Multics

C User's Guide

Honeywell Bull

MULTICS
C USER'S GUIDE

SUBJECT

Description of the Multics Implementation of the C Programming Language

SPECIAL INSTRUCTIONS

This publication supersedes the previous edition of the manual, Order No. HH07-00, dated January 1987. See the Preface for a description of changed information.

SOFTWARE SUPPORTED

Multics software release 12.1.

ORDER NUMBER

HH07-01

November 1987

Honeywell Bull

PREFACE

This manual describes the C programming language as implemented under Multics. The language is described by noting variations from a baseline version of C. The reader is assumed to be familiar with C. This manual is not a language specification, nor is it intended as a tutorial document.

Braces { } in this manual are used to enclose information from which the user must make a choice.

The following conventions are used to indicate the relative levels of topic headings used in this manual:

Level	Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, Underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

File No.: 1L23

HH07-01

SIGNIFICANT CHANGES IN HH07-01

Appendix B, "C Environment Support Commands" is new and the following RUN-TIME ROUTINES (Section 4) are either new or updated. The unmarked items are "new" and the updates are noted as "update."

access	getcwd	srand
abort	getenv	stat
alarm	getgid	strtok
asctime	getlogin	strtol
clock	getopt	strtod
ctime (update)	getpid	swab
drand48	getuid	sys_errlist
execl	gmtime	sys_nerr
execle	ioctl	times
execv	link	tzset
execve	localtime	ulimit
execlp	perror	utime
fcntl	rand	varargs
fstat	sleep	vprintf

CONTENTS

CONTENTS

CONTENTS

TABLES

Section 1
INTRODUCTION

C is a general-purpose, low-level programming language. It was developed under a UNIX* operating system but is now available for use with a number of computers and operating systems.

This manual describes the C programming language as implemented on Multics release 12.1. The language is described by noting variations from a baseline version of C.

The reader is assumed to be familiar with C and Multics. This manual is not a complete reference document, nor is it intended as a tutorial document.

DEFINITION OF BASELINE C

The version of C used in this manual as the baseline for comparison is described in:

> UNIX System V Release 2.0 Programming Guide
> Published by AT&T April 1984

The phrase baseline C refers to that version of C. You are assumed to have a copy of the UNIX operating system book on hand when you refer to this manual.

*UNIX is a registered Trademark of AT&T.

CONTENTS OF THIS MANUAL

The rest of this manual is organized as follows:

Section 2 notes variations in the Multics implementation of the C language.

Section 3 describes the command available for invoking the C compiler under the Multics environment.

Section 4 lists the C standard library of run-time routines.

Appendix A lists the C compiler diagnostic messages.

Appendix B lists the C environment support commands.

A glossary defines terms for a UNIX operating system, C, and Multics.

Section 2
IMPLEMENTATION OF THE C LANGUAGE

This section lists variations from the baseline C as described in the UNIX System Programming Guide (refer to Section 1).

This section contains only statements of variations. If a feature is not described in this section, it is fully supported by the C compiler, and behaves exactly the same as in baseline C.

LEXICAL CONVENTIONS [2]

The following variations on baseline C lexical conventions exist in Multics C.

Hardware Characteristics

The size of C data types are:

Data Type	Size (bits)
char	9
unsigned char	9
int	36
unsigned int	36
short	36
long	72
unsigned long	72
float	36
double	72

WHAT'S IN A NAME?

The C compiler supports all arithmetic types. C data types are described below.

A character variable (char) is a one-byte, signed binary integer consisting of eight significant bits and a high-order sign bit. It is always byte-aligned. Use the signed character data type for integer data with a domain of -512 to 511 (at most).

An unsigned character variable (unsigned char) is a one-byte, unsigned binary integer consisting of nine significant bits. It is never negative and always byte-aligned.

An integer variable (int) is a four-byte, signed binary integer consisting of 35 significant bits and a high-order sign bit. It is always word-aligned. This is the default data type for any variable.

An unsigned integer variable (unsigned int) is a four-byte, unsigned binary integer consisting of 36 significant bits. It is never negative and always word-aligned.

A long variable (long) is an eight-byte, signed binary integer consisting of 71 significant bits and a high-order sign bit. It is always double-word aligned.

An unsigned long variable (unsigned long) is an eight-byte unsigned binary integer consisting of 72 significant bits. It is always positive and always double-word aligned.

A floating-point variable (float) is a four-byte, word-aligned, signed real number. It can contain a value in the approximate range 1.0E-38 to 1.0E+38, with up to seven digits of precision.

A double-precision variable (double) is an eight-byte, double-word-aligned, signed real number. It can contain a value in the approximate range 1.0E-38 to 1.0E+38, with up to 16 digits of precision.

CONVERSIONS

The following variations on baseline C operand conversion exist in Multics C.

Characters and Integers

The C compiler performs sign extension on characters and on unsigned characters on assignment. Character variables range in value from -256 to 255, and unsigned characters range in value from 0 to 512.

Float and Double

The C compiler converts a double-precision variable to a floating-point variable by truncation.

Additive Operators

When adding or subtracting from pointers, be careful not to exceed segment bounds. When subtracting two pointers, the data being pointed to should be in the same segment or the result may be meaningless.

Shift Operators

When a right shift is performed on a signed quantity, the sign is propagated. For instance, in the expression E1>>E2, where E1 is a signed quantity, the vacated bit positions are filled by a copy of the sign bit.

When a right shift is performed on an unsigned quantity, vacated bit positions are filled with zeros.

DECLARATIONS

The Multics implementation of C does not use register variables.

All "static" functions must be declared before their first use.

Structure and Union Declarations

The C compiler only recognizes integer fields. The compiler does not initialize structures containing bit fields. The compiler assigns bit fields left to right within the word.

TYPES REVISITED

The following variations on baseline C types exist in Multics C.

Structures and Unions

Multics C does not allow the passing of structures or unions to or from functions.

Explicit Pointer Conversions

A pointer-to-long or a long-to-pointer conversion simply moves data between the two variables. A pointer-to-int conversion moves only the low-order bits of the pointer.

C PROGRAM PORTABILITY

There is no guarantee that a program written in C on one system will port easily to another system. Even when the programmer has been careful to write the program with portability in mind, problems may be caused by differences in the target-machine hardware or differences between the source and the target compiler.

The following is a discussion of the possible problems that may occur when porting C applications to Multics. It is assumed that most of the programs that are to be ported will have been written under a UNIX operating system on the VAX* or PDP** series machines.

Size of Data Types

A program may need to know the size of a particular data type. If this is hardcoded into the program, the code will not be portable if the data type sizes between the two machines differ. For example, a program may use the maximum size of an int in an expression as follows:

```
#define MAXINT 32767  /* largest int on a machine with a 16 bit int */
   .
   .
   .
if (y == MAXINT) .......
```

This code could cause incorrect results on Multics. Multics has a 36-bit integer so the value of MAXINT is not the maximum integer size on the Multics hardware. The following code would correct the situation:

```
#define MAXINT ((int) (((unsigned ) - 1) >> 1))
```

The following examples illustrate nonportable and portable ways of coding word definitions:

Nonportable Example:

```
#define word 4 /* hardcoded number of bytes in an integer*/
```

Portable Example:

```
#define word sizeof(int)
```

*VAX 11/780 is a trademark of Digital Equipment Corporation.

**PDP-11 and PDP-7 are trademarks of Digital Equipment Corporation.

Structures and Unions

Structures or unions cannot be passed to functions, nor can functions return either type.

Bit Fields

Multics bit fields may not be any data type other than integers or unsigned integers. However, various other compilers allow bit fields to be other than the integers or unsigned integers. This may cause portability problems.

The order in which the bits are put into memory may cause problems. The Kernighan and Ritchie C specification does not indicate the order in memory of bits in a bit field.

These problems would become apparent when using masks and unions to test bits, since the bits may not be where the tests think they are. The following is an example of the type of code that may cause problems when ported to Multics:

```
union {

        struct {
                int num: 4;
                int total: 6;
                int pad: 6;
                } st1;

        struct {
                int word;
                } st2;
        } extract;
```

In the above example, st2 is used to extract information from st1. The order of bits laid down in bit field determines the value of extract.st.word.

Pointers

One of the most common causes of nonportable code is the casting of pointers to integers. In most machines, pointers and integers are the same size. The programmer can then easily put pointers into integers with no loss of data. This is very common when returning pointers from functions.

The return value of a function is by default an integer. Since in most machines pointers and integers are the same size, it is quite easy to return a pointer from a function using the default return size of integer. This does not work on Multics because pointers are larger than integers.

To fix the problem, the Multics C programmer must define these functions as returning a long or a pointer. While returning a pointer in a long is allowed, it is not recommended.

The following code samples illustrate nonportable and portable cases of returning a pointer as an integer.

Nonportable Example:

```
my_func()                       /* function returns an int as default */
{
int *p;                         /* a pointer to an int */

if (some_test) return (p);      /* returns a poitner in an integer */

}
```

Portable Example:

```
int *my_func()                  /* function returns a pointer to an int */
{
int *p;                         /* a pointer to an int */

return (p);                     /* returns a pointer in a pointer location */
}
```

The following example shows a nonportable case of an integer holding a pointer, followed by a portable fix for Multics:

Nonportable Example:

```
int i;
struct {
                    char y[10];
                    int p;
                    } qbert, *t;

t = &qbert;                             /* t points to structure */
i = t;                                  /* assign pointer to integer */
i += 5;                                 /* point to y[5] */
```

Portable Example:

```
char *p;

p =  &qbert.y[0];                       /* point to start of y */
p += 5;                                 /* point to y[5] */
```

Be careful when you use pointers. Pointers on a VAX implementation are automatically initialized to zero when the stack frame is first allocated. Since zero is also the NULL value, the pointers can be used with no pre-initialization.

On Multics, you get no automatic initialization, therefore a pointer must be explicitly initialized to NULL before it is used. The following examples illustrate nonportable and portable cases of this:

Nonportable Example:

```
int *y[10];

if  (y[3] == NULL).....;          /* no guarantee that y[3] has */
                                  /* been assigned to NULL */
```

Portable Example:

```
int *y[10] = { NULL, NULL, ....};

        /* explicitly initialize array of pointers to NULL */
```

The Null Pointer Value

In most implementations the null pointer value NULL is defined to be the int value 0. It is not uncommon to see NULL used as a substitute for 0. On Multics the pointer value NULL is not 0, but -1|1.

