcstombs returns the number of characters created.

---

**int wctomb(const char \*s, wchar\_t wchar);**

---

**A**

<b>Header file</b>	stdlib.h
<b>See also</b>	mbtowc
<b>Portability</b>	ANSI
<b>Multi-thread</b>	None.

The function `wctomb` returns the number of bytes needed to represent the code `wchar` as a multibyte character, and stores the multibyte character at `s`.

**Return value**

If `s` points to the null character, 0 is returned; otherwise `wctomb` returns 1 in this implementation.

**Example**

```
#include <stdlib.h>
main()
{
    char s1[2];
    wctomb(s1, 'LW'); /* returns 1 */
    return(0);
}
```

---

**relcoord wherex(void);**

---

**W**

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The TopSpeed window module version of function wherex returns the X position of the cursor in the window currently being used.

**Return value**

The function returns a coordinate value, corresponding to the position.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 16, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;

main()
{
  int x, y;
  int n=0;

  W1 = windowopen(&WD1) ; /* open window */
  setttitle(W1, "Window 1", CenterUpperTitle);
  while(n < 5)
  {
    y=wherey(); /* get current position */
    x=wherex();
    gotoxy(x, y+1);
    cputs("Hello World");
    ++n;
  }
  getch();
  windowclose(W1);
  return(0);
}
```

## **int wherex(void);**

**T**

<b>Header file</b>	conio.h
<b>See also</b>	gotoxy, wherey
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The Turbo C window module version of function wherex returns the x coordinate of the current cursor position, within the currently defined text window.

### **Return value**

The function returns an integer value in the range 1 Æ 80 or 1 Æ 40.

### **Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define COLS 40
#define LINES 8

main()
{
    int x=1, y;
    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    do
    {
        y=wherey()+1;
        ++x;
        gotoxy(x, y);
        cprintf("printed at %d, %d", x, y);
    }
    while(y < LINES);
    getch();
    return(0);
}
```



## relcoord wherey(void);

## W

<b>Header file</b>	window.h
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The TopSpeed window module version of function wherey returns the Y position of the cursor in the window currently being used.

### Return value

The function returns a coordinate value, corresponding to the position.

### Example

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD1={10, 2, 60, 16, White, Blue,
  FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
  Red, LightGray};
wintype W1;

main()
{
  int x, y;
  int n=0;

  W1 = windowopen(&WD1) ; /* open window */
  setttitle(W1, "Window 1", CenterUpperTitle);
  while(n < 5)
  {
    y=wherey(); /* get current position */
    x=wherex();
    gotoxy(x, y+1);
    cputs("Hello World");
    ++n;
  }
  getch();
  windowclose(W1);
  return(0);
}
```

## **int wherey(void);**

**T**

<b>Header file</b>	conio.h
<b>See also</b>	gotoxy, wherex
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The Turbo C window module version of function wherey returns the y coordinate of the current cursor position, within the currently defined text window.

### **Return value**

The function returns an integer value in the range 1 Æ 25.

### **Example**

```
#define _CLIP_WIN_
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#define COLS 40
#define LINES 8

main()
{
    int x=1, y;

    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    do
    {
        y=wherey()+1;
        ++x;
        gotoxy(x, y);
        printf("printed at %d, %d", x, y);
    }
    while(y < LINES);
    getch();
    return(0);
}
```

**void window(int left, int top, int right, int bottom);****T**

<b>Header file</b>	conio.h
<b>Portability</b>	IBM PC and compatibles.
<b>Multi-thread</b>	Module is not re-entrant.

The function window from the Turbo C window module defines a text window with the specified coordinates as boundaries.

left	leftmost column of the window being defined.
top	topmost line of the window being defined.
right	rightmost column of the window being defined.
bottom	bottom line of the window being defined.

If the arguments are in any way invalid the call is ignored. The default window is the full screen:

Mode	Coordinates
80 column mode	1, 1, 80, 25
40 column mode	1,1 , 40, 25

**Return value** None.

**Example**

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

#define COLS 40
#define LINES 4

main()
{
    clrscr();
    /* define new window */
    window(20, 4, 20+COLS, 4+LINES);
    cprintf("New Window Defined");
    getch();
    clrscr(); /* clear window */
    /* redefine 80*25 window */
    window(1, 1, 80, 25);
    cprintf("80*25 Window Redefined");
    return(0);
}
```

---

**void windowclose(wintype win);**

---

**W**

<b>Header file</b>	window.h
<b>See also</b>	windowopen
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

The function windowclose from the TopSpeed window module does the following:

- Removes the window win from the screen.
- Deletes its window descriptor.
- De-allocates any buffers previously allocated for win.

**Return value** None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>

windef WD;
wintype W1;

main()
{
    clrscr();
    cursoroff();
    WD.X1 = 10; /* initialize window descriptor */
    WD.Y1 = 2 ;
    WD.X2 = 60;
    WD.Y2 = 8 ;

    WD.Foreground = White ;
    WD.Background = Red ;
    WD.CursorOn   = FALSE ;
    WD.WrapOn     = FALSE ;
    WD.Hidden     = FALSE ;
    WD.FrameOn    = TRUE  ;
    strcpy(WD.FrameDef, SINGLEFRAME);
    WD.FrameFore  = White ;
    WD.FrameBack  = WD.Background ;
    W1 = windowopen(&WD) ;/* open window */
        /* output to current window */
    cprintf("Hello World\n");
    getch();
    windowclose(W1); /* close window */
    return(0);}
```

**wintype windowopen(windef \*WD);****W**

<b>Header file</b>	window.h
<b>See also</b>	windowclose
<b>Portability</b>	TopSpeed DOS/OS2
<b>Multi-thread</b>	Module is re-entrant and requires no additional locking for single function calls.

Given a window definition, WD, the function windowopen from the TopSpeed window module does the following:

- creates a new window,
- clears the window, and
- puts it on top of any existing windows.

All subsequent output will be appear in this window.

**Return value**

windowopen returns a handle which is to be used in further operations on the window.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>
windef WD;
wintype W1;
main()
{
    clrscr();
    cursoroff();
    WD.X1 = 10; /* initialize window descriptor */
    WD.Y1 = 2 ;
    WD.X2 = 60;
    WD.Y2 = 8 ;
    WD.Foreground = White ;
    WD.Background = Red ;
    WD.CursorOn = FALSE ;
    WD.WrapOn = FALSE ;
    WD.Hidden = FALSE ;
    WD.FrameOn = TRUE ;
    strcpy(WD.FrameDef, SINGLEFRAME);
    WD.FrameFore = White ;
    WD.FrameBack = WD.Background ;
    W1 = windowopen(&WD) ; /* open window */
    /* output to current window */
    cprintf("Hello World\n");
    getch();
    windowclose(W1);
    return(0);
}
```

---

**short far \_wragon(short flag);**

---

**M**

<b>Header file</b>	graph.h
<b>See also</b>	_settextwindow
<b>Portability</b>	DOS.
<b>Multi-thread</b>	Function is not re-entrant.

