

Green Hills Software User's Manual

C Library

September 1, 1986

Green Hills Software, Inc.

Copyright © 1986 by Green Hills Software, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Green Hills Software, Inc.

Green Hills Software, Inc.
425 E. Colorado, Suite 710
Glendale, CA 91205
(818)246-5555

UNIX is a trademark of Bell Laboratories.
VAX, DEC, and VMS are trademarks of Digital Equipment Corporation.
4.2BSD is a trademark of the Board of Regents of the University of California at Berkeley.

The information in this publication is subject to change without notice.

Chapter 1

Customizing and Compiling the Green Hills Software C Library

This chapter explains how to customize and compile the Green Hills Software C Library source files. If you received the Green Hills Software C Library already compiled then you can ignore this chapter and begin using it immediately.

The material in this chapter assumes that you are familiar with The Green Hills Software User's Manual for your version of C.

1.1. Compile Time Options

On the compilation command line for each module you can specify certain definitions that will affect the functionality of the resulting object code library. The definitions are specified with the `-D` command line definitions of C (if you don't know what `-D` is, you are in over your head already). The available compile time options are given below. Each of these options affects only a few files, but you should specify the same options for all files. If you specify different options for different files you may create an incorrect library.

`-Dno_float`

If you do not intend to do formatted floating point input or output with `printf` or `scanf` you may specify `-Dno_float`. If you specify `-Dno_float`, `printf` and `scanf` will ignore the floating point format characters `e`, `f`, and `g`. The result of specifying floating point arguments to `printf` and `scanf` will be quite confusing (and definitely wrong).

The reason that this option is provided is that many embedded programs do not use floating point, but they need `printf` for diagnostic messages. `Printf` and `scanf` learn the types of their arguments only at runtime by examining the format character string argument. The linker cannot know whether any floating point arguments will ever be given to `printf` or `scanf`. Therefore, if `printf` and `scanf` are compiled with the default they will load in a huge amount of floating point library code in anticipation of having to do formatted floating point I/O, even if the program does not use any floating point at all. Specifying `-Dno_float` will suppress the loading of floating point code solely for the purpose of implementing formatted I/O.

`-DUN_BUFFERED_STD`

By default, the `stdin` and `stdout` streams are line buffered for compatibility with UNIX. This buffering takes 1024 bytes of buffer space. In many embedded systems this overhead is intolerable. If `-DUN_BUFFERED_STD` is specified then `stdin` and `stdout` will not be buffered. Output will occur immediately after a `putchar()` is executed instead of waiting until the end of the line and `getchar()` will return as soon as a character is typed instead of waiting for an end of line to be typed.

`-Dprompts`

If `-Dprompts` is specified when compiling the library, `args()` will prompt the user on standard output with `">"`. At this point the user can type in a UNIX style command. The UNIX redirection operator `"<"` can be used to open `stdin`. The UNIX redirection operator `">"` can be used to open `stdout`.

`-Ddo_roman`

If you want to be able to input and output Roman Numerals in `printf` and `scanf` specify `-Ddo_roman` when compiling the library.

`-Dvaxd_float`

If your floating point format is the standard Green Hills MC68000 VAX Floating Point Format Software Emulation you may specify `-Dvaxd_float` when compiling the library. It will improve the floating point library performance.

-Dieee_float

If your Target uses IEEE floating point numbers you may specify `-Dieee_float` when compiling the library. It will improve the floating point library performance.

If `-Dieee_float` is specified you must examine the `math.h` include file. At the top of `math.h` are a number of `#ifdef`'s for various computer architectures. If the type of your target machine is specified in the `#ifdef`'s then there is nothing more for you to do.

If your computer architecture is not specified in the `#ifdef`'s you must find out whether your processor conforms to the LittleEndian or BigEndian format. LittleEndian must be defined if your target machine stores the exponent part of an IEEE floating point number in memory in the last two bytes of the representation. BigEndian must be defined if your target machine stores the exponent part of an IEEE floating point number in memory in the first two bytes of the representation. If the machine is neither BigEndian or LittleEndian you may not use `-Dieee_float`.

1.2. Customizing the Low Level System Interface

The module `"os.c"` is a simple implementation of the low level system interface for the Green Hills Software C Library. As it is written it is only minimally useful. Since every system has different memory addresses and I/O devices you must customize these low level functions to work with your system in order to be able to use the Green Hills Software C library.

The functions as written in `"os.c"` will execute unchanged in most environments with limited functionality. You will be unable to do any output until you modify `OUTPUT()` to read a character from some terminal or other I/O device. You will be unable to do any input until you modify `INPUT()` to read a character from some terminal or other I/O device. You will be unable to gracefully terminate execution until you modify `_exit()` to return control to you. You will be unable to access command line arguments in your main program until you modify `args()` to find them.

But most important of all you will not be able to do anything at all until you figure out how to execute the `START()` function in `"crt0.c"`! This is the most system dependent part of the library. `START()` may be called by the operating system, a monitor, a debugger, or on system reset. YOU must figure out how to execute the `START()` function. In your environment it may be automatic. If not, you will have to inform the Link Editor (Linker) that program execution is to start at the top of the `START()` function.

In addition, you may have to modify the `START()` function for your particular environment. `START()` is responsible for initializing the processor, stack pointer, I/O, and memory systems. Then it calls the program's `main()` function with the command line arguments. If the `main()` function returns to `START()`, `START()` exits by calling `exit()`.

As written, `START()` is a C function. It may be possible to leave `START()` written in C (possibly using `"asm"` statements), but more than likely you will find it necessary to compile `START()` into assembly language and then edit the assembly code so that it can be used. In particular, there may be procedure entry code at the top of `START()` to save registers. This code may depend on a stack pointer having already been initialized. But since it is the job of `START()` to initialize the stack pointer (if it hasn't already been initialized), you may need to delete the function entry code.

The implementation of the system call interface in the `"os.c"` module is very simple. It provides only simple formatted I/O for a single terminal and dynamic allocation of memory from a static array. It assumes the following:

- (a) There is a way of entering the program at the function `START()`.
- (b) There is a way to exit the program and return control to the operating system, monitor, or debugger. This is done by calling `_exit(code)`. To exit cleanly from programs you must modify the `_exit()` routine in this module. As it is written, it will terminate with a divide by zero exception. If your execution environment does not stop on a divide by zero you must modify `_exit()` or the program will never stop and terrible things will certainly happen!
- (c) There is one input device of interest. Characters can be read from it by calling the function `INPUT()` in this module. It returns a value 0..255 or EOF for error. As it is written it always returns EOF (end of file). To

do any input at all you must modify this routine!

- (d) There is one output device of interest. Characters can be written to it by calling OUTPUT(ch). Its return will be ignored. As it is written it doesn't output anything. To do any output at all you must modify this routine!
- (e) 10000 bytes is the maximum amount of dynamically allocated memory that will be needed. This 10000 bytes is permanently allocated in a static array in "os.c". If this size is too large or too small you should change the constant MEMSIZE in "os.c" to the correct amount of dynamically allocated memory.

After implementing items (a) through (e) above, you have the functionality needed in an embedded system, simple printf and scanf to a terminal and memory allocation from a fixed pool. No file I/O is supported.

If you are ambitious and you want to use all of the functions of the Green Hills Software C library you must fully implement the functions outlined in "os.c". Each function is a standard UNIX system call. A complete implementation of the UNIX system call is not necessary. The functionality that is necessary is specified with each function definition. Some hints and suggested implementations are given in comments in "os.c".

1.3. Customizing Include Files

The include files should operate perfectly well as you have received them, but you may wish to tune them to your system. The include files should be in a directory named "include".

BUFSIZ

The constant BUFSIZ in stdio.h is the buffer size used in buffered I/O. The smaller BUFSIZ is, the less memory will be used in I/O buffers (one for each open file). The larger BUFSIZ, the fewer I/O operations must be done to disk (or any other I/O device).

BLOCKSIZE

The constant BLOCKSIZE in stdio.h is the block size which malloc uses to call sbrk() to get more memory from the operating system. The larger BLOCKSIZE is, the fewer times sbrk() will be called. The smaller BLOCKSIZE is, the less memory will be wasted at the greatest extent of the program.

_NFILE

The constant _NFILE in stdio.h is the maximum number of files that can be open at one time (this number includes stdin, stdout, and stderr). It is also possible that the actual limit is less, if the underlying operating system or the "os.c" module has a lower limit on the number of simultaneously open files.

1.3.1. Custom Floating Point Formats

You do not need to read this section if your system has IEEE floating point, the Green Hills Software MC68000 VAX Floating Point Format Software Emulation, or no floating point at all.

If your system has a non-standard floating point format then you may either settle for the default floating point configuration which will work with any floating point format fitting the constraints given below in the Implicit Library Assumptions section, or you may customize the math.h include file for your floating point format. If you chose to customize the math.h include file, be prepared for some tricky debugging.