The following two examples show implementations in which NULL is 0, which could cause portability problems:

Example 1:

```
int p;

if (p == NULL ).....              /* use NULL as substitute for zero */
```

Example 2:

```
int *t[10];

t[NULL] = 0;                      /* NULL used as subscript zero */
```

System Calls and the Runtime Library

Most of the commonly available portable programs have been developed on UNIX operating systems, and as such may have calls to system routines or runtime routines that are either not available on Multics or that have been implemented differently (see the documentation of these routines ahead). If this is the case it can be dealt with by creating the routine, removing the call, or writing a stub.

Section 3
INVOKING THE C COMPILER

This section describes the syntax of the cc command that invokes the Multics C compiler.

cc is the Multics C compiler. It accepts as input C source programs and/or assembled or compiled programs creating one of various output file types.

SYNTAX:

cc filename1, ..., filenamen {-control_args}

ARGUMENTS:

filename

Any file name with a suffix of .c is taken as a C source file and is compiled. Any file name suffixed with .alm is passed to alm. Any file name suffixed with .cpp is passed to the compiler. All other file names are given as input to the Linkage Editor.

CONTROL ARGUMENTS:

-brief, -bf

Suppresses printing of messages stating the current pass being performed (default).

-definition args, -def args

Specifies define names to be defined or undefined in the preprocessor. Where args is a list of define names separated by commas with no spaces in the following form:

-def n,x=2,^y

The first arg specifies that n is to be defined as 1 in the same way as \define n would define n to 1. The second arg specifies that x is to be given a definition of 2, and the last arg specifies that y is to be undefined in the preprocessor.

-include paths, -incl paths

Specifies the pathnames of include file directories the user wishes the preprocessor to look into for include files. All arguments up to the next control argument are treated as include directory pathnames.

-library paths, -lb paths

Specifies the pathnames of library directories, archives, or object files the user wishes the linkage editor to use when resolving external references. All arguments up to the next control argument are treated as include library pathnames.

-long, -lg

Specifies that a message should be printed specifying the completion of each pass of the compiler for each specified file name.

-optimize, -ot

Runs all compiled files through the optimizer (not implemented).

-output_file pathname, -of pathname

Forces the output to be placed in the file defined by pathname.

-profile, -pf

Generates profile information (not implemented).

-stop_after pass, -spaf pass

Specifies to cc to stop after the specified pass of the compiler. Valid values for pass are:

preprocessor, pp

Generates a .cpp file which is the output from the preprocessor.

c

Generates a .alm file which is an alm source file outputted from the C compiler.

alm

Generates a .cob file which is the intermediate executable file generated from the assembler. This file is to be used as input to the Linkage Editor.

-table, -tb

Specifies that the compiler should generate symbol table information. At the moment this generates a listing via the Linkage Editor.

Section 4
THE C STANDARD LIBRARY

This section lists the standard functions and subroutines provided with the Multics C compiler.

The routines provided with the C compiler attempt to present C programs with the same interface they would enjoy under a UNIX operating system. However, due to the inherent differences in the two operating systems, some routines are altered, have restrictions not found on UNIX operating systems, or are not supported at all. For instance, routines that involve pathnames adhere to Multics pathname conventions, not UNIX operating systems pathname conventions; the process management and "super user" functions are not available. Also excluded are these functions:

- Data base
- Multiplexed file
- Multiprecision integer arithmetic
- Plotter I/O
- Packet driver
- Interprocess communication
- Semaphore
- Archive
- X.25.

Table 4-1 lists C system functions and subroutines, sorted by name; Table 4-2 lists the same functions sorted by function group. Table 4-3 lists commonly used UNIX operating system functions (taken from System V UNIX) not supported under Multics C.

The Multics standard C include directories are located in >Sl3p>c_compiler>include.

Table 4-1. Multics C Standard Library (Sorted by Name) (Sheet 1 of 5)

Name	Function	Function Group
abort	Generate IOT fault	Process
abs	Absolute value of integer	Mathematical
access	Determine accessibility of file	File control
acos	Arc cosine	Mathematical
alarm	Schedule signal after interval	Process
alloc	Main memory allocation	Storage
asctime	Convert time to ASCII	System
asin	Arc sin	Mathematical
atan	Arc tangent	Mathematical
atan2	Arc tangent	Mathematical
atof	Convert ASCII to floating-point	String
atoi	Convert ASCII to integer	String
atol	Convert ASCII to long integer	String
calloc	Main memory allocation	Storage
ceil	Ceiling function	Mathematical
clearerr	File status inquiry	Input/output
clock	Report CPU time used	Process
close	Close file	File control
cos	Cosine	Mathematical
cosh	Hyperbolic cosine	Mathematical
creat	Create new file	File control
ctime	Convert date/time to ASCII	System
drand48	Generate uniformly distributed pseudorandom numbers	Mathematical
ecvt	Output conversion	String
execl	Execute a file	Process
execle	Execute a file	Process
execlp	Execute a file	Process
execv	Execute a file	Process
execve	Execute a file	Process
execvp	Execute a file	Process
exit	Terminate a process	Process
exp	Exponential function	Mathematical

Table 4-1. Multics C Standard Library (Sorted by Name)
(Sheet 2 of 5)

Name	Function	Function Group
fabs	Absolute value of real value	Mathematical
fclose	Close a file	Input/output
fcntl	Control over open files	File control
fcvt	Output conversion	String
fdopen	Open a file	Input/output
feof	File status inquiry	Input/output
ferror	File status inquiry	Input/output
fflush	Flush a file	Input/output
fgetc	Get character from word or file	Input/output
fgets	Get string from file	Input/output
fileno	File status inquiry	Input/output
floor	Floor function	Mathematical
fmod	Return remainder function (a/b)	Mathematical
fopen	Open a file	Input/output
fprintf	Formatted output conversion	Input/output
fputc	Put character or word on file	Input/output
fputs	Put string on file	Input/output
fread	Buffered binary input	Input/output
free	Main memory allocation	Storage
freopen	Reopen a file	Input/output
frexp	Split into mantissa and exponent	Mathematical
fscanf	Formatted input conversion	Input/output
fstat	Get file status	File control
fwrite	Buffered binary output	Input/output
gcvt	Output conversion	String
getc	Get character from word or file	Input/output
getchar	Get character from word or file	Input/output
getcwd	Get current working directory	File control
getenv	Get environment name	Process
getgid	Get group ID	Process
getlogin	Get login name	Process
getopt	Get option letter from arg	String
getpid	Get process ID	Process
gets	Get string from file	Input/output
getr	Get record	Input/output
getuid	Get user ID	Process
getw	Get word from file	Input/output
gmtime	Convert to Greenwich mean time	System
hypot	Euclidean distance	Mathematical

Table 4-1. Multics C Standard Library (Sorted by Name)
(Sheet 3 of 5)

Name	Function	Function Group
ioctl	Control device	Input/output
isalnum	Character classification	String
isalpha	Character classification	String
isascii	Character classification	String
isascii8	Character classification	String
isatty	Get name of terminal	System
iscntrl	Character classification	String
isdigit	Character classification	String
isgraph	Character classification	String
islower	Character classification	String
isprint	Character classification	String
ispunct	Character classification	String
isspace	Character classification	String
isupper	Character classification	String
isxdigit	Character classification	String
ldexp	Split into mantissa and exponent	Mathematical
link	Link to a file	File control
localtime	Convert date/time to local time	System
log	Natural logarithm	Mathematical
log10	Common logarithm	Mathematical
longjmp	Non-local goto	System
malloc	Main memory allocator	Storage
memccpy	Memory-to-memory copy	Storage
memchr	Point to character in memory	Storage
memcmp	Compare memory areas	Storage
memcpy	Memory-to-memory copy	Storage
memset	Initialize memory	Storage
mktemp	Make unique file name	File control
modf	Split into mantissa and exponent	Mathematical
kill	Send signal to process	Process
open	Open file	File control
perror	Print system error message	System
pow	Power function	Mathematical
printf	Formatted output conversion	Input/output
putc	Put character or word on file	Input/output
putchar	Put character or word on file	Input/output
putr	Put record on a file	Input/output
puts	Put string on file	Input/output
putw	Put word on file	Input/output

Table 4-1. Multics C Standard Library (Sorted by Name)
(Sheet 4 of 5)

Name	Function	Function Group
rand	Random number generator	Mathematical
read	Read from a file	Input/output
realloc	Reallocate memory	Storage
sbrk	Change memory allocation	Storage
scanf	Formatted input conversion	Input/output
setbuf	Assign buffering to a file	Input/output
setjmp	Prepare for non-local goto	System
sin	Sine	Mathematical
sinh	Hyperbolic sine	Mathematical
sleep	Suspend execution for interval	Process
sprintf	Formatted output conversion	Input/output
sqrt	Square root	Mathematical
srand	Random number generator	Mathematical
sscanf	Formatted input conversion	Input/output
stat	Get file status	File control
strcat	Character-string concatenation	String
strchr	First C occurrence	String
strcmp	Compare	String
strcpy	Copy	String
strcspn	Compare length of strings	String
strlen	Length	String
strncat	Concatenate N characters	String
strncmp	Compare N characters	String
strncpy	Copy N characters	String
strpbrk	Find first S_1 in S_2	String
strrchr	First C occurrence	String
strspn	Length of S_1 substr of S_2 chars	String
strtod	Convert string to double precision numbers	String
strtok	Token separator	String
strtol	Convert string to long integer	String
swab	Swap bytes	String
system	Execute a command line	System
sys_errlist	Vector of system error messages	System
sys_nerr	Largest system error message number	System
tan	Tangent	Mathematical
tanh	Hyperbolic tangent	Mathematical
time	Get time	Process
times	Get process times	Process
tmpnam	Create temporary file name	File control
toascii	Character translation	String
tolower	Character translation	String
toupper	Character translation	String
tzset	Set time zone	System