The function `_wragon` from the graphics module specifies whether text displayed in graphics mode will be clipped when output reaches the right edge of the text window or whether the text will wrap to a new line.

`flags` specifies whether to wrap or not. This parameter can take either of the following values:

`_GWRAPOFF` clip text at window edge.

`_GWRAPON` wrap text at window edge.

**Return value**

The function returns the *previous* value of the wrap setting. There is no error return.

**Example**

```
#include <conio.h>
#include <graph.h>
main()
{
    _setvideomode(_ERESCOLOR);
    _settextwindow(10, 20, 20, 43);
    _wragon(_GWRAPON);
    _outtext(
        "This sentence wraps onto the next line\n");
    getch();
    _wragon(_GWRAPOFF);
    _outtext(
        "This doesn't wrap onto the next line");
    getch();
    _setvideomode(_DEFAULTMODE);
    return(0);
}
```

```
void _wrbufferln(  
    wintype win,  
    relcoord X,  
    relcoord Y,  
    void *src,  
    unsigned Len);
```

---

**W**

**Header file**      window.h

**See also**          \_rdbufferln

**Portability**      TopSpeed DOS/OS2

**Multi-thread**    Module is re-entrant and requires no additional locking for single function calls.

The function \_wrbufferln from the TopSpeed window module accesses the window directly, writing one line at a time. The function writes a specified number of character/attribute words to the window, beginning at a specified location.

win specifies the window.

X   specifies the horizontal location of the current position.

Y   specifies the vertical location of the current position.

Src points to the location containing the material to be written.

Len specifies the number of words to write.

**Return value**   None.

**Example**

```
#define _JPI_WIN_
#include <conio.h>
#include <string.h>
#define BUFFER_LENGTH 18
windef WD1={10, 2, 50, 8, White, Brown,
    FALSE, FALSE, FALSE, TRUE, SINGLEFRAME,
    Red, LightGray};
wintype W1;
main()
{
    int buffer[BUFFER_LENGTH];
    int n=0;
    clrscr();
    W1 = windowopen(&WD1) ;
        /* close window * open window */
    windowclose(W1);
    setttitle(W1, "Window 1", CenterUpperTitle);
    gotoxy(1, 1);
    cprintf("Press Key to blink");
    getch();
        /* get line of text */
    _rdbufferln(W1, 1, 1, buffer, BUFFER_LENGTH);
    while(n < BUFFER_LENGTH)
    {
        buffer[n]|=0x8000; /* set blink bits */
        ++n;
    }
        /* write line of text */
    _wrbufferln(W1, 1, 1, buffer, BUFFER_LENGTH);
    getch();
    return(0);
}
```



---

**int write(int handle, void \*buf, unsigned num);**

---

**U****Header file**           io.h**See also**             close, creat, lseek, open, read.**Portability**         UNIX/DOS/OS2**Multi-thread**       Low level I/O functions are not protected by library.

The function write writes num bytes from buf to the file associated with handle. The write operation begins at the current file position (if any). After the write operation, the file position indicator points to the next byte past the last byte written.

handle specifies the handle for the file to which the information will be written.

buf points to the location from which the information will be read.

num specifies the number of bytes to write.

**Return value**

The function returns the number of bytes actually written.

A return value of less than num indicates that the device ran out of space.

A return value of -1 indicates that an error may have occurred, although a successful write of 65535 bytes would return the integer value -1. In this case errno should be used to indicate an error. In the event of an error, errno is set to one of the following values:

EBADFile handle invalid.

ENOSPC Device full.

**Note :** Text mode translation causes linefeed characters to be replaced by carriage return /linefeed pairs. This does not affect the return value. The ^Z character is treated as an end-of-file indicator. If the file associated with handle is a device, output is merely terminated. If it is a file, the end of file indicator is set, and the file must be closed and reopened to clear it.

**Example**

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

main() {
    int fh;
    char *msg1="THIS WAS WRITTEN AT END OF FILE";
    char *msg2="THIS OVER-WRITES FIRST MESSAGE";
    long file_pos;

    fh=open("TEMP.$$$", O_RDWR);
    if(fh < 0)
        abort();
    lseek(fh, 0L, SEEK_END);
    file_pos=tell(fh);
    write(fh, msg1, strlen(msg1));
    lseek(fh, file_pos, SEEK_SET);
    write(fh, msg2, strlen(msg2));
    close(fh);
    return(0);
}
```

## **int \_write(int handle, void \*buf, unsigned num);**

---

<b>Header file</b>	io.h
<b>See also</b>	close, _open, _read
<b>Portability</b>	DOS
<b>Multi-thread</b>	DOS is not re-entrant

The function `_write` writes a specified number of bytes to a file. The file can be a disk file or a device. The function accomplishes this by performing INT 21H function 40H.

`handle` specifies the handle for the file to be written. This handle must be from a call to `_creat`, `_open`, `_dup`, or `_dup2`.

`buf` points to the location at which the information to be written is stored.

`num` specifies the number of bytes to write to the file.

Because `_write` treats all files as binary, linefeed characters are *not* translated to carriage return linefeed pairs.

If the file is a device, the bytes are sent directly to the device; if the file is a disk file, the function begins writing at the current file position. If the file was opened with `O_APPEND`, `_write` does *not* position the file pointer to the end of the file before writing.

### **Return value**

The function returns the number of bytes actually written.

A return value of less than `num` indicates that the device ran out of space.

A return value of -1 indicates that an error may have occurred, although a successful write of 65535 bytes would return the integer value -1. In this case `errno` should be used to indicate an error. In the event of an error, `errno` is set to one of the following values:

`EACCES` access denied (e.g., because the file is a directory).

`EBADF` invalid handle.

### **Example**

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

#define BUFFER_SIZE 512

main()
{
    int fh;
    char buffer[BUFFER_SIZE];
    int ret;

    memset(buffer, 'X', BUFFER_SIZE);
    fh=_open("TEMP.$$$", 2);
    if(fh < 0)
    {
        fprintf(stderr, "Error opening file\n");
        return(1);
    }
    ret=_write(fh, buffer, BUFFER_SIZE);
    if(ret != BUFFER_SIZE)
        fprintf(stderr,
            "Media full error on read\n");
    _close(fh);
    return(0);
}
```

# APPENDIX A

## *Include files*

There is a list of include files with a brief description of their contents. Those files marked with `<stdio.h>` are specified by the ANSI C Standard.