The performance of the floating point functions can be greatly improved if you CORRECTLY define the following macros in the include file math.h. It should be located in a directory called "include".

KNOWN_FLOATING_POINT_FORMAT

Announces to the library that you have CORRECTLY defined the following macros, so the library can use them at will.

EXTRACTEXPONENT(x)

Return the exponent field of the double, x, such that for some y:

$$0.5 \leq y \text{ \&\& } y < 1.0 \text{ \&\& } x == y * 2^{**}(\text{EXTRACTEXPONENT}(x))$$

EXTRACTFLOATEXPONENT(x)

Return the exponent field of the float, x, such that for some y:

```

0.5 <= y && y < 1.0 && x == y*2**(EXTRACTFLOATEXPONENT(x))
SETEXPONENT(x,exp)
    For double x: x = x * 2**(exp - EXTRACTEXPONENT(x))
SETFLOATEXPONENT(x,exp)
    For float x: x = x * 2**(exp - EXTRACTFLOATEXPONENT(x))
AUGMENTEXPONENT(x,inc)
    For double x: x = x * 2**inc
AUGMENTFLOATEXPONENT(x,inc)
    For float x: x = x * 2**inc
MINIMUMEXPONENT
    The minimum possible value for EXTRACTEXPONENT
MINIMUMFLOATEXPONENT
    The minimum possible value for EXTRACTFLOATEXPONENT
MAXIMUMEXPONENT
    The maximum possible value for EXTRACTEXPONENT
MAXIMUMFLOATEXPONENT
    The maximum possible value for EXTRACTFLOATEXPONENT
HUGE  A number very close to the maximum legal value for a double floating point number. A little less than
      2**MAXIMUMEXPONENT.
HUGEFLOAT
    A number very close to the maximum legal value for a float floating point number. A little less than
      2**MAXIMUMFLOATEXPONENT.

```

1.4. Installing the Green Hills Software C Library

When you have made all of the customizations required for your system, compile every module of the library except “test-c.c”. For each compilation specify the compile time options that you selected from the “Compile Time Options” section above. Be sure to specify the same compile time options on every file. Do not compile the file “test-c.c”, it is a test program and should not be included in the library.

When all of the files have been compiled, they must be collected together into a library. The exact means for creating a library varies from system to system. You should consult your librarian or linker documentation to find out how to make a library.

If you cannot figure out how to make a library, you can just leave all of the object files for the library in a directory. Every time you link a program, link in all of the C library object files. This will produce a very large program. If this is not acceptable, you can delete any library files that you do not use. In this way you can get the same effect as if you had a librarian.

1.5. Green Hills Software C Library Test Program

When the library has been built you are ready to compile and run the Green Hills Software C Library Test Program. This Test Program is designed to test the Green Hills Software C Library. The source for the Green Hills Software C Library Test Program is the file “test-c.c”.

Before you compile “test-c.c” you may have to make a few adjustments. There are three compile time options that can be specified when compiling the Test Program.

```

-Dtest_stdin
    Test input from stdin. By default only output is tested.
-Dtest_files
    Test I/O on files. By default only terminal I/O is tested.

```

`-DTempFilename=file-name`

If you specify `-Dtest_files` you must also define the symbol `TempFilename` to be the name of a file that can be created and written to. If `TempFilename` is not specified it defaults to `FOO.BAR`.

When the Test Program is run, the first thing that should happen is that it should print the following line on your terminal:

Begin test

It does this by calling `write` with the single character 'B' and specifying file number 1 (standard output); followed by a call to `"write"` with the single character 'e' specifying file number 2 (standard error); next it calls `write` to the standard output with the four character string `"gin "`; finally it calls `write` to standard error with the five character string `"test\n"`. If this line does not come out correctly you have not implemented the `"write"` function correctly.

The second line of output should be:

C library test version 1.0 (test putchar,fflush)

This line is output by a series of `putchar` calls followed by `fflush(stdout)`. If this line doesn't come out you probably have problems with your include files. Make absolutely sure that the `"stdio.h"` file in the include directory supplied with this library is being included and not some other `"stdio.h"` file from another directory (such as `/usr/include/stdio.h`).

The third line of output should be:

Basic eprintf test

This line is output by a call to `eprint` with a single string argument. If this line doesn't come out there is a serious problem with your compiler.

The fourth line of output should be:

More advanced eprintf test(all error msgs are produced with eprintf)

This line is output by a call to `eprintf`. If this line does not come out you probably have a problem with the `<varargs.h>` file. It is absolutely essential that you have a correctly working `VARARGS(3)` facility in order to compile the `printf` and `scanf` modules. Make sure that the `"varargs.h"` file that is being included in these compilations is the correct one for your target machine. For more information, consult "The `VARARGS(3)` Facility" section of the Green Hills Software User's Manual for your version of C.

The fifth line of output should be:

Basic printf test

This line is output by a call to `printf` with a single string argument. If this line doesn't come out there is something wrong with your file number 1 (standard output).

The next three lines of output print out the `argc`, `argv`, and environment arguments that are passed to `main()`. The default output is:

```
argc=1
argv={" " " "}
env={}
```

If you have implemented a more complex command line argument scheme, all of the information passed to `main` will be printed out.

If there are no errors detected, no more output will be produced. Whenever an error is detected a message is printed out on standard error.

This test program is not exhaustive, but may prove helpful.

1.6. Implicit Library Assumptions

This C Library implementation makes certain assumptions about its environment. If these assumptions are invalid this C Library cannot be expected to work without modifications.

If your machine does not conform to these constraints you will probably have some problems with the library. We are interested in hearing about (and trying to fix) any problems you may have in running this library on other types of machines.

This C Library has been implemented to work with the Green Hills Software C compilers. It is not intended that anything in the library would prohibit it from working with other C compilers, but it has never been tested with any other compiler, and under Green Hills Software licensing policies it is only licensed to be used with Green Hills Software compilers. The only Green Hills Software specific aspect of the library is that the “register” declaration has been used less often than is normal in C programs. This is because the Green Hills Software compiler allocates variables to registers even if “register” is not specified. It is recommended that “register” not be used so that the compiler has the greatest freedom to optimize the code. But old habits die hard and there are still “register” declarations in the code.

The Green Hills Software C Library implicitly assumes several things about the machine it is running on. It assumes that it is a byte addressed machine. That is, there are 8 bits in a char, and `sizeof(char)` equals 1. It assumes there are 16 bits in a short, and `sizeof(short)` equals 2. It assumes there are 32 bits in a long, and `sizeof(long)` equals 4. It should work if `sizeof(int)==sizeof(short)` or if `sizeof(int)==sizeof(long)`. However, it has never been tested when `sizeof(int)==sizeof(short)`. The library works on both a Little-Endian architecture (least significant byte of a multiple byte integer is stored at the lowest address) and a Big-Endian architecture (most significant byte of a multiple byte integer is stored at the lowest address).

The library assumes two's complement integer arithmetic.

The library assumes (but only in a few places, such as `ctype.c`) that the character set is `ascii`.

The library assumes that `sizeof(float)==4` and `sizeof(double)==8`. It assumes that floating point exponents are radix 2 (as opposed to radix 10). It assumes that exponents are not less than 8 bits long and not greater than 11 bits long. It assumes that there is at most 56 bits of precision in double precision floating point, and at most 24 bits of precision in single precision floating point. The library also assumes that if $x \neq 0.0$ then $1/x$ will not overflow. IEEE floating point with gradual underflow and trap on overflow will fail this assumption!

Chapter 2

C Library Function Groups

This chapter is a list of the functions and variables implemented by the Green Hills Software C Library organized by function groups. The C library source file name is given in parentheses at the end of the line.

