COMPUTER INNOVATIONS, INC.

Optimizing C86™ Compiler

593-0094
CONSISTS OF:
595-3375
597-4000

Printed in the
United States of America

ZENITH data systems

HEATH

COMPUTER INNOVATIONS, INC.

Optimizing C86(TM) Compiler

ZENITH | data systems

HEATH

NOTICE

This software is licensed (not sold). It is licensed to sublicensees, including end-users, without either express or implied warranties of any kind on an "as is" basis.

The owner and distributors make no express or implied warranties to sublicensees, including end-users, with regard to this software, including merchantability, fitness for any purpose or non-infringement of patents, copyrights or other proprietary rights of others. Neither of them shall have any liability or responsibility to sublicensees, including end-users, for damages of any kind, including special, indirect or consequential damages, arising out of or resulting from any program, services or materials made available hereunder or the use or modification thereof.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of this publication.

Technical consultation is available for any problems you encounter in verifying the proper operation of this product. Sorry, but we are not able to evaluate or assist in the debugging of any programs you may develop. For technical assistance, write:

Zenith Data Systems Corporation
Software Consultation
Hilltop Road
St. Joseph, Michigan 49085

or call:

(616) 982-3884	Application Software/SoftStuff Products
(616) 982-3860	Operating Systems/Languages/Utilities

Consultation is available from 8:00 AM to 7:30 PM (Eastern Time Zone) on regular business days.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Computer Software clause in DAR 7-104.9(a). Contractor/Manufacturer is Zenith Data Systems Corporation of Hilltop Road, St. Joseph, Michigan 49085.

Trademarks and Copyrights

C86 and Optimizing C86 are trademarks of Computer Innovations, Inc.
CP/M-86, MP/M-86 and ASM-86 are trademarks of Digital Research, Inc.
IBM is a registered trademark of International Business Machines.
MS is a trademark of Microsoft Corporation
SB-DOS is a trademark of Lifeboat Laboratories
UNIX is a trademark of Bell Telephone Laboratories.

Essential Requirements for using Optimizing C86 Compiler:

a. Distribution Media: Three 5.25-inch, soft-sectored, 48-tpi disks
b. Machine Configuration (minimum): Z-100 PC, 192K memory, two floppy disk drives and CRT

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

ZENITH DATA SYSTEMS CORPORATION
ST. JOSEPH, MICHIGAN 49085

September 1984

This document describes the OPTIMIZING C86 compiler, associated programs, and support library. It is not a language reference manual. It will assist users in preparing C programs, for compilation and execution under CPM-86, MPM-86, IBM PC-DOS, MS-DOS, SB-DOS and 86DOS.

OPTIMIZING C86 USER'S MANUAL

Document Version: 2.20

Software Version: 2.20 (and later)

COMPUTER INNOVATIONS, INC

980 Shrewsbury Ave, Suite 310
Tinton Falls, N.J. 07724

TABLE OF CONTENTS

Appendices

1. GENERAL INFORMATION

1.1. Introduction.

Welcome to the world of "C" and the OPTIMIZING C86 C Compiler. We hope you will find this a useful and satisfactory product. If you have any questions, or problems, please call, telex or write.

1.1.1. Information.

This manual provides the information needed to use the Computer Innovations OPTIMIZING C86 implementation of the C Programming Language. It assumes a basic knowledge of C, your machine and your operating system. You will also need a copy of "The C Programming Language" by Kernighan and Ritchie.

If you are learning C, you will find that the examples in K&R will usually run without change. Although K&R is a very good book, we suggest that you also obtain some of the other books now available on the language. They provide valuable perspective.

1.1.2. Language features.

All language features are supported, and all programs in Kernighan and Ritchie should run. Our library includes all the standard library functions mentioned in K&R, a selection of UNIX V7 routines, and a set of machine and operating system dependent functions. Together they should let you exploit the full power of your computer, and the portability of most C code.

1.1.3. Operating systems

The OPTIMIZING C86 compiler runs on an 8086 or 8088 processor under two different operating systems. Throughout this manual we use "CPM" to refer to CPM-86, CONCURRENT-CPM86 and MPM-86 and "DOS" to refer to MS-DOS.

The documentation provided with your operating system is needed for a full understanding of some library functions. We have found that it is very important to have a technical reference for your operating system. The Z-100 PC technical reference manuals or similar ones for MS-DOS machines have some invaluable information concerning operating system interfacing from C. This information is not readily found in most DOS manuals.

1.1.4. Hardware.

To run the compiler you will need 128Kb of memory. This includes an allowance of 16Kb of memory for your operating system. You will also need at least 256Kb of disk space for the compiler, utility programs and some working space. Two disk drives are recommended. The compiler can be run on a singe drive Z-100 PC, but this may be hazardous to your sanity.

1.1.5. Support services.

If you have difficulty with the C86 compiler or documentation, we would like to hear from you. Zenith maintains a software support line for all of our software products which operates from 8:00 AM to 7:30 PM Eastern Time. This support line should be used only for questions or bug reports concerning the compiler and its documentation - we cannot help you with programming tasks. If you need assistance, or wish to report a problem with the compiler, call (616) 982-3860. This support line is provided to you at no cost so that you can make best use of the C86 compiler.

When calling our software support line, please try to have all pertinent information regarding your problem at hand, including:-

* The C86 compiler version and serial number. To obtain the version number, TYPE the file "VERSION.C86" (this file is located on the first distribution disk).
* Any options you are using to invoke the compiler.
* Names of any object libraries you link with your code.
* The type of computer you are using, along with any optional hardware which is installed in your machine.
* The version of the operating system you are using.
* The version of the ROM in your computer.

Also, please try to isolate the problem you are having as much as possible, since this will help us in solving the problem. If it is at all possible, try to have your machine available when you call - this can be very helpful to us when we are diagnosing your problem report.

For more complex issues you should send the problem to us in written form along with a listing of the code sequence which is not working, as it is very difficult to communicate long portions of code over the phone. When writing, please remember to include the information shown above. Our customer support address is:-

Zenith Data Systems
Software Consultation
Hilltop Road
St. Joseph, MI 49085

1.2. Optimizing C86 features.

Features of the C86 compiler since version 2.10:-

* Code is typically 10% to 20% smaller and twice as fast as the code generated by our version 1.33D compiler. Speed gains of up to 4 times have been noted.

* The big model switch allows programs up to the limit of your memory size in both code and data regions. Performance testing indicates that typical big memory model programs with fairly heavy pointer usage run at about the same speed as under our old 1.33 compiler.

* The compiler now produces Microsoft type object files. These files may be linked using the regular DOS linker. Thus we no longer supply our own linker or the programs that convert object codes as we did in CPM versions and older versions of the DOS compiler.

* The compiler has an option which will cause it to produce assembly source that can be assembled with MASM.

* Floating point operations using in-line 8087 code run substantially faster.

* The 8087 Trig and math library has been re-coded in assembly language to use the full power of the 8087. Speed gains in this area are impressive.

* A I/O package that takes full advantage of DOS V2.00 is now included. This gives full access to files in other directories, and is substantially smaller and faster than the DOSALL I/O package.

* Machine dependant support for DOS and the Z-100 PC has been extended with basic graphics support. A number of other functions have been added to the libraries.

* We supply DOS format object libraries, and a program (marion) to maintain these libraries. We also have a source code archiver (arch).

1.3. Version 2.20 Features

There were many new additions to version 2.20 of C86. There is improved documentation (note new section 1.14 - What to do if things go wrong), improved 8087 code generation, pathname support on include files, optimization switches for the 80186 and 80286 processors, optimizations for the compiler itself, and many new functions for the run-time libraries.

1.3.1. Ctype.h

We have added ctype.h. If you include this file the table lookup of the simple character testing functions will override the source code function calls in the run-time library. The table has what we consider to be the most used of the character testing routines and is much faster than calling the functions to do the same job.

1.3.2. New switches

There have been three new switches added to the compiler in this release as follows:

-1 produces code that is optimized to take advantage of the 80186/80286 architecture. Code produced by this switch will NOT run on the 8086 or 8088.

-h where to search for #include files (see ccl).

-e extended ASCII enabled. All characters with their 8th bits set will be converted to a 3 digit octal escape sequence \xxx.

For more information on the switches see the ccl documentation.

1.3.3. New functions

There were many new functions added to the C86 libraries. There were many functions added to take advantage of the Z-100 PC bios calls. For example, we added more graphics functions, communications functions, printer and keyboard functions that work on Z-100 PC, and MS-DOS machines. We have also added functions to make our library more compatible with version 5.0 UNIX. The following list shows the new functions:

- com_flsh
- com_getc
- com_putc
- com_rdy
- com_rst
- com_stat
- crt_cls
- crt_gmod
- crt_home
- crt_roll
- envfind
- exit_tsr
- filedir
- freopen
- intrrest
- key_getc
- key_scan
- key_shft
- prt_busy
- prt_err
- prt_putc
- prt_rst
- prt_scr
- prt_stat
- rewind
- strchr
- strpbrk
- strrchr
- toascii

For a full description of the functions see chapter 3.

1.4. Installation guide.

The OPTIMIZING C86 C Compiler package is delivered on one or more write protected diskettes. DON'T ever write on your distribution disk(s).

1.4.1. Unsqueeze the files.

You should format one or more diskettes, which will be used to hold your unsqueezed master copies of the compiler. Follow your operating system instructions to do this.

Then place your formatted diskette in drive X and one of the compiler master disks in drive Y. In the following instructions, you should substitute the actual drive letters on your system for X and Y. Usually they will be A and B.

Set X as your default drive by typing "X:".

Get the directory of the distribution disk one in drive Y by entering "dir Y:". You should see file names that are similar to the file names listed under the heading "DOS FILES". For all except the first three files, the second character of the file name extention will be the letter "Q", indicating that the files are in "squeezed" format. This format reduces the amount of disk space required to ship the files, and also provides checksums to check the file content.

Find the disk containing the file "read.me". Type this file by entering "type Y:read.me". If we need to provide any additional information, this file will contain it. Follow any such instructions.

Copy the unsqueeze program "usq.exe" to your working disk by entering "copy Y:usq.exe/v". You will need to do this for each of your unsqueezed master diskettes.

Then for each file on the master diskette with a "q" as the second letter of it's file name extention run the unsqueeze program. For example, to unsqueeze the file "stdio.hq" on the master disk enter "usq Y:stdio.hq". The unsqueezed version of this file will be placed on drive "X" with the name "stdio.h". Repeat this process for each of the squeezed master files. The unsqeeze program will take wildcards for the filenames but this should be done carefully. You must take into account the disk space limitations on your computer.

Note that the "unsqueezed" files are usually much larger than the "squeezed" files. Any problems reported indicate some machine malfunction, software problem or faulty distribution disks. If you conclude that your distribution disks are faulty, please contact your supplier. If all else fails, call us.

Finally write protect and label your unsqueezed master diskettes.

1.4.2. DOS files.

MS-DOS distribution disks contain the following files. Note that only the first three files are shipped in "unsqueezed" format.

	version.c86	The version of the files on this disk.
	read.me	Final instructions and notes.
	usq.exe	Unsqueeze program.
*	stdio.h	Standard header file.
*	ccl.exe	Compiler pass 1.

*	cc2.exe	Compiler pass 2.
*	cc3.exe	Compiler pass 3.
*	cc4.exe	Compiler pass 4.
	arch.exe	Source library maintenance program.
	marion.exe	Relocatable library maintenance program.
	base.arc	The basic support library.
	dosall.arc	I/O library for all versions of DOS.
	dos2.arc	I/O library for DOS 2.0+ (includes pathnames)
	mathbase.arc	The basic math library.
	mathsft.arc	Software 8087 math routines.
	math87.arc	Hardware 8087 math routines.
	zdspc.arc	Z-100 PC routines (non-portable)
*	c86sas.lib	Library; small, dosall, mathsft.
	c86san.lib	Library; small, dosall, math87.
	c86s2s.lib	Library; small, dos2, mathsft.
	c86s2n.lib	Library; small, dos2, math87.
	c86bas.lib	Library; big, dosall, mathsft.
	c86ban.lib	Library; big, dosall, math87.
	c86b2s.lib	Library; big, dos2, mathsft.
	c86b2n.lib	Library; big, dos2, math87.
	zdspcs.lib	Library; small, Z-100 PC routines.
	zdspcb.lib	Library; big, Z-100 PC routines.
	prologue.h	Assembly header file.
	epilogue.h	Assembly trailer file.
	model.h	Assembly big/small control file.
	error.h	Error definition header file.
	ctype.h	Character class table

1.4.3. Transfer files.

Copy the files you need from your unsqueezed master disks to working disks. For your initial use you will need the files marked with an asterisk ("*") above. The remaining files are not needed at this time.

You should also copy any other needed utility programs, such as a text editor, to your working disk.

1.4.4. Create a test program.

Create a source program. We suggest you use the editor supplied with your operating system, although any editor that creates standard text files should be satisfactory. Since C is a case sensitive language "main" is different from "MAIN". You will need to type your source code in lower case for it to work properly. As an example we will use "hello.c", which contains:-

```
#include "stdio.h"

main()
{
     printf("Hello, World\n");
}
```

1.4.5. Compilation.

To compile the program, type the following four lines:-

```
     X:cc1 Y:hello
     X:cc2 Y:hello
     X:cc3 Y:hello
     X:cc4 Y:hello
```

where disk drive "X:" contains the compiler and disk drive "Y:" contains the source program. You should substitute the correct designators for your system in the above command lines. These are usually "A:" and "B:". They may be omitted for files residing on the default drive (Usually drive "A:").

The result of this process will be the file "Y:hello.obj".

1.4.6. Linking.

Link the program to get the executable version by typing:-

```
     X:link Y:hello,,con/map,X:c86sas
```

We assume that the library ("c86sas.lib") resides on the same disk as the programs. More information on the use of the linker may be obtained by consulting you DOS manual.

1.4.7. Execution.

The program should be executed by typing "Y:hello" (without the quotes), followed by a carriage return (or enter). The program will then type the message "Hello, World" on the console.

1.5. Using the compiler.

1.5.1. Batch files.

The commands required to run the compiler may be placed in a "batch" file to reduce typing and errors. All the programs return termination status, so that under DOS 2.0+ the batch file may be arranged to terminate on error. We use the following batch file (named cc.bat) for most compilations.

```
     cc1 %2 %1
     if errorlevel 1 goto done
     cc2 %1
     if errorlevel 1 goto done
     cc3 %1
     if errorlevel 1 goto done
     cc4 %1
     if errorlevel 1 goto done
     goto allok
     :done
     pause error in compilation
     :allok
```

The first argument is the name of the source file, the second is optional, and is any compiler switches. for example:-

```
     cc xyz              Would do a simple compilation.
     cc xyz -nb          Compile "big" and "8087".
```

See your Operating System documentation for more information.

1.5.2. Using the DOS linker.

The standard DOS linker should be used to link your compiled code with one of our libraries. Multiple ".obj" files and multiple libraries may be used to create one ".exe" file.

As far as we have been able to check, an object file will always over-ride a library member which has the same public symbols. However, library searching is performed once only, from left to right. Therefore, if you link with the libraries "A+B+C", library B cannot call out any modules from library A, and library C cannot call out anything from A or B. As a result, you should always place our library last, since it will never call out any module from any other library.

See your link documentation for information on producing maps and on automatic response files.

1.5.3. Using CHKDSK.

Aborting any program that has an open output disk file causes DOS to "forget" that part of your disk space. If you do it often enough, you will have no usable disk space left. This space may be reclaimed by the DOS program CHKDSK. We suggest that you run CHKDSK once a day on your working disks.

1.6. File system.

The i/o packages are intended to present a UNIX like interface to the programmer. This section provides information that will help you understand the implications of the design.

A standard UNIX system has two groups of input/output functions. The basic group provides i/o buffered by the operating system, and very few services for programmers. The second group provides within-program buffering and a large collection of services.

We have provided both sets of functions. They react as a UNIX based program would expect. The following sections provide more information.

1.6.1. Basic services.

The basic services are provided by the functions:-

open	Open an existing file.
creat	Erase any existing file, then open a new (empty) file.
close	Close a file.
read	Read bytes from a file.
write	Write bytes to a file
lseek	Position in a file.
ltell	Report current position in file.

All the above functions identify the file by a file descriptor, which is returned by open and create, and input to all other functions. By UNIX convention, file descriptors are small non-negative integers, and open/create must return the smallest possible file descriptor for any call. You can depend on this. As a result, you can predict the file descriptors returned by a sequence of open/creat/close calls.

1.6.2. Stream services.

All other input/output functions use a stream identifier. This provides a variety of useful services including formatted input and output. These functions should be used by your programs for portability, and generality.

A stream identifier is returned by the function "fopen". Stream identifiers are by UNIX convention pointers to type "FILE", where "FILE" is defined in "stdio.h". The actual definition is implementation dependent, and our DOSALL and DOS2 libraries have different definitions. You should not assume any relationship between file descriptors and stream identifiers in your programs.

1.6.3. The DOSALL I/O library.

This I/O library will run on all versions of DOS. It provides all the services available under DOS 1.1, along with correct redirection handling under any version of DOS. It does support path names when executed under DOS 2.0. You should use this library if you are writing programs that will have to run under DOS 1.0, 1.1 or 1.25.

1.6.4. The DOS2 I/O library.

You should use this library if you can. It is smaller, faster and more extensive than the DOSALL library, and will have more of our attention. It provides full use of the DOS 2.0 I/O system, including path names, but can only be used with DOS 2.0+ operating systems. The choice is yours.

1.6.5. File opening modes.

Because DOS and UNIX have different end of line and end of file conventions, we have had to make a distinction between binary and

ASCII data. This was done in the file opening logic to minimize program conversion effort. Thus we have ASCII and BINARY open modes for files. Once the mode is chosen, the remainder of the program should work without problems, unless you mix ASCII and binary data in one file.

1.6.6. DOS character devices.

We provide special processing for "CON:", "PRN:" and "AUX:", which refer to the console, printer and auxiliary communications port respectively. These files may not be opened in update mode, but may be opened more than once in a program. Under the DOSALL library bdos calls 1 through 5 are used to transfer this data in unbuffered mode. These are treated as regular files under the DOS2 library. We have heard of some bugs in DOS 2.0 that cause some problems when treating the serial port as a file. You should use the new functions that communicate with the serial port in the V2.20 C86 library if you possibly can.

1.6.7. Console input.

Special processing is provided for the CONSOLE ("CON:") if it is opened in ASCII input mode. In this case we always input a complete line of input from the keyboard. Thus the line editing characters are available to the user. Note that files open to the keyboard do NOT share a common buffer, so that interlaced reads on more than one file opened to the keyboard may give surprising results. This may change in a future release.

This mechanism provides that console input is read a line at a time. Thus your program will wait until a return has been input before proceeding. If you want to input single characters, and do not require the carriage return, you should open another channel to the console in binary mode, or use bdos() calls, which will also give you control of echoing.

1.6.8. Standard files.

Three files, named stdin, stdout and stderr, are opened at program initiation. These files are available to the program for the function indicated by their names. By default, these files are opened to the console ("CON:") in ASCII mode.

The defaults for stdin and stdout may be changed by a process known as redirection on the command line that invokes the program. Redirection is specified in the command line as follows:-

```
     progname <newin >newout other arguments
```

Which causes "progname" to read its input from the file "newin" instead of the keyboard and write its output to the file "newout" instead of the console. The filenames may be any legal filename or any of the character device file names. Of course, you can't open the printer for input.

If stdin and stdout are both redirected to the same disk file, THE INPUT FILE WILL BE DESTROYED, USUALLY BEFORE ALL THE INPUT HAS BEEN READ.

You may also append data to the end of an existing ASCII file, using the command:-

```
progname >>appout other arguments
```

If file "appout" does not exist, it will be created.

1.7. Language information.

The following information is needed for a full definition of the C86 programming environment. It should be used in conjunction with K&R.

1.7.1. Data types.

The supported data types and their sizes are:-

char	8 bits	
unsigned char	8 bits	
short	16 bits	
unsigned short	16 bits	
int	16 bits	
unsigned int	16 bits	
pointer	16 bits or 32 bits	
long	32 bits	
unsigned long	32 bits	
float	32 bits	(8087 format)
double	64 bits	(Used for f.p. calculations)

Pointers are 16 bits long in the small model and 32 bits in the big model. All pointers in a single program must be the same length.

The floating point data storage uses the 8087 data formats, even if you are using the floating point software. Our bulletin board has a complete description of the floating point formats. To have access to the bulletin board you need to join our user group, which can be done by contacting Computer Innovations. Float data has an 8 bit exponent, allowing numbers up to approximately 1e+38. Double precision has an 11 bit exponent, allowing numbers up to approximately 1e+308. Floats are ALWAYS converted to double before being used or passed to a function. ALL floating point calculations are done in double precision.

1.7.2. Storage types.

Auto, extern and static are as defined in K&R. Register class is converted to auto for the moment, but this will change as we enable more of our optimization logic. If you are writing new code, then you should apply register class where appropriate in preparation for future releases of the compiler.

1.7.3. Definition of extern.

The use of the keyword "extern" varies radically from compiler to compiler. Our definition is taken from K&R page 206 item 11.2. It is:-

> Any external data item must occur exactly once without the keyword "extern". This entry may have an initializer. It causes storage to be reserved for the data item.
>
> All other entries must include the keyword "extern" and may not contain an initializer. They do not cause storage to be reserved.

1.7.4. Debugging.

Most debugging should be done by adding print statements and recompiling the program.. However, your standard debugging package may be used to debug programs. You can obtain a memory map from your linker. You may then place break points on the first instruction of a function, and examine local variables and arguments. We have found that the DOS debug package is much more difficult to use under the big model than the small, so work with the small model if you can.

When all else fails, the assembly source output from the compiler should help you to find bugs. Note that because of problems in the assembler, the assembled assembly source code may not match the direct object code exactly, although functionally it will be identical. Typically the assembler may chose another op code for the same instruction. For example it may use a regular three byte jump instruction, where the compiler will use a two byte short jump.

In the generated assembly code, labels lines take the form:-

```
     .0XXX              ;NNN
```

where the xxx is the approximate address of the label in the assembled code and NNN is the approximate line number of the source code that resulted in the surrounding assembly code.

It is usually easier to add a few print statements and recompile. A symbolic debugger is available. It is called PFIX86 PLUS and is a very useful symbolic debugger. For more information contact Computer Innovations.

1.8. Converting to V2.20 from V1.33.

All changes made to the library are intended to make OPTIMIZING C86 conform to the standards in K&R. If you find unusual things happening, you should examine the library documentation, or even the library source code for changes in definitions. Very few changes have occurred, but it is probable that the 2.0 and DOS-ALL libraries have minor differences. In particular, the definition of STRNCPY was incorrect, and has been changed to conform to UNIX usage.

1.8.1. Pointers.

If you are converting running code to the big model, your most probable errors will be mis-use of pointers. Since the big model uses 16 bit ints and 32 bit pointers, pointers and ints are no longer equivalent. You must declare pointers as pointers for the correct code to be generated. You must also declare functions that return pointers to be returning pointers. This has caused us significant problems, as the compiler does not provide much help in detecting such errors.

Code that uses the function "segread" under the small model almost certainly should NOT use it in the big model. Examine library functions, such as "_open" in DOSALL, for examples of the correct code.

1.8.2. 8087 support.

This version of the compiler generates in-line 8087 code and also includes a trig and math package for direct use of the 8087. Note that this library is distinct from our C language trig and math functions for use with floating point software. This should result in your floating point code running about three times faster.

1.8.3. 2.0 I/O library.

This is provided as an alternative to the standard DOSALL library. It provides additional functions, such as change directory, make directory and full path name support. This will finally emerge as our standard library, and most I/O performance enhancements will be done to this library only. Use this library if you can. The DOS2 libraries all have a "2" as the middle character in the library descriptor (e.g. C86s2s).

1.9. Assembly language functions.

This section provides basic information required to write assembly language functions. This information may change with future versions of the compiler. If you must use assembly code, keep the function short and to the point. Most things can be done in C.

Something we should point out now is that the layout and format for assembly language source files that you are linking with C86 are important. You should follow the layout of some of our assembly functions in the run-time archives. It would be a very good idea if you could read and understand such things as where local data goes, how to interact with the big model, segment names, etc. from our source code. A very good example is the "write.asm" source in the DOS2 run-time source archive.

1.9.1. Header files.

Three header files are provided to support assembly language programming. They are:-

model.h	Defines the model to assemble.
prologue.h	Defines segment names.
epilogue.h	Closes the assembly file.

These files are intended to make your assembly code easier to convert if any changes are needed for future releases. You should include each of these three files in any assembly code you write. Your code should be placed after the inclusion of prologue. See a small library routine like peek.asm for an example.

We would like to note that common versions of masm have a number of problems, and that the code you write may not be the code you get. You must check the resultant code with the debugger in a test program. The most common problems are:-

Incorrect data addresses.	(bad relocation)
Incorrect instructions.	(wrong op code or address fields)
Incorrect code addresses	(segment nesting fails)

1.9.2. Model.h

This file should be included at the head of any assembly language function. It is used to define the assembly language switch "@BIGMODEL", which currently takes the value "0" for a small model assembly, and "1" for big model assemblies. Note that we plan other values for later releases.

1.9.3. Prologue.h

The file "prologue.h" should be included immediately after "model.h". It defines all the basic segment names in the correct order to result in a correct link and also defines the symbol "@AB" which is the argument base for the chosen model, provided you have used the standard entry logic, which consists of:-

```
PUSH     BP
MOV      BP,SP
```

The standard return logic consists of:-

```
MOV      SP,BP
POP      BP
```

The segment names defined by prologue.h and the order that they have to occur in memory after linking are:-

@CODE	Code goes here in the small model.
@DATAB	Beginning of data segment.
@DATAC	Character strings and constants.
@DATAI	Initialised global data.
@DATAT	To find out where DATAU begins.
@DATAU	Uninitialised global data.
@DATAV	To find out where DATAU ends.

The compiler will create additional code segment names if you are compiling with the big model switch. These segments will always preceed the data area in memory when the code is linked.

All your code should be placed in @CODE, and any data in @DATAI. You can put your data in the other segments, as long as you qualify all references by "DGROUP:". Note that the DATAU area is initialized to zero at program startup time.

The file PROLOGUE.H should be included in all assembly language code. If you invent any additional segment names, make sure they are linked into reasonable places. The run time system makes assumptions about the order of segments in memory.

1.9.4. Calling conventions for functions.

Calls to functions, in C or assembly language, use the following conventions:-

Calling C functions push the left-most argument last, and the right most argument first. Thus the left-most argument is at the "top" of the stack.

All character arguments are converted to int before being placed on the stack.

Registers ax,bx,cx,dx,si and di may be used by the called function, and do not have to be saved or restored. THIS MAY CHANGE IN A FUTURE RELEASE.

The called function is entered by a "near" call in the small model and a "far" call in the big model.

Registers cs, ss, ds, es and bp must be preserved. Register bp is the frame pointer.

1.9.5. Returned results.

char, short and int are returned in register ax.

long is returned in registers ax and dx, ax is least significant.

Doubles are returned in ax, dx, bx and cx where ax is least significant and cx contains the exponent. This WILL change in the near future.

Big model pointers are returned in ax, dx.

Arguments are popped from the stack by the calling function, since the called function does not know how many arguments were supplied.

FUNCTIONS WHOSE NAMES BEGIN WITH A DOLLAR SIGN ARE INTERNAL COMPILER KNOWN FUNCTIONS AND DO NOT NECESSARILY FOLLOW THESE CONVENTIONS. These may be changed but be careful to read the existing code to determine the conventions used. They vary from function to function.

See the distributed archives for samples of assembly language code.

1.10. Compiler options.

1.10.1. Big model switch.

If you do not use this switch, you will generate a small model program, which allows you up to 64Kb of code plus 64Kb of data. Over 90% of all C programs will run in the small model, and they will be smaller and run faster.

The big model switch will allow the whole program to be up to 1000Kb in size, but the following additional limits apply:-

No one source file may generate more than 64Kb of code. This limit would be hard to exceed.

The total of global and static data must not exceed 64Kb. This limit may be exceeded by having large arrays in your program. The way around that is to create a pointer to the array data and allocate space to the pointer at run time. As a result, no single array may contain more than 64Kb of data. But you can have more than one array. This should not result in having to change much of your code.

The total stack space is limited to 64Kb. Again, local arrays could use up all available stack space, and the solution is to use dynamic allocation.

The remainder of memory is available from the heap in chunks just less than 64Kb in length.

In the big model pointers are 32 bits. The pointers hold both the segment and offset parts of the address. The most significant two bytes of the double word pointer holds the segment and the least significant word hold the offset. The following shows how to break up the pointer into its parts:

```
/* big model example */
char *bigpointer;
int segment, offset;

segment = ((unsigned long)bigpointer)>>16;
offset  = bigpointer;
```

There are examples on how to get the proper formats all through the run-time library source code. Look at the code for further examples.

1.10.2. 8087 switch.

This switch generates code to use 8087 hardware instead of the software package. The result is much faster execution. Note that 8087 code run on a machine without an 8087 will cause the machine to hang, and you will have to re-boot.

1.11. Overlays.

We no longer support our 1.33 overlay mechanism. The only available solution is to use PLINK86.

1.11.1. Plink86.

It will handle overlay systems under both the small and big models, but we have had some reports that it is unable to handle really complex overlay structures. You will need to obtain version 1.30 (or later) of PLINK86, which apparently cures a number of problems in the previous versions.

It is important that you use a "class" statement, to force the data areas to be placed after ALL your code, and that the data segments be in the correct order. To get this to work you must therefore put the "class" statement at the end of your PLINK86 commands. Do not put the "class" statement in your PLINK86 file if you are not using overlays, it will not work. We used the following "class" statement:-

```
class DATAB,DATAC,DATAI,DATAT,DATAU,DATAV,HEAP,STACK
```

1.12. Hints and other comments.

The following notes may help you with coding problems that are frequently reported to us.