<code>&lt;alloc.h&gt;</code>	Prototypes for memory management functions.
<code>&lt;assert.h&gt;</code>	Defines assert debugging macro.
<code>&lt;bios.h&gt;</code>	Function prototypes for BIOS routines.
<code>&lt;conio.h&gt;</code>	Prototypes and definitions for the console I/O functions and constants.
<code>&lt;ctype.h&gt;</code>	Macro definitions and prototypes for the character classification and conversion routines.
<code>&lt;dir.h&gt;</code>	Prototypes and definitions for path and filename functions.
<code>&lt;direct.h&gt;</code>	Prototypes and definitions for path and filename functions.
<code>&lt;dos.h&gt;</code>	Prototypes and definitions for DOS and for 80x86-specific functions.
<code>&lt;errno.h&gt;</code>	Error code definitions and declarations.
<code>&lt;fcntl.h&gt;</code>	Constant definitions used with open.
<code>&lt;float.h&gt;</code>	Parameter and constant definitions for floating point module.
<code>&lt;io.h&gt;</code>	Prototypes and definitions for low level I/O functions, constants and variables.
<code>&lt;limits.h&gt;</code>	ANSI definitions of machine dependent constants.
<code>&lt;locale.h&gt;</code>	Prototypes and definitions for localization functions.
<code>&lt;locking.h&gt;</code>	Parameter definitions for the locking function.
<code>&lt;malloc.h&gt;</code>	Prototypes for memory management functions.
<code>&lt;math.h&gt;</code>	Prototypes and declarations for math functions and the <code>matherr</code> exception structure.
<code>&lt;mem.h&gt;</code>	Prototypes for memory manipulation functions.
<code>&lt;memory.h&gt;</code>	Prototypes for memory manipulation functions.

OS2.H	OS2 System definitions, declarations and prototypes.
PROCESS.H	Prototypes and definitions for process management functions.
RTCHECK.H	Runtime check handler constants and declaration
SEARCH.H	Prototypes for searching and sorting functions.
SETJMP.Hå	Prototypes and declarations for setjmp and longjmp and the jmp_buf structure.
SHARE.H	Parameter definitions for file sharing functions.
SIGNAL.Hå constants.	Prototypes and definitions for signal functions and constants.
STAT.H	Constant definitions for opening and creating files.
STDARG.Hå	Definitions of variable argument list macros.
STDDEF.Hå	Definitions of commonly used constants.
STDIO.Hå	Prototypes and definitions for stream I/O functions, constants and variables.
STDLIB.Hå	Prototypes and definitions for commonly used functions constants and variables.
STRING.Hå	Prototypes for string and memory manipulation functions.
TIME.Hå	Prototypes and declarations for time functions and structures.
TIMEB.H	Prototypes and declarations for Microsoft-compatible time structure.
VALUES.H	Definitions of machine dependent constants.

In addition to these files, several other includes files are associated with specific modules (such as the JPI process module). These files are described in connection with the libraries for the modules.

# APPENDIX B

## Implementation dependent behavior

The ANSI C standard defines implementation-defined behavior as behavior that depends on the characteristics of the implementation. It is assumed that the program construction and data are correct.

The implementation-defined features for the TopSpeed C library are described here. The features are numbered in accordance with Appendix F3 (Implementation defined behavior) in the ANSI C standard.

### Library Functions

---

- NULL expands to 0 in all 16-bit data pointer memory models, 0L in all others.
- Assert outputs the following message before program termination:  
  
Assertion failed: AS, file FN, line LN\n  
  
where AS is the stringized assertion test, FN is the file name of the current compilation unit and LN is the line number of the failed assertion.  
  
On termination, all exit lists are executed, all streams are flushed and file closed. An error information file is created containing details of the error.
- **isalnum** returns true for the following characters:  
  
‘0’ - ‘9’  
  
‘a’ - ‘z’  
  
‘A’ - ‘Z’  
  
**isalpha** returns true for the following characters:  
  
‘a’ - ‘z’  
  
‘A’ - ‘Z’  
  
**isctrl** returns true for the following character values:  
  
0 - 0x1F

0x7F

**islower** returns true for the following characters:

‘a’ - ‘z’

**isupper** returns true for the following characters:

‘A’ - ‘Z’

**isprint** returns true for the following character values:

0x20 - 0x7E

- Return values for math functions on domain errors: all functions return 0.0
- Math functions set `errno` to `ERANGE` on underflow errors.
- **fmod** returns 0.0 and sets `errno` to `EDOM` when the second parameter is 0.0.

## Signals

---

- The set of signals for the `signal` function and their semantics are:
 

SIGINT	interrupt - corresponds to DOS 3.x int 23H
SIGILL	illegal opcode.
SIGFPE	floating point error.
SIGSEGV	segment violation.
SIGTERM	Software termination signal from kill
SIGUSR1	User defined signal 1 OS2 only.
SIGUSR2	User defined signal 2 OS2 only.
SIGUSR3	User defined signal 3 OS2 only.
SIGABRT	abnormal termination triggered by abort call.

### Signal default actions:

- |        |                    |
|--------|--------------------|
| SIGINT | terminate process. |
|--------|--------------------|



SIGILL terminate process.

SIGFPE terminate process.

SIGSEGV terminate process.

SIGTERM terminate process.

SIGUSR1 terminate process.

SIGUSR2 terminate process.

SIGUSR3 terminate process.

SIGABRT terminate process.

- Signals are not blocked if the equivalent of `signal(sig, SIG_DFL)` is not performed.
- Default signal handling is reset by all signal handlers on receiving a signal.

## Streams and files

---

- The last line of a text stream requires no special terminating character.
- Space characters immediately preceding a newline appear when read in.
- No null characters may be appended to a binary stream.
- The file position indicator is not set initially to the end of file in append mode.
- A write on text stream does not truncate the associated file.
- Files can be fully buffered, line buffered or unbuffered. If a file is buffered a default buffer of 512 bytes is allocated when the file is opened.
- A zero length file does exist.
- A valid file name comprises the following:
  - An optional drive specifier consisting of a letter followed by a colon.
  - An optional path consisting of 0 or many directory names of maximum length 8 characters separated by ‘\’ characters.
  - A filename consisting of up to 8 characters.

- An optional extension consisting of a period followed by a maximum of three characters.
- The same file may be opened multiple times.
- The effect of remove on an open file is not managed by the library. It is the responsibility of the programmer to check for this.
- If a file with the new name exists prior to a call to rename the function will return -1 and set errno to EEXIST.
- The output for the %p conversion is four hex digits (XXXX) in 16 bit data pointer models and 8 hex digits separated by a colon in other models (XXXX:XXXX).
- The input for scanf with the %p conversion corresponds to the output format of printf.
- The '-' character is always treated as a normal character in the scan list for the scanf '%[' conversion.

## Errno

---

- ftell sets errno to EBADF on errors caused by an invalid handle.
- perror prints the user's message followed by ": " followed by one of the following, depending on the value of errno.

	-1	Unknown Error
EZERO	0	Error Zero
EINVFN	1	Invalid Function Code
ENOENT	2	File not found
ENOPATH	3	Path not found
EMFILE	4	Too many open files
EACCES	5	Access denied
EBADF	6	Invalid handle
E2BIG	7	Memory blocks destroyed
ENOMEM	8	Insufficient Memory

EINVMEM	9	Invalid memory block address
EINVENV	10	Invalid environment
EINVFMT	11	Invalid format
EINVACC	12	Invalid access code
EINVDAT	13	Invalid data
ENODEV	15	Invalid drive
ECURDIR	16	Attempt to remove CurDir
ENOTSAM	17	Not same device
ENMFILE	18	No more files
EINVAL	19	Invalid argument
ENOTDIR	20	Not directory
EISDIR	21	Is directory
ENOEXEC	22	Corrupted exec file
EMLINK	32	Cross-device link
EDOM	33	Math argument
ERANGE	34	Result too large
EDEADLOCK	36	file locking deadlock.