Math functions

abs:	Integer absolute value (abs.c)
cabs:	Complex absolute value (cabs.c)
fabs:	Floating point absolute value (fabs.c)
ffs:	Find first set bit (ffs.c)
frexp:	Floating point return exponent and mantissa (frexp.c)
labs:	Long absolute value (labs.c)
ldexp:	Load exponent (ldexp.c)
modf:	Return integer and fractional part of floating point (modf.c)

Allocation

calloc:	C allocate memory (calloc.c)
cfree:	C free memory (cfree.c)
free:	Free memory (malloc.c)
malloc:	Memory allocate (malloc.c)
realloc:	Reallocate memory (malloc.c)

Formatted I/O

atof:	Ascii to floating conversion (atof.c)
atoi:	Ascii to integer conversion (atoi.c)
atol:	Ascii to long conversion (atol.c)
ecvt:	E format conversion (ecvt.c)
fprintf:	Print formatted output to standard error (printf.c)
fcvt:	F format conversion (ecvt.c)
fprintf:	Print formatted output to a FILE (printf.c)
fscanf:	Read formatted input from a FILE (scanf.c)
gcvt:	G Format conversion (gcvt.c)
perror:	Print error message (perror.c)
printf:	Print formatted output to standard output (printf.c)
scanf:	Read formatted input from standard input (scanf.c)
sprintf:	Print formatted output to a string (printf.c)
sscanf:	Read formatted input from a string (scanf.c)
strtol:	String to long conversion (strtol.c)
sys_errlist:	System error list (syserrlst.c)

FILE I/O

<code>_cleanup:</code>	Clean up (flush) all file buffers (<code>cleanup.c</code>)
<code>clearerr:</code>	Clear error indicator on a FILE (<code><stdio.h></code>)
<code>EOF:</code>	End of FILE character value (<code><stdio.h></code>)
<code>exit:</code>	Close all files and exit from program (<code>exit.c</code>)
<code>fclose:</code>	FILE Close (<code>fclose.c</code>)
<code>fdopen:</code>	FILE open with descriptor (<code>fopen.c</code>)
<code>feof:</code>	End of FILE query (<code><stdio.h></code>)
<code>ferror:</code>	Error on FILE query (<code><stdio.h></code>)
<code>fflush:</code>	FILE flush buffer (<code>fflush.c</code>)
<code>fgetc:</code>	FILE get character (<code>fgetc.c</code>)
<code>fgets:</code>	FILE get string (<code>fgets.c</code>)
<code>_filbuf:</code>	Fill FILE input buffer (<code>filbuf.c</code>)
<code>FILE:</code>	FILE type definition (<code><stdio.h></code>)
<code>fileno:</code>	I/O channel number associated with a FILE (<code><stdio.h></code>)
<code>_flsbuf:</code>	Flush FILE output buffer (<code>flsbuf.c</code>)
<code>fopen:</code>	FILE open (<code>fopen.c</code>)
<code>fputc:</code>	FILE put character (<code>fputc.c</code>)
<code>fputs:</code>	FILE put string (<code>fputs.c</code>)
<code>fread:</code>	FILE read (<code>fread.c</code>)
<code>freopen:</code>	FILE re-open (<code>fopen.c</code>)
<code>fseek:</code>	FILE seek (<code>fseek.c</code>)
<code>ftell:</code>	FILE tell position (<code>ftell.c</code>)
<code>fwrite:</code>	FILE write (<code>fwrite.c</code>)
<code>getc:</code>	Get a character from file (<code><stdio.h></code>)
<code>getchar:</code>	Get character from standard input FILE (<code><stdio.h></code>)
<code>getchar:</code>	Get character from standard input FILE (<code>getchar.c</code>)
<code>getl:</code>	Get long from FILE (<code>getl.c</code>)
<code>gets:</code>	Get string from FILE (<code>gets.c</code>)
<code>getw:</code>	Get int from FILE (<code>getw.c</code>)
<code>NULL:</code>	Null FILE and pointer value (<code><stdio.h></code>)
<code>putc:</code>	Output a character to a FILE (<code><stdio.h></code>)
<code>putchar:</code>	Output a character to standard output FILE (<code><stdio.h></code>)
<code>putchar:</code>	Output a character to standard output FILE (<code>putchar.c</code>)
<code>putl:</code>	Output a long to a FILE (<code>putl.c</code>)
<code>puts:</code>	Output a string to standard output FILE (<code>puts.c</code>)
<code>putw:</code>	Output a word (int) to a FILE (<code>putw.c</code>)
<code>rewind:</code>	Seek to beginning of a FILE (<code>rewind.c</code>)
<code>setbuf:</code>	Set a user I/O buffer for a FILE (<code>setbuf.c</code>)
<code>setlinebuf:</code>	Set line buffering on a FILE (<code>setlinbuf.c</code>)
<code>stderr:</code>	Standard error file (<code><stdio.h></code>)
<code>stdin:</code>	Standard input file (<code><stdio.h></code>)
<code>stdout:</code>	Standard output file (<code><stdio.h></code>)
<code>ungetc:</code>	Put a character back on a FILE (<code>ungetc.c</code>)

Character types

ctype: Character type array (ctype.c)
 isalnum: Is alphanumeric (<ctype.h>)
 isalpha: Is alphabetic character (<ctype.h>)
 isascii: Is ascii character (<ctype.h>)
 iscntrl: Is control character (<ctype.h>)
 iscsym: Is C identifier character (<ctype.h>)
 iscsymf: Is C identifier initial character (<ctype.h>)
 isdigit: Is decimal digit (<ctype.h>)
 isgraph: Is a graphics character (<ctype.h>)
 islower: Is lower case ascii character (<ctype.h>)
 isprint: Is printable character (<ctype.h>)
 ispunct: Is punctuation character (<ctype.h>)
 isspace: Is white space character (<ctype.h>)
 isupper: Is upper case ascii character (<ctype.h>)
 isxdigit: Is hexadecimal digit (<ctype.h>)
 toascii: Convert character to ascii (<ctype.h>)
 tolower: Convert character to lower case (<ctype.h>)
 _tolower: Convert character to lower case (<ctype.h>)
 toupper: Convert character to upper case (<ctype.h>)
 _toupper: Convert character to upper case (<ctype.h>)

String functions

index: Return the index of a character in a string (index.c)
 rindex: Reverse search for character in string (strchr.c)
 strcat: String concatenate (strcat.c)
 strchr: Return the position of a character in a string (strchr.c)
 strcmp: String Compare (strcmp.c)
 strcpy: String Copy (strcpy.c)
 strindex: String index (strindex.c)
 strlen: String length (strlen.c)
 strncat: String concatenate with maximum length (strncat.c)
 strncmp: String compare with maximum length (strncmp.c)
 strncpy: String copy with maximum length (strncpy.c)
 strrchr: Reverse search for character in string (strchr.c)
 strrindex: Reverse search for string in string (strrindex.c)
 strsave: String save (strsave.c)

Byte strings

bcmp: Byte string compare (bcmp.c)
 bcopy: Byte string copy (bcopy.c)
 bufcpy: Buffer copy (bufcpy.c)
 bzero: Byte string zero (bzero.c)
 filln: Fill n bytes of memory (filln.c)
 clearn: Clear n bytes of memory (clearn.c)
 swab: Swap byte string (swab.c)

Miscellaneous

args: Return argument list, argument count, and environment (os.c)
errno: Error number (errno.c)
Init_IO: Initialize I/O (os.c)
INPUT: Input a character from the controlling terminal (os.c)
mktemp: Make temporary name (mktemp.c)
OUTPUT: Output a character to the controlling terminal (os.c)
rand: Return a random integer (rand.c)
srand: Initialize random number generator used in rand() (rand.c)
START: Program transfer address (crt0.c)

Unix system call functions

access: Check accessibility of file name (os.c)
close: Close an I/O channel (os.c)
creat: Create file and open an I/O channel to it (os.c)
_exit: Exit Program (os.c)
fstat: Get device and file number connected to I/O channel (os.c)
ftruncate: Truncate file connected to I/O channel (os.c)
getpid: Get Process ID (os.c)
isatty: Is a tty connected to I/O channel (os.c)
lseek: Long seek on an I/O channel (os.c)
open: Open an I/O channel (os.c)
read: Read from an I/O channel (os.c)
sbrk: Set memory address break (os.c)
stat: Get device and file number of file name (os.c)
unlink: Unlink (Delete) file (os.c)
write: Write to an I/O channel (os.c)

Chapter 3

Alphabetized List of C Library Functions

This chapter is an alphabetized list of the functions, macros, and variables exported by the Green Hills Software C Library. The source file containing the definition is given with each function in parentheses.