Table 4-1. Multics C Standard Library (Sorted by Name)
(Sheet 5 of 5)

Name	Function	Function Group
ulimit	Get and set user limits	File control
ungetc	Push character back into input file	Input/output
unlink	Remove directory entry	File control
utime	Set file time stamps	File control
varargs	Handle variable argument list	Input/output
vprintf	Print formatted output of a varargs argument list	Input/output
wait	Wait for process to terminate	Process
write	Write on file	Input/output

Table 4-2. Multics C Routines (Sorted by Function Group) (Sheet 1 of 4)

Group	Name	Function
File control	access	Determine accessibility of file
	close	Close file
	creat	Create new file
	fcntl	Control over open files
	fstat	Get file status
	getcwd	Get current working directory
	link	Link to a file
	mktemp	Make unique file name
	open	Open file
	stat	Get file status
	tmpnam	Create name for temporary file
	ulimit	Get and set user limits
	unlink	Remove directory entry
	utime	Set file time stamps
Input/output	clearerr	File status inquiry
	fclose	Close a file
	fdopen	Open a file
	feof	File status inquiry
	ferror	File status inquiry
	fflush	Flush a file
	fgetc	Get character from word or file
	fgets	Get string from file
	fileno	File status inquiry
	fopen	Open a file
	fprintf	Formatted output conversion
	fputc	Put character or word on file
	fputs	Put string on file
	fread	Buffered binary input
	freopen	Reopen a file
	fscanf	Formatted input conversion
	fwrite	Buffered binary output
	getc	Get character from word or file
	getchar	Get character from word or file
	gets	Get string from file
	getw	Get word from file
	ioctl	Control device
	printf	Formatted output conversion
	putc	Put character or word on file
	putchar	Put character or word on file
	puts	Put string on file
	putw	Put word on file
	read	Read from file
	scanf	Formatted input conversion
	setbuf	Assign buffering to file
	sprintf	Formatted output conversion
	sscanf	Formatted input conversion

Table 4-2. Multics C Routines (Sorted by Function Group) (Sheet 2 of 4)

Group	Name	Function
Input/output (cont.)	ungetc	Push character back into input file
	varargs	Handle variable argument list
	vfprintf	Print formatted output of a varargs argument list
	vsprintf	Print formatted output of a varargs argument list
	write	Write on file
Mathematical	abs	Absolute value of integer
	acos	Arc cosine
	asin	Arc sin
	atan	Arc tangent
	atan2	Arc tangent
	ceil	Ceiling function
	cos	Cosine
	cosh	Hyperbolic cosine
	drand48	Generate uniformly distributed pseudorandom numbers
	exp	Exponential function
	fabs	Absolute value of real value
	floor	Floor function
	fmod	Return remainder function (a/b)
	frexp	Split into mantissa and exponent
	hypot	Euclidean distance
	ldexp	Split into mantissa and exponent
	log	Natural logarithm
	log10	Common logarithm
	modf	Split into mantissa and exponent
	pow	Power function
	rand	Random number generator
	sin	Sine
	sinh	Hyperbolic sine
	sqrt	Square root
	srand	Random number generator
	tan	Tangent
	tanh	Hyperbolic tangent
Process	abort	Generate IOT fault
	alarm	Schedule signal after interval
	clock	Report CPU time used
	execl	Execute a file
	execle	Execute a file
	execlp	Execute a file
	execv	Execute a file
	execve	Execute a file
	execvp	Execute a file
	exit	Terminate a process

Table 4-2. MOD 400 C Routines (Sorted by Function Group) (Sheet 3 of 4)

Group	Name	Function
Process (cont.)	getenv	Get environment name
	getgid	Get group ID
	getlogin	Get login name
	getpid	Get process ID
	getuid	Get user ID
	kill	Send signal to process
	signal	Catch signal
	time	Get time
Storage	alloc	Main memory allocation
	calloc	Main memory allocation
	free	Main memory allocation
	malloc	Main memory allocator
	memccpy	Memory-to-memory copy
	memchr	Point to character in memory
	memcmp	Compare memory areas
	memcpy	Memory-to-memory copy
	memset	Initialize memory
	realloc	Reallocate memory
	sbrk	Change memory allocation
String	a64l	Convert base-64 ASCII to long
	atof	Convert ASCII to floating point
	atoi	Convert ASCII to integer
	atol	Convert ASCII to long integer
	ecvt	Output conversion
	fcvt	Output conversion
	gcvt	Output conversion
	getopt	Get option letter from arg
	isalnum	Character classification
	isalpha	Character classification
	isascii	Character classification
	isascii8	Character classification
	iscntrl	Character classification
	isdigit	Character classification
	isgraph	Character classification
	islower	Character classification
	isprint	Character classification
	ispunct	Character classification
	isspace	Character classification
	isupper	Character classification
	isxdigit	Character classification
	strcat	Character-string concatenation
	strchr	First C occurrence
	strcmp	Compare
	strcpy	Copy
	strcspn	Compare length of strings

Table 4-2. MOD 400 C Routines (Sorted by Function Group) (Sheet 4 of 4)

Group	Name	Function
String (cont.)	strlen	Length
	strncat	Concatenate N characters
	strncmp	Compare N characters
	strncpy	Copy N characters
	strpbrk	Find first S_1 in S_2
	strrchr	First C occurrence
	strspn	Length of S_1 substring of S_2
	strtod	Convert string to double precision numbers
	strtok	Token separator
	strtol	Convert string to long integer
	swab	Swap bytes
	toascii	Character conversion
	toascii8	Character conversion
	tolower	Character conversion
	_tolower	Character conversion
	toupper	Character conversion
	_toupper	Character conversion
System	asctime	Convert time to ASCII
	ctime	Convert date/time to ASCII
	errno	Error message number
	gmtime	Convert to Greenwich mean time
	localtime	Convert date/time to local time
	longjmp	Non-local goto
	perror	Print system error message
	setjmp	Prepare for non-local goto
	system	Execute a command line
	sys_errlist	Vector of system error messages
	sys_nerr	Largest system error message number
	tzset	Set time zone

Table 4-3. C Routines Not Supported (Sheet 1 of 4)

Name	Function
a64l	Convert base-64 ASCII to long
assert	Program verification
brk	Change memory allocation
bsearch	Binary search
chdir	Change working directory
chmod	Change mode of file
chown	Change owner
chroot	Change root directory
ctermid	Get terminal ID
crypt	DES encryption
cuserid	Get user ID
dbminit	Data base subroutine
delete	Data base subroutine
dial	Dial external line
dup	Duplicate open file descriptor
encrypt	DES encryption
edata	End of program initialized data location
end	End of program data location
endgrent	Close group file
endpwent	Close password file
equal_name	Equal-names convention
erf	Return error function of arg
erfc	Return 1-erf(x)
errno	Error message number
etext	End of program code location
fetch	Data base subroutine
find file	Find a file
firstkey	Data base subroutine
fork	Spawn a new process
fseek	Reposition a file
ftell	Reposition a file
ftw	File tree walk

Table 4-3. C Routines Not Supported (Sheet 2 of 4)

Name	Function
gamma	Log absolute value gamma function
getdir	Get pathname of system directory
getegid	Get effective group ID
geteuid	Get effective user ID
getgrent	Get group file entry
getgrgid	Get group file entry
getgrnam	Get group file entry
getpass	Read password
getpgrp	Get process group
getppid	Get parent process ID
getpwent	Get passwork record entry
getpwnam	Get password record by login name
getpwuid	Get password record by user ID
getptcb	Get parent TCB
gettcb	Get TCB
gsignal	Get signal
hcreate	Create heap
hdestroy	Destroy heap
hsearch	Search heap
init_mem	Initialize memory
j0	Bessel function
j1	Bessel function
jn	Bessel function
l3tol	Convert 3-byte integer to long
l64a	Convert long to base-64 ASCII string
lgdiv	Long divide
lgmul	Long Multiply
lgrem	Long remainder
logname	Login name of user
lsearch	Linear search
lseek	Change file currency
ltol3	Convert long integer to 3-byte
matherr	Math routine error handler
mcl	Execute Multics macrocall
mknod	Make node (directory or file)
monitor	Prepare execution profile
mount	Mount volume
mpx et al	Create and manipulate multiplexed files
nextkey	Data base subroutine
nice	Change priority of a process
nlist	Get entries from name list

Table 4-3. C Routines Not Supported (Sheet 3 of 4)

Name	Function
pause	Stop until signal
pclose	Close a pipe
pipe	Interprocess communication
plock	Process lock
popen	Open a pipe to/from process
posr	Position file record pointer
profil	Execution profile
pthto6	Convert UNIX pathname to Multics
ptrace	Trace a process
putpwentry	Write password entry
qsort	Quicker sort
regcmp	Compile regular expression
regx	Execute regular expression
runl	Create new process
runlp	Create new process
runv	Create new process
runvp	Create new process
same_file	Compare pathnames
send_sig	Send signal to process
setgid	Set group ID
setgrent	Rewind group file
setkey	DES encryption
setpgrp	Set process group
setprint	Set print attribute of stream
setpwent	Rewind password file
setuid	Set user ID
sgetl	Get long numeric
sig	Signal
signal	Catch signal
smopen	Open for block read/write
smread	Read block
smwrit	Write block
sputl	Put long numeric
star_check	Validate star name
star_match	Validate and match star name
star_name	List star name matches
stime	Set time
store	Data base subroutine
stty	Set terminal characteristics
synch	Update superblock

Table 4-3. C Routines Not Supported (Sheet 4 of 4)

Name	Function
tdelete	Delete tree
tell	Change file currency
tmpfile	Create temporary file
tmpnam	Temporary name
tsearch	Search tree
ttyname	Get name of terminal
twalk	Walk tree
tzname	Get time zone
ucf_defc	Create file
ucf_defr	Create file
ucf_finish	Create file
ucf_init	Create file
uldiv	Long unsigned divide
ulrem	Long unsigned remainder
umemchr	Point to character in memory
umemcmp	Compare memory areas
umemcpy	Memory-to-memory copy
umemset	Initialize memory
unmount	Dismount volume
y0	Bessel function
y1	Bessel function
yn	Bessel function

C SUPPORT OF MULTICS FILE TYPES

C supports sequential files with most functions.