## Memory

---

- calloc and malloc return NULL and set errno to ENOMEM if an attempt is made to allocate zero bytes. realloc does this if the pointer parameter is also NULL.

## abort

---

- if abort is called open files will be flushed and temporary files will be deleted.

NB. If abort is called a second time from a function in the terminate list the process will terminate immediately with no further actions.

## exit

---

- exit will return a status value between zero and MAX\_UCHAR if a value greater than MAX\_UCHAR is passed to exit. This is achieved by truncating the status value returned by exit.

## getenv

---

- There is no predefined set of environment variables other than those installed by the DOS SET function.

The putenv function changes the environment variables only for the current process and its children.

## system

---

- The system function interprets its string argument as a DOS command . The command processor is executed and the string is passed as a command. The string may be any legal DOS command sequence.

## strerror

---

- The contents of the error message are documented under perror.

## Locale behavior

---

- The timezone is GMT and DST = 0.
- The era for the clock function is from the beginning of program.
- The execution character set is that supported by the IBM PC.
- The direction of printing is from left to right.
- The decimal point character is ‘.’
- All character testing and case mapping functions are the same at run time as at compile time.
- The collation sequence of the execution character set is that of the signed value of their ASCII representation.
- Date and time are implemented as ANSI except for timezone variable which specifies a signed number of minutes difference from GMT.

## Other

---

- White space characters directly before the end of line character are output by stream I/O functions.
- If tmpnam is called for more than TMP\_MAX files, the value NULL is returned)

# Index

## Symbols

#link pragma 9  
 \_A\_ARCH 177  
 \_A\_HIDDEN 169, 177  
 \_A\_NORMAL 169, 177  
 \_A\_RDONLY 169, 177  
 \_A\_SUBDIR 177  
 \_A\_SYSTEM 169, 177  
 \_A\_VOLID 169, 177  
 \_arc 47, 63  
 \_beginthread 14, 54, 80  
 \_bios\_disk 53, 94  
 \_bios\_equiplist 54, 96  
 \_bios\_keybrd 53, 98  
 \_bios\_memsize 53, 100  
 \_bios\_printer 53, 101  
 \_bios\_serialcom 54, 103  
 \_bios\_timeofday 53, 106  
 \_chain\_intr 50, 114  
 \_chmod 21, 119  
 \_clear87 28, 122  
 \_clearscreen 47, 124  
 \_CLIP\_WIN\_ 40, 45  
 \_close 21, 127  
 \_control87 28, 134  
 \_creat 21, 146  
 \_creat\_new 148  
 \_cube 47, 154  
 \_DEFAULTMODE 49  
 \_directwrite 43, 163  
 \_disable 50, 164  
 \_displaycursor 47, 165  
 \_dos\_allocmem 51  
 \_dos\_clos 51  
 \_dos\_creat 51  
 \_dos\_creatnew 51  
 \_dos\_findfirst 52, 169  
 \_dos\_findnext 52, 172  
 \_dos\_freemem 51, 173  
 \_dos\_getdate 34, 174  
 \_dos\_getdiskfree 52, 175  
 \_dos\_getdrive 52, 176  
 \_dos\_getfileattr 51, 177  
 \_dos\_getftime 52, 179  
 \_dos\_gettime 34, 181  
 \_dos\_getvec 50  
 \_dos\_getvect 182  
 \_dos\_keep 50, 183  
 \_dos\_open 51, 184  
 \_dos\_read 51, 188  
 \_dos\_setblock 51, 186  
 \_dos\_setdate 34, 189  
 \_dos\_setdrive 52, 190  
 \_dos\_setfileattr 52, 191  
 \_dos\_setftime 34, 193  
 \_dos\_settime 34, 195  
 \_dos\_setvec 50  
 \_dos\_setvect 196  
 \_dos\_write 51, 197  
 \_dosbeginthread 15, 54, 167  
 \_dosendthread 15, 54, 171  
 \_doserrno 62  
 \_dosfreestack 54  
 \_dup 21, 199  
 \_dup2 21, 201  
 \_ellipse 47, 204  
 \_enable 50, 206  
 \_enable\_herc 49, 207  
 \_endthread 14, 54, 207  
 \_eof 21, 209  
 \_exit 35, 214  
 \_expand 30, 32, 215  
 \_F\_RST 23  
 \_fcalloc 31, 218  
 \_fexpand 32, 226  
 \_ffree 31, 227  
 \_fheapchk 32, 233  
 \_fheapset 32, 233  
 \_fheapwalk 32, 233  
 \_floodfill 47, 238  
 \_fmalloc 30, 31  
 \_fmemccpy 26  
 \_fmemchr 26  
 \_fmemcmp 26  
 \_fmemcpy 26  
 \_fmemicmp 26  
 \_fmemset 26  
 \_fmsize 32  
 \_fpreset 29, 249  
 \_frealloc 32, 255  
 \_freect 32, 256

`_fstpcpy` 23, 268  
`_fstrcat` 24, 268  
`_fstrchr` 24, 268  
`_fstrcmp` 24, 268  
`_fstrcoll` 23, 268  
`_fstrcpy` 23, 268  
`_fstrcspn` 24, 268  
`_fstrdup` 25, 268  
`_fstricmp` 24, 268  
`_fstrlen` 25, 269  
`_fstrlwr` 25, 269  
`_fstrncat` 24, 269  
`_fstrncmp` 24, 269  
`_fstrncpy` 23  
`_fstrnicmp` 24, 269  
`_fstrnset` 24, 269  
`_fstrprbk` 24, 269  
`_fstrrchr` 24, 269  
`_fstrrev` 25, 269  
`_fstrset` 24, 270  
`_fstrspn` 24, 270  
`_fstrstr` 24, 270  
`_fstrtok` 25, 270  
`_fstrupr` 25, 270  
`_fstrxfrm` 23, 270  
`_fullscreen` 41  
`_getbkcolor` 47, 276  
`_getcolor` 47, 282  
`_getcurrentposition` 47, 284  
`_getdrive` 52  
`_getfillmask` 48, 293  
`_getimage` 48  
`_getlinestyle` 48  
`_getlogcoord` 48  
`_getphyscoord` 48  
`_getpixel` 48  
`_gettextcolor` 48  
`_gettextposition` 48  
`_getTID` 36  
`_getvideoconfig` 48  
`_harderr` 51  
`_hardresume` 50  
`_hardretn` 50  
`_heapchk` 32  
`_heapset` 32  
`_heapwalk` 32  
`_hugeshift` 8  
`_imagesize` 48  
`_JPI_WIN_` 40, 45  
`_lineto` 48  
`_lrotl` 28  
`_lrotr` 28  
`_makepath` 52  
`_MAX_EXT` 570  
`_MAX_NAME` 570  
`_memavl` 32  
`_memmax` 32  
`_moveto` 48  
`_ms_cursor` 39  
`_ms_driversize` 39  
`_ms_getmotion` 39  
`_ms_getpage` 39  
`_ms_getpress` 39  
`_ms_getrelease` 39  
`_ms_getsensitivity` 39  
`_ms_getstatus` 39  
`_ms_lightpen` 39  
`_ms_reset` 39  
`_ms_restoredriver` 39  
`_ms_savedriver` 39  
`_ms_setdouble` 39  
`_ms_setgraphcursor` 39  
`_ms_setinterrupt` 39  
`_ms_setmickeys` 39  
`_ms_setpage` 39  
`_ms_setposition` 39  
`_ms_setrange` 39  
`_ms_setsensitivity` 39  
`_ms_settextcursor` 39  
`_ms_swapinterrupt` 39  
`_ms_updatescreen` 40  
`_msize` 32  
`_ncalloc` 31  
`_nexpand` 32  
`_nfree` 31  
`_nheapchk` 32  
`_nheapset` 32  
`_nheapwalk` 32  
`_nmalloc` 7, 30, 31  
`_nmsize` 32  
`_nrealloc` 32  
`_nstrdup` 25  
`_open` 21  
`_osmode` 14  
`_outtext` 48  
`_pie` 48  
`_polygon` 48  
`_putimage` 48  
`_rdbufferIn` 43  
`_read` 21  
`_rectangle` 48  
`_remapallpalette` 48  
`_remappalette` 48  
`_rmtmp` 506