abs: Integer absolute value (abs.c)

```
int abs(x)
int x;
/*****
/* Return the absolute value of "x".
*****/
```

access: Check accessibility of file name (os.c)

```
access(name, prot)
char *name;
/*****
/* The Green Hills Software C Library does not use the access function, it is only
/* needed if the Green Hills Software Fortran Library is being used.
/*
/* "name" is the name of a file, "prot" is the logical or of the following flags
/*
/*      4: readable
/*      2: writable
/*      1: executable or directory searchable (Unused)
/* If "prot" is zero then test only for file existence.
/* Return 0 if all of the specified operations are possible on the file.
/* Return -1 if any of the specified operations are not possible on the file
/* and set errno appropriately.
*****/
```

args: Return argument list, argument count, and environment (os.c)

```
char **args(argc, envp)
int *argc;
char ***envp;
/*****
/* Parse a UNIX style argument list. Return a pointer to a NULL terminated
/* array of null terminated strings. Each string in "*envp" corresponds
/* to one blank or tab terminated argument on the command line. The first
/* argument is the name of the command. The number of arguments is returned
/* in "*argc". "*envp" is a pointer to a NULL terminated array of null terminated
/* strings. Each string in the array corresponds to environment variables
/* passed down from the invoking program (such as the UNIX shell).
*****/
```

```

atof:  Ascii to floating conversion (atof.c)
       double atof(str)
       register char *str;
       /***/
       /* Convert a null terminated ascii string to a "double" and return it.
       /***/

atoi: Ascii to integer conversion (atoi.c)
       int atoi(str)
       char *str;
       /***/
       /* Convert a null terminated ascii string to an "int" and return it.
       /***/

atol:  Ascii to long conversion (atol.c)
       long atol(str)
       char *str;
       /***/
       /* Convert a null terminated ascii string to a long and return it.
       /***/

bcmp:  Byte string compare (bcmp.c)
       bcmp(b1, b2, length)
       register char *b1;
       register char *b2;
       /***/
       /* Compare two character strings of length, length. Return a negative
       /* number if "b1" < "b2"; a 0 if "b1" == "b2"; and a positive number if
       /* "b1" > "b2". The machine dependent character ordering is used.
       /* Null ( ' ' ) is considered as an ordinary character.
       /***/

bcopy: Byte string copy (bcopy.c)
       bcopy(from, to, n)
       register char *to;
       register char *from;
       register int n;
       /***/
       /* Copy n bytes from the address, "from", to the address, "to".
       /***/

bufcpy: Buffer copy (bufcpy.c)
       bufcpy(to, from, n)
       register char *to;
       register char *from;
       register int n;
       /***/
       /* Copy "n" bytes from the address, "from", to the address, "to".
       /***/

```

bzero: Byte string zero (bzero.c)

```
bzero(n, pt)
register int n;
register char *pt;
/*****
/* Store zeros in the “n” bytes pointed to by “pt”.
*****/
```

calloc: C allocate memory (calloc.c)

```
char *calloc(num, size)
unsigned num, size;
/*****
/* Allocate a block of zero filled memory large enough to hold “num”
/* items of size, “size”. Return a pointer to the new block of memory.
*****/
```

cfree: C free memory (cfree.c)

```
cfree(item)
char *item;
/*****
/* Free a block of memory allocated by calloc(). The memory is no
/* longer available after it is freed. It is imperative that there
/* exist no active pointers to the memory after it is freed.
*****/
```

_cleanup: Cleanup file buffers (cleanup.c)

```
_cleanup()
/*****
/* fclose() all opened FILEs.
*****/
```

clearerr: Clear error indicator on FILE (<stdio.h>)

```
clearerr(file)
FILE *file;
/*****
/* Macro to clear the error indicator returned by ferror() for the FILE, “file”.
*****/
```

clearn: Clear n bytes of memory (clearn.c)

```
clearn(n, pt)
register int n;
register char *pt;
/*****
/* Store zeros in the “n” bytes pointed to by “pt”.
*****/
```

close: Close an I/O channel (os.c)

```
close(fno)
/*****
/* Close the file associated with the file number “fno” (returned by open()
/* or creat()). Return 0 if all goes well. Return -1 if something went
/* wrong and set errno appropriately.
*****/
```

creat: Create file and open an I/O channel to it (os.c)

```
creat(filename, prot)
char *filename;
/*****
/* Create and open for writing a file named “filename” with protection
/* as specified by “prot”. “filename” is a null terminated string.
/* “prot” is expressed in UNIX format, it is the logical or of:
/*
/*      0400: owner read
/*      0200: owner write
/*      0100: owner execute/search directory
/*      0040: group read
/*      0020: group write
/*      0010: group execute/search directory
/*      0004: other (world) read
/*      0002: other (world) write
/*      0001: other (world) execute/search directory
/*
/* For things like terminals that cannot be created it should just open the
/* device. If successful, return a small integer file number. On failure
/* return -1 and set errno appropriately.
*****/
```

ctype: Character type array (ctype.c)

```
unsigned char _ctype_[];
/*****
/* _ctype_ defines the character types for the <ctype.h> character type macros.
*****/
```

_doprnt: Print variable argument list formatted output to file (printf.c)

```
_doprnt(format, args, stream)
char *format;
va_list *args;
FILE *stream;
/*****
/* Write formatted output to the output file, “stream”. The
/* format specification is given by the null terminated string, “format”.
/* The output variables are obtained from the variable argument list, “args”,
/* by the varargs mechanism. Normally only used by printf, sprintf, fprintf,
/* and eprintf.
*****/
```

`_doscan`: Read variable argument list formatted input from a FILE (scanf.c)

```
_doscan(format, args, stream)
char *format;
va_list *args;
FILE *stream;
/*****/
/* Read formatted input from the input FILE, "stream". The format
/* specification is given by the null terminated string, "format".
/* The input variables are obtained from the variable argument list, "args",
/* by the varargs mechanism. Normally only used by scanf, sscanf, and fscanf.
/*****/
```

`ecvt`: E format conversion (ecvt.c)

```
char *ecvt(value, ndig, decpt, sign)
double value;
int ndig;
int *decpt;
int *sign;
/*****/
/* Convert the floating point value, "value", to an ascii string and return a
/* pointer to the string. Generate a properly rounded digit string for "ndig"
/* digits of precision in ("%e") format. Leading zeros may be suppressed.
/* The decimal point position is specified by "*decpt" as relative to the first
/* character of the returned string. If "*decpt" is zero the decimal point is
/* immediately to the left of the first character of the returned string. If
/* "*decpt" is positive, the decimal point is to the left of the character
/* numbered "*decpt" (the first character is numbered zero). If "*decpt" is
/* negative, leading zeros have been suppressed and the decimal point occurs
/* "*decpt" characters to the left of the first character of the string. "*sign"
/* is set to non-zero if "value" is negative, otherwise "*sign" is set to 0.
/*****/
```

`EOF`: End of FILE character value (<stdio.h>)

```
#define EOF (-1)
/*****/
/* The end of FILE character returned by getc() when
/* the end of file is reached.
/*****/
```

`eprintf`: Print formatted output to standard error (printf.c)

```
int fprintf(format, va_alist)
char *format;
va_dcl
/*****/
/* Write formatted output to the standard error file. The format specification
/* is given by the null terminated string, "format". The output variables are
/* obtained from the variable argument list by the varargs mechanism. Return
/* the number of characters output.
/*****/
```

errno: Error number (errno.c)

```
int errno;
/*****
/* The error number global variable. It is initialized to zero. Whenever
/* an error occurs the error number, as defined in <errno.h> and sys_errlist.c
/* is stored into errno. errno is not cleared when no error occurs!
*****/
```

_exit: Exit Program (os.c)

```
_exit(code)
/*****
/* Exit from the program with a status code specified by code.
/* DO NOT RETURN!
*****/
```

exit: Close all files and exit from program (exit.c)

```
exit(val)
int val;
/*****
/* Call _cleanup() to flush all files. Then call _exit() to exit the
/* program. Never returns.
*****/
```

fabs: Floating point absolute value (fabs.c)

```
double fabs(x)
double x;
/*****
/* Return the absolute value of x.
*****/
```

fclose: FILE Close (fclose.c)

```
fclose(file)
register FILE *file;
/*****
/* Flush out any data in the buffer corresponding to the FILE, "file", out
/* to the I/O channel corresponding to "file". Then close "file".
/* Returns 0 if the close is successful, -1 if the close fails.
*****/
```

fcvt: F format conversion (ecvt.c)

```
char *fcvt(value, ndig, decpt, sign)
double value;
int ndig;
int *decpt;
int *sign;
/*****/
/* Convert the floating point value, "value", to an ascii string and return a
/* pointer to the string. Generate a properly rounded digit string for "ndig"
/* digits to the right of the decimal point in fixed point ("%f") format.
/* Leading zeros may be suppressed. The decimal point position is specified by
/* "*decpt" as relative to the first character of the returned string. If "*decpt"
/* is zero the decimal point is immediately to the left of the first character
/* of the returned string. If "*decpt" is positive, the decimal point is to the
/* left of the character numbered "*decpt" (the first character is numbered zero).
/* If "*decpt" is negative, leading zeros have been suppressed and the decimal
/* point occurs "-*decpt" characters to the left of the first character of the
/* string. "*sign" is set to non-zero if "value" is negative, otherwise "*sign" is
/* set to 0.
/*****/
```

fdopen: FILE open with descriptor (fopen.c)

```
FILE *fdopen(fno, mode)
char *mode;
/*****/
/* Open and return a FILE associated with the I/O channel, "fno", with
/* the access mode specified by the null terminated string, "mode" (as
/* described below in fopen()).
/*****/
```

feof: End of FILE query (<stdio.h>)

```
feof(file)
FILE *file;
/*****/
/* Return non-zero if the FILE, "file" is at the end of the file.
/*****/
```

ferror: Error on FILE query (<stdio.h>)

```
ferror(file)
FILE *file;
/*****/
/* Return non-zero if an error has occurred on the FILE, "file".
/*****/
```

fflush: FILE flush buffer (fflush.c)

```
fflush(file)
register FILE *file;
/*****/
/* Flush out any data in the buffer corresponding to the FILE, "file", out
/* to the I/O channel corresponding to "file".
/*****/
```

ffs: Find first set bit (ffs.c)

```
int ffs(i)
int i;
/*****
/* Return the number of the least significant bit in "i" that is a one (counting
/* the least significant bit as 1!). If "i" is zero, return -1.
*****/
```

fgetc: FILE get character (fgetc.c)

```
fgetc(file)
register FILE *file;
/*****
/* Read and return a character from the input FILE, "file". Returns EOF on
/* end of file. fgetc() is identical in operation to getc(), except that
/* it is a function instead of a macro.
*****/
```

fgets: FILE get string (fgets.c)

```
char *fgets(str, n, file)
char *str;
int n;
register FILE *file;
/*****
/* Read a new line terminated line from the FILE, "file". Place up to the first
/* n-1 characters from the line, including the new line character into the
/* buffer, "str". Terminate the line with a null character. Return "str".
*****/
```