1.12.1. Kernighan and Ritchie.

Many of the programs in K&R will not compile and/or link correctly unless you insert the following as the first line of the program:-

```
#include "stdio.h"
```

You will also have to change "open" and "fopen" calls to match our conventions, which differ slightly from the standard UNIX conventions.

1.12.2. Initializing structures and arrays.

K&R do not point out that local arrays and structures (ie defined inside a function) cannot be initialized at compile time unless they are declared static. Thus inside the function main() the following is valid:-

```
     main()
     {
       static char *days[]={"mon","tue","wed"};

     }
```

Without the word static, you will get the warning "initializer needs lval". This type of initialization is always valid at a global level, with or without the word "static".

1.12.3. String initializers for character arrays in structures.

If you have a character array as a member of a structure, and you wish to initialize it with a string, enclose the string in braces. eg {"initializer string"}.

1.12.4. Structure and union member names.

In this version of the compiler, all structure and union member names share the same name space. Therefore, the compiler considers that any member can be part of any structure/union. The alternative is to assume that each structure/union defines a separate name space, and to provide much tighter checking of member names. We plan to add a switch to provide this checking in a future release.

To remain fully portable, we recommend that each member name be unique. We always begin each member name with some mnemonic for the structure/union that contains it.

1.12.5. Assigning pointer and int data types.

In most C Compilers integers and pointers are of the same size (in bits). Programmers frequently save an integer in a pointer, or a pointer in an integer. This is NOT a portable construction, and K&R specifically warns against this practice.

In this version of C86, if a program is compiled with the big model option, pointers are 32 bits, and these type of assignments will provide interesting debugging experiences. You should also make sure that any function that returns a pointer (including library functions), is declared before it's first use. The compiler does not warn about these constructs, because in some cases they are legal. Take care.

1.12.6. Redefinition of function name error message.

A number of users have had problems with this error message. It is caused by calling a function before it is defined. When a function is used before it is defined, the compiler assumes that the function returns an int. If you later define the function to return any other data type, then you really have redefined the function. To eliminate this error message, add an extern definition of the function, with the correct data type, before its first use.

1.12.7. Run time error messages.

There are a number of run time error messages embedded deep within the system. The message is the name of a function or generic type of function in upper case. For example "WRITE", or "ALLOC". These are used to deal with totally impossible error situations, which should only be encountered during debugging. These error messages are always written to the console, and are always followed by an abort. See "_exit" for help in locating these problems.

We would also like to mention the "BAD FILE" error message, which is caused by trying to use a file that is not currently open. This message will only appear in DOSALL libraries.

Unusual error messages can also be produced by programs that have destroyed memory.

1.12.8. Undocumented functions.

We often add functions to the library before we add library documentation. An examination of the library source archives may be interesting. It will also be useful to read the source code in our libraries when you need some examples of C code.

1.12.9. Eliminating the standard functions.

If you don't need any of the standard library functions, you should modify the functions "_exit", "$main" and "_main". These functions call in most of the standard library code. You can probably eliminate all of "_main", since it is mainly responsible for file redirection and argument parsing.

You should see the function "_fmtout" and "_fmtin" if you want to eliminate the floating point support library. This will save about 1.5Kb of code.

Since the linker only includes functions needed by the program being linked, this will reduce the size of the final program.

1.12.10. Trig library.

The C language versions of the trig functions were written using the book "Software Manual for the Elementary Functions" by Cody and Waite (Prentice-Hall 1980). You may find this book to be a useful reference for more information on the algorithms used in these functions. The 8087 assembler trig functions were created from code donated by a number of generous compiler owners.

1.12.11. Creating COM files.

The only reason that we have found to create a COM file is for a device driver. If you are writing a device driver and would like to know all of the procedures on how to create a COM file the information is on our user group bulletin board. To find out more about our user group contact our sales staff for more information. Also check future versions of our user group newsletter for the description on how to do this, it might just show up there also.

1.12.12. Creating ROM files.

We are now selling a set of routines to help you create "romable" code at a nominal fee. This package is called ROMPAK and to find out more about it contact Computer Innovations.

1.12.13. Using 8 bit characters in strings.

There is now a new switch for the compiler which allows extended ascii characters to appear in your C programs. The '-e' will convert all characters with their 8th bit set to a 3 digit octal escape sequence \xxx. There is now no need for the "allbits" program of earlier versions of C86. Be warned that some text editors will not work with this switch. See ccl documentation for more details.

1.12.14. Converting BDS-C programs.

You will have to modify some library function calls and the global data definitions.

If you have a number of programs, it may be quicker to modify our library functions to match the BDS-C definitions. We hope that the C86 users group will be able to provide such a library.

We recommend the following method of converting the global data definitions. We assume that the global definitions are in one file which is "#included" with each of the source files.

Add the word "EXTERN" in front of each existing definition.

In one file, before the "#include" statement for the global definitions, add the definition:-

```
#define EXTERN
```

In all other files add the definition:-

```
#define EXTERN extern
```

You should also use the "-u" switch with program CC2.

1.12.15. Pathnames.

There is now full pathname support in both the DOSALL and DOS2 libraries. There are also full pathnames available for the #include preprocessor statement. See the documentation for ccl for more details. One other note under this section, if you ever include a pathname in a character string constant make sure that you have two backslashes for every one in the string. In other words, the backslash is the escape character in C.

1.12.16. Porting code to OPTIMIZING C86.

A number of users have transported code that compiled under UNIX and other C Compilers to OPTIMIZING C86. Generally there was little or no conversion effort involved, except for the very non-standard i/o libraries provided with some compilers.

Some of our compile time switches are intended to reduce the problems of portability. See the program description sections for details.

1.13. Memory layout under DOS.

The following shows the layout of memory when a typical C program (such as the compiler) is executing under DOS.

```
 ---------------------------------------
|interrupt vectors                      |
|                                       |
|---------------------------------------|
|DOS code and dos controlled areas      |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|program segment prefix                 |
|---------------------------------------|
|code of c program                      |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|data (static) of c program             |
|                                       |
|                                       |
|    ------------------------------     |
|heap (dynamic)  Expands towards stack  |
|                                       |
|    ------------------------------     |
|unused, available for heap or stack    |
|growth                                 |
|                                       |
|    ------------------------------     |
|                                       |
|                                       |
|                                       |
|stack (grows towards the heap)         |
|---------------------------------------|
|                                       |
| unused memory                         |
|                                       |
|---------------------------------------|
| command.com (resident part)           |
|                                       |
 ---------------------------------------
```

With the small model, code and data areas are limited to 64Kb each. In the big model, total code and data are limited by the size of memory, but the static and stack sub-sections of data are limited to 64Kb each.

1.14. What to do if things go wrong.

This section of the manual will be the place where hints and help will be given out for what we consider to be the most recurring problems with C86 usage.

1.14.1. Problems with functions in the library.

In general, if you are having problems with a library function, try and create a sample program. You could try to run the example program found in chapter 3 under the function description. Sometimes it is as simple as not declaring a function that returns a non-integer.

1.14.2. Big Model and memory limits.

A common problem with C86 big model users is that they say that they have "a 256K machine and I can only allocate about 80 or 90K". The reason why is that we only access 96K for the stack and the heap as a default. This way users of C86 can both use the big model on just about all machines and also use the "system()" function. Since the "system()" function needs unused memory outside of where the program is loaded you cannot access all of the machine's memory for the data segment if you want to use the function. You can change the value of _MAXFMEM in _default.c, (see _default in ch. 3) recompile the function, and include the object module in your link edit step to access more memory for the big model. The number in _MAXFMEM is the number of 16 byte paragraphs. If you access all of memory you will not be able to use the system function.

1.14.3. Problems with opening a file.

Make sure you always check the returns on a file open. Also make sure that when you include a path name that you use two backslashes for one inside double quotes. Sometimes you will also run into a file limit when using the DOS2 libraries. The DOS2 file limit defaults to 8 and 3 or 4 files are already opened before your program starts to execute. This default can be changed in your DOS config.sys file. Fopen also needs about 1K on the heap in the small model and 2K in big model for the file buffer.

1.14.4. System function.

If you are having problems with fitting your program into memory when using the system function there are some things you can do. If you can somehow use less data space for you programs (either the one calling the system function or the program called by it) you can shrink the size needed by changing _default.c. The default variable that you want to change is _MAXFMEM. When you shrink this value, make sure you are careful. It is much better to err on the side of having extra unused memory than to make the program unusable.

If nothing happens when you invoke the system function, make sure that command.com is found either in your path or on the default drive and directory.

Also system will not work with your switchar set to another character. You can either modify the source to the system function to cope with your switch character or you can reset it to the default.

1.14.5. Interrupts and intrinit.

Since this is a somewhat tricky area you should really understand the documentation and examples in chapter 3 under the intrinit function. Something that is very important to mention again here is that the functions that will be invoked by the interrupt handler must be very, very short. Don't ever do a printf() call inside one of those functions. It will be too much code. Certain interrupts are so frequent (the timer tick for instance) that you can hardly do anything inside the interrupt function.

1.14.6. Serial port communications.

The new functions added to version 2.20 of C86 should help greatly with the communications problem. If you can live with the buffering and speed limitations this is definitely the way to go. Treating the COM ports as a file and using the I/O system are hazardous under DOS 2.0, so that should be avoided. On faster machines we have achieved 9600 baud rates using our library functions, so it can be done. If you really need to get high data transfer rates you should write your own interrupt driven communications package.

1.14.7. Funny errors out of ccl.

If you are getting funny errors out of ccl, run that pass of the compiler over again with the '-p' switch set on so you can see what the preprocessor is generating. Usually it becomes apparent what is wrong when you see this output. Some of the more common mistakes are that you have a "/*" in your program. A beginning of comment inside double quotes must be esacaped when it appears in the program (i.e. "/*"). Also check that all included files have a newline at the end of the file. This would cause a line wrapping problem.

1.14.8. Big model pointer arithmetic.

Version 2.20 of C86, assumes that big model programs do all pointer arithmetic in the same data segment. There is a good chance this may change in later versions. But for now you must do pointer arithmetic in absolute addresses by using the ptrtoabs() and abstoptr() functions if you cannot assume the same data segment in the big model.

1.14.9. Undefined results, (or a lesson in uninitialized pointers)

If you find that you are getting memory corruption or that some other strange, undefined results are happening, you should check for uninitialized pointers. In C when you declare a variable to be a pointer to a certain data type you must then have it point to a reserved space in memory to hold that data type. Just declaring a pointer does not save space for the variable. If you are confused by this whole area of C, there is a clear discussion of pointers in the book "The C Programming Tutor" by Leon A. Wortman and Thomas O. Sidebottom. Chapter 7 of this book is very well written.

Another very common cause of strange results and memory corruption is array subscripts out of bounds. This should be checked very carefully when tracking down strange results.

1.14.10. Scanf and it's use.

Scanf and fscanf are for formatted input. Since humans are not formatted individuals scanf should not be used to gather human inputs. Scanf works best when it is reading inputs that were formatted by machines (e.g. from printf). When dealing with human input it is much better to use gets or fgets to read the input into a buffer. The buffer can then be handled much easier by the programmer for error checking, formatting, etc. than if you were trying to do the same thing from stdin.

Scanf also works on ascii representations of the data. To read a binary file of integers for example, you would not use scanf. To read the binary formats you would have to use read or fread to read the data directly into a memory location.

1.14.11. Strncpy

You should be very careful of the use of the strncpy function. It does not always put a NULL after the destination string. Make sure you carefully read the description of the function in chapter 3 if you think you are having trouble with it. You may want to rewrite this function to suit your needs.

1.14.12. "Fixup offset exceeds field width"...

If you are getting this message from the DOS linker that you are using you are most likely doing one of the following things:

* You have mixed near calls with far calls. This means you are linking together object code compiled with the big model and object code compiled with the small model. Double check this and re-compile if you are not sure how some of your object modules were compiled.

- *If you are linking in any assembler modules you probably are not following the layout and formats needed for the assembler source code. You should definitly understand the section on assembly language functions in this chapter. Also, make sure you follow the layout and format of one of our assembler functions that make up the run-time library. A good example is "write.asm" in the DOS2 archive. The local data of the assembler function must be put in one of the data segments defined by our prologue.h file. Also, the names of the code and data segments have changed from previous versions of C86. Double-check to make sure that all of your assembler code is in the correct format.

- * For some strange reason the linker gives this error message if you exceed the 64K limit in global and static data. What this usually means is that you either have one large global array or many small ones which when added up exceed the limit. The best way to check this out is to add the sizes of the data segments from the link map output. If this exceeds 64K you must shrink it down by dynamically allocating some of the data regions rather than having then be global.

1.14.13. If all else fails...

We would appreciate it if you did a few things for us before you called with a problem with C86. First try and read the manual again in the appropiate places for some information. As strange as it may seem, the answer to your problem might be found that way. If you feel you have found a bug after this, please have an example that is as simple as possible. This helps us greatly. Also, include in your correspondence your serial number or have it ready if you call. If you send a letter please include a phone number so we can call you if we need more information.

2. PROGRAM DESCRIPTIONS

2.1. ccl, preprocessor.

2.1.1. Function.

This pass of the compiler reads the source program, processes lines beginning with the preprocessor control character ("#"), and outputs a file of lexemes.

2.1.2. Usage.

The command line required to execute the preprocessor is:-

```
     ccl [-labcdehinpstu] filename
```

If the file name does not include a period, a default extension of ".c" is assumed. The output file has a ".$cl" extension.

2.1.3. Flags.

All compiler flags, for all passes of the compiler may be input to ccl, which makes batch files less complex. The flags are:-

-l Produce code that is optimized to take advantage of the 80186 or the 80286 architecture. Code produced by this switch WILL NOT run on the 8088 or the 8086.

-a (Effects cc4) Generate assembly source code instead of a ".obj" output file.

-b Big memory model switch. If this switch is on all pointers are four bytes in length and functions that return pointers MUST be declared before use. All parts of an executable program MUST be compiled or assembled with the same setting of this switch.

-c Indicates that comments are nested. With this flag the compiler expects each opening comment token ("/*") to have a matching closing comment token ("*/"). Without it, comments are processed as specified in K&R. We recommend AGAINST the use of this switch. If you wish to comment out blocks of code, use an #ifdef with a symbol that is never defined. Because this switch exists, comment tokens inside quotes in your program have to be separated with a back-slash. WARNING: this switch will go away in the near future.

-d Defines the characters that follow, up to the next white space, as if they had been specified in a "#define" statement before the first line of the source program. More than one of these flags may be present in the command line. You can only define the presence of the name; you cannot give it a value.

-e enables processing of C source files with extended ASCII characters in them. Any character encountered in the source file with it's 8th bit set will be converted to a 3 digit octal escape sequence. This flag may cause very strange results when used with certain editors. You should note that the only safe place to include extended ascii characters is inside character strings.

-h where to search for #include files. (See notes)

-i Identifiers are significant to 31 characters instead of 8. The result is non portable but more maintainable.

-n (Effects cc3) Generate code to use the 8087 Numeric data processor chip for all floating point operations. Otherwise code will be produced for the floating point package.

-p Print a listing of the source program to stdout after all preprocessor actions have been performed. This is a valuable debugging tool for problems caused by #define statements.

-s (Effects cc2) Process string literals as an array of "unsigned char" instead of an array of "char".

-t (Effects cc3) Generate calls for program tracing. This currently generates a call to $entry before any other code for each function. The $entry function may be changed as desired to assist in program debugging. See the description of $entry in the library.

-u (Effects cc2) Treat all occurrences of the reserved word "char" as an occurrence of "unsigned char". This is provided for compatibility with some 8080 C Compilers. It also sets the "-s" flag.

2.1.4. Notes.

How to use the '-h' switch:

-hsystem[,project]

system is the name of the path to search for system files. project is the name of the path to search for project files. These places will be searched if the filename does not contain a drive or pathname specifier. The order of searching is as follows:

For files which are #included with "" :

search the path specified by the source file. (e.g. if you are compiling c:\c86\program.c and you #include "stdio.h" the compiler will attempt to open c:\c86\stdio.h first).

For files which are #included with <> or "" :

search according to the path specified by the project pathname.

search according to the path specified by the system pathname.

Example:

```
ccl program -h\c86\,\new\
```

program.c contains: #include <stdio.h>

the compiler will attempt to open stdio.h in the following order:

```
"\new\stdio.h"
"\c86\stdio.h"
```

Example:

```
ccl \c86\program -hc:\system\,\project\
```

program.c contains #include "stdio.h"

the compiler will attempt to open stdio.h in the following order:

```
"\c86\stdio.h"
"\project\stdio.h"
"c:\system\stdio.h"
```

Remember that if you have a ':', '/', or '\' in the #include filename, the compiler will try to open it "as is" and no other searching will be done. It is also important to remember that project and system path names must be termination by "\".

2.1.5. Features.

All preprocessor lines begin with a "#" in column one of a line. As a special feature of the current version of ccl, this character may NOT be followed by any spaces.

The preprocessor implements the following features:-

#include "filename" - Include the content of a file. To find out how to use it with the '-h' switch see the notes above. If you do not use the '-h' switch it will look in the following places for the filename to be included:-

1. The directory that the C source file is in.
2. The default directory.

#include <filename> - See notes above.

#define name xxx - Replace each instance of "name" (name must be an identifier) by xxx in all the following text of the program. The replacement text, "xxx" may be any sequence of characters, spaces and tabs, terminated by the end of the source line.

#define name(args) replacement text - Replace each instance of "name(args)" by the replacement text. The args in the defined replacement text are replaced by the supplied arguments. The number of supplied arguments must match the number of actual arguments.

#ifdef name - Include following code if "name" has been defined in a "#define" statement.

#ifndef name - Include following code if "name" has not been defined in a "#define" statement.

#if expr - Include the following code if the expression is "true" (not zero). Only constants are allowed in the expression.

#else - Include or exclude the following code based on the inversion of the matching previous conditional expression.

#endif - Terminate the action caused by the previous conditional.

#undef name - Remove the most recent definition of name, if any.

2.1.6. Line continuation.

All lines, not just quoted literals, may be continued by placing a backslash as the last character of the line. This is handy for long literals and macro definitions. No line or literal may be longer than 512 characters.

2.1.7. Error messages.

We hope they are self explanatory. We report the line number at which we detected an error, which may not be the same as the line that contained the cause of the error. If the message does not seem to apply to the line reported, look at the previous line(s).

We report out of balance (), [] and {} pairs within a file. The message contains the line where the error was detected and the line containing the opening (, [or {. Depending on the type of error these may be hundreds of lines apart. If you can not find the problem, add a closing),] or } at the end of the range and run through cc2. It will report the unbalanced condition with more precision.

We find that the "-p" flag is handy for problems related to the expansion of macro text. It will also help find problems caused by two lines of code separated by many lines of comments or preprocessor statements.

2.1.8. Notes.

Avoid circular definitions such as:-

```
     #define qwerty qwerty
```

They will cause ccl to crash, possibly without an error message. The "-p" option will probably help you find the cause.

The defined constant "_C86_BIG" is useful for conditionally compiling big and small model source code. Specifying '-b' automatically defines "_C86_BIG" so you do not have to use #define to define it. Examples of it's use are shown throughout the run-time source code archives.

2.2. cc2, parser

2.2.1. Function.

Reads the lexemes output by cc1, parses the program and outputs files of initialized data, a symbol table and parse trees.

2.2.2. Usage.

```
cc2 [-su] filename
```

2.2.3. Flags.

NOTE: All flags should go in cc1 for simplicity. The following flag effects this pass of the compiler:

-u Treat all occurrences of the reserved word "char" as an occurrence of "unsigned char". This is provided for compatibility with some 8080 C Compilers. It also sets the "-s" flag.

-s Process string literals as an array of "unsigned char" instead of an array of "char".

Both the above switches should be avoided unless you are porting code from other systems.

2.2.4. Error messages

Most of these should be obvious. The most difficult is "syntax error", which means you added or omitted a required reserved word or operator. In this case, the parser will report an error when it sees a word that could not legally follow the valid phrase already processed. We intend to provide better diagnostics and a listing of error messages and corrective actions in the near future.

If you cannot see the cause of the error, re-run cc1 with the -p option. This frequently makes the cause obvious.

2.2.5. Notes.

In 8080 compilers, character variables are frequently used to minimize code size. For the 8086 they should be changed to int, as character variables lead to awkward code sequences. Of course, this does not apply to character strings, or places where character variables are natural.

We also recommend the use of "unsigned char" in preference to "char". The optimizer can produce much better code for the unsigned variety on the 8086.

2.3. cc3, code generator.

2.3.1. Function.

Inputs a file of parse trees, and generates intermediate object code for cc4.

2.3.2. Usage.

```
cc3 [-nt] filename
```

2.3.3. Flags.

NOTE: All flags should go in ccl for simplicity. The following flag effects this pass of the compiler:

-n Generate code to use the 8087 Numeric data processor chip for all floating point operations. Otherwise code will be produced for the floating point package.

-t Generate calls for program tracing. This currently generates a call to $entry before any other code for each function. The $entry function may be changed as desired to assist in program debugging. See the description of $entry in the library.

2.3.4. Error messages.

Any error messages output by this pass are fatal. They are there in case we forgot something. If some invalid C code does get past cc2 it is reported in this pass. We have added a line number to errors reported in this pass, but we are unable to determine the filename. This should help in locating your problem.

2.4. cc4, optimizer.

2.4.1. Function.

Inputs a file of basic code information, performs optimization and outputs object code or assembly source code.

2.4.2. Usage.

```
cc4 [-a] filename
```

2.4.3. Flags.

NOTE: All flags should go in ccl for simplicity. The following flag effects this pass of the compiler:

-a Generate assembly source code instead of a ".obj" output file.

2.4.4. Error messages.

Any error messages output by this pass are fatal. Most are internal control messages. The only message caused by your code is the "name case conflict" message. Since the masm assembler converts all names to upper case, we do too, and we report if two names map into the same name. We do this test here, since it only applies to global names, and this should reduce the changes to your source code if this condition should arise.

If you had the infamous 'ALLOC' message come out of this pass of the compiler it means that the optimizing pass ran out of data space. It is usually an indication that you have either one gigantic function or switch statement. The "work-around" is to split up your source file, large function, or statement and compile the separate parts.

2.5. Arch, source librarian.

2.5.1. Function.

This program allows you to maintain a single library file containing a number of individual source files. You avoid cluttering up a disk directory with a number of tiny source files. It is provided to enable you to maintain the system source library archive file.

2.5.2. Usage.

```
arch [-dmprtux] libname filenames....
```

2.5.3. Flags.

Only one of the following flags may be used for a single execution of the program:-

-d Delete the filename(s) from the library.

-m Just like '-t' flag but just the filenames, one per line.

-p Just like '-x' flag except that the file doesn't get written to disk and is displayed at the console.

-r Read files from stdin, one per line.

-t Table (print) the content of the library on stdout. Only the library name is allowed. Prints the file header lines, which are described below.

-u Update the library by adding or replacing the filenames to the library. Creates a new copy of libname and renames the old copy to ".bak". Will create a new library if it does not exist.

-x Extract a copy of the named files from the archive and place in files with the same names.

2.5.4. Notes.

The default extension for the library is ".arc".

The default extension for the filenames is ".c". Note that the extension is considered part of the filename within the library, but a drive specification is not.

The archive file is a standard ascii file, and may be printed without any editing other than tab expansion, except if you have ^Z's (End-of-file) in your C files.

The following is a handy way to extract all of the files in an archive using the flags cleverly:

```
     arch -m ARCHNAME | arch -rx ARCHNAME
```

ARCHNAME is the name of the archive file.

Within the library, each module consists of a header line followed by a file body. The header line contains:-

- The letters "-ARCHIVE-" or "+ARCHIVE+"
- The name of the file
- The number of characters in the file body.
- The date the file was last changed.
- The time the file was last changed.

2.6. Marion, object librarian.

2.6.1. Function.

Maintains a Micro-Soft format library of object modules. This avoids cluttering up the disk with a large number of file names, and typing all those names on the link command line.

2.6.2. Usage.

```
marion [-bdelmux] libname filename...
```

2.6.3. Flags.

You may use the "-b", "-e" and one other flag from the following list for each execution of the program.

The flags are:-

-b Create a backup library before doing anything else. If the flag is followed by a letter, the new version of the library is created on the drive designated by the letter.

-d Delete the files from the library.

-e Suppress checksum error messages for record types 80, f0 and f1. These messages can be ignored because the Microsoft equivalent of marion does not calculate correct checksums for these record types. When marion updates a module, all checksums in that module and all following modules are correctly calculated, and written to the file.

-l List the names of modules, module sizes and defined global symbols in the library.

-m List the names of modules in the library in their order of occurrence in the library. This option is useful for constructing batch files.

-u Update the library by adding/replacing modules in the library. A new library will be created if one does not exist. Module names are the input filenames without any extension.

-x Extract copies of the named modules from the library.

2.6.4. Notes.

The backup flag and the error flag may be used in conjunction with one other flag. Since the library update is done OVER the input library, we strongly recommend that you have a safe copy of your input library, or use the backup switch. If this program should fail or be aborted, the library is likely to be UNUSABLE.

2.7. Usq, File unsqueezer.

2.7.1. Function.

Transforms a squeezed file into the original unsqueezed version.

2.7.2. Usage.

```
usq filename
```

2.7.3. Flags.

None.

2.7.4. Notes.

The filename may include a drive designator and/or wildcards. The input is a squeezed file, the output is (we hope) a copy of the original file, before it was squeezed. The output file will be written to your default drive and directory.

To unsqueeze all the compiler files use:-

```
     usq X:cc?.eqe
```

This program will produce an error message if the output file does not checksum correctly, and has been modified to report errors if the output file will not fix on the disk. However, we would advise that you check that there is some free disk space after performing the unsqueeze, just in case.

This is a copy of a public domain program by Dick Greenlaw. The source of sq.c and usq.c is available on our user group bulletin board.

NOTES:

3. LIBRARY FUNCTIONS

3.1. INTRODUCTION

The following pages describe the functions provided in the library. There are equivalents of most of the commonly used UNIX functions, plus various functions which let you have full control of your operating enviroment. Note that some of the functions are only supported under DOS 2.00 and later.

Most of the operating system and hardware dependent functions are by definition non portable, however most operating environments offer similar capabilities. With some care, you can write code that may be ported to other systems with few problems.

3.1.1. Source libraries.

The following source libraries are distributed in squeezed, archived form on your distribution disk:-

*	base.arc	Basic support code for a C program.
*	dosall.arc	DOS 1.1 level I/O library.
*	dos2.arc	DOS 2.0 level I/O library.
*	mathbase.arc	Basic math support.
*	mathsft.arc	Software 8087 support code.
*	math87.arc	Hardware 8087 support code.
*	zdspc.arc	Z-100 PC specific routines (non-portable)

The distributed libraries contain the complete content of one of the archives from each of the above four groups (this excludes the non-portable zdspc.arc). The DOS and math options are made by the appropriate archive selection, in conjunction with the setting of the big model and 8087 compiler switches.

The Z-100 PC specific routines are all in "zdspc.arc". They are in a library that is separate from the standard C86 libraries. THESE FUNCTIONS ARE HIGHLY NON-PORTABLE. They should not be used if direct portability is a major concern. Also, because of the way some linkers work you will need to include the Z-100 PC library before the standard libraries. An example is as follows:-

```
link program+object,,/map,zdspcs+c86s2s
```

3.1.2. Recompiling library functions.

All the functions in one library must be compiled with the same settings of the big model switch (-b) and the numeric data processor switch (-n), and these are the only switches you should use for library functions. Note that the big model switch for assembly code is in the file "model.h".

In some cases, an archive will contain both C and assembly language versions of the same function. In this case please inspect the assembly code for any usage restrictions (eg small model only). Usually the assembly version is the one to use.

You should try to update an existing library with those functions that need re-compilation, rather than re-compiling the whole lot.

3.1.3. Understanding the library descriptions.

Each function, or related group of functions, is described on a separate page. The first entry on the page is the name of the function and a brief description of its purpose. The remaining information is presented under a set of standard headings. The headings are:-

* Synopsis. A definition of the calling sequence for the function, the order and type of the arguments of the function, and the type of value it returns (if any).

  This information is presented from the perspective of a person about to write the function being described, so that you may know what type of data to provide in a call to the function.

  For example, if an argument is described as being a pointer to int (int *), then you should use one of the following forms for that argument in a call to the function:-

  ```
  & integer_variable
  name_of_an_array_of_int
  pointer_to_int_variable (suitably initialised).
  ```

  Faulty call arguments are one of the most common problems we encounter.

* Function. What the library function does.

* Returns. What the return values are and what they mean.

* Notes. Information that did not fit anywhere else, and any information that might help you in your usage of the function.

* Example. Some samples of the use of the function.

* DOS. Special notes about interactions with your operating system.

* Operating System. Which operating systems the functions works with.

* See also. Names of other functions that provide supporting or equivalent services.

* Use with. Functions that are tightly related to this function. Usually indicates that other functions which cannot be used in conjunction with this function are also included in the library.

3.2. _default, Define default conditions.

3.2.1. Function

This file is used to establish run time default values for a C program. We may add entries to this file from time to time, so look at the source code for more up to date information.

When running a program that was compiled with the big model the default is only about 90K+ for the data segment. We do not access all of memory so that there is enough memory for the system function (load and execute a progam) to run. You should change the default value in _MAXFMEM if you need more data space than 96K in the big model.

You may change these values to suit your needs, but if you do so include the complete source of _default.c in your program. You can also just link in the recompiled object module.

Values currently defined are:-

_MAXFMEM The maximum number of paragraphs of stack+heap space that the program will use. This will allow you to control the size of programs that use the terminate and stay resident calls. Unused memory may be used for other purposes. If more memory is needed in the big model this default should be changed. This is the value you are most likely to change in this file.