`_rotl` 28  
`_rotr` 28  
`_searchenv` 53  
`_selectpalette` 48  
`_setactivepage` 48  
`_setbkcolor` 48, 520  
`_setcliprgn` 48, 524  
`_setcolor` 48, 525  
`_setfillmask` 48, 529  
`_setlinestyle` 48, 536  
`_setlogorg` 48, 538  
`_setpixel` 48, 543  
`_settextcolor` 48, 544  
`_settextposition` 48, 546  
`_settextwindow` 48, 547  
`_setvideomode` 48, 49, 554  
`_setviewport` 48, 49, 556  
`_setvisualpage` 49, 557  
`_splitpath` 52, 570  
`_stackfill` 47, 576  
`_status87` 29, 580  
`_strdate` 33, 591  
`_strerror` 593  
`_strtime` 33, 609  
`_tolower` 29  
`_toupper` 29  
`_wupon` 49, 659  
`_wrbufferln` 660  
`_write` 21, 664  
80x86 6, 31, 666  
80x86 machines 6  
80x87 coprocessor 28

## A

`abort` 12, 35, 673  
`abs` 28, 57  
`abscoord` 115, 135  
`absolute value` 28  
`absread` 52, 58  
`abswrite` 52, 59  
`access` 20, 60  
`acos` 27  
`acosl` 27  
`allocmem` 51, 62  
ANSI 3, 5, 6, 9, 16, 34, 666, 668  
ANSI compatibility 23  
`append mode` 670  
ASCII 29  
`asctime` 34, 65  
`asin` 27  
`asinl` 27

`Assert` 668  
`assert` 36, 68, 666  
`at` 43, 69  
`atai` 27  
`atan2` 27  
`atan2l` 27  
`atanl` 27  
`atexit` 11, 35, 72  
`atof` 25  
`atoi` 25, 74  
`atol` 25, 75  
`atoul` 25, 76  
`Awaited` 77  
`awaited` 36

## B

`background color` 47  
`bar charts` 47  
`bdos` 50, 78  
`bdosptr` 50, 79  
BIOS 53, 666  
`bioscom` 54, 81  
`biosdisk` 53, 84  
`biosequip` 54, 87  
`bioskey` 53, 89  
`biosmemory` 53, 91  
`biosprint` 53, 92  
`biostime` 53, 93  
`Bitwise rotation operations` 28  
`bsearch` 8, 33, 107  
`buffer string-handling functions` 25  
`Buffering` 22  
`BUFSIZ` 22

## C

C++ 5  
`cabs` 28  
`cabsl` 28  
`calloc` 31, 110, 672  
`carriage returns in files` 21  
`ceil` 28  
`ceill` 28  
`cgets` 40, 44, 49, 113  
`change` 43, 115  
`character conversion functions` 29  
`Character handling functions` 26  
`character set` 674  
`character test functions` 29  
`chdir` 52, 117  
`Child process` 35



- chmod 20, 118
- chsize 20, 120
- cleanup procedure 11
- clearerr 19, 123
- clipping 49
- clipping rectangle 48
- clipping region 556
- clock 34, 125, 674
- close 20, 126
- closing files
  - on exit 11
- clreol 43, 46, 128, 129
- clrscr 43, 46, 130, 131
- CnsHandler 13, 51, 132
- collation sequence 674
- Color typedef 41
- COMMAND.COM 617
- compare operations 25
- comparing strings 24
- Compatibility
  - with Turbo C 9, 45
- compatibility 31
- complex 28
- complex structure 109
- COMSPEC 617
- concatenating strings 24
- conio.h 40
- console 666
- control-break processing 50
- conversion 25, 29
- convertcoords 43, 135
- converting strings 25
- coordinates
  - in graphics functions 48
  - of a window 41, 42
- coprocessor 28
- copy operations 25
- copying strings 23
- coreleft 32, 137
- cos 27, 138
- cosh 27
- coshl 27
- country 53, 140
- country dependent information 53
- country structure 140
- cprintf 40, 44, 49, 142
- cputs 40, 44, 49, 143
- creat 20, 144
- creatnew 147
- creattemp 20, 149
- cscanf 40, 44, 49, 151
- ctime 34, 152

- ctrlbrk 50, 153
- Ctype 29
- cuboid 47
- cursoroff 43, 156
- cursoron 43, 157

## D

- date 33, 674
- date DOS functions 34
- date structure 286, 526
- debugger 10
- decimal point character 674
- default data segment 30
- default heap 31
- Delay 158
- delay 36, 159
- delline 43, 46, 160, 161
- denied 664
- dfree structure 287
- dftime 34, 162
- direct memory access 51
- directories 52
- disable 164
- DisableBreakCheck 51, 164
- disk transfer area 52
- diskfree\_t 175
- diskfree\_t structure 175
- diskinfo\_t structure 94
- div 28, 166
- div\_t structure 166
- divide by zero 11
- division 28
- DLLs
  - and near heap 15
- DOS 6, 666
- DOS directory operations 52
- DOS paths 52
- DOS process control 36
- DOS system call 34, 50
- DOS system function 673
- DOS time and date handling 34
- DosCreateThread 14
- dosdate\_t structure 174, 189
- doserror structure 171
- DosExit 171
- dosexterr 51
- dostime\_t structure 181, 195
- double
  - function variants 2, 9, 27
- DOUBLEFRAME 42
- DTA 52