The creat function creates a sequential file. Sequential processing of pre-existing string-relative files will be compatible with a UNIX operating system.

SUBROUTINES AND LIBRARIES

C subroutines and libraries include input/output and mathematical functions. While these functions are not directly callable from C, you can use these functions with include statements of the form:

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

Functions in the math library may return conventional values 0 or HUGE (largest size precision floating number) when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable errno is set to the value EDOM or ERANGE.

The descriptions of some functions refer to the null pointer (NULL). This value will not match that of any legitimate pointer, so many functions that return pointers return it, for example, to indicate an error. NULL is defined in <stdio.h> as (void*)0; you can include your own definition if you are not using <stdio.h>.

The standard I/O package consists of the stdio.h header file and a set of functions. The inline macrocalls getc and putc handle characters quickly. The macrocalls getchar, putchar, and the higher level routines fgetc, fgets, fprintf, fputc, fputs, fread, fscanf, fwrite, gets, getw, printf, puts, putw, and scanf all use getc and putc; they can be freely intermixed.

A file with associated buffering is declared to be a pointer to a defined type FILE. The fopen function creates certain descriptive data for a file and returns a pointer to designate the file in all further transactions. Normally, there are three open files with constant pointers declared in the "include" file and associated with the standard open files:

```
stdin  -- Standard input file (Multics user_input)
stdout -- Standard output file (Multics user_output)
stderr -- Standard error file (Multics error_output).
```

An integer constant EOF (-1) is returned when a function encounters the end of a file or an error (see the individual descriptions for details).

Any application that uses this package must include the header file of pertinent macrocall definitions, as follows:

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

The functions and constants mentioned in the input/output functions are declared in that "include" file and need no further declaration. The constants and the following "functions" are macrocalls (redeclaration of these names is perilous):

- clearerr
- feof
- fileno
- getc
- getchar
- putc
- putchar.

TRAPS AND SIGNALS

Generally, Multics traps are mapped to their UNIX operating system equivalents, to provide an emulation of a UNIX operating system environment. After catching a signal in a UNIX operating system, a program can continue as if the signal had not been sent merely by returning from the signal catcher (as opposed to calling exit).

Multics traps will be mapped into UNIX operating system signals as described in Table 4-4. This table shows the Multics conditions available to the C user and their corresponding UNIX operating system signal value.

Table 4-4. Multics Trap Support of UNIX Operating System Signals

Multics Condition	UNIX Operating System Signal
sus	SIGHUP
program_interrupt	SIGINT
quit	SIGQUIT
illegal_opcode, illegal_modifier	SIGILL
mmel	SIGTRAP
overflow, underflow	SIGFPE
io_error	SIGBUS
out_of_bounds	SIGSEGV
command_error, active_function_error	SIGSYS
alrm	SIGALRM

Refer to the Multics Programmer's Reference Manual for a description of the Multics conditions.

Table 4-5 lists software-generated signals. Note that "pid" is the process ID.

Table 4-5. Software-Generated Signals

C Calling Sequence	Meaning
kill (pid, 0)	Test signal, always ignored
kill (pid, SIGHUP)	1 Hangup
kill (pid, SIGINT)	2 Interrupt
kill (pid, SIGQUIT)	3 Quit
kill (pid, SIGILL)	4 Invalid instruction
kill (pid, SIGTRAP)	5 Trace trap
kill (pid, SIGIOT)	6 IOT instruction[a]
kill (pid, SIGEMT)	7 EMT instruction[a]
kill (pid, SIGFPE)	8 Floating-point exception
kill (pid, SIGKILL)	9 Kill
kill (pid, SIGBUS)	10 Megabus error
kill (pid, SIGSEGV)	11 Segmentation violation
kill (pid, SIGSYS)	12 Bad argument to function
kill (pid, SIGPIPE)	13 Write to pipe having no readers
kill (pid, SIGALRM)	14 Alarm clock
alarm (delta)	14 Alarm clock (after delta secs)
kill (pid, SIGTERM)	15 Terminate
kill (pid, SIGUSR1)	16 User-defined signal 1
kill (pid, SIGUSR2)	17 User-defined signal 2
kill (pid, SIGCLD)	18 Death of a child
kill (pid, SIGPWR)	19 Power-fail restart

[a] On some processors

In a UNIX operating system, the interrupt and quit signals are sent to every process in the process group that is not ignoring the signal. Processes created to run a command in the background (asynchronously) are created with these signals being ignored. Processes created to run a command in the foreground (synchronously) are created with default handling of these signals unless otherwise specified via the trap command. All other processes inherit the handling of these (and all other) signals from their parent.

On Multics, signals will be trapped and handled by each execution unit. If no signal mechanism exists at the current execution, the Multics default_error_handler will be invoked. (See the Multics Programmer's Reference Manual for a description of this handler.)

ERROR RETURNS

Most functions have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always -1; the individual descriptions specify the details.

Reporting Errors Via errno

A UNIX operating system error number is returned in the external integer variable errno. The variable errno is not cleared on successful calls, so it should be tested only after an error has been indicated.

UNIX Operating System Errors

All of the possible error numbers are not listed in each function description because many errors are possible for most of the calls. The following is a complete list of the error numbers, manifest constants, and names as defined in <error.h>.

1 EPERM Not owner.

In a UNIX operating system, this error typically indicates an attempt to modify a file in some way forbidden except to its owner or super-user.

2 ENOENT No such file or directory.

This error occurs when a file name is specified and the file should exist but does not, or when one of the directories in a pathname does not exist.

3 ESRCH No such process.

No process can be found corresponding to that specified by the process ID in kill.

4 EINTR Interrupted process.

An asynchronous signal (such as interrupt or quit), which the user has elected to catch, has occurred during a function. If execution is resumed after processing the signal, it appears as if the interrupted function returned this error condition. This is a UNIX operating system error only; it never occurs in Multics.

5 EIO I/O error.

Some physical I/O error. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address.

In a UNIX operating system, this occurs when I/O on a special file refers to a device that does not exist, or is beyond the limits of the device. It may also occur when, for example, a tape drive is not online or no disk pack is loaded on a drive.

7 E2BIG Argument list too long.

An argument list longer than 5120 characters is presented to a member of the exec family.

8 ENOEXEC Exec format error.

In a UNIX operating system, this occurs when a request is made to execute a file that, although it has the appropriate access, does not start with a valid magic number. This is a UNIX operating system error only; it never occurs in Multics.

9 EBADF Bad file number.

A file descriptor refers to no open file, a read request is made to a file that is open only for writing, or a write request is made to a file that is only open for reading.

10 ECHILD No children.

A wait was executed by a process that has no existing child processes; or by a process already waiting for all its children.

11 EAGAIN No more processes.

A fork failed because you are not allowed to create any more processes.

12 ENOMEM Not enough memory.

During an exec, sbrk, or other function, a program asks for more space than Multics can supply. This is not a temporary condition; the maximum space is a system parameter.

13 EACCES Permission denied.

An attempt has been made to access a file to which you have insufficient access.

14 EFAULT Bad address.

On Multics this error is performed by a fault and is not available to the user.

15 ENOTBLK Block device required.

In a UNIX operating system this occurs when a nonblock file is mentioned where a block device is required; for example, in mount.

16 EBUSY Mount device is busy.

In a UNIX operating system this occurs when an attempt is made to mount a device that is already mounted, or an attempt is made to demount a device on which there is an active file (open file or current directory). It also occurs if an attempt is made to enable accounting when it is already enabled.

17 EEXIST File already exists.

An existing file is mentioned in an inappropriate context; for example, link.

18 EXDEV Cross-device link.

In a UNIX operating system this occurs when a link to a file on another device is attempted.

19 ENODEV No such device.

An attempt has been made to apply an inappropriate function to a device; for example, read a write-only device.

20 ENOTDIR Not a directory.

A file is specified where a directory is required, for example in a path prefix or as an argument to chdir.

21 EISDIR Is a directory.

An attempt has been made to write on a directory.

22 EINVAL Invalid argument.

Some invalid argument has occurred; for example, mentioning an undefined signal in signal, or kill. This error is also set by the mathematical functions.

23 ENFILE File table overflow.

The system table of open files is full, and temporarily no more opens can be accepted.

24 EMFILE Too many open files.

No process can have more than 20 file descriptors open at a time.

25 ENOTTY Not a typewriter.

The device is not a terminal.

26 ETXTBSY Text file busy.

In a UNIX operating system this occurs when an attempt has been made to execute a pure procedure program that is currently open for writing (or reading), or an attempt has been made to open for writing a pure procedure program that is being executed.

27 EFBIG File too large.

In a UNIX operating system this occurs when the size of a file exceeds the maximum file size or ULIMIT.

28 ENOSPC No space left on device.

During a write to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek.

In a UNIX operating system this occurs when an lseek has been issued to a pipe.

30 EROFS Read-only file system.

An attempt was made to modify a file or directory on a device mounted read-only; that is, with the write-protect switch set.

31 EMLINK Too many links.

In a UNIX operating system this occurs on an attempt to make more than the maximum number of links (1000) to a file.

32 EPIPE Broken pipe.

In a UNIX operating system this occurs when a write has been attempted on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument not in function's domain.

The argument of a function in the math package is out of the domain of the function.

34 ERANGE Math function's result too large.

The value of a function in the math package is not representable within machine precision.

35 ENOMSG No message of desired type.

An attempt was made to receive a message of a type that does not exist on the specified queue. This is a UNIX operating system error only; it never occurs in Multics.

36 EIDRM Identifier removed.

This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space. This is a UNIX operating system error only; it never occurs in Multics.

RUN-TIME ROUTINES

The rest of this section describes the run-time routines (either functions or macrocalls) available under Multics C. The descriptions are arranged alphabetically by routine name. Refer to Tables 4-1 and 4-2 for a complete list of routines.

abort

Terminate a C program.

SYNTAX:

```
int abort ( )
```

ARGUMENTS:

None.

DESCRIPTION:

The abort function causes an IOT signal to be sent to its own process. The default signal catcher causes program termination.