_STAKMEM The minimum number of bytes that must be left as stack space after a successful call to sbrk(). Remember that there are no other run time stack checks.

_MINRMEM The number of Paragraphs at the top of memory that may not be used by the program. We intended this to be used to preserve command.com, but command.com has grown to about 15K, and that is too much memory to skip in 128Kb and smaller systems. For now our default value is zero paragraphs. This control could be used to create a communication region between programs, by reserving additional space.

_MINFMEM The minimum number of paragraphs of stack+heap space that must be available for the program to run. Checked at start-up time.

_BUFSIZE The buffer size for the I/O system in the DOS2 library. It is different for the small and big models. You can vary this default and make the buffers any size that you want.

3.2.2. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.2.3. See also

$main().

3.3. $entry, Entry to a function.

3.3.1. Synopsis

```
int $entry()
```

3.3.2. Function

This function is called as the first instruction of any function that was compiled under the trace flag ("-t"). The supplied version does a stack overflow check, and reports "NO CORE" on an error.

3.3.3. Returns

Nothing.

3.3.4. Notes

You may modify this function to provide any checking that you need. It could also be modified to call a C function if the code is extensive. Just be careful that the called C function is NOT compiled with the "-t" flag.

3.3.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.4. _exit, Terminate program execution without closing files.

3.4.1. Synopsis

```
int _exit(status)
int status;
```

3.4.2. Function

Performs the standard program termination procedure. No cleanup procedures are performed, file buffers are NOT flushed and files are NOT closed. The supplied status value is returned to the operating system as the termination status of the program.

Normally the function "exit" should be used to terminate a program. This procedure should only be used in cases of extreme emergency.

By convention, non zero termination status values indicate abnormal program termination.

3.4.3. Debugging feature

This routine provides a debugging feature that has proved to be very useful. At the time "_exit" is called, the value provided to _exit and the global character variable "_exittbc" are tested according to the following condition table:-

_exittbc	value	action
zero	any	no stack trace
negative	any	stack trace
positive non-zero	zero	no stack trace
positive non-zero	non-zero	stack trace

The resulting print out is a list of call addresses, showing the addresses of all active calls at the time "_exit" was called. This list may be compared with the linker map output to determine the names of the active functions.

The flag "_exittbc" may be set using code at the beginning of your program, or using your debugger.

3.4.4. Notes

Your operating system may close your files anyway.

3.4.5. DOS before V2.00.

Terminates by issuing interrupt 0x20. The termination status is not available. The version of dos is determined at run time and saved in the variable "_SYSVERS" by the function $main().

3.4.6. DOS 2.0+.

Terminates by issuing interrupt 0x21, sub code 0x4C. The exit status is placed in register AL. This status may be tested in batch files or by the invoking program. Note that only one byte of status is returned.

3.4.7. Example

To turn on the debugging feature.

```
main()
{
  extern char _exittbc;

  _exittbc=0xff;        /* turn it on */
  ...
  ...                   /* run the code of the program, the */
  ...                   /* trace will be printed on most crashes */
  ...
  return 0;             /* return all ok (trace done anyway) */
}
```

3.4.8. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.4.9. See also

exit

3.5. _main, Initialize for program execution.

3.5.1. Synopsis

```
int _main()
```

3.5.2. Function

This function initializes memory for the execution of a C program. It performs the following actions:-

* Builds the argc and argv data for the function "main".
* Detects any re-direction of stdin and/or stdout. (Not DOS 2.0)
* Opens stdin, stdout and stderr in ASCII mode.
* Executes the program by performing the statement:-

```
          exit(main(argc,argv));
```

3.5.3. Returns

Never returns.

3.5.4. Error messages.

The following error messages may occur. The program will abort.

"TOO MANY ARGS"	More than 20 parameters occurred on the command line.
"REDIRECTION ERROR"	One of stdin, stdout or stderr could not be opened.

3.5.5. Notes

Since the name of the program being executed is not available, the first argument provided to "main" is always the lower case letter "c". Therefore "main" will always have at least one argument.

The length of the command line is limited to about 128 characters by the operating system, and generally no indication is provided that the command line is too long. Be careful.

3.5.6. DOS

If the program is running under DOS 2.0+, stdin, stdout and stderr will use the default files and redirection provided by DOS. Under earlier versions of DOS, this function creates stdin, stdout and stderr and processes the redirection information.

3.5.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.6. abort, Abort execution of a program with a message.

3.6.1. Synopsis

```
int abort(format,args...)
char *format;
int args...;
```

3.6.2. Function

Prints a newline and the message "ABORT:- " to stderr. It then prints the abort message to stderr. Finally it prints another newline and calls exit with a hex value of "7FFF".

This is a convenient way to terminate a program when unexpected errors occur.

3.6.3. Notes

See fprintf for details of available format control codes.

This function is provided in most UNIX systems, but it does not take any arguments. There it causes a core dump and program termination, and not the output of a message.

3.6.4. Example

To abort a program if a required file is not available:-

```
#include "stdio.h"

main()
{
  FILE *fopen();
  FILE *fd;

  fd=fopen("filename.dat","r");
  if(fd==NULL)abort("could not open filename.dat");
  ...
}
```

3.6.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.6.6. See also

exit, printf, fprintf, sprintf

3.7. abstoptr, Absolute memory address to pointer.

3.7.1. Synopsis

```
char *abstoptr(address)
long address;
```

3.7.2. Function

Convert a 20 bit absolute memory address to the standard big pointer format. The offset part of the pointer returned by this function will always be in the range 0 through 15 (decimal). Thus the resultant pointer provides access to the next 64Kb of memory.

3.7.3. Return

Returns a big model pointer for the 20 bit absolute memory address.

3.7.4. Notes

This function is not usable in the small memory model.

If you are doing big model pointer arithmetic you will need this function along with the ptrtoabs() function. Without these functions big model pointer arithmetic assumes that the data segments are the same for the pointers. Since this function is very non-portable you should not use it in the big model if it is at all possible. We may change the way pointer arithmetic is done in future versions.

Pointers returned by this function should not be decremented, as you will wrap to the end of the segment. If you need to decrement such a pointer see ptrtoabs.

This function was written mainly for use in malloc and free.

3.7.5. Example

To obtain the absolute memory address of a buffer.

```
          char buffer[23];
          extern long abstoptr();

          printf("Address of buffer is %D\n",abstoptr(buffer));
```

3.7.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.7.7. See also

ptrtoabs

3.8. alloc, Allocate a storage region on the heap.

3.8.1. Synopsis

```
char *alloc(size)
unsigned int size;
```

3.8.2. Function

Allocates a region "size" bytes in length in the heap and initializes it to zeros. If there is not enough space on the heap to allocate a region of the required size, it prints the message "ALLOC" to stderr and calls "_exit".

3.8.3. Returns

The address of the first byte of the allocated region.

3.8.4. Notes

Using the big memory model, blocks of up to 65516 (0xFFE8) bytes may be requested. In the bigmodel, the default amount of memory available is about 96K for the heap and the stack. This can be changed (either increased to access all of memory on your machine or decreased to leave more unused memory) by editing the file _default.c. We have found that most users of C86 can live with about 96K of heap and stack space in the bigmodel.

Total memory available using the small memory model is about 64000 bytes.

Obtains memory from the free list (maintained by "free") if possible. Otherwise obtains memory using the function sbrk. See the description of "free" for more information.

The function "coreleft" may be used to check that memory is available before calling "alloc".

WARNING: In the big model this function must be declared (see the example below). In general, you should get used to the idea of declaring all functions which do not return an integer.

3.8.5. Example

To allocate space for a 1000 byte array:-

```
    char *alloc();          /* define alloc to return a pointer */

    char *array;
    array=alloc(1000);
```

3.8.6. Example

To obtain a buffer dynamically for string storage

```
{
  extern char *alloc();  /* important in the big model!! */
  extern char *fgets();  /* important in the big model!! */
  extern int fputs() , free();
  char *buffer;
  extern unsigned int coreleft();

      if(coreleft() < 1000)abort("you need to buy more core");

      buffer = alloc(255); /* buffer points to 255 byte cells */

      fputs("Enter a line of data > ",stdout);
      fgets(buffer,255,stdin);      /* for example */
      fputs(buffer,stdout);
      free(buffer);          /* free makes this memory available */
                             /* on successive memory allocations */
}
```

3.8.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.8.8. See also

malloc, calloc, realloc, sbrk, free, coreleft

3.9. atof, Convert ASCII to floating point

3.9.1. Synopsis

```
double atof(string)
char *string;
```

3.9.2. Function

Convert a string containing a 'scientific notation' floating point number to a double precision floating point number.

The string can contain optional leading whitespace, an integer part, a fraction part, and an exponent part. The integer part consists of an optional sign followed by zero or more decimal digits. The fraction part is a decimal point followed by zero or more decimal digits. The exponent part consists of an 'E' or 'e' followed by an optional sign followed by a sequence of decimal digits. There must be an integer part or a fraction part at least. The exponent part is optional.

3.9.3. Returns

The converted number.

3.9.4. Notes

This function must be declared, since it returns a double. Use:-

```
     extern double atof();
```

The largest number that may be entered is about "+1E+300", and it can have about 15 significant digits.

Preceeding plus signs in the number are not allowed. They will return zero in the software floating point run-time code.

3.9.5. Example

To convert an input string to double:-

```
     {
          char buffer[132];
          double dnum;
          extern double atof();

          fgets(buffer,sizeof buffer,stdin);
          dnum=atof(buffer);
     }
```

3.9.6. Example

```
{
 extern double atof();  /* atof converts string to float */
 double result;
 char *string;

       string = "-1.56678899";
       result = atof(string);
       printf("\nATOF\n%s = %g\n",string,result);

       /* up to 15 significant digits allowed */

       string="3.54009e10";
       result = atof(string);
       printf("%s = %g\n",string,result);

}
```

3.9.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.9.8. See also

ftoa, sscanf

3.10. atoi, Convert ASCII to integer (long).

3.10.1. Synopsis

```
long atoi(string)
char *string;
```

3.10.2. Function

Convert a string containing the ASCII representation of a number to an int or long.

The string can contain optional leading whitespace, an optional sign and a series of decimal digits.

3.10.3. Returns

The converted number as a long. If the number is in the range +32767 to -32768 the function may be used as though it returned an integer.

3.10.4. Notes

This function does not test for errors during the conversion process. Thus if the input number is too large, or any illegal character is encountered in the input string, the function will silently return an incorrect result.

3.10.5. Example

```
{
  extern long atoi(); /* atoi converts an ASCII string to long */
  long lresult;
  char str[255];

     strcpy(str,"32767");
     lresult = atoi(str);
     printf("\nATOI\n%s = %D\n",str,lresult);

     /* the ASCII string can have a sign */

     lresult = atoi("-45000");
     printf("\nATOI\n%s = %ld\n","-45000",lresult);
}
```

3.10.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.11. basicget, Get a "record" written by a basic program.

3.11.1. Synopsis

```
int basicget(stream,buff,bufflen,fieldptr,fieldcnt)
FILE *stream;                  /* where to read data */
unsigned char buff;            /* where to put it */
int bufflen;                   /* how much to put */
unsigned char *fieldptr[];     /* field pointers */
int fieldcnt;                  /* max number of fields */
```

3.11.2. Function

Reads a line of up to "bufflen" characters from file "stream" into the buffer "buff". Then constructs an array of up to "fieldcnt" pointers in "fieldptr".

If the first character of a field is a quotation mark, the field terminates at the next quotation mark, otherwise the field terminates at the next comma. In either case, the field termination characters are stripped from the field.

3.11.3. Returns

- Minus one at end of file.
- Minus two if the input line was too long to fit in "buff".
- Zero if there were more than "fieldcnt" fields.
- Otherwise the number of fields in the record.

3.11.4. Example

```
#include "stdio.h"

#define BSIZE 128
#define MAXFIELD 10

main()
{
  char bf[BSIZE+2];
  char *fp[MAXFIELD];
  int j;

  j=basicget(stdin,bf,BSIZE,fp,MAXFIELD);
  printf("number of fields was %d\n",j);
  for(j=0;j<MAXFIELD;++j){
    printf("Field %d is %s\n",j,fp[j]);
  }
}
```

If you enter the following input:-

```
This,is,"a field with a , embedded",in,it
```

You should get back:-

```
number of fields was 5
Field 0 is This
Field 1 is is
Field 2 is a field with a , embedded
Field 3 is in
Field 4 is it
```

3.11.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.12. bdos, Execute a basic DOS function.

3.12.1. Synopsis

```
int bdos(fcode,dx)          /* SMALL MODEL */
int fcode;                  /* the function code for your O/S */
unsigned dx;                /* an optional argument */

int bdos(fcode,dx_ds)       /* BIG MODEL */
int fcode;                  /* the function code for your O/S */
unsigned long dx_ds;        /* an optional argument */
```

3.12.2. Function

This function lets you execute most basic operating system defined functions. The value "fcode" specifies the requested action. The value "dx" is optional, and will be placed in register dx before calling your operating system.

3.12.3. Returns

The value returned by your operating system in register ax.

3.12.4. Notes

Operating functions that need input in registers other than dx or return values in registers other than ax may be called using the function "sysint", or "sysint21".

If you are using the big model, the value supplied is assumed to be a pointer. The first word of the value is placed in dx, and the second word in ds. This produces a correct call for functions that need an fcb address. If you don't need the ds value, you may pass an int or omit the value altogether.

DOS

The value "fcode" is placed in register ah.

3.12.5. Example

To check the console to see if the user wants to abort the program:-

Under DOS use:-

```
    bdos(11);
```

3.12.6. Example

```
{
  extern int bdos();
  char disk, status,a_character;

  a_character=bdos(1)&0xff;  /* get a character from keyboard */

  bdos(2,'A');     /* display the letter 'A' on the crt */

  a_character=bdos(3)&0xff;  /* get a character from comm line */

  bdos(4,'A');     /* writes the letter 'A' to the comm line */

  bdos(9,"Print this dollar terminated string on the crt$");

                             /* to set default disk: */
  disk = 'b';
  bdos(14,disk - 'a');          /* will select disk b: */

                             /* to get console status: */
  status = bdos(11)&0xff;         /* if status, char is ready */

}
```

3.12.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.12.8. See also

makefcb, sysint, sysint21

3.13. calloc, Allocate a block of memory.

3.13.1. Synopsis

```
char *calloc(nelem,elsize)
unsigned nelem;                     /* number of elements */
unsigned elsize;                    /* size of each element */
```

3.13.2. Function

Obtains a region nelem*elsize bytes in length from the heap, sets the area to zero and returns its address. If no such area is available, returns zero.

3.13.3. Notes

Using the big memory model, blocks of up to 65516 (0xFFE8) bytes may be requested. In the bigmodel, the default amount of memory available is about 96K for the heap and the stack. This can be changed (either increased to access all of memory on your machine or decreased to leave more unused memory) by editing the file _default.c. We have found that most users of C86 can live with about 96K of heap and stack space in the bigmodel.

WARNING: You need to declare this function in the big model!

3.13.4. Example

```
#define NUMBER 255
#define SIZE 1
{
  extern char *calloc(); /* needed in big model!! */
  extern char *fgets();  /* needed in big model!! */
  extern int fputs();
  extern int free();
  char *buffer;

     /* to allocate NUMBER*SIZE bytes : */
     buffer = calloc(NUMBER,SIZE) ;
     if(!buffer)abort("Ug, TOO BIG\n");
     fputs("Enter data followed by CTRL-Z >",stdout);
     /* buffer can now be used: */
     while(fgets(buffer,NUMBER,stdin))
          fputs(buffer,stdout);
     /* the above will echo console input until EOF */
     free(buffer);
     /* free returns the area of store to the heap for
        later memory allocation calls */
}
```

3.13.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.13.6. See also

alloc, malloc, realloc

3.14. ceil, Ceiling function.

3.14.1. Synopsis

```
double ceil(arg)
double arg;
```

3.14.2. Returns

The a double precision number which contains the smallest integer greater or equal to arg.

3.14.3. Notes

WARNING: This function needs to be declared to work properly.

3.14.4. Example

```
{
  extern double ceil();          /* returns double */
  double dval, dresult;

           /* ceil returns the smalles integer >= argument */

     dval = 178.3456;
     dresult = ceil(dval);
                                          /* dresult contains 179. */
     printf("\nCEIL\nceil %e = %e\n",dval,dresult);

     dval = -34.2333e3;
     dresult = ceil(dval);
                                          /* dresult contains -34233 */
     printf("\nceil %e = %e\n",dval,dresult);

     dval = 12.00 ;
     dresult = ceil(dval);
                                           /* dresult contains 12.00 */
     printf("\nceil %e = %e\n",dval,dresult);
}
```

3.14.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.14.6. See also

floor

3.15. chdir, Change to a new working directory.

3.15.1. Synopsis

```
int chdir(pathname)
char *pathname;
```

3.15.2. Function

Calls the operating system to set a new working directory. The path name must be reachable from your current directory, and is operating system dependent. You should refer to your operating system documentation for more information.

3.15.3. Returns

EOF if an error is detected, otherwise zero.

3.15.4. Notes

This function is only usable on a system running DOS V2.0+.

If you code path names in strings, don't forget that you need two backslash characters to enter one backslash in the string.

3.15.5. Example

To change to directory \c86 on your default disk.

```
    chdir("\\c86");
```

To change to directory "xxx\yyy\zzz" on drive A:.

```
    chdir("A:xxx\\yyy\\zzz");
```

3.15.6. Operating System

DOS 3.0, DOS 2.0+

3.15.7. Use with

mkdir, rmdir, all file i/o logic

3.16. chmod, Change the mode of a file.

3.16.1. Synopsis

```
int chmod(filename,mode)
char *filename;
int mode;
```

3.16.2. Function

Calls the operating system to set the mode bits for the file to those requested. The available mode bits and their meanings are operating system dependent, and you should refer to your operating system technical reference manual for more information.

3.16.3. Returns

EOF if an error is detected, otherwise zero.

3.16.4. Notes

This function is only usable with DOS V2.0 and later.

Using undocumented bits in the requested mode is not recommended.

Legal file modes:

01H	Read Only
02H	Hidden
04H	System
08H	Volume label
10H	Subdirectory
20H	Archive

3.16.5. Example

```
{
int mode;
#define NORMAL   0x00          /* normal file */
#define READONLY 0x01
#define HIDDEN   0x02
#define SYSTEM   0x04
#define VOLUME   0x08
#define SUBDIR   0x10
#define ARCHIVE  0x20

  mode = NORMAL;        /* set to a normal file */
  mode |= READONLY;     /* set to readonly */
  mode |= HIDDEN;       /* set hidden attribute */
  mode |= SYSTEM;       /* set system attribute */

  chmod("filename.dat",mode);   /* set the file's attributes */
}
```

3.16.6. Operating System

DOS 3.0, DOS 2.0+

3.17. clearerr, Clear a stream error indicator.

3.17.1. Synopsis

```
#include "stdio.h"
int clearerr(stream)
FILE *stream;
```

3.17.2. Returns

Nothing

ACTION

Clears an error indicator maintained for the stream. The error indicator may be read using the function "ferror".

3.17.3. Example

```
{
  extern int clearerr();  /* clears a stream error indicator */
  extern int ferror();
  extern FILE *fopen();
  extern int fclose();
  FILE *stream;

     stream = fopen("a:foo.bar","a");
     if(stream==NULL)abort("can't open a:foo.bar\n");
     ...
     ...                         /* more processing of stream */
     ...
     if(ferror(stream)){
          printf("\nError associated with foo.bar\n");
          clearerr(stream);  /* clears the error for stream */
          ...  /* take corrective action */
          }
     ...
     ...
     ...
}
```

3.17.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.17.5. See also

ferror, fopen, fclose, fread

3.18. close, Close a file.

3.18.1. Synopsis

```
int close(fd)
int fd;
```

3.18.2. Function

Flushes any outstanding output data to the file, and then closes the file designated by file descriptor fd. Returns the file control block and buffers to the heap, and makes fd available for the next call to open or creat.

3.18.3. Returns

.zero if successful
.minus one if any error was detected

3.18.4. Notes

The use of this routine is discouraged. Use the alternative routine fclose. In this release, even if an error is reported, the file buffers and control blocks are released, so the file is really 'closed'.

3.18.5. Example

```
{
  extern int open();
  extern int close();
  int fptr, success;

          fptr = open("CON:",AREAD);
          if(fptr<0)abort("can't open CON:\n");

          /* console gets read here */

          success = close(fptr);

          /* if success==0, close was successful,
             if success==-1 error occured */
}
```

3.18.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.18.7. Use with

open, creat, read, write

3.18.8. See also

fopen, fclose

3.19. Z-100 PC Communications Functions: (Z-100 PC ONLY!)

com_flsh, com_getc, com_putc, com_rdy, com_rst, com_stat

3.19.1. Synopsis

```
int com_flsh(channel)
int channel;

int com_getc(channel)
int channel;

int com_putc(channel,ch)
int channel;
char ch;

int com_rdy(channel)
int channel;              /*  0=COM1 1=COM2 */

int com_rst(channel,baud,parity,stop,length)
int channel,baud,parity,stop,length;

unsigned int com_stat(channel)
int channel;              /* 0 = COM1 1 = COM2 */
```

3.19.2. Function

com_flsh attempts to flush the channel associated with the function and returns the modem status word also.

com_getc will wait until a character is ready at the channel and then return it. It does no conversion on the character.

com_putc outputs a character to the channel specified and returns the status word (see notes) after the attemt to send it.

com_rdy returns a 1 if a character is waiting or a zero if not.

com_stat gets the communication channel status and returns the modem status word. See the notes section for a description of what **com_stat** returns.

com_rst resets the baud rate, parity, stop bit and length of the channel specified and returns the modem status word. See the notes for a description of the format for the parameters.

3.19.3. Notes

com_stat modem status words:

bit position	mask (Hex)	meaning
0	0x0001	delta clear to send
1	0x0002	delta data set ready
2	0x0004	trail edge ring detector
3	0x0008	delta rec. line sgnl dtct
4	0x0010	clear to send
5	0x0020	data set ready
6	0x0040	ring indicator
7	0x0080	rec. line signal detect
8	0x0100	data ready
9	0x0200	overrun error
A	0x0400	parity error
B	0x0800	framing error
C	0x1000	break detect
D	0x2000	xmitter holding reg empty
E	0x4000	xmitter shift reg empty
F	0x8000	timeout occurred

Notes on com_rst parameters:

BAUD RATES:

value	baud rate (bits / second)
0	110
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	9600

PARITY:

value	Parity setting
0	NONE
1	ODD
2	NONE
3	EVEN

STOP BITS:

value	Number of stop bits
0	1
1	2

WORD LENGTH:

value	Word length (bits)
0	5
1	6
2	7
3	8

All of the above information is available in your friendly neighborhood technical reference manuals or programmer's guide.

3.19.4. Examples

* Here is a small piece of code to illustrate the use of com_rdy, com_getc, key_scan, and com_rst.

```
com_rst(0,7,0,1,3);     /* 9600, no parity, 1 stop, 8 data */

for(;;)                 /* display characters when they come */
  {
  while(com_rdy(0) == 0)   /* wait for a character */
      ;
  ch = com_getc(0);        /* read it */
  putchar(ch);             /* write it to the screen */
  if(key_scan() != EOF)    /* a key was pressed */
      break;
  }
```

* In order to see if a character is ready:

```
ready = com_stat(0) & 0x0100;
```

* In order to see if clear to send signal is set:

```
clear = com_stat(0) & 0x0010;
```

* To set COM1 to 9600 baud, no parity, 1 stop bit, 8 data bits use:

```
com_rst(0,7,0,1,3);
```

* To set COM2 to 300 baud, odd parity, 2 stop bits, 7 data bits use:

```
com_rst(1,2,1,1,2);
```

3.19.5. Operating System

PC DOS 3.0, PC DOS 2.0+, PC DOS 1.1+

3.20. coreleft, Get size of unused stack.

3.20.1. Synopsis

```
unsigned int coreleft();
```

3.20.2. Function

Returns the number of bytes unused on the stack. This is the number of bytes between s-break and the content of register 'sp'.

3.20.3. Notes

The number returned is frequently negative, so that the function should be declared to return an unsigned integer if correct sign operations are to be performed.

This function makes no allowances in the returned number. You should subtract a safety margin from the returned value for files to be opened and local storage in functions. After this the value may be used to allocate memory.

Free areas in the heap are not considered. Thus after performing an alloc it is possible that the coreleft value has not been altered.

A major use of this function is with algorithms that can use all available core, such as text editors for text storage. Thus the size of buffers can be set dynamically at run time.

In the big model this function returns 64K until there is less than 64K of heap space available. When there is less than 64K of heap space available it will return the amount of memory available.

3.20.4. Example

```
{
  extern unsigned int coreleft(); /* returns available memory */
  unsigned int core;

        core = coreleft();       /* memory available */
        printf("\nCoreleft: %x Hexadecimal\n",core);
        printf("Coreleft: %u Unsigned decimal\n",core);
}
```

3.20.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.20.6. See also

alloc, malloc, calloc, realloc, free

3.21. creat, Create a new empty file.

3.21.1. Synopsis

```
int creat(filename,mode)
char *filename;
unsigned mode;
```

3.21.2. Function

Filename must be valid for your operating system or one of the special names defined below.

If a file with the name "filename" exists it is deleted. A new file is then created. Valid open modes are defined in "stdio.h", and have the values:-

AREAD	0	open for ASCII read
AWRITE	1	open for ASCII write
AUPDATE	2	open for ASCII update
BREAD	4	open for binary read
BWRITE	5	open for binary write
BUPDATE	6	open for binary update

Please use the names in your code, because the values are subject to change without notice.

3.21.3. Returns

.A negative number if any error was detected
.A positive number (a file descriptor) otherwise

3.21.4. Notes

The use of this routine is discouraged. Use fopen.

The open mode is not recorded with the file, therefore when the file is next used it may be opened in any one of the six modes.

If a file is created in ASCII mode:-

* Carriage return/linefeed pairs in the file will be converted to newlines ('\n') on input.
* Newlines will be converted to carriage return/linefeed pairs on output.
* Control-z in the file will be returned as end of file on input.

If a file is created in binary mode only physical end of file (as returned by your operating system) can be returned by the i/o system. Logical EOF conventions must be implemented by the user programmer.

3.21.5. DOS

The following names are processed specially by the i/o system:-

. The console	"CON:"
. The printer	"PRN:"
. The com device	"AUX:"

If the above names are used (including the colon), data is processed in unbuffered mode (except for console input).

If the console is opened in ASCII input mode, input data must be terminated by a carriage return (just like UNIX). If it is opened in binary mode, input is character by character (raw mode).

3.21.6. Example

```
{
  extern int creat();    /* opens a new file */
  extern int close();
  int fd;

     fd = creat("foo.bar",AUPDATE);
     if (fd<0) abort("\ncreat error occured\n");
     printf("\nfoo.bar created\n");  /* use file here */
     close(fd);
}
```

3.21.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.21.8. Use with

open, read, write, close, lseek, ltell

3.21.9. See also

fopen, fclose

3.22. Z-100 PC video display routines: (Z-100 PC ONLY!)

crt_cls, crt_gmod, crt_home, crt_line, crt_mode, crt_rdot, crt_roll, crt_scrp, crt_wdot

3.22.1. Synopsis

```
crt_cls()

crt_home()

crt_gmod()

int crt_line(x1,y1,x2,y2,color)
unsigned int x1,y1;            /* from co-ordinate */
unsigned int x2,y2;            /* to co-ordinate */
int color;                     /* color of line */

int crt_mode(mode)
int mode;                     /* desired mode code */

int crt_rdot(row,column)
int row;
int column;

int crt_roll(top,bottom,left,right,n)
int top;
int bottom;
int left;
int right;
int n;

int crt_srcp(row,column,page)
int row;
int column;
int page;

int crt_wdot(row,column,color)
int row;
int column;
int color;
```

3.22.2. Function

crt_cls clears the screen on page 0 on the Z-100 PC and brings the cursor home.

crt_home positions the cursor to the upper left hand corner of the Z-100 PC monitor on page 0.

crt_gmod returns the mode of the crt for the Z-100 PC (see notes).

crt_line draw a line on the monitor in graphics mode (see crt_mode). It takes two coordinates and a color as parameters.

crt_mode sets the mode of the monitor. For valid modes see the notes description.

crt_rdot returns the 'color' of the specified point on the monitor (Only in graphics mode).

crt_roll scrolls a section of what is on the monitor. The section from the top and left corner and the bottom and right corner is scrolled either up or down n lines. You might experiment with this function in a small test case first.

crt_srcp sets the cursor to the specified row, column and page on the monitor. Page is usually set to zero. If you want to use the other pages be warned that some other functions in this group assume the use of page zero.

crt_wdot writes a dot of the specified 'color' at the given row and column (Only in graphics mode).

3.22.3. Notes

Valid modes for the Z-100 PC monitor:

mode	meaning
0	40 x 25 BW (default)
1	40 x 25 COLOR
2	80 x 25 BW
3	80 x 25 COLOR
4	320 x 200 COLOR
5	320 x 200 BW
6	640 x 200 BW
7	80 x 25 BW CARD

For the functions crt_line, crt_rdot, and crt_wdot the monitor must be in a graphics mode. The limiting values of all supplied arguments (e.g. coordinates) are a function of the chosen mode. Using out of range values will get you into problems for any of these functions. The default colors for these functions are as follows:

COLOR	PALETTE 0	PALETTE 1
0	bckgrd	bckgrd
1	green	cyan
2	red	magenta
3	brown	white

The default palette is 1 and the default background is black. To change the palette and background see your Z-100 PC Technical Reference manual for more information.