dup 20, 198  
dup2 20, 200

## E

E2BIG 671  
EACCES 188, 197, 671  
EBADF 188, 197, 671  
ECURDIR 672  
ecvt 15, 25, 202  
EDEADLOCK 672  
EDOM 669, 672  
EEXIST 671  
EINVACC 672  
EINVAL 672  
EINVDAT 672  
EINVENV 672  
EINVFMT 672  
EINVFN 671  
EINVMEM 672  
EISDIR 672  
ellipse 47  
EMFILE 671  
EMLINK 672  
enable 206  
EnableBreakCheck 51, 206  
end of file 670  
end-of-file 11  
ENMFILE 672  
ENODEV 672  
ENOENT 671  
ENOEXEC 672  
ENOMEM 671, 672  
ENOPATH 671  
ENOTDIR 672  
ENOTSAM 672  
environment variables 673  
EOF 22  
eof 20, 208  
era 674  
ERANGE 669, 672  
errno 13, 23, 669, 671, 672  
errno: list of possible values 671  
Error code definitions 666  
error codes 23  
error handling 10, 34, 50  
ERRORINF:\$\$\$ 10  
errors 19  
    and OS2 14  
    C error handling 10  
    environment error handling 10  
    fatal 11

    in file functions 13  
    in mathematical functions 12  
    low level library 13  
    recoverable 11, 12  
    runtime checks 13  
    signal function 34  
    stderr 12  
exception conditions 22  
exception handling 10, 29, 34, 50  
exception structure 383  
execl 35  
execle 35  
execclp 35  
execclpe 35  
execution character set 674  
execv 35  
execve 35  
execvp 35  
execvpe 35  
exit 11, 35, 213  
exit mechanism 11, 35  
exit procedure 673  
exp 27  
expl 27  
exponent of a floating point number 28  
exponential functions 27  
extra large memory model 37  
EZERO 671

## F

fabs 28  
fabsl 28  
far heap 30, 31  
far pointer  
    function variants 2, 7  
far pointers 30  
    to huge objects 8  
far strings 26  
farcalloc 30, 31, 217  
farcoreleft 32, 218  
farfree 31, 218  
farmalloc 8, 31, 218  
farrealloc 32, 218  
FAT tables 52  
fatal errors 11  
fatinfo structure 291, 292  
fcb structure 455, 482, 484  
fclose 17, 219  
fcloseall 17, 220  
fcvt 15, 25, 221  
fdopen 19, 222

feof 19, 22, 224  
ferror 13, 19, 23, 225  
ffblk structure 236  
fflush 15, 19, 226  
fgetc 15, 18  
fgetchar 15, 18, 228  
fgetpos 19, 229  
fgets 15, 18, 231  
FILE 16  
file allocation tables 52  
file description string 52  
file errors 22  
file functions  
    error handling 13  
file handling  
    random access 51  
file position indicator 670  
file protection 22  
File streams 17  
FILE structure 13  
filelength 20, 234  
fileno 19, 235  
files  
    and carriage returns 21  
    and exit procedure 11  
    binary mode 21  
    buffer handling 19  
    buffering 22  
    DOS functions 51  
    DOS low-level functions 21  
    error handling 19  
    handles 19  
    initial state 23  
    implementation-dependent behavior 670  
    open and close functions 17  
    position functions 19  
    read functions 17  
    temporary file 149  
    text mode 21  
    truncation on output 670  
    UNIX low-level functions 20  
    write functions 18  
fill mask 48  
filling strings 23  
find\_t structure 169  
findfirst 52  
findnext 52  
Floating point coprocessor 28  
floor 28  
floorl 28  
flushall 19, 241  
flushing files 19  
fmod 28, 669  
fmode 21  
fmodl 28  
fnmerge 52, 244  
fnsplit 53, 245  
fopen 17, 247  
FP\_OFF 51, 253  
FP\_SEG 51  
fprintf 17, 18, 249  
fputc 15, 18, 250  
fputchar 15, 18, 251  
fputs 15, 18, 252  
fractional part of a number 28  
FrameOn 42  
framestr 42  
fread 8, 15, 18, 254  
free 8, 30, 31, 255  
freeing memory 31  
freeing memory:using DOS 51  
freemem 51, 257  
freopen 17, 258  
frexp 28  
frexpl 28  
fscanf 17, 18, 261  
fseek 15, 19, 22, 262  
fsetpos 19, 264  
fstat 20, 266  
ftell 15, 19, 271, 671  
ftime 34, 272  
ftime structure 295, 531  
fullscreendef 41  
fwrite 8, 15, 18, 273

## G

gcvt 25  
gcvtl 25  
generic console input-output 40, 44, 46, 49, 666  
geninterrupt 50, 275  
get\_ftime 33  
getc 15, 18, 277  
getcbrk 50, 278  
getch 44, 279  
getchar 15, 18, 280  
getche 44, 49, 281  
getcurdir 52, 283  
getcwd 52, 285  
getdate 33, 286  
getdfree 52, 287  
getdisk 52, 288  
getdta 52, 289  
getenv 53, 290, 673

getfat 52, 291  
 getfatd 52, 292  
 getftime 20, 295  
 getimage 296  
 GetInProgramFlag 54, 298  
 getlinestyle 299  
 getlogcoord 300  
 getphyscoord 301  
 getpid 36, 302  
 getpixel 303  
 getpsp 53, 304  
 gets 15, 18, 305  
 gettext 46, 306  
 gettextcolor 308  
 gettextinfo 46, 309  
 gettextposition 310  
 getTID 311  
 gettime 33, 311  
 getvect 50, 312  
 getverify 52, 313  
 getvideoconfig 314  
 getw 18, 316  
 gmtime 34, 317  
 gotoxy 43, 46, 318, 319  
 graphics 47  
 graphics functions 9

## H

halloc 8, 31, 320  
 handle 16, 19, 20  
     window handle 42  
 harderr 321  
 hardresume 324  
 hardretn 325  
 hardware structure 436  
 hbsearch 33, 326  
 header file 4  
 header files  
     location 4  
 heap 30  
     near heap and DLLs 15  
 heap checking 32  
 Heap information 32  
 Heap integrity 32  
 heapinfo structure 329  
 heapwalk 329  
 hfreed 18, 331  
 hfree 8, 31, 331  
 hfwrite 18, 331  
 Hidden 42  
 hide 43, 332

highvideo 46, 333  
 hlfind 33, 333  
 hlsearch 33, 333  
 hmalloc 31  
 hmemccpy 8, 26, 333  
 hmemchr 8, 26, 333  
 hmemcmp 8, 26, 333  
 hmemcpy 8, 26, 333  
 hmemicmp 8, 26, 333  
 hmemset 8, 26, 334  
 hqsort 33, 334  
 hrealloc 8, 32, 334  
 huge objects 30  
     far pointers to 8  
 huge pointer  
     function variants 2, 7  
 Huge pointers 8  
 huge strings 26  
 hyperbolic functions 27  
 hypot 27  
 hypotl 27