_filbuf: Fill FILE input buffer (filbuf.c)

```
_filbuf(file)
register FILE *file;
/*****
/* Fill the buffer associated with the FILE, "file", from the corresponding
/* I/O channel. The buffer must be empty when this operation is done.
/* Returns the next character from "file" or EOF if the end of file
/* is reached. Normally this function is only called from the getc() macro.
*****/
```

FILE: FILE type definition (<stdio.h>)

```
#define FILE struct iobuf
/*****
/* The type definition for the FILE data type.
*****/
```

fileno: Channel number associated with a FILE (<stdio.h>)

```
fileno(file)
FILE *file;
/*****
/* Macro to return the channel number associated with the FILE, "file".
*****/
```

filln: Fill n bytes of memory (filln.c)

```
filln(n, pt, fill)
register int n;
register char *pt;
register char fill;
/*****
/* File the n bytes of memory pointed to by "pt" with the character, "fill".
*****/
```

_flsbuf: Flush out FILE output buffer (flsbuf.c)

```
_flsbuf(ch, file)
register FILE *file;
char ch;
/*****
/* Output the character "ch" to the FILE, "file". Then output any data stored in
/* the buffer associated with the FILE to the corresponding I/O channel.
/* Normally this function is only called by the putc() macro.
*****/
```

fopen: FILE open (fopen.c)

```
FILE *fopen(name, mode)
char *name, *mode;
/*****
/* Open the file named by the null terminated string, "name", with the
/* access mode specified by the null terminated string, "mode". The access
/* modes are "r", "w", "a", "r+", "w+", "a+". "r" specifies read only.
/* "w" specifies write only and that the file is to be created or replaced.
/* "a" specifies write only and that the file is to be created if doesn't
/* exist or if it exists it is to be appended to. "r+" specifies that the file
/* must exist, that it may be read or written, and that it is to be initially
/* opened at the beginning. "w+" specifies that the file is to be created or
/* replaced but that it may either read or written. "a+" specifies that the
/* file is to be created if it does not exist. If it does exist it is to be
/* opened such that the first operation will occur at the end of the file.
/* The file may be either read or written.
*****/
```

fprintf: Print formatted output to file (printf.c)

```
int fprintf(stream, format, va_alist)
FILE *stream;
char *format;
va_dcl
/*****
/* Write formatted output to the output file, "stream". The
/* format specification is given by the null terminated string, "format".
/* The output variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of characters output.
*****/
```

fputc: FILE put character (fputc.c)

```
fputc(ch,file)
int ch;
register FILE *file;
/*****/
/* Output the character, ch, to the output FILE, "file". fputc is identical
/* in operation to putc, except that it is a function instead of a macro.
/*****/
```

fputs: FILE put string (fputs.c)

```
fputs(str,file)
register char *str;
register FILE *file;
/*****/
/* Output the null terminated string, "str", to the output FILE, "file".
/* The terminating null character is not written to stdout.
/*****/
```

fread: FILE read (fread.c)

```
fread(pt,size,nitems,file)
register char *pt;
register FILE *file;
/*****/
/* Read "nitems" items of size, "size", beginning at the pointer, "pt", to the
/* FILE, "file". Return the number of items actually read. Return 0
/* at end of file.
/*****/
```

free: Free memory (malloc.c)

```
free(pt)
char *pt;
/*****/
/* Free a block of memory allocated by malloc(). The memory is no
/* longer available after it is freed. It is imperative that there
/* exist no active pointers to the memory after it is freed.
/*****/
```

freopen: FILE re-open (fopen.c)

```
FILE *freopen(name, mode, file)
char *name, *mode;
FILE *file;
/*****/
/* If the FILE, "file", is open, close it. Then open the file named
/* by the null terminated string, "name", with the access mode,
/* "mode" (as specified in fopen above), on the FILE, "file".
/*****/
```

frexp: Floating point return exponent and mantissa (frexp.c)

```
double frexp(value, eptr)
double value;
int *eptr;
/*****/
/* Return the mantissa of the floating point value, "value". Return the
/* binary radix exponent value in "*iptr". The return value is chosen
/* such that 0.5 <= frexp(value, &iptr) < 1.0 and such that
/* value == frexp(value, &iptr) * 2 ** iptr
/*****/
```

fseek: FILE seek (fseek.c)

```
fseek(stream, offset, ptrname)
long offset;
FILE *stream;
/*****/
/* Seek to a new position within the file, "stream". If "ptrname" is 0, seek to
/* offset bytes from the beginning of the file. If "ptrname" is 1, seek to
/* offset bytes from the current position in the file. If "ptrname" is 2 seek
/* to offset bytes from the end of the file. fseek does no I/O. The next
/* I/O operation to the file, "stream", will begin at the new position in the
/* file. fseek() undoes the effect of any ungetc().
/*
/* Return 0 if the seek is successful. If an error occurs return -1.
/*****/
```

fscanf: Read formatted input from a FILE (scanf.c)

```
fscanf(stream, format, va_alist)
FILE *stream;
char *format;
va_dcl
/*****/
/* Read formatted input from the input FILE, "stream". The format
/* specification is given by the null terminated string, "format".
/* The input variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of input variables assigned
/* to.
/*****/
```

fstat: Get device and file number connected to I/O channel (os.c)

```
fstat(fno, statptr)
struct stat *statptr;
/*****/
/* The Green Hills Software C Library does not uses the fstat function, it is only
/* needed if the Green Hills Software Fortran Library is being used.
/*
/* “fno” is a file number returned by open() or creat(). In the two fields
/* st_dev and st_ino place values which uniquely identify the file or device
/* opened on file number “fno”. In UNIX these are the device number and the
/* inode (file) number on the device.
/*
/* Return 0 if the file status is correctly returned. Return -1 if no file
/* is opened on file number “fno”, and set errno appropriately.
/*****/
```

ftell: FILE tell position (ftell.c)

```
long ftell(stream)
FILE *stream;
/*****/
/* Return the current position from the beginning of the FILE, “stream”.
/*****/
```

ftruncate: Truncate file connected to I/O channel (os.c)

```
ftruncate(fno, length)
/*****/
/* The Green Hills Software C Library does not uses the ftruncate function, it is only
/* needed if the Green Hills Software Fortran Library is being used.
/*
/* “fno” is a file number returned by open() or creat(). The file number “fno”
/* must be opened for writing. If the file opened for writing on file number
/* “fno” is longer than “length” bytes, it is truncated to “length” bytes.
/*
/* Return 0 if the named file is truncated or was previously less than
/* length bytes. Return -1 if the file is not open for writing or if the
/* file cannot be truncated, and set errno appropriately.
/*****/
```

fwrite: FILE write (fwrite.c)

```
fwrite(pt,size,nitems,file)
register char *pt;
register FILE *file;
/*****/
/* Write “nitems” items of size, size, beginning at the pointer, pt, to the
/* FILE, “file”. Return the number of items actually written.
/*****/
```

gcvvt: G Format conversion (gcvvt.c)
char *gcvvt(value, ndig, buf)
double value;
int ndig;
char *buf;
/*****
/* Convert the floating point value, "value", to an ascii string in the
/* buffer, "buf". If possible generate "ndig" digits after the decimal
/* point in "%f" format. If not possible generate in "%e" format.
/* Suppress trailing zeros. Return a pointer to the string.
*****/

getc: Get a character from a FILE (<stdio.h>)
getc(f)
FILE *f
/*****
/* A macro to read and return a character from the input FILE, "file".
/* Returns EOF on end of file.
*****/

getchar: Get a character from the standard input FILE (<stdio.h>)
getchar()
/*****
/* A macro to read and return a character from the standard input FILE, stdin.
/* Returns EOF on end of file. It has the same effect as the getchar()
/* function below, but it takes more code space, but less execution time.
*****/

getchar: Get a character from the standard input FILE (getchar.c)
getchar()
/*****
/* A function to read and return a character from the standard input
/* FILE, stdin. Returns EOF on end of file. It has the same effect as the
/* getchar() macro above, but it takes less code space, but more execution
/* time.
/*
/* If you have included <stdio.h>, in order to use this function you must
/* place a line containing "#undef getchar" before the first use. From then
/* on the getchar() macro will be inaccessible.
*****/

getl: Get long from FILE (getl.c)
long getl(file)
register FILE *file;
/*****
/* Read and return a "long" from the FILE, "file".
*****/

getpid: Get Process ID (os.c)

```

getpid()
/*****
/* getpid returns the current process number.  getpid is used to create
/* filenames which will not conflict with filenames created by another
/* process at the same time.
*****/