3.22.4. Example

```
* simple graphics example
{
int i;

     crt_mode(4);         /* set 320 X 200 color graphics */
     crt_line(100,150,200,150,2);  /* draw a line */
     crt_line(200,150,200,100,2);  /* another... */
     crt_line(200,100,100,100,2);  /* and another... */
     crt_line(100,100,100,150,2);  /* to make a square!! */

     for (i=0;i<50;i+=3)
       crt_wdot(125+i,150,1);      /* draw dotted line */

     crt_mode(2);         /* reset monitor back to normal */
}
```

3.22.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.23. envfind, search environment for defined name.

3.23.1. Synopsis

```
unsigned char *envfind(name)   /* dos 2.0+ only */
unsigned char *name;
```

3.23.2. Function

Searches the DOS 2.0+ environment for a defined name.

3.23.3. Returns

If the name is found, it returns the translation of that name. Otherwise it returns a NULL pointer.

3.23.4. Notes

The returned string is obtained from malloc, so you must free it when you are finished with it.

The input to envfind() is case sensitive.

3.23.5. Example

```
{
  unsigned char *envfind();
  unsigned char *cp;
  unsigned char s[255];

  while(gets(s,255)) {
    upper(s);
    if (cp=envfind(s)) {
      puts(cp);
      free(cp);
    } else puts("NOT FOUND");
  }
}
```

3.23.6. Operating System

DOS 3.0, DOS 2.0+

3.24. exit_tsr - exit, terminate and stay resident

3.24.1. Synopsis

```
exit_tsr()
```

3.24.2. Function

Exit your program and then make it resident. This will allow it to be executed later by a certain system interrupt defined by the user.

3.24.3. Notes

The function will stay in memory until a re-boot.

Interrupts 0 thru 3F and 80 thru F0 are reserved for DOS, Intel and Basic. You obviously can not use any of these for you terminate and stay resident function.

A very handy tool to read and understand whenever you venture into this area is the tecnical reference manual of your favorite computer. We highly recommend purchasing one of these for your sanity's sake. To get a copy check with your computer dealer.

Since most of the programs that you would like to terminate and stay resident should be small, you probably will want to shrink the size of the default data segment in a C86 program that is going to use this function. The default value _MAXFMEM found in _default.c in base.arc should be changed. See the _default function for more information.

3.24.4. Example

```
------------------ how to set up resident program --------------

#define VECNO 0x50              /* use a free interrupt vector */
#define STACK 3000              /* save some space for stack */

int hello()
{
   bdos(9,"\r\nHello world\r\n$");
}

/*

   use this program to load "hello" into memory so that it can
   be called through interrupt 0x50. hello() could be written
   in C or assembler. Remember that 'main' and 'hello' will both
   be loaded into memory. If these programs are not going to use
   a lot of data space, you may wish to decrease the amount of
   data space allocated to your program. See the documentation
   on _default for more information.
*/
main()
{
extern int hello();

    intrinit(hello,STACK,VECNO)   /* set up vector to point
                                   to your routine */
    exit_tsr();                   /* terminate, stay resident */
}


-------------- how to call resident program ---------------


/*
   use a program such as this to call the resident process
*/

#include <stdio.h>

main()
{
struct regval { int ax,bx,cx,dx,si,di,ds,es; } srv;

  sysint(VECNO,&srv,&srv);
}
```

3.24.5. Operating System

DOS 3.0, DOS 2.0+

3.24.6. See also

exit, _exit, sysint, sysint21

3.25. exit, Terminate program execution.

3.25.1. Synopsis

```
int exit(value)
int value;
```

3.25.2. Function

This function is called to terminate the execution of a program. It flushes any output buffers and then closes any open files. It then calls "_exit" with the supplied value. By convention, non zero values indicate that the program terminated abnormally.

The advantage of this function is that a program may be terminated without returning through all the currently active calling functions.

3.25.3. Returns

Never returns.

3.25.4. Notes

The value is returned to the operating system as the termination status of the program. Currently only DOS 2 uses this information.

3.25.5. Example

```
{
     /* exit terminates program execution */
     /* an example: */
          if(FATAL_ERROR) exit(7);
}
```

3.25.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.25.7. See also

_exit

3.26. exp, Exponential function.

3.26.1. Synopsis

```
double exp(val)
double val;
```

3.26.2. Returns

The exponential function of the argument "val".

3.26.3. Notes

Returns a large value (1e+300) if the result would be too large.

3.26.4. Example

```
main()
{
  extern double exp();  /* exp(val) raises e to the val power */
  double dval, dres;

        dval = 10;
        dres = exp(dval);
        /* dres contains e ^ 10 */
        printf("\ne to the %g = %g\n",dval,dres);

        dval = -15.0;
        dres = exp(dval);
        /* dres contains e ^ (-15.0) */
        printf("\ne to the %g = %g\n",dval,dres);
}
```

3.26.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.26.6. See also

log, log10, pow, sqrt

3.27. fabs, Floating absolute value.

3.27.1. Synopsis

```
double fabs(val)
double val;
```

3.27.2. Returns

The absolute value of val.

3.27.3. Example

```
main()
{
  extern double fabs();
  double value, avalue;

        value = 1.5e25;
        avalue = fabs(value);
        /* in this case avalue contains value */
        printf("\nFABS\nfabs(%g) = %g\n",value,avalue);

        value = -2.3e10;
        avalue = fabs(value);
        /* in this case avalue contains -value */
        printf("\nfabs(%g) = %g\n",value,avalue);
}
```

3.27.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.28. farcall, Call a "far" function

3.28.1. Synopsis

```
struct reg_str{unsigned int ax,bx,cx,dx,si,di,ds,es;};

int farcall(offset,segment,srv,rrv)
int offset;              /* the offset address of the function */
int segment;             /* the segment of the function */
struct reg_str *srv;     /* supplied register values */
struct reg_str *rrv;     /* returned register values */
```

3.28.2. Function

Calls the function at the address and segment supplied after setting the registers to the supplied values. After returning, the returned register values are placed in the structure pointed to by rrv. The value of farcall is the content of the processor status register after execution of the call.

3.28.3. Notes

The returned values may overlay the values supplied with the call.

The called function must preserve the values in registers SS, BP and SP.

Generally this function follows the conventions of sysint.

You may need to modify this code, depending on the use you are making of the function.

3.28.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.28.5. See also

sysint, sysint21, segread

3.29. fclose, Close a stream.

3.29.1. Synopsis

```
#include "stdio.h"
int fclose(stream)
FILE *stream;
```

3.29.2. Function

Flushes any outstanding buffered data and then closes the stream. All buffers are returned to the heap.

3.29.3. Returns

Minus one if an error was detected, otherwise zero (NULL).

3.29.4. Notes

In previous releases the internal file information was preserved if an error was detected during the close processing. This is no longer true.

3.29.5. Example

```
{
  extern int fclose();  /* fclose is used to close a stream  */
  extern FILE *fopen(); /* obtained through a call to fopen() */
  FILE *stream;
  int errstat;

     stream = fopen("tempfile.tmp","r");    /* open tmp file */
     fputs("\nFCLOSE\n",stdout);
     errstat = fclose(stream);         /* returns -1 if error */
}
```

3.29.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.29.7. Use with

fopen, fgets, fprintf, fscanf, putc, getc

3.29.8. See also

close, read, write, open

3.30. feof, Return end of file status.

3.30.1. Synopsis

```
#include "stdio.h"
int feof(stream)
FILE *stream;
```

3.30.2. Function

Test if the stream is at end of file.

3.30.3. Returns

True (non-zero) if the stream is at end of file; otherwise zero.

3.30.4. Notes.

End of file status is a transient thing on character devices.

A disk file will remain at end of file unless you seek to another position in the file.

3.30.5. Operating System

DOS 3.0, DOS 2.0+

3.30.6. See also

fopen, fclose, ferror

3.31. ferror, Return error status of a stream

3.31.1. Synopsis

```
#include "stdio.h"
int ferror(stream)
FILE *stream;
```

3.31.2. Function

Reports the error status associated with the stream. The error indicator is set if an error has ever been detected since the stream was opened. The error indicator may be reset by the function "clearerr".

3.31.3. Returns

Zero if no error has been detected, otherwise a negative error status value.

3.31.4. Example

```
#include "stdio.h"

{
  extern int ferror();    /* returns error status of a stream */
  extern FILE *fopen();
  extern int fclose();
  int errstat;
  FILE *stream;

  stream = fopen("xyz","w");
  if(!stream)abort("can't open xyz\n");
  ...
  ...       /* processing done here */
  ...
  errstat = ferror(stream);    /* look for errors */
  if(errstat)printf("\nerrors processing file 'xyz'\n");
}
```

3.31.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.31.6. See also

clearerr, fopen, fclose, fflush

3.32. fflush, Flush a stream to disk

3.32.1. Synopsis

```
#include "stdio.h"
int fflush(stream)
FILE *stream;
```

3.32.2. Function

Writes all buffered data and file control information for the stream to disk. This provides some security for data files against program crashes.

3.32.3. Returns

Zero if successful, minus one if an error occurred.

3.32.4. Notes

Your operating system may not actually perform the disk write operations if your disk is a Winchester. Be warned.

3.32.5. Example

```
{
  extern int fflush();      /* flushes a stream to disk */
  extern FILE *fopen();
  extern int fclose();
  FILE *fd;
  int i;

  fd = fopen("foo.bar","w");
  if(!fd) return;               /* check if opened */

  /* do processing and write the outout to the file 'fd' */

  fflush(fd);       /* make sure all data is safe on disk */

  /* at this point we can perform operations that could cause the
     program to crash,  such as input from the operator,  without
     risk to data already recorded on disk */

  fclose(fd);
}
```

3.32.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.32.7. Use with

fopen, fread, fwrite, fclose

3.33. fgetc, Get a character from a stream

3.33.1. Synopsis

```
#include "stdio.h"

int fgetc(stream)
FILE *stream;
```

3.33.2. Function

Reads one character from a stream.

3.33.3. Returns

Actually returns an int, whose top byte is set to zero. If an error occurs returns minus one.

3.33.4. Notes

Macros in "stdio.h" convert function calls to getc and getchar into calls to fgetc.

This is the most basic input function for the DOS2 I/O library.

If the result of this function is assigned to a variable of type unsigned char it will not sign extend the top byte during a comparison of an integer (i.e. EOF). This may result in some loop conditionals not working correctly.

3.33.5. Example

```
{
  extern int fgetc(); /* gets a character from an input stream */
  extern FILE *fopen();
  extern int fclose(), fputc();
  int ch;

  /* the following will echo console input a buffer at a time. */

     fputs("\nFGETC\nEnter Data Terminated by CTRL-Z > ",stdout);
     while( (ch=fgetc(stdin)) != EOF) fputc(ch,stdout);
}
```

3.33.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.33.7. See also

fopen, fclose, fputc, printf, scanf

3.34. fgets, Read a string from a stream.

3.34.1. Synopsis

```
char *fgets(buffer,bufleng,stream);
char *buffer;                   /* where to put it */
unsigned int bufleng;           /* how much to read */
FILE *stream;                   /* stream to use */
```

3.34.2. Function

Reads characters from the stream into the buffer until:-

- A newline is read from the input stream
- (bufleng-1) characters have been transferred
- End of file is encountered

In all cases the string will be terminated by EOS.

3.34.3. Returns

- The address of the data buffer
- Zero at end of file or if an error is detected

3.34.4. Notes

If the file was opened in ASCII mode, carriage return and control-z characters will receive special processing.

3.34.5. Example

```
{
  extern char *fgets(); /* reads characters into a string */
  extern int fputs();
  char storage[255];
  int result, bufleng;

        bufleng = 255;
        fputs("\nFGETS\nEnter a line of data\n",stdout);
        result = fgets(storage,bufleng,stdin);
        if(!result) printf("\nEOF or ERROR\n");
        else fputs(storage,stdout);

        /* the newline character will be put onto the string */
}
```

3.34.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.34.7. Use with

fopen, fclose, putc, getc

3.34.8. See also

open, close

3.35. filedir, return a list of matching file names.

3.35.1. Synopsis

```
unsigned char *filedir(filespec,mode)
unsigned char *filespec;
unsigned int mode;
```

3.35.2. Function

Takes a wildcard file specification and file mode a returns a list of matching file names. The file specification may include a drive specification and a path name.

3.35.3. Returns

A list of filenames that do not include the drive specification and path name. Each filename is NULL terminated and the last filename is terminated by a NULL.

3.35.4. Notes

The returned list is obtained by call to malloc so it must be freed when you are finished with it.

Modes are descibed in your DOS technical reference manual. A mode of zero would return the names of all regular or "normal" file names. Check your DOS manual for the DOS Disk Directory section for more details.

Legal File Modes for chmod() and filedir():

Attribute byte:

01H	Read Only
02H	Hidden
04H	System
08H	Volume label
10H	Subdirectory
20H	Archive

3.35.5. Example

```
/* First example of filedir() */

{
  extern char *filedir();
  char *first;
  char *next;
  char filespec[255];
  int mode;

    mode = 0;                    /* regular files */
    strcpy(filespec,"C:*.C");    /* get all *.c in current dir on c: */

    first = filedir(filespec,mode);     /* get list of file names */
    if(first == NULL)
       {
       fprintf(stderr,"Couldn't find any files *.c on c:\n");
       return;
       }

    /* a NULL terminated list of file names */
    for(next = first; *next != NULL;)
       {
       printf("the file name is: %s\n",next);
       next = next + strlen(next) + 1;
       }
    free(first);        /* filedir ALLOCates space for the list */
}

/* Another example with mode examples */

#define NORMAL   0x00           /* normal file */
#define READONLY 0x01
#define HIDDEN   0x02
#define SYSTEM   0x04
#define VOLUME   0x08
#define SUBDIR   0x10
#define ARCHIVE  0x20
{
        /* to get normal files */
        list = filedir("*.*",NORMAL);

        /* to get system files */
        list = filedir("*.*",SYSTEM);

        /* to get the volume id */
        list = filedir("*.*",VOLUME);

        /* to get hidden, readonly files */
        list = filedir("*.*",HIDDEN | READONLY);
}
```

3.35.6. Operating System

DOS 3.0, DOS 2.0+

3.36. fileno, Get file handle.

3.36.1. Synopsis

```
#include "stdio.h"

int fileno(stream)
FILE *stream;
```

3.36.2. Function

Get the file descriptor used for input and/or output from/to this stream. This function lets you use file descriptor I/O on streams.

3.36.3. Returns

The file descriptor associated with the stream.

3.36.4. Notes

This function has changed since the early versions of 2.10 of C86. WARNING: There is no relationship between file descriptors and DOS file handles in the DOS2 library code. Making this assumption will definitly get you into trouble.

3.36.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.36.6. See also

fopen, read, write, close,lseek

3.37. floor, Floor function.

3.37.1. Synopsis

```
double floor(val)
double val;
```

3.37.2. Returns

A double containing the largest integer less than or equal to val.

3.37.3. Example

```
{
  extern double floor();
  double dval, dresult;

        /* floor returns the largest integer <= dval */
        dval = 1.456;
        dresult = floor(dval);          /* dresult contains 1. */
        printf("\nfloor (%g) = %g\n",dval,dresult);

        dval = -3.4;
        dresult = floor(dval);          /* dresult contains -4. */
        printf("\nfloor (%g) = %g\n",dval,dresult);

        dval = 4.0;
        dresult =  floor(dval);         /* dresult contains 4.0 */
        printf("\nfloor (%g) = %g\n",dval,dresult);
}
```

3.37.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.37.5. See also

ceil

3.38. fopen, Open a stream.

3.38.1. Synopsis

```
#include "stdio.h"

FILE *fopen(filename,fomode)
char *filename,*fomode;
```

3.38.2. Function

Open the file "filename" with the mode "fomode". Filename may be any legal file name. The open modes currently supported are:-

"r" The file must exist and is opened for ASCII read.

"r+","rw" The file must exist and is opened for ASCII update.

"w" Any existing file is deleted. A new file is created and is opened for ASCII write.

"w+","wr" Any existing file is deleted, and a new file created and opened in ASCII update mode.

"a" If the file does not exist, create it. Then open the file for writing and position at the end of file.

"a+","ar" If the file does not exist, create it. Then open the file for updating and position at the end of file.

If a "b" is concatenated to the above strings, the file is opened in binary mode, and carriage return/linefeed translation does not occur.

For UNIX v5.0 compatability, it is prefered that you use the modes "a+","r+", and "w+".

When a file is opened in update mode (using modes "r+", "rw", "w+", "wr", "a+", or "ar") both input and output can be done for the given file. WARNING: In future implementations of the library you will not be able to switch from either output to input or from input to output without an intervening fseek() or rewind() call. This way positioning of where you want to write and read from is up to the user in update mode.

3.38.3. Returns

A non zero file stream pointer if the file was successfully opened. Otherwise, zero. It is very important that you always check the return of this function before using the stream pointer.

3.38.4. Notes

This is the standard method for opening files.

3.38.5. DOS

The special filenames "CON:", "PRN:", and "AUX:" may be used provided that they can support the requested famode. In DOS 2.0+ you can open any available device as a file but you do not need to put a colon (':') at the end of the device name.

3.38.6. Example

```
#include "stdio.h"
{
  extern FILE *fopen();
  extern int fclose();
  FILE *ptr;

     /* various modes */
     ptr = fopen("foo.bar","r");  /* ASCII Read Disk File */
     if(ptr) fclose(ptr);

     ptr = fopen("CON:","wb");    /* Direct Console Write */
     fputc(0x07,ptr);
     if(ptr) fclose(ptr);

     ptr = fopen("CON:","rb");     /* Direct Console Input */
     if(fgetc(ptr)==3) printf("\nCTRLC Entered\n");
     if(ptr) fclose(ptr);

     ptr = fopen("a:filename.ext","w"); /* ASCII Write */
     if(ptr) fclose(ptr);

     ptr = fopen("PRN:","w");    /* ASCII Write to PRINTER */
     if(ptr) fclose(ptr);
     /* DOS will not allow opening "PRN:" for read */

     /* ASCII Write Starting at EOF */
     ptr = fopen("a:filename.ext","a");
     if(ptr) fclose(ptr);

     /* ASCII Update Starting at EOF */
     ptr = fopen("a:filename.ext","a+");
     if(ptr) fclose(ptr);

     /* ASCII Read/Write Update */
     ptr = fopen("a:\\bin\\filename.ext","r+");
     if(ptr) fclose(ptr);
}
```

3.38.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.38.8. Use with

fclose, getc, putc, fflush, fclose, printf

3.38.9. See also

open, creat

3.39. fprintf, Print to a stream.

3.39.1. Synopsis

```
#include "stdio.h"

int fprintf(stream,format,args...)
FILE *stream;
char *format;
see below for args;
```

3.39.2. Function

Output data under control of a format string to the file stream.

The output file is defined by the stream pointer. This is the value returned by fopen when the file was opened, or one of the special standard values "stdout" or "stderr". The file should be opened in ASCII mode, unless you need files to run with UNIX.

The format string contains characters that are copied to the output file, and conversion specifications. Each conversion specification causes conversion of one argument and the output of the converted value. You need as many arguments as there are conversion specifications in the format string.

Each conversion specification begins with a percent ("%") character, and ends with a conversion control character. Between these two characters are the following optional control fields:-

A minus sign — This indicates that the output data should be left justified,instead of the default right justification.

A fill char — A zero indicates that the field should be filled with zeros instead of spaces.

A field width — This may be an asterisk ("*") or a number, and if supplied specifies the minimum field width. The asterisk indicates that the next argument is an integer, and it is the width specification.

At least this many characters will be output for the field. More will be output if required. If the data is shorter than the field width, it will be filled with the fill character specified above.

A precision This consists of a period, followed by an asterisk or a decimal number. As above, the asterisk indicates that the precision is the next argument. The meaning (if any) of the precision specification is defined by the conversion character.

A long flag An 'l' or 'L', indicates that the corresponding argument is a long or unsigned long. This code may be used in conjunction with any of the integer conversion codes.

The allowed conversion codes are:-

d The argument is an integer. It is converted to a signed decimal number. As a non standard extension, the precision field will result in a period, precision places to the left. This is handy for money fields.

u The argument is an unsigned integer. It is converted to an unsigned decimal number.

x The argument is an integer. It is output as a hexadecimal number without a leading "0X". The precision field has no meaning for this type of field.

o The argument is an integer. It is output as an octal number without a leading zero.

b The argument is an integer. It is output as a binary number. This is an extension to the K&R specification.

e The argument is a floating point number. It is output in scientific notation, in the form "i.ffffffEeee". The precision field specifies the number of fractional places in the output number, the default being 6.

f The argument is a floating point number. It is output without an exponent field in the form "i.f". The precision field specifies the number of places after the decimal point. The default is 6 places. A precision of zero suppresses the period also.

g Outputs a floating point number using conversion code "e" or "f". It uses the conversion code that needs the least width.

s The argument is the address of a string. If precision is supplied, then at most the left-most precision characters of the string will be output.

c The argument is a single character. The precision field has no meaning for this type of field.

If the character following a percent is not part of a valid conversion specification, it is output unchanged. This allows you to output a percent sign with the format string "%%".

Upper case conversion codes "D", "X", "O", "U", and "B" are equivalent to "ld", "lx", "lo", "lu" and "lb" respectively.

3.39.3. Returns

Nothing.

For examples of the format control string, see Kernighan and Ritchie.

3.39.4. Notes

The floating point output routines have been modified to output the literal value "NAN" (standing for "Not A Number") for values that are out of range. The detection mechanism for this case is good if you are using the 8087, but you should be careful when using the floating point package. In general, any floating point logic should be written so that you can detect this type of problem before it occurs.

3.39.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.39.6. Use with

printf, sprintf, fopen

3.40. fputc, Output character to a stream

3.40.1. Synopsis

```
#include "stdio.h"

int fputc(byte,stream)
char byte;                    /* the byte to be output */
FILE *stream;                 /* where to put it */
```

3.40.2. Function

Outputs the byte to the stream.

3.40.3. Returns

The constant EOF if an error occurs, otherwise the byte.

3.40.4. Notes

This is the basic output function in the DOS2 library.

3.40.5. Example

```
{
  extern int fputc();          /* put a character to file stream */
  char ch;
  FILE *fptr;
  int res;

        fptr = stdout;
        ch = '*';
        res = fputc(ch,fptr);      /* write '*' to stdout */
        /* res contains EOF if an error occured, else ch */
}
```

3.40.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.40.7. See also

fopen, printf, fclose

3.41. fputs, Output a string to a stream.

3.41.1. Synopsis

```
#include "stdio.h"

int fputs(string,stream)
char *string;
FILE *stream;
```

3.41.2. Function

Outputs the string to the file designated by the stream pointer, until a null character (value of binary zero) is detected. If the file was opened in ASCII mode, newlines are output as carriage return/linefeed pairs.

3.41.3. Returns

The constant EOF if an error is detected, otherwise zero.

3.41.4. Example

To print a string on the standard error stream:-

```
     fputs("Print this string on standard error\n",stderr);
```

3.41.5. Example

```
#include "stdio.h"

{
  extern int fputs();  /* write a string to a file stream */
  extern char *fgets();
  extern FILE *fopen();
  extern int fclose();
  char *string;
  FILE *stream;

     stream = fopen("a:filename.dat","w");
     if(!stream) abort("can't open a:filename.dat");
     string = "Test Data";
     fputs(string,stream);
     fclose(stream);

     /* the following code will echo console input */
        fputs("\nEnter data followed by a CTRL-Z > ",stdout);
        while(fgets(string,255,stdin))
                       fputs(string,stdout);
}
```

3.41.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.42. fread, Read items from a stream

3.42.1. Synopsis

```
#include "stdio.h"

int fread(where,elsize,nelem,stream);
char *where;                /* where to put the data */
unsigned elsize;            /* size of an element in bytes */
unsigned nelem;             /* number of elements to read */
FILE *stream;               /* where to get them */
```

ACTION

Reads nelem elements of elsize bytes each.

3.42.2. Returns

The number of COMPLETE elements read. Zero is returned if the end of file is encountered or if an error is detected.

This function may return less than the requested number of elements. In files that have been opened in ASCII mode, a newline is considered to complete the input for the current element.

3.42.3. Example

```
#include "stdio.h"
{
  extern int fread();      /* read items from a stream */
  extern FILE *fopen();
  extern int fclose(), free();
  extern char *calloc();
  char *dest;
  unsigned size, number;
  FILE *source;
  int num_read;

        size = 1;       /* read bytes */
        number = 255;   /* read up to 255*size bytes */
        source = fopen("a:filename.dat","r");
        if(!source) return;
        dest = calloc(255,1);     /* get area to read into */
        num_read = fread(dest,size,number,source);    /* read */
        /* if(num_read==0) end of file or error occurred */
        /* if(num_read!=number) EOF was encountered */
        free(dest);               /* return area to heap */
}
```

3.42.4. Operating System
DOS 3.0, DOS 2.0+, DOS 1.1+

3.42.5. Use with
fopen, fseek, fscanf, fclose

3.43. free, Return a region to the heap.

3.43.1. Synopsis

```
int free(pointer)
char *pointer;
```

3.43.2. Function

Free a region of storage and return it to the heap. Pointer is the address of the region, which must have been obtained by a call to alloc, malloc, calloc or realloc.

Aborts after writing "FREE" to the console if the heap or the returned block header has been corrupted.

3.43.3. Returns

Nothing.

3.43.4. Notes

Corruption of the heap will be caused by storing outside of the allocated region. Frequently this problem is the result of:-

- allocating too few bytes to hold a structure.
- array subscript out of range.
- an uninitialised pointer.

3.43.5. Example

To allocate and subsequently free a string area 18 bytes in length:-

```
     char *string,*alloc();

     string=alloc(18);       /* aborts if not enough core */
     ...                     /* more processing */
     free(string);           /* all done now */
```

3.43.6. Example

```
{
  extern int free();              /* free up allocated space */
  extern char *calloc();
  char *ptr;
  int i;

     ptr = calloc(100,1);         /* allocate and zero 100 bytes */

     for(i=0;i<26;i++) *(ptr+i) = 'A'+i; /* use ptr for storage */

     /* the statment  ptr[i] = 'A'+i;   is same as above */

     free(ptr); /* return to heap to make available for
                   subsequent memory allocation requests */
}
```

3.43.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.43.8. See also

alloc, malloc, calloc, realloc

3.44. freopen, Close and reopen a file

3.44.1. Synopsis

```
#include "stdio.h"

FILE *freopen(filename,fomode,stream)
unsigned char *filename;        /* file name string */
unsigned char *fomode;          /* file open mode string */
FILE *stream;                   /* currently open stream ptr */
```

3.44.2. Function

Close the file associated with stream and attempt to open a new file using the file name and mode given.

3.44.3. Returns

A pointer to the new file opened or NULL if the file open failed.

3.44.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.44.5. See also

fclose, fopen

3.45. frexp, Split double into mantissa and exponent.

3.45.1. Synopsis

```
double frexp(val,eptr)
double val;
int *eptr;                        /* where to put exponent */
```

3.45.2. Function

Returns the mantissa and exponent of a double. The mantissa is less than one and greater or equal to a half. The exponent is stored at the address specified by "eptr".

3.45.3. Returns

A result such that:-

```
     val == mantissa*(2**exponent);
```

Where "**" means 2 to the power "exponent".

3.45.4. Notes

The exponent is in the range -1023 through +1023. Zero is returned if the input value is zero.

This routine is useful for "normalization" of floating point quantities.

3.45.5. Example

```
{
  extern double frexp();
  double dval, mantissa;
  int exponent;

        dval = 1.2e10;
        mantissa = frexp(dval,&exponent);

        /* frexp returns values such that:
        dval = mantissa * (2 to the power 'exponent') */

        printf("\nFREXP\n(%g) = (%g) * 2 to the (%d)\n",
                dval,mantissa,exponent);
}
```

3.45.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.45.7. See also

ldexp, modf

3.46. fscanf, Scan fields from a stream.

3.46.1. Synopsis

```
#include "stdio.h"

int fscanf(stream,format,args)
FILE *stream;                 /* where to get data */
char *format;                 /* conversion control data */
something *args;              /* where to put data */
```

3.46.2. Function

Reads ASCII characters from the input stream, interprets them under control of the format string, and stores them at the addresses specified by the remaining arguments.

3.46.3. Returns

The number of values successfully input and converted. The standard version includes the number of literal characters matched in the control string in the count.

Fscanf returns EOF if EOF is encountered as the first character.

CONVERSION CONTROL

The format control string contains:-

Blanks, tabs and newlines(white-space characters), which are ignored.

Literal characters (other than %) which must match the next non white-space character from the input stream.

Conversion specifications consisting of a % followed by an optional assignment suppression character ("*"), an optional number specifying a maximum field width, and a conversion character.

A conversion specification controls the processing of the next input field from the stream. The result of the conversion is placed at the address specified by the corresponding argument.

If the assignment suppression indicator is specified, the converted value is discarded. No argument should be specified in the argument list for such values.

Generally, white-space preceeding an input field in the input stream is discarded. If a field width is specified, no more than that number of characters will be read from the input stream.

The legal conversion control codes are:-

d Convert a decimal number and store in an integer. Input stops at the first non decimal digit.

o An octal number. Store in an integer.

x A hexadecimal number, with or without a leading 0X.

h A short decimal integer. For this machine this is identical to a "d" conversion code.

b A binary number. Store in an integer.

e A floating point number in scientific notation. A leading sign, decimal point and exponent field are optional. The result is stored in a float.

f Same as the "e" conversion code.

s The input is a string of characters, and the corresponding argument should point to an area large enough to hold the string and a terminating zero ("\0"). The input string is terminated by the first white-space character after the beginning of the string. The white space character is not stored.

c A single character is to be stored. Leading white-space characters in the input stream are not skipped by this conversion code. If a field width is specified, that number of characters will be transferred to successive memory locations. To read the first non white space character from the input stream use "%ls".

All the above conversion codes that produce an integer or float may be made to return a long or double by using the upper case letter or preceeding the lower case conversion code by the lower case letter "l".