## I

IBM PC 7  
 imagesize 336  
 Incore input-output 17  
 info 43, 337  
 Init 36, 338  
 init 36  
 inp 53, 339  
 inportb 53, 339  
 inportw 53  
 InProgramFlag 54  
 Input-output 16  
 input-output 667  
     and graphics functions 49  
     generic console functions 40, 44, 49  
     high-level 17  
     low-level 17  
     serial ports 53  
     standard functions 17  
     windows 16  
     with Turbo C window module 46  
 input-output:generic console functions 666  
 input-output:implementation-dependent behavior 674  
 inpw 53  
 inline 43, 46, 341, 342  
 INT 21H 50  
 int86 50, 343  
 int86x 50, 344  
 intdos 50, 346

- intdosx 50, 347
- interrupt vectors
  - restored on exit 11
- Interrupts 50
- interrupts 11
- intr 50, 349
- inverse trigonometric functions 27
- ioctl 20, 350
- isalnum 29, 668
- isalpha 29, 668
- isascii 29
- isatty 20, 354
- isctrl 29, 668
- isdigit 29
- isgraph 29
- islower 29, 669
- isprint 29, 669
- ispunct 29
- isspace 29
- isupper 29, 669
- isxdigit 29
- itoa 25, 355

## J

- jmp\_buf 373, 667
- jmp\_buf structure 534
- JPI graphics module 47
- JPI process and task module 36
- JPI process module 14, 15, 667
- JPI text window module 40
- JPI window module 43

## K

- kbhit 44, 356
- keep 358

## L

- labs 28
- labsl 28
- lconv structure 365
- ldexp 28, 360
- ldexpl 28, 360
- ldiv 28, 361
- ldiv\_t structure 361
- lfind 8, 33
- library
  - Graphics 9, 49
  - special library requirements 9
  - usage 4
  - window modules 9, 40

- line draw 48
- linefeeds in files 21
- lineto 364
- locale 674
- Locale functions 54
- locale structure 365
- localeconv 365
- localization functions 666
- localtime 34, 366
- locating header files 4
- Lock 36, 39, 49, 368
- lock 368
- locking 15, 20, 369, 666
- log 27
- log functions 27
- log10 27
- log10l 27
- logical co-ordinates
  - in graphics functions 48
- logl 27
- long double
  - function variants 2, 9, 27
- longjmp 50, 373, 667
- low-level file functions• 20
- low-level functions 16
- lowvideo 46, 375
- lrotl 376
- lrotr 376
- lsearch 8, 33
- lseek 20, 377
- ltoa 25, 379

## M

- makepath 380
- malloc 7, 8, 30, 31, 382, 672
- mantissa 28
- Math errors 12
- math functions 666
- Mathematical functions 27
- mathematical functions
  - error handling 12, 29
- matherr 12, 29, 383, 666
- max 28, 385
- MAX\_UCHAR 673
- maximum 28
- mblen 29, 386
- mbstowcs 30, 387
- mbtowc 29, 388
- memavl 389
- memccpy 8, 26, 390
- memchr 8, 26, 392

- memcmp 8, 26, 393
- memcpy 8, 26, 395
- memicmp 8, 26, 396
- memmax 397
- memmove 26, 398
- memory access 51
- Memory allocation 30
- memory allocation 672
  - using DOS 51
- memory buffer string-handling functions 25
- memory management 666
- memory manipulation 666
- memory models 7, 30
  - extra large 37
  - Mthread 37
- memset 8, 26, 399
- memwcpy 26, 400
- memwmove 26, 401
- memwset 26, 402
- Microsoft compatibility 3
- min 28, 403
- minimum 28
- mixed memory models 30
- MK\_FP 51, 407
- mkdir 52, 404
- mktemp 20, 405
- mktime 34, 406
- Mode
  - protected 15
- mode
  - real 15
- modf 28
- modfl 28
- module windows
  - Turbo C window module 9
- modulus 28
- mouse functions 39
- move graphics cursor 48
- move operations 25
- movedata 26, 409
- movetext 46, 410
- moveto 412
- movmem 26, 413
- ms
  - cursor 415
  - data structure 418, 420, 423
  - driversize 415
  - getmotion 416
  - getpage 417
  - getpress 418
  - getrelease 420
  - getsensitivity 422

- getstatus 423
- graphcur structure 429
- interrupt structure 430, 438
- lightpen 425
- range structure 434
- reset 426
- restoredriver 427
- savedriver 427
- sense structure 422, 435
- setdouble 428
- setgraphcursor 429
- setinterrupt 430
- setmickeys 431
- setpage 432
- setposition 433
- setrange 434
- setsensitivity 435
- settextcursor 436
- swapinterrupt 438
- textcur structure 436
- type union 436
- updatescreen 439
- ms\_motion structure 416
- msize 414
- Mthread memory model 37
- multi-thread programming 15, 36, 37
  - and graphics functions 49
  - and single byte access fu 15
  - under OS2 14
  - with a mouse 39
- Multibyte characters 29
- Multiple windows 40

## N

- near heap 31
  - and DLLs 15
- near pointer
  - function variants 2, 7
- nearcoreleft 32
- normvideo 46, 440
- nosound 50, 441
- Notify 442
- notify 36
- nrealloc 442
- nstrdup 442
- NULL 668

## O

- O\_TEXT 21
- obscuredat 43, 443

- onexit 11, 35, 444
- open 19, 20, 446, 448
- Operating system interface 50
- OS2 6, 54
  - multi-thread programming 14
  - process termination 14
- OS2 process control 36
- OS2 System definitions 667
- outp 53
- outportb 53
- outportw 53
- output
  - to a file 18
  - to a port 53
- outpw 53
- outtext 49, 451

## P

- palette 40
- Palette Windows 42
- palettec color 43, 452
- palettec color used 43, 453
- paletteopen 43, 454
- parsfnm 52, 455
- path
  - for header files 5
- path control 52
- peek 51, 456
- peekb 51, 456
- perror 13, 19, 457, 671, 673
- physical co-ordinates
  - in graphics functions 48
- physical screen 40
- pie 458
- pixel value 48
- pointer
  - variant 7
- Pointer conventions 7
- pointers
  - far pointers to huge objects 8
  - huge 8
- poke 51, 460
- pokeb 51, 460
- poly 28
- polygon 48, 462
- polyl 28
- Portability 3
- portability 5
- pow 27
- pow10 27, 465
- power functions 27

- powl 27
- print direction 674
- printf 15, 16, 18, 22, 466, 671
- Process control functions 34
- process module 14, 15, 36
- ProcessDelay 36
- Program Segment Prefix 53
- program segment prefix 304
- program termination 11, 35
- project system 4
- protected mode 14
- protections 22
- PSP 53
- psp 304
- PTM 456
- putbeneath 43
- putc 15, 18
- putch 40, 44, 49
- putchar 15, 18, 469
- putenv 53, 470
- putimage 471
- putontop 42, 43, 473
- puts 15, 18, 474
- puttext 46, 475
- putw 18, 476

## Q

- qsort 8, 33, 477

## R

- raise 34, 35, 479
- rand 28, 481
- randbrd 51, 482
- randbwr 51, 484
- random 28, 486
- random access file handling 51
- randomize 28, 486
- rcCOORD structure 310, 546
- rdbufferln 487
- re-entrancy 54
- re-entrant functions 15
- read 20, 489, 491
- read-write 22
- real mode 14
- realloc 8, 32, 493, 672
- recoverable errors 11, 12
- rectangle 48, 495
- redirection 5
- REGPACK structure 349
- REGS union 343, 344, 346, 347