```

gets: Get string from the standard input FILE (gets.c)

```

char *gets(str)
char *str;
/*****
/* Read a new line terminated string from the standard input FILE, stdin.
/* Return a pointer to a string containing the line from the file with
/* the new line changed to a null ('\0') character.
*****/

```

getw: Get int from FILE (getw.c)

```

getw(file)
register FILE *file;
/*****
/* Read and return an "int" from the FILE, "file".
*****/

```

Init_IO: Initialize I/O (os.c)

```

Init_IO()
/*****
/* Init_IO is called from START() to initialize the I/O system.
/*
/* Open file numbers 0, 1, and 2 like the UNIX shell.
/*
/* File number 0 is the standard input file.  It is usually the controlling
/* terminal for the program or a sequential character input file.
/*
/* File number 1 is the standard output file.  It is usually the controlling
/* terminal for the program or a sequential character output file.
/*
/* File number 2 is the standard error file.  It is usually the controlling
/* terminal for the program.  Error messages are usually written on this
/* file rather than standard output so that if the standard output is a file,
/* the error message can still be seen by the person running the program.
/* In batch jobs, file number 2 is usually a sequential character output file.
/*
/* On non-UNIX systems "/dev/tty" should be changed to whatever is appropriate
/* ("SYS$INPUT": on VMS, "TTY": on tops-10, etc.).
/*
/* If these files are already opened by the operating system, or by the UNIX
/* shell, delete the calls to open.
*****/

```

INPUT: Return character input from the controlling terminal (os.c)

```
int INPUT()
/*****
/* Read a character from the controlling terminal and return it.
/*
/* If you want to remain compatible with UNIX stdio and the Green Hills Software
/* C Test Program you must convert carriage returns ('\r') to line feeds ('\n').
*****/
```

isalnum: Is alphanumeric (<ctype.h>)

```
isalnum(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is an alphanumeric character. Returns 0 otherwise.
*****/
```

isalpha: Is alphabetic character (<ctype.h>)

```
isalpha(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is an alphabetic character. Returns 0 otherwise.
*****/
```

isascii: Is ascii character (<ctype.h>)

```
isascii(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is an ascii character. Returns 0 otherwise.
*****/
```

isatty: Is a tty connected to I/O channel (os.c)

```
isatty(fno)
/*****
/* Return 1 if the file number "fno" (returned by open() or creat())
/* is connected to a terminal, otherwise return 0.
*****/
```

isctrl: Is control character (<ctype.h>)

```
isctrl(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is a control character. Returns 0 otherwise.
*****/
```

isctype: Is C identifier character (<ctype.h>)

```
isctype(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" could be a character of
/* a C identifier. Returns 0 otherwise.
*****/
```

iscsymf: Is C identifier initial character (<ctype.h>)

```
iscsymf(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” could be the initial character of
/* a C identifier. Returns 0 otherwise.
*****/
```

isdigit: Is decimal digit (<ctype.h>)

```
isdigit(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a decimal digit. Returns 0 otherwise.
*****/
```

isgraph: Is a graphics character (<ctype.h>)

```
isgraph(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a graphics character. Returns 0 otherwise.
*****/
```

islower: Is lower case ascii character (<ctype.h>)

```
islower(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a lower case letter. Returns 0 otherwise.
*****/
```

isprint: Is printable character (<ctype.h>)

```
isprint(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a printable character. Returns 0 otherwise.
*****/
```

ispunct: Is punctuation character (<ctype.h>)

```
ispunct(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a punctuation character. Returns 0 otherwise.
*****/
```

isspace: Is white space character (<ctype.h>)

```
isspace(ch)
char ch;
/*****
/* Macro which returns non-zero if “ch” is a white space character. Returns 0 otherwise.
*****/
```

isupper: Is upper case ascii character (<ctype.h>)

```
isupper(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is an upper case letter. Returns 0 otherwise.
*****/
```

isxdigit: Is hexadecimal digit (<ctype.h>)

```
isxdigit(ch)
char ch;
/*****
/* Macro which returns non-zero if "ch" is a hexadecimal digit. Returns 0 otherwise.
*****/
```

index: Return the index of a character in a string (index.c)

```
char *index(str, ch)
register char *str;
register char ch;
/*****
/* Return a pointer to the first instance of the character in the string, "str" (or
/* a 0 if the character is not there. Note that this will work if the
/* character is NUL. Identical to strchr().
*****/
```

labs: Long absolute value (labs.c)

```
long labs(x)
long x;
/*****
/* Return the absolute value of x.
*****/
```

ldexp: Load exponent (ldexp.c)

```
double ldexp(value, exp)
double value;
/*****
/* Return "value" * 2 ** "exp". That is, return "value" with the binary exponent
/* incremented by "exp".
*****/
```

lseek: Long seek on an I/O channel (os.c)

```
lseek(fno, offset, end)
long offset;
/*****/
/* Seek to a new position within the file connected to the file number "fno"
/* (returned by open() or creat()). If "end" is 0, seek to offset bytes from
/* the beginning of the file. If "end" is 1, seek to offset bytes from the
/* current position in the file. If "end" is 2 seek to offset bytes from
/* the end of the file. lseek does no I/O. The next I/O operation to
/* file number "fno" will begin at the new position in the file.
/*
/* Return the offset from the beginning of the file after the seek takes
/* place. If an error occurs return -1 and set errno appropriately.
/* If an error occurs the file position is not changed.
/*****/
```

malloc: Memory allocate (malloc.c)

```
char *malloc(siz)
unsigned siz;
/*****/
/* Allocate and return a new data area of size, "siz". If no more memory
/* is available return NULL.
/*****/
```

mktemp: Make temporary name (mktemp.c)

```
char *mktemp(str)
char *str;
/*****/
/* The null terminated input string, "str", must have 'X's as its last six
/* characters. These characters are replaced by a letter and the process
/* id returned by getpid(). Return "str".
/*****/
```

modf: Modulo 1 floating point (modf.c)

```
double modf(value, iptr)
double value;
double *iptr;
/*****/
/* Split "value" into its integer and fractional parts, such that
/* 0 <= modf(value, iptr) < 1.0 && (value == modf(value, iptr) + iptr)
/*
/* *iptr = floor(x) and return (x - *iptr)
/*****/
```

NULL: Null pointer value (<stdio.h>)

```
#define NULL 0
/*****/
/* The NULL pointer value. Also used as the NULL FILE value.
/*****/
```

open: Open an I/O channel (os.c)

```
open(filename, mode)
char *filename;
/*****/
/* Open the file named "filename" with the indicated mode.
/* "filename" is a null terminated string and "mode" is one of the following
/*
/* 0      => open for reading
/* 1      => open for writing
/* 2      => open for reading & writing
/*
/* If the open is successful return a small integer file number.
/* On failure return -1, and set errno in accordance with the error.
/*****/
```

OUTPUT: Output a character to the controlling terminal (os.c)

```
OUTPUT(ch)
/*****/
/* Write the character "ch" to the controlling terminal.
/*
/* If you want to remain compatible with UNIX stdio you must convert a
/* line-feed ('\n') to a carriage-return ('\r') and a line-feed ('\n').
/*****/
```

perror: Print error message (perror.c)

```
perror(str)
char *str;
/*****/
/* Print on the standard error file, stderr, the null terminated string, "str",
/* followed by the string ":", followed by the error message corresponding
/* to the current value of global "int" variable, errno. The error message
/* list is specified by the array sys_errlist defined in sys_errlist.c.
/*****/
```

printf: Print formatted output to standard output (printf.c)

```
int printf(format, va_alist)
char *format;
va_dcl
/*****/
/* Write formatted output to the standard output file, stdout. The
/* format specification is given by the null terminated string, "format".
/* The output variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of characters output.
/*****/
```

putc: Output a character to a FILE (<stdio.h>)

```
putc(ch,f)
char ch;
FILE *f;
/*****/
/* A macro to output the character, "ch", to the output FILE, "f".
/*****/
```

putchar: Output a character to the standard output FILE (<stdio.h>)

```

putchar(ch)
/*****
/* A macro to output the character, "ch", to the standard output
/* FILE, stdout. It has the same effect as the putchar() function below, but
/* it takes more code space, but less execution time.
*****/