3.46.4. Notes

You MUST provide the ADDRESS of the variable to contain the input data. For most variables the correct expression is &var_name.

This, and related functions, are really designed to read machine generated files. If you want to read input typed by a human, see the notes under scanf and sscanf.

Any unmatched characters are pushed back onto the input stream, and are returned by the next input request from the stream. Reading end of file or an error will terminate the function.

In general, you should also read scanf and sscanf to understand how all of these functions fit together and all of their implications.

3.46.5. Examples

There is a good description of scanf and printf in K&R. It is recommended reading.

3.46.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.46.7. Use with

sscanf, fprintf, fscanf

3.47. fseek, Seek using a long offset.

3.47.1. Synopsis

```
#include "stdio.h"

long fseek(stream,offset,base)
FILE *stream;
long offset;
int base;
```

3.47.2. Function

This function allows you to alter the read/write position pointer for a disk file. This pointer defines the character that will be read/written on the next i/o operation on the file. To issue this call:-

- stream is a stream pointer returned by fopen.
- offset is an adjustment relative to base
- base is a code for the base value of the seek

Allowable base codes are:-

- 0 Relative to beginning of file. Offset must be positive
- 1 Relative to current position in file
- 2 Relative to the end of the file (SEE NOTE BELOW)

3.47.3. Returns

- minus one if an error is detected
- The current position in the file if successful.

3.47.4. Notes

fseek(fd,0L,0) will let you process the first byte in the file on the next i/o operation. fseek(fd,-1L,1) will let you process the most recently processed byte again.

Because of a serious bug in DOS, seeking before the beginning of the file or after the end of the file will cause undefined results in DOS 2.0+ version of our library. It is your responsibility to handle these cases.

Seeks on a non disk file return the error code -1. Seeks beyond End Of File should be avoided, since they may result in files with missing sectors, which could result in incorrect EOF indications in subsequent processing.

ASCII files may be used, but the presence of carriage return/line feed pairs may make it difficult to determine the seek offset.

For files open in binary mode, the end of file is assumed to be the physical end of file point. This probably was not what you intended.

The open logic for ASCII mode files reads the last sector looking for a control-z, and sets the end of file position accordingly.

3.47.5. DOS

This feature is provided, and uses the operating system provided file size. Thus it will not work with files written by programs using CPM end of file conventions.

With the DOS-ALL I/O package, files must be open in read or read/write mode for this function to operate correctly.

3.47.6. Example

```
#include "stdio.h"
#define BEGIN 0
#define CURRENT 1
#define END 2
{
  extern long fseek();  /* uses a long offset */
  extern FILE *fopen();
  extern int fclose();
  FILE *stream;
  long offset, lpos;
  int base;

     stream = fopen("a:filename.dat","r");

     /* To position on first byte: */
     offset = 0L;
     lpos = fseek(stream,offset,BEGIN);

     /* To position such that last byte will be reprocessed:  */
     offset = -1L;
     lpos = fseek(stream,offset,CURRENT);

     /* To position such that the byte at End of File - 100
        will be processed next:      */
     offset = -100L; lpos = fseek(stream,offset,END);

     /* In the above calls to fseek, lpos contains either:
        (-1) if an error detected (such as seek beyond EOF)
        or the current position relative to the beginning */
}
```

3.47.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.47.8. Use with

fopen, fclose, ftell, fread, getc, putc

3.48. ftell, Tell R/W position in a stream.

3.48.1. Synopsis

```
#include "stdio.h"

long ftell(stream)
FILE *stream;
```

3.48.2. Returns

The current read/write position in the stream.

3.48.3. Notes

On many systems, this is the only way to obtain a valid offset to use with fseek.

3.48.4. Example

```
#include "stdio.h"

{
  extern long ftell();  /* get current position in the file */
  extern FILE *fopen();
  extern int fclose();
  FILE *stream;
  long position;

     stream = fopen("a:filename.ext","a");
     fprintf(stream,"Sample Data\n");

     position = ftell(stream);
     /* position contains the current position in the file */

     fclose(stream);
}
```

3.48.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.49. ftoa, Convert float to ASCII.

3.49.1. Synopsis

```
int ftoa(value,buffer,iplaces,fplaces)
double value;                /* the value to convert */
char *buffer;                /* buffer to hold output string */
unsigned iplaces;            /* number of integer places */
unsigned fplaces;            /* number of fractional places */
```

3.49.2. Function

Converts the input value to an ASCII output string of the format "[-]iii.fffE[-]eee". The number of integer places, and the number of fractional places are under the user's control.

3.49.3. Notes

The sum of iplaces and fplaces should not exceed 15, the number of significant digits in a double precision number. If the number is too large, returns the literal value 'NAN' ("Not A Number").

3.49.4. Example

```
{
  extern int ftoa();
  extern char *calloc();
  extern int free();
  double dval;
  char *outstr;
  unsigned int_places;
  unsigned frac_places;

          outstr = calloc(255,1);

          dval = -100.0144;
          int_places = 5;
          frac_places = 7;

          ftoa(dval,outstr,int_places,frac_places);
          /* outstr now contains:  "-1.0001440E+002"  */

          free(outstr);
}
```

3.49.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.49.6. See also

atof, scanf, sscanf

3.50. fwrite, Write to a stream.

3.50.1. Synopsis

```
include "stdio.h"

int fwrite(where,elsize,nelem,stream)
char *where;                    /* pointer to data */
unsigned elsize;                /* size of one element */
unsigned nelem;                 /* number of elements to write */
FILE *stream;                   /* where to put it */
```

3.50.2. Function

Transfers elsize*nelem bytes to the specified stream.

3.50.3. Returns

The number of elements written to the stream.

3.50.4. Example

```
{
  extern int fwrite();            /* write to a stream */
  extern char *calloc();
  extern int free();
  char *buffer;
  unsigned size, number;
  FILE *output;
  int num_written;

          buffer = calloc(255,1);
          size = 1;
          number = 255;
          output = fopen("a:filename.ext","a");
          strcpy(buffer,"\nSample Data is Here\n");
          num_written = fwrite(buffer,size,number,output);
          /* num_written contains the number of elements
             written ( zero if error ) */
          free(buffer);
}
```

3.50.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.50.6. Use with

fopen, fclose, fread, fseek, fprintf

3.51. gcdir, Get the current directory.

3.51.1. Synopsis

```
char *gcdir(drivename)
char *drivename;
```

3.51.2. Function

Obtain the full pathname for the current directory on the specified drive.

If the drivename string begins with a letter in the range 'A' through 'P' and is followed by a colon (":"), then that is the drive used. Otherwise the result is for the current default drive.

3.51.3. Returns

A string allocated from the heap containing the drive and full path name of the current directory on the specified drive. If any error occurs, or you are not running on DOS 2.0+, returns zero.

3.51.4. Notes

You can dispose of the returned string using the function "free".

3.51.5. Example

To obtain the current default directory:-

```
          gcdir("");
```

To obtain the default directory on drive B: :-

```
          gcdir("b:any_rubbish");
```

3.51.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.51.7. Use with

chdir, fopen, open

3.52. getc, Read a character from a stream.

3.52.1. Synopsis

```
#include "stdio.h"

int getc(stream)
FILE *stream;
```

3.52.2. Function

Read the next input character from the stream.

3.52.3. Returns

The input character as a positive integer.
-1 on end of file or if an error was detected.

3.52.4. Notes

This function is defined as a macro in stdio.h. Actually the function fgetc is used.

If the stream is open in ASCII mode, newline processing (etc) is performed. Otherwise no special processing is performed.

3.52.5. Example

To copy standard input to standard output.

```
     int cc;

     while( (cc=getc(stdin)) != -1)putc(cc,stdout);
          /* get here at end of file */
```

3.52.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.52.7. Use with

fgetc, fopen, fclose, putc, fprintf

3.53. getchar, Get a character from stdin.

3.53.1. Synopsis

```
int getchar()
```

3.53.2. Function

Read the next character from stdin. This is normally the console, but it may be a file if redirection has been performed.

3.53.3. Returns

A positive integer if a character is returned. -1 at end of file, or if an error was detected.

3.53.4. Notes

This call is converted by a macro in stdio.h into the call fgetc(stdin).

Stdin is normally assigned to the console keyboard, which is open in ASCII mode. Under this condition, input from the console is buffered a line at a time, and will not be available to the program until the user enters a carriage return.

If you want unbuffered input from the console, use bdos() or sysint21() to make direct calls to your operating system. This will allow you to control the recognition of special characters and the echo of input characters.

3.53.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.53.6. Use with

fopen, fclose, putc, printf

3.53.7. See also

open, close

3.54. gets, Read a string from standard input.

3.54.1. Synopsis

```
char *gets(buffer,bufleng);
char *buffer;                   /* where to put it */
unsigned int bufleng;           /* how much to read */
```

3.54.2. Function

Reads characters from the stream into the buffer until:-

- A newline is read from the input stream
- (bufleng-1) characters have been transferred
- End of file is encountered

In all cases the string will be terminated by a NULL.

3.54.3. Returns

- The address of the data buffer
- Zero at end of file or if an error is detected

3.54.4. Notes

WARNING: This function is not the same as the UNIX standard! If you are concerned about portability use the function fgets as follows:

```
fgets(buffer,bufleng,stdin);
```

If the file was opened in ASCII mode, carriage return and control-z characters will receive special processing.

3.54.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.54.6. Use with

fopen, fclose, putc, fgetc

3.54.7. See also

open, close

3.55. getw, Get a word from a stream.

3.55.1. Synopsis

```
#include "stdio.h"

int getw(stream)
FILE *stream;
```

3.55.2. Function

Read the next input word from the stream.

3.55.3. Returns

The input word. EOF on end of file or if an error was detected.

3.55.4. Notes

The error and end of file indications returned by this function are also a valid data word. Use the functions feof and ferror to distinguish these cases.

If the stream is open in ASCII mode, newline processing (etc) is performed. Otherwise no special processing is performed.

3.55.5. Example

```
#include "stdio.h"
{
  extern int getw(), putw(), ferror();
  FILE *instream, *outstream,*fopen();
  int word;

  instream = fopen("a:filename.ext","r");
  if(!instream) return; /* file not found */
  outstream = fopen("a:filename.out","w");
  if(!outstream) { fclose(instream); return; }
  for(;;)                 /* do forever */
     {
     word = getw(instream);
     if(feof(instream) || ferror(instream))break; /* stop now */
     putw(word,outstream);
     }
  fclose(outstream);
  fclose(instream);
}
```

3.55.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.55.7. Use with

fgetc, fopen, fclose, putc, fprintf

3.56. index, Find a character in a string.

3.56.1. Synopsis

```
char *index(string,cc)
char *string;                 /* string to search */
char cc;                      /* char to find */
```

3.56.2. Function

Report the first occurrence of the character cc (if any) in the string.

3.56.3. Returns

Zero if the character was not found in the string. If the character is found, returns a pointer to the character.

3.56.4. Notes

The function strchr() is the same as this function. You should use the strchr() function to be more UNIX v5.0 compatible.

3.56.5. Example

```
{
  extern char *index();
  char *string;
  char ch;
        /* this could be used to implement a function to check
        if a character is part of a given set of characters,
        as well as for extracting the position of the character
        in the string. A function is_vowel(c) could be written:
                is_vowel(ch)
                char ch;
                {
                        return(index("aeiouAEIOU",ch));
                }
        Of course, this could be #defined as a macro:
        #define isvowel(ch) index("aeiouAEIOU",ch)      */

        /* another example */
        string = "123456789"; ch = '1';
        if(index(string,ch) != 0 )
                printf("\n*index(%s,%c) = %c\n",
                        string, ch, *index(string,ch));
}
```

3.56.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.56.7. See also

string functions, rindex

3.57. inportb, inportw - Input a byte or word from a port.

3.57.1. Synopsis

```
char inportb(portno)
int portno;

int inportw(portno)
int portno;
```

3.57.2. Function

Inportb inputs a byte from a user supplied port number (portno). The port number must be valid for the addressed device. In some cases a 16 bit port number is required. For older devices an 8 bit number is required, and it may have to be in either byte of portno. One possibility is to place the port number in both upper and lower bytes of portno. It returns the byte from the port.

Inportw inputs a word from a port number (portno). The port number must be valid for the addressed device. Usually a 16 bit port number is required. This function is not needed for most devices currently available, as they do not support 16 bit i/o transfers. It returns the word read from the port. The bytes may also be in reverse order.

3.57.3. Example

```
* Inportb example

{
  extern unsigned char inportb();
  unsigned int portno;
  unsigned char byte;
        portno = 1;
        byte = inportb(portno);
}

* inportw example

{
  extern unsigned int inportw();
  unsigned int portno;
  unsigned int word;
              portno = 1;
              word = inportw(portno);
}
```

3.57.4. Operating System
DOS 3.0, DOS 2.0+, DOS 1.1+

3.57.5. See also
outport functions

3.58. intrinit, intrrest - Init and restore for interrupt processing

3.58.1. Synopsis

```
intrinit(func,stack,vecno)
int (*func)();              /* function which will process interrupt */
unsigned stack;             /* # bytes of stack needed by function */
unsigned vecno;             /* # of vector for interrupt trap */

intrrest(vecno)
unsigned vecno;
```

3.58.2. Function

intrinit initializes the interrupt vector "vecno" so that the specified function is executed whenever the interrupt occurs.

intrrest restores the original interrupt handling address to the proper vector location specified by vecno.

3.58.3. Notes

This function also needs the function intrserv. Interserv() is in the C86 libraries. You need to understand your hardware and assembly language programming to use this routine correctly.

Make sure you provide enough stack space for the function, as there are no run time checks, and stack overflow will provide an interesting debugging experience. About 5000 bytes should be used for debugging. In production you need 128 bytes plus the size of your local data.

The interrupt entry is relatively fast, but you should test this mechanism before writing a lot of code. It should be fast enough to cope with the communications line (up to 9600 baud), light pen, game paddles, etc.

If you are processing a device interrupt (eg the RS-232 USART), you may need to issue an End Of Interrupt command to the 8259 interrupt controller chip. This may be done using outportb().

IMPORTANT: Interrupts are disabled while your function is executing, so keep it small and fast. Do not do a printf inside your function. Some interrupts are so fast that you can not even do another interrupt. The timer tick (see example) only has time to decrement a variable. It is very important to keep it small and fast.

The operating system is in an unstable state while an interrupt is being processed. Bdos() calls 12 and below should be safe, but check to be sure.

When your program terminates, you should restore the interrupt vector to its original content. This applies only to vectors that may be called after your program terminates. Classical instances are Clock, Break key and Key-board handlers. Otherwise strange things will happen after your program has been executed. You can cause a software interrupt with sysint. This may help you debug a prototype interrupt routine, as it can be traced with the debugger.

The 2.1 version of intrinit has been recoded for the two following reasons:

- It uses the official MSDOS entry point for pokeing the interrupt vector
- It provides for restoring the original interrupt handling address to the vector location through the function intrrest(vecno).

3.58.4. Example

```
/*
         NOTES ON USING INTERRUPT SERVICE ROUTINES

   The following is a program to demonstrate the use of intrinit
to assign new interrupt service routines to the clock tick and
cntl-break interrupts under DOS. This program will do a bdos(6,7)
to sound a beep intermittently. This can be turned off by
pressing CTRL-BREAK.

*/

#include <stdio.h>

#define STACK 5000                  /* no. bytes of stack to save */
#define TICK 0x1c                   /* Clock Tick interrupt */
#define KEYBOARD 0x1b               /* Control-Break interrupt */

long counter;
long start;
int ctrlbreak;

/* interrupt service routine for clock tick */
timer()
{
   counter--;
}

/* interrupt service routine for control-break */
ctrl()
{
   ctrlbreak = 0;
}
```

```
/* main routine */

main()
{
  printf("\nEnter approximate no. of seconds: ");
  scanf("%D",&counter);
  printf("Number entered: %D\n",counter);
  if(counter==0)counter=1;
  counter *= 20;                        /* was on 8MHz 80186 */
  start = counter;
  printf("Press CTRL-BREAK to stop this process:\n");

  intrinit(timer,STACK,TICK);           /* set up new routine */
  intrinit(ctrl,STACK,KEYBOARD);        /* set up new routine */

  for(ctrlbreak=1;ctrlbreak;){
     while(counter > 0)                 /* count down */
        if(!ctrlbreak) goto done;       /* quit */
     bdos(6,7);                         /* issue beep */
     counter = start;                   /* reset counter */
  }
done:
  intrrest(TICK);                       /* restore original */
  intrrest(KEYBOARD);                   /* restore original */
} /* end program for intrinit */


------------------ intrrest() example -------------------

/*      The following is an example of trapping the Control-C
        handler.
        It now uses intrrest() to reset the vector number.
*/
#include <stdio.h>
extern int dummy();
main()
{
  int foo;

  intrinit(dummy,256,0x23);     /* set to point to dummy */
  for(foo=0;!foo;) bdos(2,'-'); /* print out '-', while U wait */
  intrrest(0x23);               /* restore orig. ^C handler addr */
}

int dummy()
{
  int foo;
  foo = 1;
  bdos(2,'*');
}
```

3.58.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.59. issomething, Character class tests.

3.59.1. Synopsis

```
int isalnum(cc)      /* alpha-numeric */
int isalpha(cc)      /* alphabetic */
int isascii(cc)      /* a defined ASCII character */
int iscntrl(cc)      /* a control character */
int isdigit(cc)      /* a digit */
int islower(cc)      /* a lower case alphabetic */
int isprint(cc)      /* a printable character */
int ispunct(cc)      /* a punctuation character */
int isspace(cc)      /* a white space character */
int isupper(cc)      /* an upper case character */
char cc;
```

3.59.2. Function

Test the supplied character to see if is a member of a specific class.

3.59.3. Returns

One if the character is a member of the class, otherwise zero.

3.59.4. Notes

The classes tested by the functions are:-

function	class
isalnum	'a' thru 'z', 'A' thru 'Z', '0' thru '9'
isalpha	'a' thru 'z', 'A' thru 'Z'
isascii	0x00 thru 0x7f
iscntrl	0x00 thru 0x1f, 0x7f
isdigit	'0' thru '9'
islower	'a' thru 'z'
isprint	0x20 thru 0x7e
ispunct	040 thru 057, 072 thru 0100, 0133 thru 0140, 0173 thru 0176
isspace	0x20, '\t' or '\n'
isupper	'A' thru 'Z'

See ctype.h for an alternative to function calls to perform these functions.

3.59.5. Example

```
{
  int r;
       /* character class tests:
         INPUT:  character
         OUTPUT: integer value:
                1 if character in class
                0 if not
       */
       r = isalnum('#');    /* r is 0 */
       r = isalpha('C');    /* r is 1 */
       r = isascii(0xff);   /* r is 0 */
       r = iscntrl('\t');   /* r is 1 */
       r = isdigit('0');    /* r is 1 */
       r = islower('W');    /* r is 0 */
       r = isprint(0x7f);   /* r is 0 */
       r = ispunct(' ');    /* r is 0 */
       r = isspace('&');    /* r is 0 */
       r = isupper('W');    /* r is 1 */

       printf("\nisalnum(%c)  = %d\n",'#',isalnum('#'));
       printf("\nisalpha(%c)  = %d\n",0x07,isalpha(0x07));
       printf("\nisascii(%c)  = %d\n",0x07,isascii(0x07));
       printf("\niscntrl(%c)  = %d\n",'\t',iscntrl('\t'));
       printf("\nisdigit(%c)  = %d\n",'9',isdigit('9'));
       printf("\nislower(%c)  = %d\n",'a',islower('a'));
}
```

3.59.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.60. iswap, Swap two integers.

3.60.1. Synopsis

```
int iswap(inta,intb)
int *inta,*intb;
```

3.60.2. Function

Swaps the two integers pointed to by inta and intb.

3.60.3. Returns

Nothing

3.60.4. Notes

This function MUST be called with the addresses of the two integers to be swapped. Using the value(s) will lead to obscure system failures, that are difficult to find.

3.60.5. Example

```
{
  extern int iswap();           /* integer swap */
  int x, y;


        x = 5; y = 6;
        printf("\nx=%d, y=%d\n",x,y);

        iswap(&x,&y);           /* MUST supply addresses */

        /* now x equals 6 and y equals 5 */
        printf("ISWAP\n x=%d, y=%d\n",x,y);
}
```

3.60.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.61. itoa, Convert an integer to ASCII.

3.61.1. Synopsis

```
int itoa (n,buffer)
int n;                    /* value to convert */
char *buffer;             /* where to put ascii characters */
```

3.61.2. Function

Converts the input binary integer n into the equivalent ASCII string in buffer. A leading minus sign is output if the number is negative. The buffer must be at least 7 characters in length, to hold the largest possible number. The string is terminated by a binary zero.

3.61.3. Returns

The number of characters placed in the buffer. The value is equivalent to strlen(buffer) after the conversion is completed.

3.61.4. Notes

Uses the function sprintf.

3.61.5. Example

```
{
  extern int itoa();     /* integer to ascii conversion */
  extern int fputs();
  int binary;
  char string[7];
  int num;

        binary = -12;
        num = itoa(binary,string);
        /* num contains the number of chars put in string */
        fputs(string,stdout);
}
```

3.61.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.61.7. See also

sprintf, fprintf

3.62. itoh, Convert an integer to hexadecimal.

3.62.1. Synopsis

```
int itoh(n,buffer)
unsigned int n;                 /* the value to convert */
char *buffer;                   /* where to place it */
```

3.62.2. Function

Converts the input integer into the equivalent hex string in buffer. The string is terminated by a binary zero. Buffer must be at least 5 characters in length. No leading "0X" is placed in the buffer.

3.62.3. Returns

The number of characters placed in the buffer. This is equivalent to strlen(buffer) after the conversion has been done.

3.62.4. Notes

Uses the function sprintf

3.62.5. Example

```
{
  extern int itoh();
  extern int fputs();
  unsigned int n;
  char hexstr[5];
  int number;

        n = 0x3ff;

        number = itoh(n,hexstr);

        /* hexadecimal characters are stored in hexstr */

        fputs(hexstr,stdout);
}
```

3.62.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.62.7. See also

sprintf, fprintf

3.63. key_getc,key_scan,key_shft-Z-100 PC keyboardfunct. (Z-100 PC ONLY!)

3.63.1. Synopsis

int key_getc()

int key_scan()

int key_shft()

3.63.2. Function

key_getc reads the next character typed at the keyboard. Returns the ASCII value of the character (in the low byte) and the keyboard scan code (in the high byte). The character is removed from the keyboard buffer and is not echoed to the screen.

key_scan scans the keyboard for a character. It does a non-destructive read, that is, the character is not removed from the keyboard buffer. It will return an int in the same format as key_getc if a character is available, otherwise it returns -1.

key_shft returns the keyboard shift status byte as described in the notes.

3.63.3. Notes

See the Z-100 PC Technical Reference Manual for more details.

Keyboard shift status byte:

bit	mask	meaning
0	0x01	RIGHT SHIFT KEY depressed
1	0x02	LEFT SHIFT KEY depressed
2	0x04	CTRL key depressed
3	0x08	ALT key depressed
4	0x10	SCROLL state active
5	0x20	NUMBER lock engaged
6	0x40	CAPS lock engaged
7	0x80	INSERT state engaged

3.63.4. Example

* To read a character and it's scan code:

```
int c;
c = key_getc();
```

* To find out if the right shift key is being depressed:

```
rshkey = key_shft() & 0x01;
```

* To find out if the control key is being depressed:

```
ctrlkey = key_shft() & 0x04;
```

3.63.5. Operating System

MS-DOS Version 2+, MS-DOS 1.1+

3.64. longjmp, Restore an environment.

3.64.1. Synopsis

```
#include "stdio.h"          /* to define jmp_buf */

int longjmp(envp,value);
jmp_buf *envp;
int value;
```

3.64.2. Function

Restores the environment to one previously saved using the function set_jmp(). The value is returned as the exit value of set_jmp().

3.64.3. Returns

Never returns.

3.64.4. Notes

This is a very dangerous function, but if you really want to use it see set_jmp().

The environment must have been saved using set_jmp by a function that is currently active, and which is the same function or a parent of the function containing the call to longjmp.

3.64.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.64.6. Use with

set_jmp

3.65. ldexp, Load exponent

3.65.1. Synopsis

```
double ldexp(mantissa,exponent)
double mantissa;
int exponent;
```

3.65.2. Function

Returns the double:-

```
     mantissa*(2**exponent)
```

3.65.3. Notes

This is the inverse of the function "frexp".

3.65.4. Example

```
{
  extern double ldexp();
  double mantissa;
  int exponent;
  double dresult;

        mantissa = 1.444;
        exponent = 10;

        dresult = ldexp(mantissa,exponent);

        /* dresult contains: 1.444 * (2 ^ 10 ) */
        printf("\nLDEXP\n%g = %g * ( 2 to the %d)\n",
                        dresult, mantissa, exponent);
}
```

3.65.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.65.6. See also

frexp, modf

3.66. loadexec, Load or execute a program

3.66.1. Synopsis

```
int loadexec(filename,param,funcode)
char *filename;
struct pblock *param;
int funcode;
```

3.66.2. Function

Performs the DOS V2.0 load or execute a program function call. This function is called by the system function. The system function is much easier to use and is recommended over the loadexec function.

3.66.3. Returns

Zero if successful, otherwise a DOS 2.0 error code.

3.66.4. Notes

This function is ONLY available under DOS V2.0.

We really wanted to create the UNIX functions 'exec' and 'system', but that would have removed some useful abilities of this system service.

See the writeup of service 0x4b in the DOS V2.0 manual for full details of this function. Funcode is 3 to load a program, and zero to load and execute a program.

The filename is a standard DOS V2.0 file name with path specification if desired.

The parameter block is as explained in the DOS manual. The filename and param block pointers are ALWAYS big model pointers, even in a small model program. See the source code of the function "system" for an example.

Since this function will load and execute a program in unused memory you may not have enough room if you are running your program on a computer with a small amount of memory. If you do not have enough room in your machine, you will have to modify the defaults in _default, so that the program will leave some memory free. Otherwise there will be no memory available for loadexec to load the target code into.

3.66.5. Operating System
DOS 3.0, DOS 2.0+

3.66.6. See also
system

3.67. log, log10, Logarithm functions.

3.67.1. Synopsis

```
double log(val)
double val;

double log10(val)
double val;
```

3.67.2. Returns

Log returns the natural logarithm of the value.

Log10 returns the logarithm of val to the base 10.

3.67.3. Notes

Both functions return zero if the value is zero or negative.

3.67.4. Example

```
{
  extern double log();
  extern double log10();
  double dval;
  double dresult;

     dval = 45.023;
     dresult = log(dval);
     /* dresult contains 3.80717.....
          log returns the natural logarithm */

     dval = 45.023;
     dresult = log10(dval);
     /* dresult contains 1.6534....
          log10 returns the base 10 logarithm */

     printf("\nLOG\n%g log(%g)\n",24.56,log(24.56));
     printf("\nLOG10\n%g log10(%g)\n",77e3,log10(77e3));
}
```

3.67.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.67.6. See also

exp, pow, sqrt

3.68. lower, Convert a string to lower case.

3.68.1. Synopsis

```
char *lower(string)
char *string;
```

3.68.2. Function

Converts all uppercase characters in the string to lower case. All other characters are unchanged.

3.68.3. Returns

The address of the string.

3.68.4. Example

Read a filename and force it to lower case.

```
{
  char filename[80];

  printf("Enter file name: ");
  gets(filename,75);
  lower(filename);
  printf("\nname is:- %s\n",filename);
}
```

3.68.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.68.6. See also

upper

3.69. lseek, Position R/W pointer in a file.

3.69.1. Synopsis

```
long lseek(fd,offset,base)
int fd;
long offset;
int base;
```

3.69.2. Function

This function allows you to alter the read/write position pointer for a disk file. This pointer defines the character that will be read/written on the next i/o operation on the file. To issue this call:-

- fd is a file descriptor returned by open or creat.
- offset is an adjustment relative to base.
- base is a code for the base value of the seek.
- The file must be open in read or read/write mode.

Allowable base codes are:-

- 0 Relative to beginning of file. Offset must be positive
- 1 Relative to current position in file
- 2 Relative to the end of the file (SEE NOTE BELOW)

3.69.3. Returns

.minus one if an error is detected
.The current position in the file if successful.

3.69.4. Notes

Lseek(fd,0L,0) will let you process the first byte in the file on the next i/o operation. Lseek(fd,-1L,1) will let you process the most recently processed byte again.

Seeks on a non disk file return the error code -1. Seeks beyond End Of File should be avoided, since they may result in files with missing sectors, which could result in incorrect EOF indications in subsequent processing.

ASCII files may be used, but the presence of carriage return/line feed pairs may make it difficult to determine the seek offset.

The open logic for ASCII mode files reads the last sector looking for a control-z, and sets the end of file position accordingly.

3.69.5. DOS

This feature is provided, and uses the operating system provided file size. Thus it will not work with files written by programs using CPM end of file conventions.

3.69.6. Example

```
#define BEGIN 0
#define CURRENT 1
#define END 2

{
  extern long lseek();
  extern int open() , close();
  int fd, base;
  long offset, lpos;

        fd = open("a:filename.dat",AREAD);
        if(fd<0) return;

        /* To position on the first byte: */
        offset = 0L; lpos = lseek(fd,offset,BEGIN);

        /* To position so that last byte will be reprocessed:  */
        offset = -1L;
        lpos = lseek(fd,offset,CURRENT);

        /* To position such that the byte at End of File
          - 50 will be processed next:        */
        offset = -50L;
        lpos = lseek(fd,offset,END);

        /* In the above calls to lseek, lpos contains either:
             (-1) if an error detected (such as seek beyond EOF)
             or the current position relative to the beginning */
        close(fd);
}
```

3.69.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.69.8. Use with

open, creat, close, ltell

3.70. ltell, Tell the R/W position within a file.