It is possible for abort to return control if SIGIOT is caught or ignored. In this case, the value returned is that of the kill function.

abs

Integer absolute value.

SYNTAX:

```
int abs (i)
int i;
```

ARGUMENTS:

i

Integer value whose absolute value is to be returned.

DESCRIPTION:

The abs function returns the absolute value of its integer operand.

RELATED FUNCTIONS:

fabs.

access

Determine access rights or existence of a file.

SYNTAX:

```
int access (path, amode)
char *path;
int amode;
```

ARGUMENTS:

path

Pointer to a pathname naming a file.

amode

Bit pattern constructed as a sum of the following:

04 -- Read
02 -- Write
01 -- Execute (search)

DESCRIPTION:

The access function checks the access rights of the named file according to the bit pattern contained in the amode argument.

The file has access checked with respect to the read, write, and execute mode bits.

No access to the file is indicated if the information request of the file system returns an error.

RETURN VALUE:

If the requested access is permitted, a value of 0 is returned. Otherwise, a value of -1 is returned. The variable errno is set to indicate a UNIX operating system error.

acos

Arc cosine function.

SYNTAX:

```
# include <math.h>

double acos (x)
double x;
```

ARGUMENTS:

x

Double value of the cosine.

DESCRIPTION:

The acos function returns the arc cosine in the range 0 to pi.

DIAGNOSTICS:

Arguments of magnitude greater than 1 cause acos to return value 0.

RELATED FUNCTIONS:

asin, atan, atan2, cos, sin, tan.

alarm

Set a process alarm clock.

SYNTAX:

```
int alarm (sec)
int sec;
```

ARGUMENTS:

sec

Number of seconds until alarm.

DESCRIPTION:

The alarm function instructs the calling process's alarm clock to send the signal SIGALRM to the calling process after the number of real-time seconds specified by the sec argument have elapsed; see signal.

Alarm requests are not stacked; successive calls replace the calling task's alarm clock.

If sec is 0, any previously made alarm request is canceled.

RETURN VALUE:

The alarm function returns the amount of time, possibly 0, previously remaining in the calling process's alarm clock.

DIAGNOSTICS:

If alarm is unable to set the alarm clock for any reason, errno is set to indicate the reason and -1 is returned.

RELATED FUNCTIONS:

signal.

asctime

Convert date and time to ASCII.

SYNTAX:

```
# include <time.h>

    char *asctime (tm)
    struct tm *tm;
    extern long timezone;
    extern int daylight;
    extern char *tzname[2];
```

ARGUMENTS:

tm

Time, in military notation.

DESCRIPTION:

The asctime function converts the components of the time to ASCII and returns a pointer to a 26-character string in the following form (all fields have constant width):

```
Fri Aug 10 10:24:54 1984\n\0
```

The structure declaration from the include file is:

```
struct tm   {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday - 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight savings time is in effect.

The external long variable timezone contains the difference, in seconds, between GMT and local standard time (in EST, timezone is 5*60*60); the external variable daylight is nonzero if and only if the standard U.S. daylight savings time conversion should be applied.

If the environment variable TZ is not present, the asctime function assumes the local time zone is the same as the system time zone. The external variable daylight is set to zero in this case.

If TZ is present, the asctime function uses it to determine the local time zone. The value of TZ must be a time zone acronym, a time offset, and an optional daylight savings time zone acronym.

- The time zone acronym is up to four characters long.
- The time offset represents the difference between local time in the designated time zone and GMT. The difference is represented by a string of digits with an optional leading minus sign (for locations east of Greenwich, England) and with an optional trailing .5 (for locations some odd number of half-hours from Greenwich).
- The optional daylight savings time zone acronym is up to four characters long.

For example, the setting for Boston would be EST5EDT.

Setting TZ changes the values of the external variables timezone and daylight; in addition, time zone acronyms contained in the external variable tzname are set:

```
char *tzname[2] = {"EST ", "EDT "};
```

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

ctime, gmtime, localtime, time, tzset; see also the list_stz and set_stz commands.

asin

Arc sine function.

SYNTAX:

```
# include <math.h>

double asin (x)
double x;
```

ARGUMENTS:

x

Double-precision value of the sin.

DESCRIPTION:

The asin function returns the arc sine in the range -pi/2 to pi/2.

DIAGNOSTICS:

Arguments of magnitude greater than 1 cause asin to return value 0.

RELATED FUNCTIONS:

acos, atan, atan2, cos, sin, tan.

atan

Arc tangent function.

SYNTAX:

```
# include <math.h>

double atan (x)
double x;
```

ARGUMENTS:

x

Double-precision value of the tangent.

DESCRIPTION:

The atan function returns the arc tangent of x in the range -pi/2 to pi/2.

RELATED FUNCTIONS:

acos, asin, atan2, cos, sin, tan.

atan2

Arc tangent of y/x.

SYNTAX:

```
# include <math.h>

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

ARGUMENTS:

x

Double-precision value.

y

Double-precision value.

DESCRIPTION:

The atan2 function returns the arc tangent of y/x in the range -pi to pi.

RELATED FUNCTIONS:

acos, asin, atan, cos, sin, tan.

atof

Converts ASCII to floating point.

SYNTAX:

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

ARGUMENTS:

aptr

A string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optionally signed integer.

DESCRIPTION:

The atof function converts a string to floating-point representation. The first unrecognized character ends the string.

NOTE

There are no provisions for overflow.

RELATED FUNCTIONS:

atoi, atol, scanf.

atoi

Converts ASCII to integer.

SYNTAX:

```
int atoi (aptr)
char *aptr;
```

ARGUMENTS:

aptr

A string of tabs and spaces, then an optional sign, then a string of digits.

DESCRIPTION:

The atoi function converts a string to integer representation. The first unrecognized character ends the string.

NOTE

There are no provisions for overflow.

RELATED FUNCTIONS:

atof, atol, scanf.

atol

Converts ASCII to long.

SYNTAX:

```
long atof (aptr)
char *aptr;
```

ARGUMENTS:

aptr

A string of tabs and spaces, then an optional sign, then a string of digits.

DESCRIPTION:

The atol function converts a string to long integer representation. The first unrecognized character ends the string.

NOTE

There are no provisions for overflow.

RELATED FUNCTIONS:

atof, atoi, scanf.

calloc

Heaps memory allocation.

SYNTAX:

```
char *calloc (nelem, elsize)
unsigned nelem, elsize;
```

ARGUMENTS:

nelem

Number of elements.

elsize

Size of each element in characters.

DESCRIPTION:

The calloc function allocates space for an array of elements. The space is initialized to zeros.

RETURN VALUE:

The calloc function returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS:

If the heap does not contain enough memory and cannot be sufficiently expanded to meet the request, the variable errno is set to ENOMEM or ENOSPC and a null character pointer is returned.

RELATED FUNCTIONS:

free, malloc, realloc.

ceil

Ceiling function.

SYNTAX:

```
double ceil (x)
double x;
```

ARGUMENTS:

x

Double-precision value to be compared.

DESCRIPTION:

The ceil function returns the smallest integer not less than x.

RELATED FUNCTIONS:

abs, fabs, floor, fmod.

clearerr

File status inquiry -- clear error indicator.

SYNTAX:

```
# include <stdio.h>

clearerr (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The clearerr function resets the error indication on the named file.

The clearerr function is a macrocall; it cannot be redeclared.

RELATED FUNCTIONS:

feof, ferror, fileno, fopen, open.

clock

Report CPU time used.

SYNTAX:

```
long clock ( )
```

ARGUMENTS:

None.

DESCRIPTION:

Clock returns the amount of CPU time (in microseconds) used since the first call to clock. The time reported is the sum of the user and system times of the calling process.

close

Closes a file.

SYNTAX:

```
# include <stdio.h>

int close (fildes)
int fildes;
```

ARGUMENTS:

fildes

File descriptor obtained from a create or open function.

DESCRIPTION:

The close function closes and deletes a file. The close function closes the file descriptor indicated by fildes. A shared file is not removed until the last user executes a close.

RETURN VALUE:

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned. The variable errno is set to indicate the error.

DIAGNOSTICS:

The close function fails if fildes is not a valid, open file descriptor.

RELATED FUNCTIONS:

creat, open.

cos

Cosine function.

SYNTAX:

```
# include <math.h>

double cos (x)
double x;
```

ARGUMENTS:

x

Double-precision value of the angle in radians.

DESCRIPTION:

The cos function returns the cosine of a radian argument. The caller should check the magnitude of the argument to ensure that the result is meaningful.

RELATED FUNCTIONS:

acos, asin, atan, atan2, sin, tan.

cosh

Hyperbolic function.

SYNTAX:

```
# include <math.h>

double cosh (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The cosh function computes the hyperbolic cosine function for real arguments.

DIAGNOSTICS:

The cosh function returns a huge value of appropriate sign when the correct value would overflow.

RELATED FUNCTIONS:

sinh, tanh.

creat

Creates a new file or rewrites an existing one.

SYNTAX:

```
int creat (path, mode)
char *path;
int mode;
```

ARGUMENTS:

path

File pathname.

mode

File access--ignored (see below).

DESCRIPTION:

The creat function creates a new sequential file or prepares to rewrite an existing file named by the pathname pointed to by the path argument.

The mode argument (which in a UNIX operating system sets file access) is ignored. Access Control List (ACL) rights for the file are determined by whatever ACLs currently apply to the file.

If the file exists, the length is truncated to 0 and the mode and owner are unchanged.

RETURN VALUE:

Upon successful completion, the file descriptor (a non-negative integer) is returned and the file is opened for writing. The file descriptor is set to remain open across exec functions (see fcntl). The file pointer is set to the beginning of the file. No process can have more than 20 files open simultaneously.

Otherwise, a value of -1 is returned, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

close, open, read, write.

ctime

Converts date and time to ASCII.

SYNTAX:

```
# include <time.h>

char *ctime (clock)
long *clock;
```

ARGUMENTS:

clock

Long integer pointer to the time in seconds since midnight GMT, Jan. 1, 1970 (such as returned by time).