- relcoord 135, 163
- remainder 28
- remapallpalette 497
- remappalette 499
- remove 19, 502
- rename 19, 503
- Return value 273
- rewind 19, 22, 504
- rmdir 52, 505
- rmtmp 19
- rotate operators 28
- rounding 28
- Runtime check 667
- runtime checks 13

## S

- scanf 15, 16, 18, 22, 508, 671
- screen
  - physical 40
  - storing a screen image 48
  - virtual 40
- search functions 33
- searchenv 512
- searching 667
- searching strings 24
- searchpath 53, 514
- segmented addressing 31
- segread 51, 515
- selectpalette 516
- semaphore 15
- semaphores 15, 36
- SEND 37, 518
- SET 673
- setactivepage 519
- setblock 51, 521
- setbuf 19, 23, 522
- setcbkr 50, 523
- setdate 33, 526
- setdisk 52, 527
- setdta 52, 528
- setenv 673
- seterrno 15
- setframe 43, 530
- setftime 21, 531
- SetInProgramFlag 54, 533
- setjmp 50, 534, 667
- setlocale 537
- setmem 26, 539
- setmode 21, 540
- setpalette 43, 541
- setpalettecolor 43, 542
- settime 33, 548
- setting strings 23
- settitle 43, 549
- setvbuf 19, 23, 550
- setvec 50
- setvect 552
- setverify 52, 553
- setwrap 43
- SIG constants 12
- SIG\_DFL 558
- SIG\_IGN 558
- SIGABRT 12, 479, 558
- SIGFPE 12, 479, 558, 669
- SIGILL 12, 479, 558, 669
- SIGINT 12, 479, 558, 669
- SIGNAL 77, 338, 442, 518, 650
- signal 12, 35, 558
- signals 36, 669
  - default actions 669
- SIGSEGV 12, 479, 558, 669
- SIGTERM 12, 479, 558, 669
- SIGUSR 669
- sin 27
- SINGLEFRAME 42
- sinh 27
- sinhl 27
- sinl 27
- size\_t 25
- sleep 562
- snapshot 43, 563
- software structure 436
- sopen 564
- sort functions 33
- sorting 667
- sound 50, 566
- spawn 35
- spawnl 35
- spawnle 35
- spawnlp 35
- spawnlpe 35
- spawnv 35
- spawnve 35
- spawnvp 35
- spawnvpe 35
- sprintf 18, 572
- sqrt 27
- sqrtl 27
- srand 28, 574
- SREGS structure 344, 347, 515
- sscanf 18, 575
- stack 33
- stackavail 33, 576



- standard input-output 17
- Standard streams 16
- standard streams
  - in OS2 14
- StartProcess 36, 37, 576
- StartScheduler 36, 37, 577
- stat 578
- stdaux 16, 21
- stderr 14, 16, 21
- stdin 14, 16, 21
- stdio.h 16
- stdout 14, 16, 21
- stdprn 16, 21
- stime 33, 581
- StopProcess 37, 582
- StopScheduler 37, 582
- stpcpy 23, 583
- strcat 24, 584
- strchr 24, 585
- strcmp 24, 586
- strcmpi 24, 587
- strcoll 23, 588
- strcpy 8, 23, 589
- strcspn 24, 590
- strdup 25, 592
- stream 16
- Stream Buffering 22
- Stream I/O functions 20
- stream input-output 17
- stream position functions 19
- stream termination character 670
- Streams 16
- streams
  - initial state 23
- strerror 13, 19, 593, 673
- strftime 34, 594
- stricmp 24, 587
- string conversion 25
- string-handling functions 23
  - comparison 24
  - concatenation 24
  - conversion 25
  - copying 23
  - memory buffer functions 25
  - searching 24
  - setting 23
- strlen 25, 596
- strlwr 25, 597
- strncat 24, 598
- strncmp 24, 599
- strncmpi 600
- strncpy 23, 601

- strnicmp 24, 600
- strnset 23, 602
- strpbrk 24, 603
- strrchr 24, 604
- strrev 25, 605
- strset 24, 606
- strspn 24, 607
- strstr 24, 608
- strtod 25, 610
- strtodl 25, 610
- strtok 24, 611
- strtol 25, 612
- strtoul 25, 613
- strupr 25, 614
- strxfrm 23, 615
- swab 26, 616
- system 34, 617
- system call 673
- system time 33

## T

- tan 27
- tanh 27
- tanh1 27
- tanl 27
- task module 36
- tell 21, 620
- tempnam 19, 621
- temporary file 149
- temporary files 19
  - deleted on exit 11
- Terminate and stay resident program 183
- terminating character for streams 670
- termination 11
- text window module 2, 3, 40
- text\_info structure 309
- textattr 46, 622
- textbackground 43, 46, 623, 624
- textcolor 43, 46, 625, 626
- textmode 46, 627
- threads 15, 37, 54
- time 33, 628, 674
  - DOS functions 34
  - format conversion 33
- time format string 594
- time structure 311, 548
- time\_t structure 125, 152, 162
- timeb structure 272
- timezone 674
- TitleMode type 41
- tm structure 65, 317, 366

TMP\_MAX 674  
tmpfile 19, 629  
tmpnam 19, 630, 674  
toascii 29, 631  
tolower 29  
top 43, 635  
toupper 29  
transcendental functions 27  
Translation Modes 21  
TSR 183  
Turbo C 2, 6, 45, 368, 642  
    window module 9, 45  
Turbo C window module 2, 3, 40  
Turbo C windows 45  
tzset 34, 636

## U

ultoa 25, 636  
umask 21, 637  
ungetc 15, 18, 22, 638  
ungetch 44, 640  
UNIX 6, 17, 20  
unlink 21, 641  
Unlock 36, 37, 39, 49, 642  
unlock 642  
unrecoverable errors 11  
use 40, 43, 643  
used 44, 644  
utime 33, 645  
utm structure 645

## V

valid file name 670  
vfprintf 19  
vfscanf 18  
VID 10  
videoconfig structure 314  
viewport 49  
virtual screens 40  
vprintf 19  
vscanf 18  
vsprintf 19  
vsscanf 18

## W

WAIT 36, 37, 650  
wcstombs 30, 650  
wctomb 30, 651  
wedge 48  
wherex 44, 46, 652, 653

wherey 44, 46, 654, 655  
White space characters 674  
windef 42  
window 46, 656  
    text window module 2, 40  
    Turbo C window module 2, 40  
window constants 41  
windowclose 44, 657  
windowopen 42, 44, 658  
windows  
    input-output 16  
    relative and absolute co-ordinates 42  
    text window module 3, 9  
    Turbo C window module 15, 45  
wintype 42, 115, 135  
Word handling functions 26  
wrbufferln 44  
write 21, 662  
write-only 22

## X

xycoord structure 284, 300, 301, 538

## Z

zero length file 670