3.70.1. Synopsis

```
long ltell(fd)
int fd;
```

3.70.2. Function

This function returns the absolute position of the byte in the file which will be processed by the next i/o operation. To issue this call:-

- fd is a file descriptor returned by open or creat.

3.70.3. Returns

- The absolute position of the next byte in the file
- minus one if any error is detected

3.70.4. Notes

This function must be declared as returning a long integer before its use in a program. It will always return an error when used on a non disk file. This fact may be used to determine if a file is disk file.

This function is equivalent to lseek(fd,0L,1);

3.70.5. Example

```
{
  extern unsigned long ltell();
  extern int open() , close();
  int fd;
  unsigned long position;

     fd = open("a:filename.ext",AUPDATE);
     if(fd<0) return;
     fprintf(fd,"Sample Data\n");

     position = ltell(fd);
     /* position contains the current position in the file */

     printf("\nCurrent position is: %D\n",position);
     close(fd);
}
```

3.70.6. Operating System
DOS 3.0, DOS 2.0+, DOS 1.1+

3.70.7. See also
open, creat, lseek, close

3.71. ltoa, Convert a long integer to ASCII.

3.71.1. Synopsis

```
int ltoa(n,buffer)
long n;
char *buffer;
```

3.71.2. Function

Converts the input long binary integer n into the equivalent ASCII string in buffer. A leading minus sign is output if the number is negative. The buffer must be at least 12 characters in length, to hold the largest possible number. The string is terminated by a binary zero.

3.71.3. Returns

The number of characters placed in the buffer. The value is equivalent to strlen(buffer) after the conversion is completed.

3.71.4. Example

```
{
  extern int ltoa();     /* converts a long to an ASCII string */
  extern int fputs();
  long number;
  char longstr[12];      /* must be at least 12 chars */
  int numchars;

        number = 10223444L;
        numchars = ltoa(number,longstr);

        /* numchars contains number of characters converted */
        /* longstr contains "10223444" */

        fputs(longstr,stdout);
}
```

3.71.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.72. ltoh, Convert a long integer to hexadecimal.

3.72.1. Synopsis

```
int ltoh(n,buffer)
unsigned long n;
char *buffer;
```

3.72.2. Function

Converts the input long integer into the equivalent hex string in buffer. The string is terminated by a binary zero. Buffer must be at least 9 characters in length. No leading "OX" is placed in the buffer.

3.72.3. Returns

The number of characters placed in the buffer. This is equivalent to strlen(buffer) after the conversion has been done.

3.72.4. Example

```
{
  extern int ltoh();
  extern int fputs();
  unsigned long number;
  char hexstr[9];        /* must be at least 9 chars */
  int numchars;

     number = 10223444L;
     numchars = ltoh(number,hexstr);

     /* numchars contains the number of characters converted */

     fputs(hexstr,stdout);

}
```

3.72.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.73. ltos, Convert a long integer to a string.

3.73.1. Synopsis

```
int ltos(n,buffer,base)
long n;               /* the number to convert */
char *buffer;         /* where to put it */
int base;             /* the base for conversion to characters */
```

3.73.2. Function

Converts the input long integer into the equivalent string of characters in the output area 'buffer'. The string is terminated by a binary zero. The output buffer must be long enough to hold the largest possible output number, as this routine does not check.

The conversion is controlled by the value supplied as the base. If the base is positive, an unsigned conversion is performed, otherwise a signed conversion is performed, and a minus sign is output if required.

This routine may be used to convert a signed or unsigned long to an ASCII string to any base in the range 2 through 16 (decimal).

3.73.3. Returns

The number of characters placed in the buffer. This is equivalent to strlen(buffer) after the conversion has been done.

3.73.4. Notes

This routine was written to provide a common conversion routine from binary to ASCII. It is not typically available on UNIX systems.

3.73.5. Example

```
{
  extern int ltos(), fputs();
  unsigned long number;
  char str[34]; /* make sure it's long enough */
  int numchars;

  number = 10223444L;
  numchars = ltos(number,str,-2);  /* convert to signed binary */

  fputs(str,stdout);               /* print it */
}
```

3.73.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.74. main, Entry point for a C program.

3.74.1. Synopsis

```
int main(argc,argv)
int argc;             /* number of arguments on the command line */
char *argv[];         /* an array of pointers to the arguments */
```

3.74.2. Function

This function is user supplied. It is the main function of a C program. It is called with the number of arguments on the command line and an array of pointers to the argument strings.

On a UNIX system, the first argument is always the name of the program. Since this is unavailable, we substitute a lower case "c". Thus argc will always be at least one.

Entries on the command line beginning with a ">" or "<" symbol are assumed to specify redirection and are not supplied to this function. See "_main" for details.

3.74.3. Returns

Zero if the program run without errors, non-zero otherwise.

3.74.4. Example

The command line:-

```
myprog This is a Line >prn:
```

would result in main (inside the file myprog.c) being called with:-

- argc containing 5
- argv pointing to an array of pointers, which point to the strings:-

```
"c"
"This"
"is"
"a"
"Line"
```

3.74.5. Example

To print each of the command line arguments on a separate line:-

```
#include <stdio.h>

main(argc,argv)
int argc;                 /* count of arguments */
char *argv[];             /* argument strings */
{
  int i;

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

3.74.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.74.7. See also

$main, _main

3.75. makefcb, Make a file control block.

3.75.1. Synopsis

```
char *makefcb(filename)
char *filename;
```

3.75.2. Function

Makes a file control block. The input filename string may contain:-

- A single letter drive specifier followed by a colon.
- A path specification of up to 63 characters
- A file name of up to 8 characters.
- An extent consisting of a period followed by up to three characters.

The File Control Block is obtained by a call to calloc.

3.75.3. Returns

The address of the created file control blocb. On error makefcb returns NULL. The format of the file control block is specified in the file "fileio.h" (found in dosall.arc).

3.75.4. Notes

Characters less than Hex 21 are considered as errors. Filenames containing question marks may be used with this function, but should not be used with other supplied file manipulating functions. When you have finished with the fcb, you should return the area to the heap using the function 'free'.

If this function is executed under DOS 2.0+, the path name (if any) will be extracted and saved in the returned fcb. If it is executed on earlier versions of DOS, the path information will be discarded.

The path information is provided for the convenience of the DOSALL library functions. We recommend you do not use the path facilities in your own coding, as the mechanisms may change in future releases.

You should use the DOS 2.0+ calls, and avoid this service if you can.

3.75.5. Example

```
{
  extern char *makefcb();
  extern int free();
  char *filename;
  char *fcbl, *fcb2, *fcb3, *fcb4,*fcb5;

      filename = "a:filename.ext";       /* full specification */

      fcbl = makefcb(filename);

      /* fcbl contains the address of the File Control Block
         created by makefcb. The File Control Block is obtained
         through a call to alloc, and therefore the fcb can be
         freed when you are done with it. Other valid calls
         include:
      */

      fcb2 = makefcb("file");
      fcb3 = makefcb("b:????????.c");
      fcb4 = makefcb("prog.dat");

      fcb5 = makefcb("c:\\bin\\c86\\cc3.exe");

      /* '?' may be used with this function , but not with the
         other file functions (such as rename) */

      free(fcbl);
      free(fcb2);
      free(fcb3);
      free(fcb4);
      free(fcb5);
}
```

3.75.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.75.7. See also

open, creat, fopen, bdos, sysint

3.76. makefnam, Make a file name.

3.76.1. Synopsis

```
char *makefnam(input,default,result);
char *input;          /* the input file name */
char *default;        /* the default file name /
char *result;         /* where to build the result */
```

3.76.2. Function

Builds a composite disk file name in the result area, by combining components from the input and default file names.

This function considers the input and default file names to consist of the following four components:-

* A drive designator, consisting of one letter followed by a colon (':').
* A path specification, consisting of all the characters after the drive designator, if any, up to the last backslash ('\') or forward slash ('/') in the string. Reminder: the backslash ('\') is the escape character in C.
* A file name consisting of all the characters following the path specification, if any, up to a period, or the end of the string.
* A file extent, consisting of all the characters after the period terminating the file name up to the end of the string.

If the input file name contains a component, then that component is copied to the result string. If the default string contains a component that is not present in the input string, then the default component becomes part of the result string. If a component is missing from both input and default strings, then it is also missing from the result string.

In addition, each part of the path name is truncated to the first eight characters, the file name to the first eight characters, and the extent to the first three.

3.76.3. Returns

The address of the null terminated result string.

3.76.4. Notes

This function does not cope with device names, such as "con:" or "lpt:". So check that you don't use them.

This function can be used to supply default components of file names, or to force components of file names to specific values.

This function is not a standard C langauge function.

3.76.5. Example

To set a default extension of ".c" on a user suppiled file name:-

```
makefnam(userfile,".c",resfile);
```

To force a drive of "b:" and an extension of ".c" on a user supplied file name:-

```
makefnam("b:.c",userfile,resfile);
```

3.76.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.76.7. Use with

open, fopen, rename, unlink

3.77. malloc, Allocate uninitialized memory from the heap

3.77.1. Synopsis

```
char *malloc(size)
unsigned size;              /* number of bytes needed */
```

3.77.2. Function

Obtains a region of size bytes from the heap. The region is not initialized.

3.77.3. Returns

The address of the allocated region, or zero (NULL) if not enough memory was available.

3.77.4. Notes

Using the big memory model, blocks of up to 65516 (0xFFE8) bytes may be requested. In the bigmodel, the default amount of memory available is about 96K for the heap and the stack. This can be changed (either increased to access all of memory on your machine or decreased to leave more unused memory) by editing the file _default.c. We have found that most users of C86 can live with about 96K of heap and stack space in the bigmodel.

It is very important that this function is declared in the big model as returning a pointer to a character. Undefined results will occur if this function is not declared.

3.77.5. Example

```
{
  extern char *malloc();        /* important in big model!! */
  extern int free();
  char *buf;
  unsigned number_bytes;
  int i;

        number_bytes = 255;
        buf = malloc(number_bytes);     /* not initialized */
        /* you can initialize yourself if neccessary */
        for(i=0;i<number_bytes;i++)
                buf[i] = EOS;           /* EOS = '\0' */
        /* use buf */
        free(buf);
}
```

3.77.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.77.7. See also

alloc, calloc, realloc, sbrk, free, coreleft

3.78. mkdir, Make a new subdirectory.

3.78.1. Synopsis

```
int mkdir(pathname)
char *pathname;
```

3.78.2. Function

Calls the operating system to make a new subdirectory in the current working directory. The path name for the new directory is essentially limited to a single eight character name, and is operating system dependent. You should refer to your operating system documentation for more information.

3.78.3. Returns

EOF if an error is detected, otherwise zero.

3.78.4. Notes

This function is only available for DOS V2.0+.

3.78.5. Operating System

DOS 3.0, DOS 2.0+

3.78.6. Use with

chdir, rmdir

3.79. modf, Split double into integer and fraction.

3.79.1. Synopsis

```
double modf(val,iptr)
double val;                /* the input value */
double *iptr;              /* where to put the result */
```

3.79.2. Function

Stores the integer part of "val" indirectly through the pointer "iptr", and returns the fractional part. The fractional part is always greater than or equal to zero.

3.79.3. Example

```
{
  extern double modf(); /* double precision mod function */
  double val;
  double int_part;
  double frac;

        val = 1234.5555;
        frac = modf(val,&int_part);

        /* frac contains .5555 , int_part 1234 */
        printf("\nMODF\n%g = modf(%g,%g)\n",frac,val,int_part);

        val = -19.812;
        frac = modf(val,&int_part);

        /* frac contains .812, int_part -19 */
        printf("\n%g = modf(%g,%g)\n",frac,val,int_part);
}
```

3.79.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.79.5. See also

frexp, ldexp

3.80. movblock, Move a block of memory.

3.80.1. Synopsis

```
int movblock(soffset,sseg,doffset,dseg,count)
unsigned soffset;          /* source offset relative to */
unsigned sseg;             /* the source segment */
unsigned doffset;          /* destination offset relative to */
unsigned dseg;             /* the destination segment */
unsigned count;            /* number of bytes to move */
```

3.80.2. Function

Moves a block of memory from anywhere in memory to anywhere in memory. The source and destination addresses are specified by standard offset/segment double word values.

Up to 64000 bytes may be moved.

If the move is for exactly 2 words, interrupts are disabled. This allows you to move data to an interrupt vector address.

In a big model program, soffset and sseg may be supplied by ONE pointer. Likewise doffset and dseg.

3.80.3. Returns

Nothing.

3.80.4. Notes

This routine is intended for moving data to/from memory that is "outside" a program.

DOS

DOS provides a system call for setting interrupt vectors. It should be used instead of movblock.

3.80.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.80.6. See also

movmem

3.81. movmem, Move memory within a program.

3.81.1. Synopsis

```
int movmem(source,dest,count)
char *source,*dest;
unsigned int count;
```

3.81.2. Function

Copies a block of memory, count bytes in length, starting at address source to the area starting at address dest. This routine is written so that a valid copy will be made even if the source and destination regions overlap.

3.81.3. Returns

Nothing.

3.81.4. Notes

This routine is intended for moving data "within" a program. In a big model program it provides the same capabilities as movblock.

3.81.5. Example

```
{
  extern int movmem(), free();
  extern char *calloc();
  unsigned int byte_count;
  char *srce, *dest, *tmp;

     byte_count = 100;
     srce = calloc(byte_count,1);
     strcpy(srce,"Source string Written Here");
     dest = calloc(byte_count,sizeof(char));
     strcpy(dest,"Destination string Here");
     tmp = calloc(byte_count,sizeof(char));

     printf("\nBEFORE MOVMEM\n");
     printf("srce : \"%s\"\ndest : \"%s\"\ntmp : \"%s\"\n"
          ,srce, dest, tmp);

     movmem(srce,tmp,byte_count);
     printf("\nsrce : \"%s\"\ndest : \"%s\"\ntmp : \"%s\"\n"
          ,srce, dest, tmp);

     movmem(dest,srce,byte_count);
     printf("\nsrce : \"%s\"\ndest : \"%s\"\ntmp : \"%s\"\n"
          ,srce, dest, tmp);
     movmem(tmp,dest,byte_count);

     printf("\nAFTER MOVMEM\n");
     printf("srce : \"%s\"\ndest : \"%s\"\ntmp : \"%s\"\n"
          ,srce, dest, tmp);

     free(tmp); free(srce); free(dest);
}
```

3.81.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.81.7. See also

setmem, movblock

3.82. open, Open an existing file.

3.82.1. Synopsis

```
int open(filename,mode)
char *filename;
unsigned int mode;
```

3.82.2. Function

Filename must be a valid file name for your operating system or one of the special names defined below. If a file of that name currently exists it is opened, otherwise an error is returned. Valid open modes are defined in "stdio.h", and have the values:-

AREAD	0	open for ASCII read
AWRITE	1	open for ASCII write
AUPDATE	2	open for ASCII update
BREAD	4	open for binary read
BWRITE	5	open for binary write
BUPDATE	6	open for binary update

3.82.3. Returns

- A negative number if any error is detected.
- A positive number (a file descriptor) if successful.

3.82.4. Notes

Opening a file in ASCII mode means:-

- Carriage return/linefeed pairs in the file will be converted to newlines ('\n') on input.
- Newlines will be converted to carriage return/linefeed pairs on output.
- Control-z in the file will be returned as end of file on input.

No conversion is performed on files opened in binary mode.

You must use the function creat to creat a new file. Unlike the function fopen, this function will not creat a file for you, it must be done with another function call.

3.82.5. DOS

The following special file names are supported:-

```
     - To open the console    "CON:"
     - To open the printer    "PRN:"
     - To open the com device "AUX:"
```

3.82.6. Example

```
{
  extern int open();
  extern int close();
  char *filename;
  unsigned int mode;
  int fd;

        mode = AREAD;
        filename = "a:filename.ext";
        fd = open(filename,mode);
        if(fd<0) printf("\nError opening %s\n",filename);
        else  close(fd);

        mode = AWRITE;
        filename = "CON:";      /* write to console */
        fd = open(filename,AWRITE);
        if(fd<0) printf("\nError opening console for write\n");
        else close(fd);

        mode = BREAD;
        filename = "a:filename.dat";
        fd = open(filename,mode);
        /* open in binary mode, no CRLF or CTRLZ conversion */
        if(fd>=0) close(fd);
}
```

3.82.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.82.8. Use with

creat, read, write, close, lseek, ltell

3.82.9. See also

fopen, fclose

3.83. outportb, outportw - Output a byte or word to a port.

3.83.1. Synopsis

```
unsigned char outportb(portno,value)
unsigned int portno;
char value;

int outportw(portno,value)
int portno;
int value;
```

3.83.2. Function

Outportb outputs a byte to a user supplied port number (portno). The port number must be valid for the addressed device. In some cases a 16 bit port number is required. For older devices an 8 bit number is required, and it may have to be in either byte of portno. One possibility is to place the port number in both upper and lower bytes of portno. It returns the byte read from the port.

Outportw outputs a word to a user supplied port number (portno). The port number must be valid for the addressed device. Usually a 16 bit port number is required. This function is not needed for most devices currently available, as they do not support 16 bit i/o transfers. It returns the word output from the port.

3.83.3. Example

```
* outportb example

{
  extern unsigned char outportb();
  unsigned int portno;
  char byte_value;

        byte_value = 0x07;
        portno = 0;
        outportb(portno,byte_value);

}

* outportw example

{
  extern unsigned int outportw();
  unsigned int portno, word_value;

        word_value = 0xffaa;
        portno = 1;
        outportw(portno,word_value);
}
```

3.83.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.83.5. See also

inportw, inportb, outportw

3.84. peek, Examine the content of a word in memory

3.84.1. Synopsis

```
int peek(offset,seg)
unsigned offset;          /* offset of the word relative to */
unsigned seg;             /* a segment register value */
```

3.84.2. Function

Get the content of a word anywhere in memory. The offset/seg is a standard double word pointer. In the big model a regular pointer may be used.

3.84.3. Returns

The content of the requested word.

3.84.4. Notes

Sometimes you may want to poke and peek inside the memory that the program is loaded into. To find out where the program is loaded use the function segread().

3.84.5. Example

```
{
  extern int peek();
  unsigned offset;
  unsigned segment;
  int word;
  int i;

     segment = 0x100;
     putchar('\n');

     for(offset=0;offset<20;offset++)
          {
          word = peek(offset,segment);
          printf("peek(%xH,%xH) = %d\n",offset,segment,word);
          }
}
```

3.84.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.84.7. See also

pokeb, pokew, segread

3.85. poke, Store data in memory.

3.85.1. Synopsis

```
pokeb(offset,seg,byte);        /* poke a byte */
unsigned offset;                /* the offset relative to */
unsigned seg;                   /* the segment value */
char byte;                     /* the value to poke */

pokew(offset,seg,word);        /* poke a word */
unsigned offset;               /* the offset relative to */
unsigned seg;                   /* the segment value */
unsigned word;                 /* the value to poke */
```

3.85.2. Function

Put a word or byte at the specified offset/seg address in memory. The offset/seg pair are a standard double word pointer.

3.85.3. Returns

The value poked.

3.85.4. Notes

Sometimes you may want to poke and peek inside the memory that the program is loaded into. To find out where the program is loaded use the function segread().

3.85.5. Example

```
{
  extern unsigned char pokeb();
  extern unsigned int pokew();
  unsigned offset;
  unsigned segment;
  char byte;
  unsigned word;

              segment = 0x200;
              offset = 0;

              byte = peek(offset,segment) & 0xff;
              pokeb(offset,segment,byte);

              word = peek(offset,segment);
              pokew(offset,segment,word);
}
```

3.85.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.85.7. See also

peek, movblock, segread

3.86. pow, Return X to the power Y.

3.86.1. Synopsis

```
double pow(x,y)
double x,y;
```

3.86.2. Returns

The value of x raised to the power y.

3.86.3. Notes

Returns zero if both x and y are zero.

Returns 1e+300 if the result would overflow.

3.86.4. Example

```
{
  extern double pow();  /* returns base to the exponent */
  double base, exp;
  double result;

        base = 2.7182818;
        exp = -2.5;

        result = pow(base,exp);

        /* result contains 0.08208.... */
        printf("\npow(%g,%g) = %g\n",base,exp,result);

        base  = 10.0;
        exp = 6.0;

        result = pow(base,exp);

        /* result contains 1000000.0 */
        printf("\npow(%g,%g) = %g\n",base,exp,result);
}
```

3.86.5. See also

exp, log, log10, sqrt

3.86.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.87. printf, Print to stdout.

3.87.1. Synopsis

```
int printf(format,args...)
char *format;
see below for args;
```

3.87.2. Function

Output data under control of a format string to file stdout.

The output file is stdout, which is normally open to the console. However, it is subject to redirection, and may be open to a disk file or the line printer. The file should be open in ASCII mode.

See fprintf for additional information.

3.87.3. Returns

Nothing.

3.87.4. Example

For examples of the format control string, see Kernighan and Ritchie.

3.87.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.87.6. Use with

fprintf, fopen, fclose

3.88. Z-100 PC printer functions (Z-100 PC ONLY!)

prt_busy, prt_err, prt_putc, prt_rst, prt_scr, prt_stat

3.88.1. Synopsis

```
int prt_busy(printer)
int printer;    /* 0,1,2 */

int prt_err(printer)
int printer;              /* 0,1,2 */

int prt_putc(printer,character)
int printer;                    /* 0,1,2 */
char character;                 /* the character to print */

int prt_rst(printer)
int printer;                    /* 0,1,2 */

prt_scr()

int prt_stat(printer)
int printer;
```

3.88.2. Function

prt_busy checks the printer status to see if it is busy. Returns 1 if it is busy, 0 if it is not.

prt_err checks the printer status to see if device is off-line, timed out, paper out or i/o error has occurred. The parameter "printer" can be either 0, 1, or 2. Returns 1 if an error has occured, 0 if it has not.

prt_putc attempts to print a character to the printer. Valid printers are 0 through 2. Returns the printer status byte as described in the notes.

prt_rst initializes the printer port and returns the printer status byte.

prt_scr prints the screen to the printer on the Z-100 PC.

prt_stat will return the current status of the printer (see notes).

3.88.3. Notes

The printer status byte is organized as follows:

Bit	Mask	Meaning
0	0x01	Timeout Occurred
1	0x02	[Unused]
2	0x04	[Unused]
3	0x08	I/O error
4	0x10	selected
5	0x20	out of paper
6	0x40	acknowledge
7	0x80	busy

3.88.4. Example

*   This example displays the status of the printer

```
{
int stat;                /* display status of the printer */
int printer;

   printer = 0;
   stat = prt_stat(printer);

   printf("\nPrinter status:\n");
   printf("Timeout:       %d\n",(stat & 0x01) != 0);
   printf("I/O Error:     %d\n",(stat & 0x08) != 0);
   printf("Selected:      %d\n",(stat & 0x10) != 0);
   printf("Out of paper:  %d\n",(stat & 0x20) != 0);
   printf("Acknowledge:   %d\n",(stat & 0x40) != 0);
   printf("Busy:          %d\n",(stat & 0x80) != 0);
}
```

*   To print from stdin to the printer until EOF. This is a simple example and does not check for paper out, timeout, etc.

```
{
int ch;
int printer;

  printer = 0;
  prt_rst(printer);
  while((ch=getchar())!=EOF)
      {
      if(prt_err(printer))
         abort("printer error: status = %x\n",prt_stat(printer));
      while(prt_busy(printer)) printf("printer busy\n");
      if(ch == '\n') prt_putc(printer,'\r');
      prt_putc(printer,ch);
      }
}
```

3.88.5. Operating System

MS DOS 3.0, MS DOS 2.0+, MS DOS 1.1+

3.89. ptrtoabs, Convert a pointer to an absolute address.

3.89.1. Synopsis

```
long ptrtoabs(address)
char *address;
```

3.89.2. Function

Convert a long pointer to an absolute 20 bit memory address.

3.89.3. Returns

The absolute value corresponding to the supplied BIG MEMORY MODEL POINTER.

3.89.4. Notes

This function is supplied for use with big memory model pointers ONLY. It can be used to compare pointers that may be in different segment spaces.

If you must compare pointers that could be in different segments in the big model you must use this function. Comparison of two pointers in the big model without using this function will assume that the segments are the same for the two pointers. WARNING: This function is non-portable and should not be used if at all possible. Also, The definition of how pointer difference works in C86 may change in future releases.

A machine pointer consists of two words, an offset followed by a segment value. This function calculates the absolute value by the formula:-

```
          abs_val = segment * 16 + offset;
```

3.89.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.89.6. See also

abstoptr

3.90. putc, Output a character to a stream.

3.90.1. Synopsis

```
#include "stdio.h"

int putc(cc,stream)
char cc;                    /* the character to write */
FILE *stream;               /* where to write it */
```

3.90.2. Function

Outputs the character to the stream. Conversion of newlines will take place if the file was opened in ASCII mode.

3.90.3. Returns

The character cc.

3.90.4. Notes

This is defined with a macro in stdio.h, so that "stdio.h" must be included in your source program. The function fputc is actually used to output the data.

3.90.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.90.6. Use with

putchar, getchar, fopen, fclose, fputc, fgetc

3.91. putchar, Output a character to stdout.

3.91.1. Synopsis

```
#include "stdio.h"

int putchar(c)
char c;
```

3.91.2. Function

Outputs the character to stdout. This file is normally assigned to the console.

3.91.3. Notes

Defined in stdio.h to be equivalent to fputc(c,stdout);

3.91.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.91.5. Use with

putc, getc, getchar, fopen, fclose

3.92. puts, Output a string to a stdout.

3.92.1. Synopsis

```
int puts(string)
char *string;              /* the data to write */
```

3.92.2. Function

Writes the null terminated string, and then a newline to the standard output stream, stdout.

3.92.3. Returns

Zero if no error detected, otherwise -1.

3.92.4. Notes

Puts appends a newline, fputs does not. That's what UNIX does too.

3.92.5. See also

fopen, gets, putc, printf, ferror, fputs

3.92.6. Example

```
#include "stdio.h"

{
  extern int puts();     /* write a string to a stdout */
  char *string;
  int res;

        string = "This is a string of characters";
        printf("\ncalling puts\n");

        res = puts(string);

        /* writes the string and then a '\n' to stdout */
        /* res contains -1 if error detected */
}
```

3.92.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.93. putw, Output a word to a stream.

3.93.1. Synopsis

```
int putw(w,stream)
int w;
FILE *stream;
```

3.93.2. Function

Writes the word "w" to the specified stream. If the file is open in ASCII mode, newline translation will be performed on each of the two characters in the word. The least significant byte of the word is written first.

3.93.3. Returns

The word itself, or -1 if an error was detected.

3.93.4. Example

```
#include "stdio.h"

{
  extern int putw();
  int w;
  FILE *stream;
  int result;

         w = 'ab';

         stream = stderr;

         result = putw(w,stream);

         /* this will write 'ab' to stream */
         /* result contains -1 if error detected */
}
```

3.93.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.93.6. Use with

getw, putc, fopen, fclose

3.94. qsort, Sort an array of records in memory

3.94.1. Synopsis

```
qsort(array,number,width,cmpf)
char *array;            /* address of array of data to be sorted */
unsigned number;        /* number of entries in the array */
unsigned width;         /* width of an entry in bytes */
int (*cmpf)();          /* comparison function */
```

3.94.2. Function

Sorts an array containing "number" entries each of width "width" bytes using Hoare's Quicksort algorithm.

The comparison function is called with two pointers to entries in the array. It must compare the two entries and return the following values:-

-1	first<second
0	first==second
+1	first>second

3.94.3. Returns

Nothing

3.94.4. Notes

This routine will abort if it runs out of working space. Working space may be adjusted by a recompilation.

3.94.5. Example

Read a series of lines from stdin, sort into ascending sequence, and output to stdout.

This program can be used to sort an ascii file into ascending sequence. It reads it's input from stdin, and outputs to stdout.

3.94.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

```
/*   sort lines from stdin into ascending sequence
*/

#include "stdio.h"

#define MAXLINES 1000
unsigned char *line[MAXLINES];

extern unsigned char *alloc();

comp(a,b)                    /* compare two for the sort */
unsigned char **a,**b;
{

  return strcmp(*a,*b);
}

main()
{
  int j,k;
  unsigned char buffer[132];

  for(j=0;j<MAXLINES;++j){
    if(!fgets(buffer,130,stdin))break;          /* all input */
    line[j]=alloc(strlen(buffer)+1);
    strcpy(line[j],buffer);
  }
  qsort(line,j,sizeof(unsigned char *),comp);
  for(k=0;k<j;++k)printf("%s",line[k]);
}
```

3.94.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.95. read, Read characters from a file.

3.95.1. Synopsis

```
int read(fd,buffer,count)
unsigned int fd;
char *buffer;
unsigned int count;
```

3.95.2. Function

Read up to count characters from the file specified by the file descriptor fd. If the file is open in ASCII mode, newline and end of file processing will be performed.

3.95.3. Returns

This function returns the number of characters placed in the buffer. This will be the equal to count unless an end of file is detected, in which case a short record may be returned.

A returned value of zero indicates end of file, a minus one that indicates an error was detected.

This function stops reading at an end of line in an ASCII file.

3.95.4. Notes

This is the main input procedure for the DOSALL library. All other input procedures call read for their data.

3.95.5. Example

```
{
  extern int read();
  extern int open();
  extern int close();
  extern char *alloc();
  extern int free();
  unsigned int fd;
  char *buffer;
  unsigned int bytecount;
  int num_read, i;

     bytecount = 255;

     fd = open("a:filename.dat",AREAD);
     if(fd<0) { fputs("file not opened",stdout); return; }

     buffer = alloc(bytecount+1);
     for(i=0;i<bytecount;i++) *(buffer+i)='\0';

     num_read = read(fd,buffer,bytecount);
     /* num_read contains:
          if -1 error
          if  0 EOF
          if >0 number of characters put in buffer */

     close(fd);
     free(buffer);
     return;
}
```

3.95.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.95.7. Use with

write, open, creat

3.96. realloc, Change size of a heap area.

3.96.1. Synopsis