DESCRIPTION:

The ctime function converts a time into ASCII and returns a pointer to a 26-character string in the following form (all fields have constant width):

```
Sat Aug 10 10:24:54 1985\n\0
```

The structure declaration from the include file is:

```
struct tm   {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday - 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight savings time is in effect.

The external long variable timezone contains the difference, in seconds, between GMT and local standard time (in EST, timezone is 5-60-60); the external variable daylight is nonzero if, and only if, the standard U.S. daylight savings time conversion should be applied.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

asctime, gmtime, localtime, time, tzset.

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lconq48

Generate uniformly distributed pseudorandom numbers.

SYNTAX:

```
double drand48 ( )
double erand48 Xi[3];

long lrand48 ( )

long nrand48 (Xi)
unsigned short Xi[3];

long mrand48 ( )

long jrand48 (Xi)
unsigned short Xi[3];

void srand48 (seedval)
long seedval;

unsigned short *seed48 (seed16v)
unsigned short seed16v[3];

void lcong48 (param)
unsigned short param[7];
```

DESCRIPTION:

This family of functions generates pseudorandom numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions drand48 and erand48 return non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0).

Functions lrand48 and nrand48 return non-negative long integers uniformly distributed over the interval $[0, 2^{31})$.

Functions mrand48 and jrand48 return signed long integers uniformly distributed over the interval $[-2^{31}, 2^{31})$.

Functions srand48, seed48, and lcong48 are initialization entry points, one of which should be invoked before either drand48, lrand48, or mrand48 is called. (Although it is not recommeded practice, constant default initializer values will be supplied automatically if drand48, lrand48, or mrand48 is called without a prior call to an initialization entry point.) Functions erand48, nrand48, and jrand48 do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, X_i, according to the linear congruential formula:

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \quad n>=0$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless lcong48 has been invoked, the multiplier value a and the addend value c are given by:

$$a = 5DEECE66D_{16} = 273673163155_8$$
$$c = B_{16} = 13_8.$$

The value returned by any of the functions drand48, erand48, lrand48, nrand48, mrand48, or jrand48 is computed by first generating the next 48-bit X_i in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of X_i and transformed into the returned value.

The functions drand48, lrand48, and mrand48 store the last 48-bit X_i generated in an internal buffer; that is why they must be intialized prior to being invoked. The functions erand48, nrand48, and jrand48 require the calling program to provide storage for the successive X_i values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of X_i into the array and pass it as an argument. By using different arguments, functions erand48, nrand48, and jrand48 allow separate modules of a large program to generate several independent streams of pseudorandom numbers, i.e., the sequence of numbers in each stream will not depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function srand48 sets the high-order 32 bits of X_i to the 32 bits contained in its argument. The low-order 16 bits of X_i are set to the arbitrary value $330E_{16}$.

The initializer function seed48 sets the value of X_i to the 48-bit value specified in the argument array. In addition, the previous value of X_i is copied into a 48-bit internal buffer, used only by seed48, and a pointer to this buffer is the value returned by seed48. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time. Use the pointer to get at and store the last X_i value, and then use this value to reinitialize via seed48 when the program is restarted.

The initialization function lcong48 allows the user to specify the initial X_i, the multiplier value a, and the addend value c. Argument array elements param[0-2] specify X_i, param[3-5] specify the multiplier a, and param[6] specifies the 16-bit addend c. After lcong48 has been called, a subsequent call to either srand48 or seed48 restores the "standard" multiplier and addend values, a and c, specified on the previous page.

ecvt

Output conversion.

SYNTAX:

```
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
```

ARGUMENTS:

value

Double-precision value to be converted.

ndigit

Number of digits in output string.

decpt

Pointer to position of the decimal point relative to the beginning of the string (negative means to the left of the returned digits).

sign

If the sign of the result is negative, the word pointed to by sign is nonzero; otherwise it is zero.

DESCRIPTION:

The ecvt function converts a value to a null-terminated string of ndigit digits and returns a pointer thereto. If the sign of the result is negative, the word pointed to by sign is nonzero; otherwise it is zero. The low-order digit is rounded.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

fcvt, gcvt, printf.

errno

System error message number.

SYNTAX:

```
extern int errno;
```

ARGUMENTS:

None.

DESCRIPTION:

The external variable errno is set when errors occur but not cleared when nonerroneous calls are made.

execl

Execute a bound unit.

SYNTAX:

int execl(path,arg_0,arg_1,...,arg_n,(unsigned char *) 0)
unsigned char *path, *arg_0, *arg_1, ..., *arg_n;

ARGUMENTS:

path

Pointer to a pathname that identifies the new process bound unit.

arg_0, arg_1, ..., arg_n

Pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, at least arg_0 must be present and point to a string that is the same as path (or its file-name component).

DESCRIPTION:

The execl function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

A pointer to the environment of the calling process is placed in the global cell:

extern unsigned char **environ;

It is used to pass the environment of the calling process to the new process.

The execl function fails and returns to the calling process if:

- One or more components of the pathname do not exist [ENOENT].
- A directory-name component of path is not a directory [ENOTDIR].
- List access is denied for a directory named in path [EACCES].

- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].
- The path, argv, or envp argument points to an invalid address [EFAULT].

RETURN VALUE:

If execl returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execle, execv, execve, exit, getenv.

execle

Execute a bound unit.

SYNTAX:

```
int execle(path,arg0,arg1,...,argn,(unsigned char
*)0),envp)unsigned char *path, *arg0, *arg1, ..., *argn,
*envp [];
```

ARGUMENTS:

path

Pointer to a pathname that identifies the new process bound unit.

arg_0, arg_1, ..., arg_n

Pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, at least arg_0 must be present and point to a string that is the same as path (or its file name component).

envp

Array of character pointers to null-terminated strings. These strings constitute the environment for the new process. The array is terminated by a null character pointer.

DESCRIPTION:

The execle function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

The new process also inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- TTY group ID
- Time left until an alarm signal
- Current working directory
- Root directory
- File mode creation mask
- File size limit.

The execle function fails and returns to the calling process if:

- One or more components of the pathname do not exist [ENOENT].
- A directory-name component of path is not a directory [ENOTDIR].
- List access is denied for a directory named in path [EACCES].
- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].
- The path, argv, or envp argument points to an invalid address [EFAULT].

RETURN VALUE:

If execle returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execl, execv, execve, exit.

execlp

Execute a bound unit.

SYNTAX:

int execlp(file,arg_0,arg_1,...,arg_n(unsigned char *)0)
unsigned char *file, *arg_0, *arg_1, ..., *arg_n;

ARGUMENTS:

file

Pointer to the filename of the new process bound unit.

arg_0, arg_1, ..., arg_n

Pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, at least arg_0 must be present and point to a string that is the same as path (or its filename component).

DESCRIPTION:

The execlp function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

The directory containing the new process bound unit is found by searching the directories passed as the environment line "PATH= ... ".

A pointer to the environment of the calling process is placed in the global cell:

extern unsigned char **environ;

It is used to pass the environment of the calling process to the new process.

The new process also inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- TTY group ID
- Time left until an alarm signal
- Current working directory
- Root directory
- File mode creation mask
- File size limit.

The execlp function fails and returns to the calling process if:

- One or more components of a directory named in the environment line "PATH= ... " does not exist [ENOENT].
- A directory-path component of "PATH= ... " is not a directory [ENOTDIR].
- List access is denied for a directory named in "PATH= ... " [EACCES].
- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].
- The argv argument points to an invalid address [EFAULT].

RETURN VALUE:

If execlp returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execvp.

execv

Execute a bound unit.

SYNTAX:

```
int execv (path, argv)
unsigned char *path, *argv [];
```

ARGUMENTS:

path

Pointer to a pathname that identifies the new process bound unit.

argv

Array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, argv must have at least one member, and it must point to a string that is the same as path (or its file name component). The array is terminated by a null character pointer.

DESCRIPTION:

The execv function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

A pointer to the environment of the calling process is placed in the global cell:

```
extern unsigned char **environ;
```

It is used to pass the environment of the calling process to the new process.

The new process also inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- TTY group ID
- Time left until an alarm signal
- Current working directory
- Root directory
- File mode creation mask
- File size limit.

The execv function fails and returns to the calling process if:

- One or more components of the pathname do not exist [ENOENT].
- A directory-name component of path is not a directory [ENOTDIR].
- List access is denied for a directory named in path [EACCES].
- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].
- The path, argv, or envp argument points to an invalid address [EFAULT].

RETURN VALUE:

If execv returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execl, execle, execve, exit.

execv

Execute a bound unit.

SYNTAX:

```
int execve (path, argv, envp);
unsigned char *path, *argv [], *envp [];
```

ARGUMENTS:

path

Pointer to a pathname that identifies the new process bound unit.

argv

Array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, argv must have at least one member, and it must point to a string that is the same as path (or its file name component). The array is terminated by a null character pointer.

envp

Array of character pointers to null-terminated strings. These strings constitute the environment for the new process. The array is terminated by a null character pointer.

DESCRIPTION:

The execve function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

A pointer to the environment of the calling process is placed in the global cell:

```
extern unsigned char **environ;
```

It is used to pass the environment of the calling process to the new process.

The new process also inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- TTY group ID
- Time left until an alarm signal
- Current working directory
- Root directory
- File mode creation mask
- File size limit.

The execv function fails and returns to the calling process if:

- One or more components of the pathname do not exist [ENOENT].
- A directory-name component of path is not a directory [ENOTDIR].
- List access is denied for a directory named in path [EACCES].
- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].
- The path, argv, or envp argument points to an invalid address [EFAULT].

RETURN VALUE:

If execve returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execl, execle, execv, exit.

execv

Execute a bound unit.

SYNTAX:

```
int execvp (file, argv)
unsigned char *file, *argv []
```

ARGUMENTS:

file

Pointer to the filename of the new process bound unit.

argv

Array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, argv must have at least one member, and it must point to a string that is the same as path (or its file name component). The array is terminated by a null character pointer.

DESCRIPTION:

The execlp function transforms the calling process into a new process. The new process is constructed from an ordinary bound unit called the new process bound unit.

The directory containing the new process bound unit is found by searching the directories passed as the environment line "PATH= ... ".

A pointer to the environment of the calling process is placed in the global cell:

```
extern unsigned char **environ;
```

It is used to pass the environment of the calling process to the new process.

The new process also inherits the following attributes from the calling process:

- Process ID
- Parent process ID
- Process group ID
- TTY group ID
- Time left until an alarm signal
- Current working directory
- Root directory
- File mode creation mask
- File size limit.

The execvp function fails and returns to the calling process if:

- One or more components of a directory named in the environment line "PATH= ... " does not exist [ENOENT].

- A directory-path component of "PATH= ... " is not a directory [ENOTDIR].

- List access is denied for a directory named in "PATH= ... " [EACCES].

- The new process bound unit is not a bound unit, or the calling process lacks execute access to it [EACCES].

- The argv argument points to an invalid address [EFAULT].

RETURN VALUE:

If execvp returns to the calling process, an error has occurred; the return value is -1, and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

execlp.

exit

Terminate a process.

SYNTAX:

```
exit (status)
int status;
```

ARGUMENTS:

status

Status of operation.

DESCRIPTION:

The exit function terminates the calling process with the following consequences:

- All of the file descriptors open in the child (calling) process are closed.

RELATED FUNCTIONS:

signal, wait.

exp

Exponential function.

SYNTAX:

```
# include <math.h>

double exp (x)
double x;
```

ARGUMENTS:

x

Double-precision value to be operated on.

DESCRIPTION:

The exp function returns e^x.

DIAGNOSTICS:

The exp function returns a huge value when the correct value would overflow. A very large argument can also result in errno being set to ERANGE.

RELATED FUNCTIONS:

hypot, log, pow, sinh, sqrt.

fabs

Absolute value function.

SYNTAX:

```
double fabs (x)
double x;
```

ARGUMENTS:

x

Double-precision value to be operated on.

DESCRIPTION:

The fabs function returns |x| (that is, the absolute value of x).

RELATED FUNCTIONS:

abs, ceil, floor, fmod.

fclose

Close a file.

SYNTAX:

```
# include <stdio.h>

int fclose (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The fclose function causes any buffers for the named file to be written to that file, and the file to be closed. Buffers allocated by the standard input/output system are freed.

The fclose function is performed automatically upon calling exit.

RETURN VALUE:

This function returns 0 for success, and EOF if any errors were detected.

RELATED FUNCTIONS:

close, fflush, fopen, setbuf.

fcntl

File control.

SYNTAX:

```
# include <fcntl.h>

int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;
```

ARGUMENTS:

fildes

Open file descriptor obtained from a creat, open, or fcntl function.

cmd

Command (see below).

arg

Argument to cmd.

DESCRIPTION:

The fcntl function provides for control over open files.

Acceptable values for cmd are as follows:

F_DUPFD	Duplicate the lowest-numbered available file descriptor greater than or equal to arg. The file descriptor shares the same open file(s), file pointer, and access mode. The file status flags have the values of the original flags. The close-on-exec flag associated with the new file descriptor is set.
F_GETFD	Get the close-on-exec flag associated with the file descriptor fildes. If the low-order bit is zero, the file remains open across exec functions; otherwise, the file is closed on execution of exec.
F_SETFD	Set the close-on-exec flag associated with the file descriptor fildes to the low-order bit or arg.

F_GETFL Get the status flags of file.

F_SETFL Set the status flags of file to arg.

RETURN VALUE:

Upon successful completion, the value returned depends on the cmd argument, as follows:

F_DUPFD -- A new file descriptor
F_GETFD -- Value of flag (only low-order bit defined)
F_SETFD -- Value other than -1
F_GETFL -- Value of file flags
F_SETFL -- Value other than -1.

Otherwise, a value of -1 is returned and the variable errno is set to indicate the error.

DIAGNOSTICS:

The fcntl function fails it:

- The fildes argument does not point to a valid, open file descriptor [EBADF].
- The cmd argument is F_DUPFD and twenty file descriptors are currently open [EMFILE].
- The cmd argument is F_DUPFO and the arg argument is negative or greater than twenty [EINVAL] or the cmd argument is F_SETFL and the arg argument is invalid.

RELATED FUNCTIONS:

close, exec, open.

fcvt

Output conversion.

SYNTAX:

```
char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
```

ARGUMENTS:

value

Double-precision value to be converted.

ndigit

Number of digits to be returned.

decpt

Pointer to position of the decimal point relative to the beginning of the string (negative means to the left of the returned digits).

sign

If the sign of the result is negative, the word pointed to by sign is nonzero; zero otherwise.

DESCRIPTION:

The ecvt function converts a value to a null-terminated string of ndigit digits and returns a pointer thereto. The correct digit has been rounded for FORTRAN F-format output of the number of digits specified by the ndigit argument.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

ecvt, gcvt, printf.

fdopen

Open a file.

SYNTAX:

```
# include <stdio.h>

FILE *fdopen (fildes, type)
int fildes;
char *type;
```

ARGUMENTS:

fildes

Number of a file descriptor.

type

Access type (see below).

DESCRIPTION:

The fdopen function opens a file descriptor obtained from the open, dup, or creat function. The read/write indicator is set according to the type argument.

When a file is opened for update, both input and output are allowed.

The type argument consists of all valid combinations of r, w, a, +, and b. The argument has these meanings:

```
r  -- Open text file for reading only
w  -- Create text file for writing
a  -- Append to text file
r+ -- Update (read/write) text file
w+ -- Create text file for update (read/write)
a+ -- Append (read/write) at end of text file.
```

RELATED FUNCTIONS:

fclose, fopen, freopen, open.

feof

File status inquiry -- check for end of file.

SYNTAX:

```
# include <stdio.h>

int feof (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The feof function returns nonzero when EOF is read on the named input file; otherwise, it returns zero.

The feof function is a macrocall; it cannot be redeclared.

RELATED FUNCTIONS:

clearerr, ferror, fileno, fopen, open.

ferror

File status inquiry -- check for I/O error.

SYNTAX:

```
# include <stdio.h>

int ferror (file)
FILE *file
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The ferror function returns a nonzero value when an error has occurred while reading or writing the named file; otherwise, it returns zero. Unless cleared by the clearerr function, the error indication remains until the file is closed.

The ferror function is a macrocall; it cannot be redeclared.

RELATED FUNCTIONS:

clearerr, feof, fileno, fopen, open.

fflush

Flush a file.

SYNTAX:

```
# include <stdio.h>

int fflush (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The fflush function causes any buffered data for the named output file to be written to that file.

RETURN VALUE:

This function returns 0 for success, and EOF if any errors were detected.

RELATED FUNCTIONS:

close, fclose, fopen, setbuf.

fgetc

Get character from file.

SYNTAX:

```
# include <stdio.h>

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

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The fgetc function returns the next character from the named input file. The fgetc function behaves like getc, but is a genuine function, not a macrocall; it can therefore be used as an argument. The fgetc macrocall runs more slowly than getc, but takes less space per invocation.

DIAGNOSTICS:

This function returns the value -1 at end of file.

RELATED FUNCTIONS:

ferror, fopen, fread, getc, getchar, gets, getw, putc, scanf.

fgets

Gets characters from a file.

SYNTAX:

```
# include <stdio.h>

char *fgets (s, n, file)
char *s;
int n;
FILE *file;
```

ARGUMENTS:

s

Pointer to string of characters returned, including a newline character.

n

Number of characters to get -1.

file

File pathname.

DESCRIPTION:

The fgets function reads n-1 characters, or up to a newline character (which is retained), whichever comes first, from the file into the string s. The last character read into s is followed by a null character.

RETURN VALUE:

The fgets function returns its first argument.

DIAGNOSTICS:

The fgets function returns the constant pointer NULL upon the end of file or on an error.

NOTE

The fgets function retains in string s a newline character that ends input.

RELATED FUNCTIONS:

ferror, fopen, fread, getc, gets, puts, scan.

fileno

File status inquiry -- get file descriptor.

SYNTAX:

```
# include <stdio.h>

fileno (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The fileno function returns the integer file descriptor associated with the file (see open).

The fileno function is a macrocall; it cannot be redeclared.

RELATED FUNCTIONS:

clearerr, feof, ferror, fopen, open.

floor

Floor function.

SYNTAX:

```
double floor (x)
double x;
```

ARGUMENTS:

x

Double-precision value for comparison.

DESCRIPTION:

The floor function returns the largest integer (as a double-precision number) not greater than x.

RELATED FUNCTIONS:

abs, ceil, fabs, fmod.

fmod

Remainder function.

SYNTAX:

```
double fmod (x, y)
double x, y;
```

ARGUMENTS:

x

Double-precision value.

y

Double-precision value.

DESCRIPTION:

The fmod function returns x if y is 0; otherwise, it returns the number f with the same sign as x such that x = i*y + f, for some integer i, and 0 < f < y.

RELATED FUNCTIONS:

abs, ceil, fabs, floor.

fopen

Open a file.

SYNTAX:

```
# include <stdio.h>

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

ARGUMENTS:

filename

File pathname.

type

Access type (see below).

DESCRIPTION:

The fopen function opens the file named by filename and associates a file with it.

The fopen function returns a file pointer that identifies the file in subsequent operations.

When a file is opened for update, both input and output are allowed.

The type argument consists of all valid combinations of r, w, a, and +. The argument has these meanings:

```
r  -- Open text file for reading only
w  -- Create text file for writing
a  -- Append to text file
r+ -- Update (read/write) text file
w+ -- Create text file for update (read/write)
a+ -- Append (read/write) at end of text file.
```

DIAGNOSTICS:

The fopen function returns a null pointer if the file cannot be accessed.

RELATED FUNCTIONS:

fclose, fdopen, freopen, open.

fprintf

Formats output to file.

SYNTAX:

```
# include <stdio.h>

int fprintf (file, format [, arg] ... )
FILE *file;
char *format;
```

ARGUMENTS:

file

Pathname of file to receive output.

format

Format string (see below).

arg

Optional argument to be printed.