```

putchar: Output a character to the standard output FILE (putchar.c)

```

putchar(ch)
/*****
/* A function to output the character, "ch", to the standard output
/* FILE, stdout. It has the same effect as the putchar() macro above, but
/* it takes less code space, but more execution time.
/*
/* If you have included <stdio.h>, in order to use this function you must
/* place a line containing "#undef putchar" before the first use. From then
/* on the putchar() macro will be inaccessible.
*****/

```

putl: Output a long to a FILE (putl.c)

```

long putl(l, file)
long l;
register FILE *file;
/*****
/* Output the long, "l", to the FILE, "file". Return "l".
*****/

```

puts: Output a string to the standard output FILE (puts.c)

```

puts(str)
register char *str;
/*****
/* Output the null terminated string, "str", to the standard output FILE,
/* stdout. The terminating null character is not written to stdout, in
/* its place a new line ('\n') is written.
*****/

```

putw: Output a word (int) to a FILE (putw.c)

```

putw(w, file)
int w;
register FILE *file;
/*****
/* Output the int, "w", to the FILE, "file".
*****/

```

rand: Random integer (rand.c)

```

int rand()
/*****
/* Return a random positive integer.
*****/

```

read: Read from an I/O channel (os.c)

```
read(fno, buf, size)
char *buf;
/*****/
/* Read at most "size" bytes into "buf" from the file connected to "fno" (where
/* "fno" is one of the file numbers returned by open() or creat()).
/* Return the number of bytes read. Return 0 at end of file. Return -1 on
/* error and set errno appropriately.
/*
/* The Green Hills Software C Test Program assumes that (as in UNIX) on line
/* buffered terminals input carriage returns ('\r') are converted to line feeds ('\n').
/*****/
```

realloc: Reallocate memory (realloc)

```
char *realloc(old, new_size)
char *old;
unsigned new_size;
/*****/
/* The memory block pointed to by "old" must have been allocated by malloc().
/* Return the address of a new block of memory of the size, "new_size",
/* which contains the same data as the old block. The old block is freed
/* with free().
/*****/
```

rewind: Seek to beginning of a FILE (rewind.c)

```
rewind(stream)
FILE *stream;
/*****/
/* Seek to the beginning of the FILE, "stream".
/* Equivalent to fseek(stream,0L,0).
/*****/
```

rindex: Reverse search for character in string (strchr.c)

```
char *rindex(str, ch)
register char *str;
register char ch;
/*****/
/* Return a pointer to the last location of the character, "ch", in the string,
/* "str", 0 if not found.
/*****/
```

sbrk: Set memory address break (os.c)

```
char *sbrk(size)
/*****/
/* Return a pointer to at least "size" bytes of contiguous read/write memory.
/* The memory returned by sbrk on different calls need not be contiguous
/* with each other (although they should be for compatibility with UNIX).
/* Return -1 on an error and set errno = ENOMEM.
/*****/
```

scanf: Read formatted input from standard input (scanf.c)

```
scanf(format, va_alist)
char *format;
va_dcl
/*****/
/* Read formatted input from the standard input file, stdin. The
/* format specification is given by the null terminated string, "format".
/* The input variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of input variables assigned to.
/*****/
```

setbuf: Set a user I/O buffer for a FILE (setbuf.c)

```
setbuf(stream, buf)
FILE *stream;
char buf[BUFSIZ];
/*****/
/* Use the data area specified by "buf" as an I/O buffer for the FILE, "stream".
/* If "buf" is NULL, then use unbuffered output on the FILE, "stream".
/*****/
```

setlinebuf: Set line buffering on a FILE (setlinbuf.c)

```
setlinebuf(stream)
FILE *stream;
/*****/
/* Set the FILE, "stream", to be line buffered. That is buffer output
/* until a '\n' character is encountered, then flush the output to
/* the output channel.
/*****/
```

sprintf: Print formatted output to string (printf.c)

```
sprintf(s, format, va_alist)
char *s;
char *format;
va_dcl
/*****/
/* Write null terminated formatted output to the string, "str". The
/* format specification is given by the null terminated string, "format".
/* The output variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of characters output
/* to the string (not counting the null).
/*****/
```

srand: Initialize the random number generator used by rand() (rand.c)

```
srand(val)
int val;
/*****/
/* Initialize the random number generator used by rand()
/*****/
```

sscanf: Read formatted input from a string (scanf.c)

```

sscanf(str, format, va_alist)
char *str;
char *format;
va_dcl
/*****/
/* Read formatted input from the null terminated string, "str". The
/* format specification is given by the null terminated string, "format".
/* The input variables are obtained from the variable argument list
/* by the varargs mechanism. Return the number of input variables assigned to.
/*****/

```

START: Program transfer address (crt0.c)

```

START()
/*****/
/* START is the function that starts all C programs. This file must be
/* specified to the Link Editor (Linker) in such a way that program execution
/* starts at the top of this function.
/*
/* START() initializes the processor, stack pointer, I/O, and memory systems
/* then it calls the program's main() function with the command line
/* arguments. If the main() function returns to the START function, START
/* exits by calling exit().
/*
/* As written, this file is a C function. It may be possible to leave this
/* function written in C (possibly using "asm" statements), but more than
/* likely you will find it necessary to compile this function into assembly
/* language and then edit the assembly code so that it can be used. In
/* particular, there may be procedure entry code at the top of this function
/* to save registers. This code may depend on a stack pointer having already
/* been initialized. But since it is the job of this function to load a
/* stack pointer (if it hasn't already been loaded), you may need to delete
/* the function entry code.
/*****/

```

stat: Get device and file number of file name (os.c)

```

stat(name, statptr)
char *name;
struct stat *statptr;
/*****/
/* The Green Hills Software C Library does not use the stat function, it is only
/* needed if the Green Hills Software Fortran Library is being used.
/*
/* "name" is the name of a file. In the two fields st_dev and st_ino place
/* values which uniquely identify the named file or device. In UNIX, these
/* are the device number and the inode (file) number on the device.
/*
/* Return 0 if the file status is correctly returned. Return -1 if no file
/* with that name exists, and set errno appropriately.
/*****/

```

```

stdin:  Standard input file (<stdio.h>)
        #define stdin (&_iob[0])
        /***/
        /* The standard input FILE.
        /***/

stdout: Standard output file (<stdio.h>)
        #define stdout (&_iob[1])
        /***/
        /* The standard output FILE.
        /***/

stderr: Standard error FILE (<stdio.h>)
        #define stderr (&_iob[2])
        /***/
        /* The standard error FILE.
        /***/

strcat: String concatenate (strcat.c)
        char *strcat(str2, str1)
        char *str2;
        register char *str1;
        /***/
        /* Copy the null terminated string, "str1", onto the end of the null terminated
        /* string, "str2". The first character of "str1" replaces the null terminating "str2".
        /* No test is made for overflowing "str2".
        /***/

strchr: Return the position of a character in a string (strchr.c)
        char *strchr(str, ch)
        register char *str;
        register char ch;
        /***/
        /* Return a pointer to the first instance of the character, "ch", in the
        /* null terminated string, "str". Return 0 if the character does not appear
        /* in "str". Note that this will work if the character is null ( ' ' ).
        /* Identical to index().
        /***/

strcmp: String Compare (strcmp.c)
        strcmp(str1, str2)
        register char *str1;
        register char *str2;
        /***/
        /* Compare two null terminated strings. Return a negative number if
        /* "str1" < "str2"; a 0 if "str1" == "str2"; and a positive number if "str1" > "str2".
        /* The machine dependent character ordering is used.
        /***/