```
char *realloc(oldp,size)
char *oldp;                /* the address of the old region */
unsigned size;             /* the required size */
```

3.96.2. Function

Increases or decreases the size of a block of memory in the heap to "size" bytes, preserving the content of the beginning of the block.

The new block may be at a different address from the original block. The content of the block is preserved up to the size of the smaller of the sizes of the new or old blocks.

As a non-standard extension, if the address of the old region is zero, this function is equivalent to the function malloc().

3.96.3. Returns

The address of the new block, or zero if no block of the required size could be allocated. If zero is returned, the original data has been destroyed.

3.96.4. Example

```
{
  extern char *realloc();
  extern char *alloc();
  extern int free();
  char *ptr;
  unsigned size;

                size = 100;
                ptr = alloc(size);
                strcpy(ptr,"this is initial data.");
                size = 200;
                ptr = realloc(ptr,size);

                strcat(ptr,"this is added later");
                printf("\n%s\n",ptr);
                /* more space gets allocated for ptr
                   without disturbing the present contents. */
                free(ptr);    /* release memory */
}
```

3.96.5. See also

alloc, calloc, malloc, sbrk, free, coreleft

3.96.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.97. rename, Change the name of a file.

3.97.1. Synopsis

```
int rename(from,to)
char *from,*to;
```

3.97.2. Function

Change the name of a disk file from "from" to "to".

3.97.3. Returns

- zero if the rename was successful
- A negative number if an error was detected.

3.97.4. Notes

This function uses the operating system rename function. Both file names should specify the same disk drive, and neither should include asterisks or question marks.

The file must not be open when this function is called.

If executed under DOS 2.0+, both names may contain identical path information.

3.97.5. Example

```
{
  extern int rename();
  char *present, *newname;
  int ret_code;

     present = "a:oldfile.ext";
     newname = "a:newfile.new";
     ret_code = rename(present,newname);
     /* ret_code contains zero if successful */
     /* DO NOT USE '?'s IN EITHER NAME */
     printf("\n%s %s renamed to %s\n",present,
          ret_code==0 ? "was" : "was not",newname);
     return;
}
```

3.97.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.98. rewind, position to the beginning of an open file

3.98.1. Synopsis

```
long rewind(stream)
FILE *stream;
```

3.98.2. Function

Attempt to position the file pointer associated with the stream to the beginning of the file.

3.98.3. Returns

0L if successful and a negative number otherwise.

3.98.4. Notes

This is equivalent to fseek(stream,0L,0);

3.98.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.98.6. See also

fseek

3.99. rindex, Reverse index search

3.99.1. Synopsis

```
char *rindex(string,cc)
char *string;             /* the string to search */
char cc;                  /* the character to find */
```

3.99.2. Function

Find the last occurrence of the character cc in the string.

3.99.3. Returns

Zero if the character was not found, else a pointer to the character in the string.

3.99.4. Notes

The function strrchr() is the same as this function. You should use the strrchr() function to be more UNIX v5.0 compatible.

3.99.5. Example

```
{
  extern char *rindex();
  char *string;
  char ch;
  char *result;

                  string = "this is a string of data";
                  ch = 't';

                  result = rindex(string,ch);
                  /* result contains 0 if not found */
                  /* in this case result points to the 't'
                     in data near the end of the string */

                  result = rindex("1234567890",'A');
                  /* in this case result contains 0 */
                  return;
}
```

3.99.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.99.7. See also

string functions, index

3.100. rmdir, Remove a specified directory.

3.100.1. Synopsis

```
int rmdir(pathname)
char *pathname;
```

3.100.2. Function

Calls the operating system to remove (delete) the specified directory. The path name must be reachable from the current working directory, and the directory must be empty. The path name is operating system dependent. You should refer to your operating system documentation for more information.

3.100.3. Returns

EOF if an error is detected, otherwise zero.

3.100.4. Notes

This function is only available for DOS V2.0+.

3.100.5. Operating System

DOS 3.0, DOS 2.0+

3.100.6. Use with

chdir, mkdir

3.101. sbrk, Request memory at string break.

3.101.1. Synopsis

```
char *sbrk(size)
unsigned int size;
```

3.101.2. Function

Return the address of a region of length "size" bytes.

3.101.3. Returns

The address of a region of the required length, or zero if none is available.

3.101.4. Notes

This function knows about the way memory is allocated by the linker. It uses a magic cell to maintain the address of the next free block of memory, and checks that allocating this region will not overwrite the machine stack.

It also knows about another word that specifies the minimum amount of memory that must exist between the string break and the bottom of the machine stack.

Don't fiddle in this routine.

The function malloc calls sbrk when there is not enough memory in the free list.

The maximum amount of memory that can be allocated by a single call is about 0xffe8 bytes.

WARNING: If you get memory with the sbrk() function you cannot free() or realloc() it.

3.101.5. Example

```
{
  extern char *sbrk();
  extern int fputs();
  unsigned int size;
  char *ptr;

      size = 100; ptr = sbrk(size);
      if(ptr == 0)
           {
           fputs("No space available at string break\n",stdout);
           return;
           }

      /* ptr points to an area 100 bytes long if it was available,
         otherwise, ptr contains 0 */

      strcpy(ptr,"SBRK: sample data");
      strcat(ptr," can be added\n");
      fputs(ptr,stdout);
}
```

3.101.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.101.7. See also

malloc, calloc, realloc, alloc, free

3.102. scanf, Scan fields from stdin

3.102.1. Synopsis

```
int scanf(format,args)
char *format;              /* the input conversion control */
something *args;           /* pointers for result locations */
```

3.102.2. Function

Reads input from stdin and converts under control of a format string.

3.102.3. Returns

The number of arguments successfully matched and stored. EOF is returned at EOF or if an error is detected.

3.102.4. Notes

This function should not be used when dealing with human inputs. You should use the fgets or gets function to enter a human input line to a buffer and then format the buffer using sscanf. This function and fscanf should be used for machine formatted inputs that have a definite form and when a high degree of error checking is not needed. Error recovery from sscanf is much easier than scanf's.

Scanf should not be used along with the function getchar (and it's related functions). The input stream stdin could get very confused if you mix scanf and other input functions.

Scanf returns the number of arguments matched and stored. As an example, the format control string "%d/%d/%d" would return 5 on a successful scan of the input. It would match the two literal slashes ('/') and the three integers.

This is equivalent to the call:-

```
     fscanf(stdin,format,args);
```

See fscanf for details of the format control string.

See sscanf for other information on this use of this function.

3.102.5. Example

```
{
  extern int scanf();
  int num_cvt, i, num[3];

        fputs("\nSCANF:\nType three integer values ",stdout);

        /* the addresses must be provided for arguments */
        num_cvt = scanf("%d %d %d",&num[0],&num[1],&num[2]);

        for(i=0;i<num_cvt;i++)
                printf("\n%d*%d=%d\n",
                        num[i],num[i],num[i]*num[i]);

        /* calling scanf(format,args) is equivalent to calling
           fscanf(stdin,format,args). See fscanf for details. */
}
```

3.102.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.102.7. See also

fscanf, sscanf

3.103. segread, Read the segment registers.

3.103.1. Synopsis

```
int segread(rv)
struct {int scs,sss,sds,ses;} *rv; /* reg save area pointer */
```

3.103.2. Function

Reads the segment registers of the CALLING function and places the values into the four words of the structure. For use with the small model only.

3.103.3. Returns

Nothing.

3.103.4. Notes

This handy function is designed to provide information needed by various machine dependent functions. The definition is set up so that no changes are needed to use it with the big model.

When we converted our library code to run under the big model, we eliminated the use of this function in almost every case. If you are compiling under the big model, and you are still using segread, then please examine your code carefully. It is almost guaranteed to be wrong.

In the big model the data segment is included in the body of the pointer. To get the segment in the big model you need to use the following construct:-

```
     unsigned int seg,off;     /* big model */
     char *p;

     seg = ((unsigned long)p)>>16;
     offset = p;
```

3.103.5. Example

```
{
  extern int segread();

  struct { int scs, sss, sds, ses; } rrv;
  unsigned int code_segment,    data_segment,
               stack_segment,   extra_segment;

        segread(&rrv);
        /* reads the segment registers of the calling function */

        code_segment = rrv.scs;
        stack_segment = rrv.sss;
        data_segment = rrv.sds;
        extra_segment = rrv.ses;

        printf("\n8086 SEGMENT REGISTERS:\n");
        printf("\nCS: %4x\nDS: %4x\nSS: %4x\nES: %4x\n",
                code_segment, data_segment,
                stack_segment, extra_segment);
}
```

3.103.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.103.7. See also

sysint, movblock, peek, poke

3.104. setjmp, Save the environment for longjmp.

3.104.1. Synopsis

```
#include "stdio.h"        /* to define jmp_buf */

int setjmp(envp)
jmp_buf *envp;            /* where to save the environment */
```

3.104.2. Function

Saves the current environment in the memory area pointed to by the envp for subsequent use by longjmp().

3.104.3. Returns

This function returns zero itself, but if longjmp is executed it appears to return the value passed to longjmp.

3.104.4. Notes

This is a very dangerous function. You have been warned.

The typedef of jmp_buf occurs in stdio.h.

The purpose of setjmp and longjmp is to allow you to terminate a block of code, and return to a previous point in your code with an error value. It enables you to avoid long sequences of error returns, and to "fail" up a series of functions.

When called this function saves various machine status values in the environment buffer and returns zero.

After calling setjmp, you can call longjmp later in the same function, or in functions called by the same function. After exiting the function that called setjmp, you may no longer use the environment in longjmp.

Note that variables in the function that calls setjmp are not restored to their values at the time setjmp was called but will have the values in them at the time longjmp was called.

Test these functions so that you understand them before depending on them.

See the following example.

3.104.5. Example

```
#include "stdio.h"
jmp_buf environment;
main()
{
  int error_code;

  error_code=setjmp(environment);
  if(error_code!=0){          /* must be a restore via longjmp */
    ..                        /* do error thing */
  }
  ...
  ...
  another_function();
  ...
}

another_function()
{

  /* have detected an error so get out */

  longjmp(environment,error_value);    /*restore environment */
  /* longjmp never returns, so can't get here */
}
```

3.104.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.104.7. Use with

longjmp

3.105. setmem, Set memory to a byte value.

3.105.1. Synopsis

```
int setmem(address,count,value)
char *address;
unsigned int count;
char value;
```

3.105.2. Function

Set bytes of memory in the range address through (address+count-1) to the value "value". This function is frequently used to zero blocks of memory.

3.105.3. Returns

Nothing.

3.105.4. Example

```
{
  extern int setmem();  /* set block of memory to a value */
  extern char *alloc();
  extern int free();
  char *address;
  unsigned int count;
  char value;

        count = 255;
        address = alloc(count);
        value = '\0';

        setmem(address,count,value);

        /* use address */

        free(address);
}
```

3.105.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.106. sprintf, Print to a string in memory

3.106.1. Synopsis

```
int sprintf(string,format,args)
char *string;              /* where to put results */
char *format;              /* the format control string */
something args;            /* optional data to be converted */
```

3.106.2. Function

Using the arguments (if any) under control of the format string, create a string in memory containing the converted data.

3.106.3. Returns

Nothing.

3.106.4. Notes

The conversions are as described under the function fprintf. The output of the conversion is placed in the "string", which is terminated by a NULL('\0').

The area of memory reserved for the string must be long enough for the result, as no checks are performed.

3.106.5. Example

```
{
  extern int sprintf();
  extern char *calloc();
  extern int free();
  extern int fputs();
  char *destination;

               destination = calloc(255,1);
               sprintf(destination,
                       "%4d %4d %4d",
                       45, 123, 50);
               /* the string destination will contain
                  the same data as a file would if
                  fprintf were called. */

               fputs(destination,stdout);
               free(destination);
}
```

3.106.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.106.7. See also

fprintf, sscanf, fscanf

3.107. sqrt, Square root.

3.107.1. Synopsis

```
double sqrt(val)
double val;
```

3.107.2. Function

Returns the square root of the number val.

3.107.3. Notes

Returns zero if val is negative.

3.107.4. Example

```
{
  extern double sqrt();
  double dval;
  double rval;

  dval = 45.00;
  rval = sqrt(dval);
  /* rval contains the square root of 45.00 */
  printf("The square root of %g = %g\n",dval,rval);

  dval = -10.0;
  rval = sqrt(dval);
  /* rval contains 0 because dval is < 0 */
  return;
}
```

3.107.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.108. sscanf, Scan fields from a string.

3.108.1. Synopsis

```
int sscanf(string,format,args)
char *string;             /* String contains the input data */
char *format;             /* the input conversion control */
something *args;          /* pointers for result locations */
```

3.108.2. Function

Reads input from the ASCII string and converts under control of a format string.

3.108.3. Returns

The number of arguments successfully matched and stored. You cannot read past the end of the input data string.

3.108.4. Notes

This is similar to the call:-

```
     fscanf(stream,format,args);
```

except that data is read from a string in memory instead of an input file.

This function may be used to re-scan input data using more than one format.

See fscanf for details of the format control string.

See scanf for other important information on this function.

3.108.5. Example

```
#include "stdio.h"

{
  extern int sscanf();
  extern char *calloc();
  extern int free();
  extern FILE *fopen();
  extern int fclose();
  extern char *fgets();
  char *buffer;
  char *bufstart;
  FILE *input;
  char *s[12];
  int i, j;

        fputs("\nSSCANF:\nType in strings separated by spaces\n"
               ,stdout);

        input = fopen("CON:","r");
        buffer = calloc(255,1);
        for(i=0;i<10;++i) s[i] = calloc(100,1);
        fgets(buffer,255,input);

        bufstart = buffer;
        for(i=0;i<10;++i)
                {
                if(sscanf(buffer,"%s",s[i])<1) break;
                if(strlen(buffer) > strlen(s[i])+1)
                        buffer += (strlen(s[i])+1);
                else break;
                }

        for(j=0;j<=i;++j)
                printf("\ts[%d] = %s\n",j,s[j]);

        for(i=0;i<10;++i) free(s[i]);
        free(bufstart);
        fclose(input);
        return;
}
```

3.108.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.108.7. See also

fscanf, scanf, sprintf, fprintf

3.109. String functions

strcat, strchr, strcmp, strcpy, strlen, strncat, strncmp, strncpy, strpbrk, strrchr

3.109.1. Synopsis

```
char *strcat(string1,string2)
char *string1,*string2;

unsigned char *strchr(s,c)
unsigned char *s,c;

int strcmp(string1,string2)
char *string1,*string2;

char *strcpy(to,from)
char *to,*from;

unsigned strlen(string)
char *string;

char *strncat(string1,string2,max)
char *string1,*string2;
unsigned max;

int strncmp(string1,string2,n)
char *string1,*string2;
unsigned n;

char *strncpy(to,from,n)
char *to,*from;
unsigned n;

unsigned char *strpbrk(s1,s2)
unsigned char *s1,*s2;

unsigned char *strrchr(s,c)
unsigned char *s,c;
```

3.109.2. Function

strcat appends a copy of string2 to the end of string1. It also returns a pointer to the first character of string1.

strchr finds the first occurrence of the character c in s and returns a pointer to it. **strrchr** finds the last occurrence of the character c in s and returns a pointer to it. Both functions will return a NULL if the character is not found.

strcmp compares the two strings, character by character, and returns an indication of which string is lower in the ASCII collating sequence. The following shows the result of strcmp:-

- Minus one	if string1 is less than string2
- Zero	if string1 is equal to string2
- Plus one	if string1 is greater than string2

strcpy makes a copy of the string at the address "from" in the buffer at address "to" and returns a pointer to the destination string.

strlen returns the length of the string. In C, character strings are terminated by the first byte with a value of binary zero.

strncat appends a copy of string2, or the first "max" characters of string2 (whichever is the smaller), to the end of string1. It also returns a pointer to the destination string.

strncmp compares the two strings, character by character, and returns an indication of which string is lower in the ASCII collating sequence. The comparison stops after n characters have been compared, or the end of a string has been detected. The returns of strncmp are the same as the returns of strcmp (see above).

strncpy makes a copy of the string at address "from" in the buffer at address "to". Copy at most "n" characters. If the input string is less than "n" characters in length, pad the remainder of the destination field with binary zeros. If the source contains "n" or more characters, the string "to" WILL NOT BE TERMINATED BY AN END OF STRING CHARACTER. This function returns the address of the destination string.

strpbrk returns a pointer to the first occurrence in string s1 of any character from string s2, or NULL if no character from s2 exists in s1.

3.109.3. Notes

WARNING: In the copy (strncpy and strcpy) and append (strcat and strncat) functions it is the user's responsibility to ensure that there is enough memory to hold the result. This routine cannot perform such checks.

THE DEFINITION OF STRNCPY WAS INCORRECT UNDER OUR RELEASE V1.33. The current definition conforms to UNIX conventions.

The functions strchr and strrchr are the UNIX names for C86's index and rindex. You should use these functions because they are more portable to UNIX systems.

3.109.4. Example

```
/* STRCAT example */

{
  extern char *strcat();
  extern char *calloc();
  extern int free(), fputs();
  char *s1, *s2, *s3, *outstr;

      outstr = calloc(255,1);

      s1 = "string #1 ";
      s2 = "string #2 ";
      s3 = "string #3 ";

      strcpy(outstr,"outstring: ");
      strcat(outstr,s1);
      strcat(outstr,s2);
      strcat(outstr,s3);
      fputs(outstr,stdout);

      free(outstr);
}


/* STRCMP example */

{
  extern int strcmp();
  char *string1, *string2;
  int result;

      string1 = "this";
      string2 = "the";
      result = strcmp(string1,string2);
      /* result contains +1 */

      result = strcmp("sample","sample string");
      /* result contains -1 */

      result = strcmp("a","z");
      /* result contains -1 */

      result = strcmp(string1,"this");
      /* result contains 0 */

      result = strcmp("a","at last");
      /* result contains -1 */
}
```

```
/* STRCPY example */

#define BLOCK 255

{
  extern char *strcpy();
  extern char *calloc(), *fgets();
  extern int free(), fputs();
  char *srce, *dest;

     srce = calloc(BLOCK,1);
     dest = calloc(BLOCK,sizeof(char));

     fgets(srce,BLOCK,stdin);
     strcpy(dest,srce);
     fputs(dest,stdout);

     free(srce);
     free(dest);
}
```

```
/* STRLEN example */

{
  extern int strlen();  /* returns length of string */
  unsigned length;
  char *string;

     string = "123456789";
     length = strlen(string);                /* length is 9 */

     length = strlen("a:filename.ext");      /* length is 14 */

     *string = '\0';
     length = strlen(string);                /* length is 0 */
}
```

```
/* STRNCAT example */

{
  extern char *strncat();
  extern char *alloc();
  extern int free();
  char *s1, *s2, *s3, *dest;

      dest = alloc(255);
      *dest = EOS;

      s1 = "number 1 ";
      s2 = "number 2 ";
      s3 = "number 3 ";

      strcpy(dest,"dest: ");
      strncat(dest,s1,3);   /* dest is: "dest: num" */

      strncat(dest,s2,5);   /* dest is: "dest: numnumbe" */

      strncat(dest,s3,255); /* dest: "dest: numnumbenumber 3 " */

      free(dest);
}
```

```
/* STRNCMP example */

{
  extern int strncmp(); /* compare strings up to a point */
  char *s1 , *s2; int result;

      s1 = "this is sample data";
      s2 = "this is not real data";

      result = strncmp(s1,s2,255);
      /* result is +1 because 's' > 'n' */

      result = strncmp(s1,s2,6);
      /* result is 0 because strings are the same for the first
         6 characters. */

      result = strncmp("first","second",3);
      /* result is -1 because 'f' < 's' */

      result = strncmp(s1,"this",4); /* result is 0 */

      result = strncmp("book","ballast",1);
      /* result is 0 because only the first character from each
         string is compared */
}
```

```
/* STRNCPY example */

{
  extern char *strncpy();
  char *calloc() , *srce, *dest;

     srce = calloc(255,1); dest = calloc(255,1);

     strncpy(srce,"this is sample data.",5);
     /* srce contains "this " */

     strncpy(dest,"also, on the hill were three",100);
     /* dest contains the entire string */

     strncpy(dest,srce,2); /* dest contains "th" */

     strncpy(dest,"versions of the code were",10);
     /* dest contains "versions o" */
}
```

3.109.5. See also

sprintf, sscanf

3.109.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.110. sysint, Execute an INT instruction.

3.110.1. Synopsis

```
struct regval { int ax,bx,cx,dx,si,di,ds,es; };

int sysint(vec,sreg,rreg)
unsigned char vec;                  /* interrupt to execute */
struct regval *sreg;                /* registers before int */
struct regval *rreg;                /* registers after int */
```

3.110.2. Function

Execute an INT instruction after setting registers to the values in the structure pointed to by sreg. The values in the registers after the instruction are placed into the struct pointed to by rreg before returning.

3.110.3. Returns

The value of the 8086 flag register after completion of the interrupt. The values used to test the returned status bits are:-

0x001	Carry flag.
0x002	not used.
0x004	Parity flag.
0x008	not used.
0x010	Auxiliary Carry flag.
0x020	not used.
0x040	Zero flag.
0x080	Sign flag.
0x100	Trap flag.
0x200	Interrupt Enable flag.
0x400	Direction flag.
0x800	Overflow flag.

3.110.4. Notes

Registers cs, ss, and bp cannot be set up using this function.

The structures for input and output may overlap, or be the same structure if desired.

This function can be used to request operating system actions that cannot be requested via the bdos call.

3.110.5. Example

The following is a function to get the system date and time using sysint.

```
/*
        getdate: get system date and time

        DOS 1.xx and 2.xx
*/
getdate(date)
int date[4];
{
struct regval { int ax,bx,cx,dx,si,di,ds,es; } srv;

  srv.ax = 0x2a00;
  sysint(0x21,&srv,&srv);
  date[0] = srv.cx;
  date[1] = srv.dx;

  srv.ax = 0x2c00;
  sysint(0x21,&srv,&srv);
  date[2] = srv.cx;
  date[3] = srv.dx;
}

/*
   print out the current date and time
*/
main()
{
int date[4];
int year,month,day,hour,minutes,seconds,hundredths;

   getdate(date);

   year = date[0];
   month = date[1] >> 8;
   day = date[1] & 0xff;
   hour = date[2] >> 8;
   minutes = date[2] & 0xff;
   seconds = date[3] >> 8;
   hundredths = date[3] & 0xff;

   printf("date: %2d/%02d/%4d %2d:%02d:%02d.%02d\n",
        month,day,year,hour,minutes,seconds,hundredths);
}
```

3.110.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.110.7. See also

sysint21, bdos, segread

3.111. sysint21, Execute an INT 21H instruction.

3.111.1. Synopsis

```
struct regval { int ax,bx,cx,dx,si,di,ds,es; };

int sysint21(sreg,rreg)
struct regval *sreg;                /* registers before int */
struct regval *rreg;                /* registers after int */
```

3.111.2. Function

Execute an INT 21H instruction after setting registers to the values in the structure pointed to by sreg. The values in the registers after the instruction are placed into the struct pointed to by rreg before returning.

3.111.3. Returns

The value of the 8086 flag register after completion of the interrupt. The values used to test the returned status bits are:-

0x001	Carry flag.
0x002	not used.
0x004	Parity flag.
0x008	not used.
0x010	Auxiliary Carry flag.
0x020	not used.
0x040	Zero flag.
0x080	Sign flag.
0x100	Trap flag.
0x200	Interrupt Enable flag.
0x400	Direction flag.
0x800	Overflow flag.

3.111.4. Notes

Registers cs, ss, and bp cannot be set up using this function.

The structures for input and output may overlap, or be the same structure if desired.

This function can be used to request operating system actions that cannot be requested via the bdos call.

This function is somewhat more efficient for the major calls to the DOS operating system.

3.111.5. Example

The following is a function to get the system date and time using sysint21.

```
/*
        getdate: get system date and time

        DOS 1.xx and 2.xx
*/
getdate(date)
int date[4];
{
struct regval { int ax,bx,cx,dx,si,di,ds,es; } srv;

  srv.ax = 0x2a00;
  sysint21(&srv,&srv);
  date[0] = srv.cx;
  date[1] = srv.dx;

  srv.ax = 0x2c00;
  sysint21(&srv,&srv);
  date[2] = srv.cx;
  date[3] = srv.dx;
}

/*
   print out the current date and time
*/
main()
{
int date[4];
int year,month,day,hour,minutes,seconds,hundredths;

   getdate(date);

   year = date[0];
   month = date[1] >> 8;
   day = date[1] & 0xff;
   hour = date[2] >> 8;
   minutes = date[2] & 0xff;
   seconds = date[3] >> 8;
   hundredths = date[3] & 0xff;

   printf("date: %2d/%02d/%4d %2d:%02d:%02d.%02d\n",
        month,day,year,hour,minutes,seconds,hundredths);
}
```

3.111.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.111.7. See also

sysint, bdos, segread

3.112. system, Execute a program.

3.112.1. Synopsis

```
int system(string)
char *string;
```

3.112.2. Function

The string consists of the name of a program, followed by its command line arguments, exactly as it could have been entered at the keyboard to execute the program. This function uses loadexec to load a copy of command.com, and passes the string to command.com.

3.112.3. Returns

The error code returned by loadexec if command.com could not be loaded, otherwise zero. Note that the termination status of the requested program is NOT available.

3.112.4. Notes

The full device and pathname for command.com must be given by an entry in the environment, with "COMSPEC=". This may be checked by using the dos command "set" at the command level.

If nothing seems to happen when you call this function, remember to check the return code of the system call. It might indicate that command.com was never found and could not be loaded.

If you find that you are running out of memory when using this function, check and see if you can alter the size of the stack and heap by lowering the default in _default.c (see _default function). This may be enough to let you run bigger programs on your computer from the system call.

The big model heap and stack now leave room in unused memory for the use of this function. The default in _default.c can be varied to make the configuration of memory you need with both the big model and the system function.

3.112.5. Example

To obtain a directory of C source files:-

```
     system("dir *.c");
```

To run pass one of the compiler:-

```
     system("cc1 prog -b");
```

3.112.6. Operating System

DOS 3.0, DOS 2.0+

3.112.7. See also

loadexec

3.113. toascii, tolower, toupper - Convert characters

3.113.1. Synopsis

```
char toascii(c)
char c;

int tolower(c)
char c;

int toupper(c)
char c;
```

3.113.2. Function

Toascii returns it's argument with all bits turned off that are not part of a standard ASCII character (range: 0 - 0x7f). This is included to be compatible with other operating systems.

Tolower returns the lower case equivalent if the input character is upper case, else return the input character.

Toupper returns the upper case equivalent if the input character is lower case, otherwise it returns the input character.

3.113.3. Example

```
* tolower example
{
  extern int tolower();
  char chl, ch2;

     chl = 'C';
     ch2 = tolower(chl);
     /* ch2 contains 'c', as expected */

     ch2 = tolower('#');
     /* ch2 contains '#' */

     chl = 'x';
     ch2 = tolower(chl);
     /* ch2 contains 'x' */
}

* toupper example

{
  extern int toupper();
  char chl, ch2;

     chl = 'c';
     ch2 = toupper(chl);
     /* ch2 contains 'C', as expected */

     ch2 = toupper('#');
     /* ch2 contains '#' */

     chl = 'X';
     ch2 = toupper(chl);
     /* ch2 contains 'X' */
}
```

3.113.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.114. trigonometric functions

3.114.1. Synopsis

```
double sin(val)
double val;

double cos(val)
double val;

double tan(val)
double val;

double asin(val)
double val;

double acos(val)
double val;

double atan(val)
double val;

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

3.114.2. Function

These functions all take or return radian arguments.

3.114.3. Notes

asin and acos return zero if the argument is greater than 1.0.

tan returns a number greater than 1e+300 for arguments close to pi/2.

sin and cos return zero for arguments greater than 1e+8.

3.114.4. Example

```
#define PI 3.1415927;
#define PIHALF 1.5707963

{
  extern double sin(),  /* sin(x) where x is in radians */
                cos(),  /* cos(x) where x is in radians */
                tan(),  /* tan(x) where x is in radians */
                asin(), /* arcsin(x)  -1 < x < +1       */
                acos(), /* arccos(x)  -1 < x < +1       */
                atan(), /* arctan(x)  -1 < x < +1       */
                atan2();/* arctan(x/y) to return value in
                                 proper quadrant */
  double aval, bval, cval, dval;
  double argument;

                argument = PI;                   /* PI radians */
                aval = sin(argument);            /* aval is 0 */

                aval = cos(argument);            /* aval is 1 */

                bval = tan(argument);            /* bval is 0 */

                argument = PIHALF;               /* 90 degrees */

                cval = sin(argument);            /* cval is 1 */

                cval = cos(argument);            /* cval is 0 */

                cval = tan(argument);
                /* cval > 1e300  (tan 90 degrees is infinite) */

                dval = sin(1.2);     /* dval is 0.9320 */

                dval = cos(1.2);     /* dval is 0.36235 */

                aval = asin(1.0);    /* aval is PIHALF */

                aval = acos(1.0);    /* aval is 0 */

                aval = acos(47.0);   /* illegal, aval is 0 */

                dval = atan(33.0);   /* aval is 1.5405 radians */

                dval = atan2(10.0,3.0); /* dval is arctan(10/3)   */
}
```

3.114.5. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.115. ungetc, Push back an input character.

3.115.1. Synopsis

```
#include "stdio.h"

int ungetc(c,stream)        /* unget to stream */
char c;
FILE *stream;

#include "stdio.h"

int ungetch(c)              /* unget to stdin */
char c;
```

3.115.2. Function

Push the character "c" back into the stream. Only one character of push back is allowed. This character will be delivered on the next input function directed at the file. The input function may be any of the functions defined in this document that use a stream.

The function ungetch pushes a character back to the stream "stdin". It is defined by a macro in the standard header file. To use this function you must include the file "stdio.h" as part of your source program.

3.115.3. Returns

- The character itself.
- Minus one if any error is detected.

3.115.4. Notes

If an fseek, ftell, fwrite or fflush function is performed on the file, any character that was pushed back onto the stream will be forgotten.

3.115.5. Example

```
#include "stdio.h"

{
  extern int ungetc();  /* push a character back on a stream */
  extern FILE *fopen();
  extern int fclose();
  char ch; FILE *stream;

     stream = fopen("a:filename.dat","r");

     while((ch=fgetc(stream))!=EOF) if(isspace(ch)) break;

     ungetc(ch,stream); ch = fgetc(stream);
  /* ch is now the next whitespace character */

     fclose(stream);
}
```

3.115.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.115.7. Use with

getc, getchar, fscanf

3.116. unlink, Erase a disk file.

3.116.1. Synopsis

```
int unlink(filename)
char *filename;
```

3.116.2. Function

Erase a disk file. The file must not be open when this call is executed.

3.116.3. Returns

- Zero if the file was successfully deleted.
- A negative number if any error was detected.

3.116.4. Notes

File names containing question marks will result in all matching files being deleted in the DOSALL library. In the DOS2 library unlink will not erase filenames with wildcards (either '*' or '?') in them.

The DOSALL library supports path names when executing under DOS 2.0+.

3.116.5. Example

```
{
  extern int unlink();  /* delete a disk file */
  char *filename;
  int result;

     filename = "a:delfile.xxx";
     result = unlink(filename);

     printf("\nFILE: %s %s\n",filename,
          result < 0 ? "not deleted" : "deleted");
}
```

3.116.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.117. upper, Convert a string to upper case.

3.117.1. Synopsis

```
char *upper(string)
char *string;
```

3.117.2. Function

Converts all lowercase characters in the string to upper case. All other characters are unchanged.

3.117.3. Returns

The address of the string.

3.117.4. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.117.5. See also

lower ,tolower, toupper

3.118. utoa, Unsigned integer to ASCII conversion.

3.118.1. Synopsis

```
int utoa(value,buffer)
unsigned int value;
char *buffer;
```

3.118.2. Function

The value is converted to an unsigned ASCII digit string and stored in buffer. Buffer must be at least six bytes in length.

3.118.3. Returns

The count of the number of digits stored in buffer, excluding the trailing NULL.

3.118.4. Notes

This function uses sprintf to do the conversion

3.118.5. Example

```
{
  extern int utoa();
  extern char *alloc();
  extern int free();
  extern int fputs();
  unsigned int val;
  char *buffer;
  int count;

     buffer = alloc(10);
     *buffer = EOS;
     val = 5689;
     count = utoa(val,buffer);
     /* count contains 4, the number of characters */
     /* buffer is "5689" */
     fputs(buffer,stdout);
     free(buffer);
}
```

3.118.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.119. wqsort, Sort a set of records in memory.

3.119.1. Synopsis

```
int wqsort(n,cmpf,xchgf,data)
unsigned n;                     /* number of records to sort */
int (*cmpf)();                  /* key comparison function */
int (*xchgf)();                 /* record exchange function */
char *data;                     /* data for cmpf and xchgf */
```

3.119.2. Function

Uses Hoares quicksort algorithm to perform an in-core sort of n records. The function calls a user supplied function to compare keys of two candidates to be compared. The form of the call is:-

```
     (*cmpf)(x,y,&data);
```

where x and y are unsigned integers less than n. The function cmpf must return:-

```
     -1   if x < y
     0    if x == y
     +1   if x > y
```

If two records are to be exchanged, qsort performs the call:-

```
     (*xchgf)(x,y,&base);
```

3.119.3. Returns

Nothing

3.119.4. Notes

This function is provided for users converting code from another well known C Compiler. Its use is not recommended.

Information required by the functions cmpf and xchgf (for example the base address of the array of data) may be provided by:-

- Embedding the information within the functions
- Putting the data in a structure and using the address of the structure as an argument to qsort
- Placing the data as arguments to qsort and accessing them by using base as a pointer to a structure on the stack. This method is convenient but will result in NON-PORTABLE CODE.

3.119.5. Example

To sort an array of pointers to names into alphabetical order. We assume that the names are input and output by other code.

```
/*   sort example
*/

char *names[1000];               /* pointers to name strings */
unsigned number;                 /* number of names in array */

main()
{
  ...                            /* read in the names */
  wqsort(number,&namecmp,&nameswap,names);    /* do the sort */
  ...                            /* output the names */
  return 0;                      /* all ok */
}

/*   compare two names
*/

namecmp(i,j,base)
unsigned i,j;                    /* subscripts to name array */
char **base;                     /* pointer to name array */
{

  return strcmp((*base)[i],(*base)[j]);     /* not easy is it */
}

/*   exchange two name table entries
*/

nameswap(i,j,base)
unsigned i,j;                    /* subscripts to name array */
char **base;                     /* pointer to name array */
{
  iswap((*base)+i,(*base)+j);   /* do the swap */
}
```

3.119.6. Another way

The above is general but not too obvious. The two statements above could be written:-

```
                return strcmp(names[i],names[j]);
     and:-      iswap(names+i,names+j);
```

This is far less general, since the variable "names" is embedded in the code, but is probably quite acceptable for most purposes.

3.119.7. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.120. write, Write characters to a file.

3.120.1. Synopsis

```
int write(fd,buffer,count)
unsigned int fd;
char *buffer;
unsigned int count;
```

3.120.2. Function

Output count characters from buffer to the file specified by the file descriptor fd. If the file is open in ASCII mode, newline translation will be perfomed.

3.120.3. Returns

The number of characters written, which will be the same as count unless an error occurs. Returns minus one if an error is detected, and no characters were written. If it returns a number which is less than the count parameter, this is to be considered an error. The most likely cause is that the disk is full and no more characters could be written.

3.120.4. Notes

The infamous "WRITE" message is written by this function. It is an error message that indicates either the file was not open for output mode or that the file indicator (either a stream pointer or a file descriptor depending on how the file was opened) was invalid. Since this function is the basis for most of the output in the C86 library any of the output functions could result in a "WRITE" message occuring.

3.120.5. Example

```
{
  extern int write();
  extern int open();
  extern int close();
  unsigned int fd;
  char *buffer;
  unsigned int count;
  int num_wr;

     buffer = "data to be written to file";
     count = strlen(buffer);

     /* file must exist to be opened , else use creat */
     fd = open("a:xxx.xxx",AWRITE);
     if(fd < 0) { fputs("file not opened",stdout); return; }

     num_wr = write(fd,buffer,count);
     /* num_wr contains the actual number written */

     printf("\nWRITE:\n%u bytes written to file\n",num_wr);
     close(fd);
}
```

3.120.6. Operating System

DOS 3.0, DOS 2.0+, DOS 1.1+

3.120.7. See also

open, close, read

NOTES:

A. APPLICATION NOTES

The following pages are directly from our bulletin board. They are included to help people develop certain applications and learn certain aspects of C programming. Good luck! The sources can be downloaded from our user group bulletin board. To join the user group contact our sales staff.

A.1. PLINK DEMONSTRATION

```
****file: root.c****
#include <stdio.h>
main()
{
  overlay1();
  overlay2();
}

****file: overlay1.c****
#include <stdio.h>
overlay1()
{

  printf("overlay1\n");
}

****file: overlay2.c****
#include <stdio.h>
overlay2()
{

  printf("overlay2\n");
}

****file: test.lnk****
output ovlytest.exe
file root
library c86s2s
begin
  section file overlay1
  section file overlay2
end
class datab,datac,datai,datat,datau,datav,heap,stack


****file: test.bat****
plink86 @test
```

A.2. CREATING .COM FILES

A.2.1. New prologue.h

```
;       prologue.h      11/5/83

;       standard prologue for c86 assembly code

;       DEFINE ARGUMENT BASE RELATIVE FROM BP

IF      @BIGMODEL
@AB     EQU     6
ELSE
@AB     EQU     4
ENDIF

@CODE   SEGMENT BYTE PUBLIC 'CODE'
@CODE   ENDS
@DATAB  SEGMENT PARA PUBLIC 'DATAB'
@DATAB  ENDS
@DATAC  SEGMENT BYTE PUBLIC 'DATAC'
@sb     label   byte
@sw     label   word
@DATAC  ENDS
@DATAI  SEGMENT BYTE PUBLIC 'DATAI'
@ib     label   byte
@iw     label   word
@DATAI  ENDS
@DATAT  SEGMENT BYTE PUBLIC 'DATAT'
@DATAT  ENDS
@DATAU  SEGMENT BYTE PUBLIC 'DATAU'
@ub     label   byte
@uw     label   word
@DATAU  ENDS
@DATAV  SEGMENT BYTE PUBLIC 'DATAV'
@DATAV  ENDS
DGROUP  GROUP   @DATAB,@DATAC,@DATAI,@DATAT,@DATAU,@DATAV
@CODE   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:@CODE,DS:DGROUP

;       END OF PROLOGUE.h
```

A.2.2. New $main.asm

The following file is the modified $main.asm which must be used to create a com file:

```
;       title   'c86 basic support package'

;       this is the starting point for all C programs
;       modified for dos 2.0

        include model.h

;       following is copy of prologue.h
;       this is included so you can vary the assume statement
;       this will allow the creation of 8080 and modified big model format

;       define the following to be true for 8080 (.com) file

@COMFILE        EQU     1
IF      @COMFILE
IF      @BIGMODEL
ABORT-THERE IS NO WAY THIS IS REASONABLE
ENDIF
ENDIF

;       DEFINE ARGUMENT BASE RELATIVE FROM BP

IF      @BIGMODEL
@AB     EQU     6
ELSE
@AB     EQU     4
ENDIF

@CODE   SEGMENT BYTE PUBLIC 'CODE'
@CODE   ENDS
@DATAB  SEGMENT PARA PUBLIC 'DATAB'
@DATAB  ENDS
@DATAC  SEGMENT BYTE PUBLIC 'DATAC'
@sb     label   byte
@sw     label   word
@DATAC  ENDS
@DATAI  SEGMENT BYTE PUBLIC 'DATAI'
@ib     label   byte
@iw     label   word
@DATAI  ENDS
@DATAT  SEGMENT BYTE PUBLIC 'DATAT'
@DATAT  ENDS
@DATAU  SEGMENT BYTE PUBLIC 'DATAU'
@ub     label   byte
@uw     label   word
@DATAU  ENDS
@DATAV  SEGMENT BYTE PUBLIC 'DATAV'
@DATAV  ENDS

IF      @COMFILE
```

```
DGROUP  GROUP   @CODE,@DATAB,@DATAC,@DATAI,@DATAT,@DATAU,@DATAV
ELSE
DGROUP  GROUP   @DATAB,@DATAC,@DATAI,@DATAT,@DATAU,@DATAV
ENDIF

@CODE   SEGMENT BYTE PUBLIC 'CODE'

IF      @COMFILE
        ASSUME  CS:DGROUP,DS:DGROUP
        ORG     100H
ELSE
        ASSUME  CS:@CODE,DS:DGROUP
ENDIF


;       END OF PROLOGUE.h

@code   ends

;       add stack and heap segments

@HEAP   SEGMENT WORD PUBLIC 'HEAP'
@HEAPBASE       LABEL   BYTE
@HEAP   ENDS

IFE     @COMFILE
@STACK  SEGMENT PARA STACK 'STACK'
        DW      128 DUP (?)
@STACK  ENDS
ENDIF

@DATAB  SEGMENT
        DW      0                       ;DATA SEGMENT CAN NOT START AT ZERO
@DATAB  ENDS

@DATAC  SEGMENT

        public  _systype,_sysvers,_PSPSEG,_heaptop,_SYSENDP

;       the following identifies the base operating system

_systype        dw      1       ;o/s type (ms-dos)
_sysvers        dw      0       ;o/s version (low byte 0 if < dos 2.00 )
_SYSENDP        DW      0       ;LENGTH OF PROG IN PARAGRAPHS
if      @bigmodel
_heaptop        DW      @UDEND+2,seg @udend     ;pointer to base of heap
else
_heaptop        DW      OFFSET DGROUP:@UDEND+2  ;pointer to base of heap
endif

COREMES DB      0AH,0DH,'NO CORE$'

_PSPSEG DD      0                       ;32 BIT POINTER TO THE PROG SEG PREFIX

@DATAC  ENDS
```

```
@DATAT  SEGMENT
@UDBEGIN        LABEL   BYTE
@DATAT  ENDS
@DATAV  SEGMENT
@UDEND          LABEL   BYTE
@DATAV  ENDS
@DATAI  SEGMENT
        EXTRN   _MINFMEM:WORD,_MAXFMEM:WORD,_MINRMEM:WORD
@DATAI  ENDS


IF      @BIGMODEL
        EXTRN   _MAIN:FAR,_EXIT:FAR
@code   segment
ELSE
@code   segment
        EXTRN   _MAIN:NEAR,_EXIT:NEAR
ENDIF

$MAIN   PROC    FAR
        JMP     SHORT BEGIN

;       PLACED HERE FOR SHORT JUMP PROBLEMS


NOCORE:
        MOV     DX,OFFSET DGROUP:COREMES
        MOV     AH,9
        INT     21H
        MOV     AX,-1           ;SAY BAD ERROR
        PUSH    AX
        CALL    _EXIT           ;NEVER RETURNS

        public  $main

;       $main   entry point for c programs

BEGIN:  esc     1ch,bx                  ;reset the 8087 if any
        cld                             ;just in case
IFE     @COMFILE
        MOV     AX,DGROUP
        MOV     DS,AX                   ;SET UP DS REGISTER
ENDIF
        MOV     WORD PTR DGROUP:_PSPSEG+2,ES    ;SAVE THE PROG SEG PREFIX
        MOV     AX,ES:2                 ;GET TOP OF CORE IN PARA UNITS
        CMP     AX,DGROUP:_MINRMEM      ;IS RESERVED MEMORY AVAILABLE ?
        JBE     NOCORE                  ;NOPE
        SUB     AX,DGROUP:_MINRMEM      ;SO RESERVE IT

IFE     @BIGMODEL
        MOV     DI,DS                   ;LIMIT SIZE TO 64 K
        ADD     DI,1000H                ;PARAGRAPHS IN 64K
        CMP     DI,AX                   ;MORE MEMORY THAN WE NEED ?
        JAE     DM01                    ;NOPE
        MOV     AX,DI                   ;RESET IT
```

```
DM01:
ENDIF

IF      @COMPILE
        MOV     SI,OFFSET @HEAP
        ADD     SI,15
        SHR     SI,1
        SHR     SI,1
        SHR     SI,1
        SHR     SI,1
        MOV     CX,DS
        ADD     SI,CX
ELSE
        MOV     SI,@HEAP                        ;GET PARA OF HEAP
ENDIF
        CMP     SI,AX                           ;IS HEAP ABOVE 'TOP OF MEM' ?
        JAE     NOCORE                          ;YEP
        SUB     AX,SI                           ;# OF FREE PARAGRAPHS
        CMP     AX,DGROUP:_MINFMEM              ;GOT OUR MINIMUM
        JB      NOCORE                          ;NOPE
        CMP     AX,DGROUP:_MAXFMEM              ;GOT TOO MUCH ?
        JBE     DM02                            ;NOPE
        MOV     AX,DGROUP:_MAXFMEM              ;RESET IT
DM02:
        ADD     SI,AX                           ;GET THE NEW STACK TOP PARAGRAPH
        MOV     DI,DS                           ;GET THE DATA SEG
        MOV     AX,SI
        SUB     AX,DI                           ;GET TOT NUMBER OF PARAS
        CMP     AX,1000H                        ;DOES IT EXCEED 1 SEGMENT
        JBE     DM03                            ;NOPE
        MOV     AX,1000H                        ;USE THE WHOLE STACK
        SUB     SI,AX                           ;AND THIS IS THE SS VALUE WE NEE
        MOV     DI,SI                           ;IN THE CORRECT PLACE
DM03:
        mov     si,ax                           ;save stack size
        MOV     CL,4
        SHL     AX,CL                           ;SCALE THE SP VALUE
        PUSHF                                   ;GET THE FLAGS
        POP     CX                              ;IN A SAFE PLACE
        CLI                                     ;TURN OFF INTERRUPTS
        MOV     SS,DI                           ;RESET SS
        MOV     SP,AX                           ;AND THE STACK POINTER
        XOR     BP,BP                           ;CLEAR BP
        PUSH    BP
        MOV     BP,SP
        PUSH    CX
        POPF                                    ;RESTORE FLAGS AND INTERRUPTS

;       get the operating system version (for version dependant i/o)

        push    es              ;so we don't forget it
        add     di,si           ;get end paragraph address
        push    di              ;save for later
        mov     ah,30h
        int     21h
        or      al,al
        jnz     isv2                ;is a ver 2.00 system
        xor     ah,ah               ;reset ah too if below 2.00
        pop     bx                  ;dump the end of prog paragraph
isv2:
        mov     DGROUP:_sysvers,ax
        jz      notv2

;       set length of program for use of memory after program

        pop     bx
        pop     ax                  ;get seg of psp
        mov     es,ax               ;in es too
        sub     bx,ax               ;get length of program
        MOV     _SYSENDP,BX         ;SAVE FOR USER
        push    es
        mov     ah,4ah
        int     21h
notv2:
        mov     bx,ds                       ;save the ds value
        pop     ds                          ;get the prog seg prefix value
        push    ss
        pop     es                          ;set dest
        mov     si,80h                      ;command line offset
        mov     cl,[si]                     ;get command line count
        add     cl,3
        and     cx,0feh                     ;force count even
        sub     sp,cx                       ;get stack pointer value
        mov     di,sp
        rep     movsb                       ;move the string
        mov     ds,bx                       ;restore ds
        mov     ax,sp
if      @bigmodel
        push    ss
endif
        push    ax                          ;set pointer to command line **

;       clear the uninitialized global storage region

        mov     es,bx                       ;set es to data seg value
        MOV     DI,OFFSET DGROUP:@UDBEGIN
        MOV     CX,OFFSET DGROUP:@UDEND
        SUB     CX,DI                       ;GET THE NUMBER OF BYTES
        XOR     AX,AX
        REP     STOSB                       ;CLEAR THE AREA

;       call the routine DGROUP:_main to do other initialisation

        call    _main                       ;enter c system at 'DGROUP:_main'
        push    ax                          ; Put the exit value
        call    _EXIT                       ;ALL DONE NOW, this never returns

$MAIN   ENDP
        INCLUDE EPILOGUE.H
        end     $MAIN
```

A.2.3. Notes on getting com files created:

How to build .COM files from files produced by the C compiler:

First:

Get assembly output for any functions that you need. This includes stuff like fopen, _main, _exit, etc, that you may not normally think of.

Then:

Assemble all the files with the new prologue.h. You will have to turn the @COMFILES switch to 1 in order for this to work. Don't forget to assemble $maincom.asm. Have fun running the assembler.

Then:

Link as normal WITH THE $MAIN FILE FIRST: Do not do any funny stuff with the linker. You will get the message NO STACK SEGMENT from the linker. This is to be expected.

Now:

You now have a file with the .EXE extension.

Run EXE2BIN.EXE on it to produce a .BIN file.

If EXE2BIN does not display any messages, you are ok.

Rename the .BIN file to .COM.

Run the program and enjoy.

A.3. TECHNICAL NOTES ON THE 8087 FLOATING POINT FORMAT

The floating point format used by the compiler is the same as that for the Intel 8087 numeric data processor. The format is as follows:

FLOAT:

```
|-|------------------|-----------------------------------|
|S|  biased exponent |    significand                    |
|-|------------------|-----------------------------------|
31                   23                                  0
                     /\ (implied binary point)
```

the exponent for a float (SHORT REAL) is stored with a bias of 7f hex. this means that 7f is added to the exponent when it is stored in this format.

DOUBLE:

```
|-|------------------|-----------------------------------|
|S|  biased exponent |    significand                    |
|-|------------------|-----------------------------------|
63                   52                                  0
                     /\ (implied binary point)
```

the exponent for a double (LONG REAL) is stored with a bias of 3ff hex. this means that 3ff is added to the exponent when it is stored in this format.

For more information on this data format, see the iAPX 86/20, 88/20 Numerics Supplement. This is part of the iAPX 86,88 User's Manual and can be obtained from Intel at:

Intel Corporation
Literature Dept. SV3-3
3065 Bowers Avenue.
Santa Clara, CA 95051

A.4. Variable length tables at run time

TECHNICAL NOTES ON ALLOCATING VARIABLE LENGTH TABLES AT RUN-TIME

Here is an example of how to allocate a pointer table at runtime. This can be used for most any data type. It is assumed here that the data is of type DATA. This could be a structure or a simple data type.

```
/*
*/
main()
{
int i;                      /* table index variable */
int tabsize;                /* length of table */
DATA **p;                   /* pointer to DATA table */
char *calloc();             /* calloc returns a pointer */

  /* determine number of entries at run-time */
  tabsize = 100;

  /* get pointer table */
  p = (DATA *)calloc(tabsize,sizeof (DATA *));

  /* allocate data area for each entry in table */
  for(i = 0; i < tabsize; i++){
        p[i] = (DATA *)calloc(1,sizeof(DATA));
  }

  /* fill in data areas */
  for(i = 0; i < tabsize; i++){
        filldata(p[i]);
  }

  /* use them somehow */
  for(i = 0; i < tabsize; i++){
        usedata(p[i]);
  }

  /* free up data areas */
  for(i = 0; i < tabsize; i++){
        free(p[i]);
  }

  /* free up pointer table */
  free(p);

}
```

```
/*
        now filldata and usedata can be written such that they
        do not know that they are part of an array:
*/
filldata(d)
DATA *d;                /* filldata gets a pointer to DATA */
{
        /* initialize the data somehow */
}

usedata(d)
DATA *d;                /* usedata gets a pointer to DATA */
{
        /* do something with the data */
}
```

A.5. Calling a function with a pointer.

TECHNICAL NOTES ON CALLING A FUNCTION THROUGH A POINTER

```
noargs()
{
int (*p)();        /* p is a pointer to a function returning int */
extern int a();    /* a is a function returning int              */

    p = a;         /* set p to point to a */
    (*p)();        /* call a */

}

int a()
{
   printf("hello there\n");
}

withargs()
{
int (*p)();        /* p is a pointer to a function returning int */
extern int b();    /* b is a function returning int              */
int x,y;
   x = 10;
   y = 20;
   p = b;
   (*p)(x,y);
}

int b(arg1,arg2)
int arg1,arg2;
{

   printf("arg1 = %d arg2 = %d\n",arg1,arg2);
}
```

A.6. TECHNICAL NOTES ON READING A NUMBER FROM THE CONSOLE

```
/* Here is a function to read an INTEGER from the console: */
int getnum(message)
char *message;
{
extern char *fgets();
char buffer[128];
int number;

        fputs(message,stderr);
        if(fgets(buffer,128,stdin) == NULL) return 0;
        if(sscanf(buffer,"%d",&number) != 1)
           {
           fputs("Invalid input, Please enter an integer\n",stderr);
           return getnum(message);
           }
        return number;
}

/* This can then be called in the following fashion: */
program()
{
int number;
        number = getnum("\nPlease enter the number: ");
        printf("\nThe number entered is: %d\n",number);
}

/* For FLOATING POINT numbers, you must use a different
conversion code in sscanf: */
double getnum(message)
char *message;
{
extern char *fgets();
char buffer[128];
double number;

        fputs(message,stderr);
        if(fgets(buffer,128,stdin) == NULL) return 0.0;
        if(sscanf(buffer,"%lf",&number) != 1)
           {
           fputs("Invalid input, Please enter an number\n",stderr);
           return getnum(message);
           }
        return number;
}

/* This can be called in a similar fashion: */
program()
{
double number;
double getnum();
        number = getnum("\nPlease enter the number: ");
        printf("\nThe number entered is: %lf\n",number);
}
```

A.7. TECHNICAL NOTES ON THE USE OF MOVBLOCK

Here is an example of how to use movblock in the BIG MODEL:

Suppose that you wish to move 10 bytes of data from your data buffer to the screen memory:

```
{
char data[10];
unsigned int dest_seg;
unsigned int dest_off;
int count;

        dest_seg = address of the segment you wish to write to;
        dest_off = offset within that segment;
        count = 10;

        movblock(data,dest_off,dest_seg,10);
}
```

This should do the trick. You will have to find out what segment and offset values are needed for your memory map.

In the SMALL MODEL, you will have to determine the segment address of your data segment. This can be done via segread as follows:

```
{
struct segregs { unsigned int scs,sss,sds,ses; } srv;
char data[10];
unsigned int dest_seg;
unsigned int dest_off;
int count;

        segread(&srv);
        dest_seg = address of the segment you wish to write to;
        dest_off = offset within that segment;
        count = 10;
        movblock(srv.sds,data,dest_off,dest_seg,10);
}
```

A.8. TECHNICAL NOTES ON DEFAULT MEMORY VALUES

This is for use with the file _default.c in base.arc:

HEAP + STACK	_MAXFMEM
32K	800
64K	1000
96K	1800
128K	2000
160K	2800
192K	3000
224K	3800
256K	4000
288K	4800
352K	5800
416K	6800
480K	7800
544K	8800
608K	9800
672K	A800
736K	B800
800K	C800
864K	D800
928K	E800
992K	F800

A.9. TECHNICAL INFO ON THE CORRECT USE OF FOPEN()

Here is some information on the correct use of fopen(). In the BIG MODEL, it is imperative that the function fopen() be declared as a function returning type FILE *. Consider the following example:

```
main()
{
extern FILE *fopen();
int c;
FILE *fptr;

        fptr = fopen("\\c86\\stdio.h","r");
        if(fptr == NULL)
                abort("could not find file: \\c86\\stdio.h");
        while((c = fgetc(fptr)) != EOF) putchar(c);
        fclose (fptr);

}
```

Note that you should always check the return code of fopen.
Note that you need two backslashes for each one you want.

A.10. TECHNICAL INFORMATION ON LOW-LEVEL Z-100 PC ROM CALLS

C is a uniquely powerful language which is often used for development of new systems software. C86 provides many extensions to the standard language in order to allow you to better exploit the capabilities of your machine, at the expense of portability. Some applications, however, may need even greater access to low-level I/O than the included library routines.

For more information on low-level interfaces to the Z-100 PC, consult the Z-100 PC Technical Reference Manual. This document contains information on using the ROM firmware contained in your machine.

Additionally, Zenith sells a package of utilities known as the MS-DOS Version 2 Programmer's Utility Package. This package contains an 8086 Macro-Assembler which is compatible with C86, a powerful full-screen program editor, and valuable documentation on interfacing to MS-DOS and the firmware in the Z-100 and Z-100 PC series.

To order these documents, contact your local Zenith dealer or distributor.

A.11. TECHNICAL INFO ON BIG MODEL POINTERS

Here is some code to demonstrate how to split a big model pointer into it's segment and offset values:

```
char *p;                        /* big model pointer */
unsigned int seg;               /* where to store segment val*/
unsigned int off;               /* where to store offset */

    seg = ((unsigned long) p) >> 16;    /* segment value */
    off = (unsigned int) p;             /* offset value  */
```

A.12. TECHNICAL INFO ON DOING SEND / RECEIVE FOR IBM-PC

Here are some functions for doing low-level serial io on the IBM PC.

```
struct regval { unsigned int ax,bx,cx,dx,si,di,ds,es; } ;
#define COM1 0
#define COM2 1

/*
        send:   send a character to a COM1

        returns:
                The value of the character sent or
                EOF if an error occurred
*/
send(ch)
char ch;
{
struct regval srv;

        srv.ax = 0x100 | (ch & 0xff);           /* ah = 1, al = ch   */
        srv.dx = COM1;                          /* select COM1       */
        sysint(0x14,&srv,&srv);                 /* send it           */
        if(srv.ax & 0x8000) return EOF;         /* an error occurred */
        return ch;                              /* no error occurred */
}


/*
        recv: wait until a character is ready at COM1 and return it
        Returns: the character received
*/
recv()
{
struct regval srv;

  do {                                  /* try to receive it */
        srv.ax = 0x200;                 /* select function   */
        srv.dx = COM1;                  /* select COM1       */
        sysint(0x14,&srv,&srv);         /* try to get it     */
     }
   while (srv.ax & 0xff00);             /* it's not ready yet */
   return (srv.ax & 0xff);              /* return the character */
}
```

A.13. TECHNICAL INFORMATION ON USING THE ANSI.SYS DEVICE DRIVER

Some pointers on using the ANSI.SYS device driver to control the screen:

The ANSI.SYS device driver is a can be used for a variety of screen handling uses. These range from clearing the screen to positioning the cursor to setting the mode for the screen.

Codes to control this device are given in an Appendix of the MS DOS 2.0 manual.

Here is how to install the ANSI.SYS device driver:

1. Create or add to the file CONFIG.SYS in the root directory of the disk that you boot from by adding the statement:

```
DEVICE = ANSI.SYS
```

2. Make the file ANSI.SYS available on your boot disk. This file is provided as part of the standard distribution of your operating system.

3. Reboot your machine.

Once this is done, you will be able to do many screen and keyboard operations easily. For example:

```
printf("\033[2J");            /* clear the screen */

printf("\033[%d;%dH",row,col);/* position cursor to row,col */

printf("\033[%dA",n);         /* move cursor up n rows */

printf("\033[%dB",n);         /* move cursor down n rows */

printf("\033[%dC",n);         /* move cursor right n columns*/

printf("\033[%dD",n);         /* move cursor left n columns */
```

In addition, there are escape codes for doing things like crt mode selection and keyboard reassignment, but you will have to look those up yourself if you want them.

INDEX