```

strcpy: String Copy (strcpy.c)

```
char *strcpy(str2, str1)
char *str2;
register char *str1;
/*****
/* Copy the null terminated string, "str1", to "str2" up to and including the null
/* character that terminates "str1". Note that there is no check for an overflow
/* of "str2".
*****/
```

strindex: String index (strindex.c)

```
strindex(str, sub)
char *str;
char *sub;
/*****
/* Find the position of a null terminated substring, "sub", in another null
/* terminated string, "str". Return the offset to the substring's position
/* (0 based). Return -1 if the substring is not found in the string.
*****/
```

strlen: String length (strlen.c)

```
strlen(str)
char *str;
/*****
/* Return the length of the null terminated string "str".
*****/
```

strncat: String concatenate with maximum length (strncat.c)

```
char *strncat(str2, str1, n)
char *str2;
register char *str1;
register int n;
/*****
/* Copy characters from "str1" onto the end of the null terminated string,
/* "str2", until a null ('\0') is moved in "str1" or until n characters
/* are moved.
*****/
```

strncmp: String compare with maximum length (strncmp.c)

```
strncmp(str1, str2, n)
register char *str1;
register char *str2;
register int n;
/*****
/* Compare two null terminated strings. Return a negative number if
/* "str1" < "str2"; a 0 if "str1" == "str2"; and a positive number if "str1" > "str2".
/* Uses the machine dependent character ordering. If "str1" and "str2" do not
/* differ in the first "n" characters then they are equal (assuming there is
/* not a null ( ' ') in the first "n" characters)
*****/
```

strncpy: String copy with maximum length (strncpy.c)

```
char *strncpy(str2, str1, n)
char *str2;
register char *str1;
register int n;
/*****
/* Move exactly “n” characters into “str2”, these characters are taken from “str1”
/* until a ‘\0’ appears there, after which “str2” is padded with nulls (‘ ’).
*****/
```

strrchr: Reverse search for character in string (strrchr.c)

```
char *strrchr(str, ch)
register char *str;
register char ch;
/*****
/* Return a pointer to the last location of the character, “ch”, in the
/* null terminated string, “str”. Return 0 if the character does not
/* appear in the string.
*****/
```

strrindex: Reverse search for string in string (strrindex.c)

```
strrindex(str,sub)
char *str;
char *sub;
/*****
/* Search for the last occurrence of a null terminated substring, “sub”, in
/* another string, “str”. Return the offset to the substring’s position (0
/* based). Return -1 if the substring is not found in the string.
*****/
```

strsave: String save (strsave.c)

```
char *strsave(str)
char *str;
/*****
/* Return a pointer to a newly allocated copy of the null terminated string,
/* “str”. Return 0 if no memory can be allocated.
*****/
```

strtol: String to long conversion (strtol.c)

```
long strtol(str, ptr, base)
register char *str;
register char **ptr;
register int base;
/*****
/* Convert the null terminated ascii string, “str”, into a long value and
/* return it. Interpret the ascii string in the base (radix) specified
/* by “base”. Return a pointer to the end of the digit string in “*ptr”.
*****/
```

swab: Swap byte string (swab.c)

```
swab(from, to, nbytes)
register char *from;
register char *to;
register int nbytes;
/*****
/* Copy "nbytes" bytes from "from" to "to", swapping adjacent bytes.
/* If "nbytes" is odd then move "nbytes"-1 bytes.
*****/
```

sys_errlist: System error list (syserrlst.c)

```
char *sys_errlist[] = {...};
int sys_nerr = sizeof(sys_errlist)/sizeof(char *);
/*****
/* sys_errlist maps error numbers from errno.h to error message strings.
*****/
```

toascii: Convert character to ascii (<ctype.h>)

```
toascii(ch)
char ch;
/*****
/* Convert the character, "ch", to a legal ascii character, by clearing
/* any high order bits.
*****/
```

tolower: Convert character to lower case (<ctype.h>)

```
tolower(ch)
char ch;
/*****
/* Convert the upper case character, "ch", to its lower case equivalent.
/* Identical to tolower().
*****/
```

_tolower: Convert character to lower case (<ctype.h>)

```
_tolower(ch)
char ch;
/*****
/* Convert the upper case character, "ch", to its lower case equivalent.
/* Identical to _tolower().
*****/
```

toupper: Convert character to upper case (<ctype.h>)

```
toupper(ch)
char ch;
/*****
/* Convert the lower case character, "ch", to its upper case equivalent.
/* Identical to _toupper().
*****/
```

`_toupper`: Convert character to upper case (<ctype.h>)

```

/*****
/* Convert the lower case character, "ch", to its upper case equivalent.
/* Identical to toupper().
*****/
_toupper(ch)
char ch;

```

`ungetc`: Un-get character from a FILE (`ungetc.c`)

```

ungetc(ch,file)
int ch;
register FILE *file;
/*****
/* Put the character, "ch", back onto the input FILE, "file", such that
/* the next getc() or other input operation on "file" will return ch.
/* A call to fseek() will erase the effect of a ungetc(). Only one
/* call to getc() may be done between input requests from a FILE.
*****/

```

`unlink`: Unlink (Delete) file name from directory (`os.c`)

```

unlink(name)
char *name;
/*****
/* The Green Hills Software C Library does not uses the unlink function, it is only
/* needed if the Green Hills Software Fortran Library is being used.
/*
/* "name" is the name of a file. The named file is deleted.
/* Return 0 if the named file is deleted. Return -1 if there is no such
/* file or if the file cannot be deleted, and set errno appropriately
*****/

```

`write`: Write to an I/O channel (`os.c`)

```

write(fno, buf, size)
char *buf;
/*****
/* Write at most "size" bytes into the file connected to "fno" (where "fno" is
/* one of the file numbers returned by open() or creat()) into "buf".
/* Return the number of bytes written, or -1 to indicate an error and
/* set errno appropriately. On line buffered terminal output (as in UNIX)
/* line feeds ('\n') are converted to a carriage return ('\r') and a
/* line feed ('\n').
*****/

```

Chapter 4

Incompatible Changes From Previous Releases

If you have worked with previous releases of the Green Hills Software C Library you may need to know about some incompatible changes that have been made. For the most part this new version is fully compatible with previous versions. However, certain bugs and inconsistencies with UNIX have been reconciled by sacrificing compatibility. In addition, confusing default configuration options have been changed to make the Library more easily understood by the new user. The following changes are INCOMPATIBLE with the previous (May 1985) release.

The file `crtlib.h` and its programming style customization macros have been eliminated. The indenting style has been converted to Green Hills Standard.

The default configuration is now to load in floating point code for `printf` and `scanf`. In the previous release it was necessary to specify `-Ddo_float` to have `printf` and `scanf` recognize floating point arguments. This caused some people to believe that floating point was not supported at all. Now, the default is as if `-Ddo_float` was specified in the previous version. Now, if `-Dno_float` is specified, then `printf` and `scanf` no longer recognize floating point arguments, as was the case in the previous version by default.

To be compatible with UNIX, the second argument of both `index` and `rindex` is now of type `char`, instead of `(char *)` as in May 1985 release. The previous version was in error. The old `index` function has been renamed to `strindex`. The old `rindex` function has been renamed to `strrindex`.

The function `putw` now outputs an “int” instead of a “short”. The previous version was in error.

The function `getw` now inputs an “int” instead of a “short”. The previous version was in error.

The new library functionality demands more user supplied functions in the low level I/O module. For full functionality the additional UNIX calls `lseek()`, `isatty()`, and `getpid()` are needed. Trivial versions of these functions are supplied in the file “`os.c`”.

`fopen(name, “a”)` now calls a user supplied routine `lseek()` to seek to the end of the file. The `_OP_EOF` flag to `read()` has been discontinued.

The new function `fseek()` calls the user supplied routine `lseek()`. The function `mktemp()` calls the user supplied routine `getpid()`.

The Green Hills Software C Library Test Program has been renamed from “`testsuite.c`” to “`test-c.c`”. The Math Library Test Program has been renamed from “`testmath.c`” to “`test-m.c`” for consistency with other test program names.

The Green Hills Software C Library Test Program option `-Dtest_nofiles` has been replaced. In the previous version, the library as delivered would only pass the Test Program if `-Dtest_nofiles` was specified. This was confusing. Now, by default no test is made which requires a file system. That is, in this release the default is as if `-Dtest_nofiles` was specified to the previous version. In order to test a user implemented file system interface with the Green Hills Software C Library Test Program, it is necessary to specify `-Dtest_files` when compiling “`test-c.c`”.

Chapter 5

Revision history

May, 1984

First Release

May, 1985

Fixed fwrite

Fix so open/creat/close do not have to initialize _iob structures

Added more comments about how to test

Added perror, sys_errlist, sys_nerr

Added fdopen

Fixed *printf, strcpy, strncpy to return as documented under Sys V.

Fixed floating point output, corrected some problems in others

Added gcvt, ecvt, fcvt

August, 1986

Many bugs fixed.

“crtlib.h” deleted.

Math library separated and algorithms improved.

_cleanup, fseek, freopen added.

“a+”, “r+”, “w+” modes added to fopen, freopen, fdopen.

setbuf, setlinbuf, and line buffering added.

varargs added to printf and scanf, _doprnt, _doscan added

labs, cabs, hypot, bzero, ffs, bcmp, bcopy added

Floating point customizability added to math.h

Customization documentation improved.

Fields in stdio.h substantially modified.

Modules reorganized and renamed in many cases.

Documentation much improved.

-

Table of Contents

1. Customizing and Compiling the Green Hills Software C Library	4
1.1. Compile Time Options	4
1.2. Customizing the Low Level System Interface	5
1.3. Customizing Include Files	6
1.3.1. Custom Floating Point Formats	6
1.4. Installing the Green Hills Software C Library	7
1.5. Green Hills Software C Library Test Program	7
1.6. Implicit Library Assumptions	9
2. C Library Function Groups	10
3. Alphabetized List of C Library Functions	14
4. Incompatible Changes From Previous Releases	42
5. Revision history	43

-