DESCRIPTION:

The fprintf function places output on the named output file. This function converts, formats, and prints its arguments under control of the format. The format is a character string that contains two types of objects: plain characters, which are simply copied to the output file, and conversion specifications, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

Each conversion specification is introduced by the percent (%) character. After the percent character, the following appear in sequence:

- Zero or more flags, which modify the meaning of the conversion specification.
- An optional decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it is blank-padded on the left (or right, if the left-adjustment flag has been given) to make up the field width.

- A precision that gives the minimum number of digits to appear for the d, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e and f conversions, the maximum number of significant digits for the g conversion, the maximum number of characters to be printed from a string in s conversion, or the minimum number of digits to appear in the word address portion of a converted pointer for the p or P conversions. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero.

- An optional l specifying that a following d, o, u, x, or X conversion character applies to a long integer argument.

- A character that indicates the type of conversion to be applied.

A field width or precision can be indicated by an asterisk (*) instead of a digit string. In this case, an integer argument supplies the field width or precision. The argument that is actually converted is not fetched until the conversion letter is seen, so the arguments specifying field width or precision must appear before the argument (if any) to be converted.

The flag characters and their meanings are:

-	The result of the conversion is left-justified within the field.
+	The result of a signed conversion always begins with a sign (+ or -).
blank	If the first character of a signed conversion is not a sign, a blank precedes the result. This implies that if the blank and + flags both appear, the blank flag is ignored. The p and P conversions ignore this flag.

#	The value is to be converted to an "alternate form." For c, d, s, and u conversions, the flag has no effect. For o conversions, it increases the precision to force the first digit of the result to be a zero. For x (X) conversion, a nonzero result will have 0x (0X) preceding it. For e, E, f, g, and G conversions, the result always contains a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeros are not removed from the result (as they normally are). For p or P conversions, the word-address and character-address portions of the converted pointer will each be preceded by 0x or 0X, except when the portion's value is zero.

The conversion characters and their meanings are:

d,o,u,x,X	The integer argument is converted to signed decimal, unsigned octal, unsigned decimal, or unsigned hexadecimal notation (x and X), respectively; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a 0 value with a precision of 0 is a null string (unless the conversion is o, x, or X and the # flag is present).
f	The float or double argument is converted to decimal notation in the style "[-]ddd.ddd", where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, six digits are output; if the precision is explicitly 0, no decimal point appears.

e,E — The float or double argument is converted in the style "[-]d.ddde+dd", where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, six digits are produced; if the precision is 0, no decimal point appears. The E format code produces a number with E instead of e introducing the exponent. The exponent always contains exactly two digits.

g, G — The float or double argument is printed in style e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted; style e is used only if the exponent resulting from the conversion is less than -4 or greater than the precision. Trailing zeros are removed from the result; a decimal point appears only if it is followed by a digit.

c — The character argument is printed.

s — The argument is taken to be a string (character pointer) and characters from the string are printed until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed.

% — Print a %; no argument is converted.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by fprintf are printed as if putchar had been called.

RETURN VALUE:

This function returns the number of characters transmitted.

DIAGNOSTICS:

If this function encounters an invalid string pointer, it behaves as if it has encountered a valid pointer to a null string. An error condition is indicated to the calling function by a negative return value.

EXAMPLES:

To print a date and time in the form "Sunday, July 3, 10:02", where weekday and month are pointers to null-terminated strings:

```
fprintf(temp,"%s, %s %d, %.2d:%.2d",weekday,month,day,hour,min);
```

To print pi to five decimal places:

```
    fprintf(output,"pi = %.5f", 4*atan(1.0));
```

RELATED FUNCTIONS:

ecvt, printf, putc, scanf, sprintf.

fputc

Puts a character on a file.

SYNTAX:

```
# include <stdio.h>

fputc (c, file)
FILE *file;
```

ARGUMENTS:

c

Character to write to file.

file

File pathname.

DESCRIPTION:

The fputc function appends the character c to the named output file. Unlike putc, it is a genuine function rather than a macrocall; it can therefore be used as an argument. The fputc function runs more slowly than putc, but takes less space per invocation.

RETURN VALUE:

The fputc function returns the character written.

DIAGNOSTICS:

The fputc function returns the constant EOF when it encounters an error. Since this is a good integer, ferror should be used to detect putw errors.

RELATED FUNCTIONS:

ferror, fopen, fwrite, getc, printf, putc, putchar, puts, putw.

fputs

Puts a string on a file.

SYNTAX:

```
# include <stdio.h>

int fputs (s, file)
char *s;
FILE *file;
```

ARGUMENTS:

s

String to be written to the file.

file

File pathname.

DESCRIPTION:

The fputs function copies the null-terminated string s to the named output file.

This function does not copy the terminating null character.

DIAGNOSTICS:

This function returns EOF if it encounters an error.

NOTE

The fputs function does not append a newline character.

RELATED FUNCTIONS:

ferror, fopen, fwrite, gets, printf, putc, puts.

fread

Buffered input.

SYNTAX:

```
# include <stdio.h>

fread (buf_ptr, size, nitems, file)
int size;
int nitems;
char *buf_ptr;
FILE *file;
```

ARGUMENTS:

buf_ptr

Buffer address pointer.

size

Item size in characters.

nitems

Number of items to read.

file

File pathname.

DESCRIPTION:

The fread function reads, into an array beginning at buf_ptr, nitems of size characters each from the named input file.

RETURN VALUE:

The fread function returns the number of items actually read.

RELATED FUNCTIONS:

fopen, fwrite, getc, gets, printf, putc, puts, read, scanf, write.

free

Frees heap memory.

SYNTAX:

```
void free (ptr)
char *ptr;
```

ARGUMENTS:

ptr

Pointer to a block previously allocated by calloc or malloc; this space is made available for further allocation.

DESCRIPTION:

The malloc and free functions together provide a simple, general-purpose memory allocation package.

DIAGNOSTICS:

Unspecified results occur if free acts on some random number.

RELATED FUNCTIONS:

calloc, malloc, realloc.

freopen

Reopens a file.

SYNTAX:

```
# include <stdio.h>

FILE *freopen (filename, type, file)
char *filename, *type;
FILE *file;
```

ARGUMENTS:

filename

New file pathname.

type

Access type (see below).

file

Old file pathname.

DESCRIPTION:

The freopen function substitutes the named file in place of the open file. It returns the original value of file. The original file is closed, regardless of whether the open ultimately succeeds.

The freopen function is used to attach the pre-opened constant names stdin, stdout, and stderr to specified files.

When a file is opened for update, both input and output are allowed.

The type argument consists of all valid combinations of r, w, a, and +. The argument has these meanings:

```
r  -- Open text file for reading only
w  -- Create text file for writing
a  -- Append to text file
r+ -- Update (read/write) text file
w+ -- Create text file for update (read/write)
a+ -- Append (read/write) at end of text file.
```

DIAGNOSTICS:

The freopen function returns a null pointer if filename cannot be accessed.

RELATED FUNCTIONS:

fclose, fdopen, fopen, open.

frexp

Splits into mantissa and exponent.

SYNTAX:

```
double frexp (value, eptr)
double value;
int *eptr;
```

ARGUMENTS:

value

Double-precision value to be processed.

eptr

Pointer to exponent.

DESCRIPTION:

The frexp function returns the mantissa, x, of the double-precision value as a double-precision quantity. The magnitude of x is less than 1 and greater than 1/16. It stores the exponent at the location pointed to by eptr. The exponent is the integer n such that value = $x*2^n$.

RELATED FUNCTIONS:

ldexp, modf.

fscanf

Formatted input conversion.

SYNTAX:

```
# include <stdio.h>

fscanf (file, format [, pointer]...)
FILE *file;
char *format;
```

ARGUMENTS:

file

Input file pathname.

format

Control string format (see below).

pointer

Set of arguments indicating where the converted input should be stored.

DESCRIPTION:

The fscanf function reads from the named input file. This function reads characters, interprets them according to a format, and stores the results in its arguments. It requires a control string format described below, and an optional set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs, or newline characters, which cause input to be read up to the next non-white-space character.

2. An ordinary character (not %), which must match the next character of the input file.

3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of nonspace characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are valid:

%	A single % is expected in the input at this point; no assignment is done.
d	A decimal integer is expected; the corresponding argument should be an integer pointer.
o	An octal integer is expected; the corresponding argument should be an integer pointer.
x	A hexadecimal integer is expected; the corresponding argument should be an integer pointer.
s	A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which is added automatically. The input field is terminated by a space or newline character.
c	A character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next nonspace character, use %1s. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.
e,f	A floating-point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating-point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an E or an e, followed by an optionally signed integer.

[Indicates a string that is not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not a circumflex (^), the input field consists of all characters up to the first character that is not in the set between the brackets; if the first character after the left bracket is a circumflex, the input field consists of all characters up to the first character that is in the set of the remaining characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o, and x can be capitalized and/or preceded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e and f may be capitalized and/or preceded by l to indicate that a pointer to double rather than to float is in the argument list.

The fscanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input file.

RETURN VALUE:

The fscanf function returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

NOTE

Trailing white space (including a newline character) is left unread unless matched in the control string.

DIAGNOSTICS:

This function returns EOF at the end of input and a short count for missing or illegal data items.

NOTE

The success of literal matches and suppressed assignments is not directly determinable.

EXAMPLES:

The call:

```
int i; float x; char name[50];
fscanf (names, "%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 brenda
```

assigns to i the value 25, to x the value 5.432, and name contains brenda\0. Or:

```
int i; float x; char name[50];
fscanf (data, "%2d%f%*d%[1234567890]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to i, 789.0 to x, skip 0123, and places the string 56\0 in name. The next call to getchar returns a.

RELATED FUNCTIONS:

atof, getc, printf, scanf, sscanf.

fstat

Get file status.

SYNTAX:

```
# include <types.h>
# include <stat.h>

int fstat (fildes, buf)
int fildes;
struct stat *buf;
```

ARGUMENTS:

fildes

File descriptor of the open file.

buf

Pointer to a static structure into which information is placed concerning the file.

DESCRIPTION:

The fstat function obtains information about an open file known by the file descriptor fildes, obtained from a successful open, creat, or dup function.

The contents of the structure pointed to by buf include the following members:

```
ushort    st_mode;   /*File mode                          */
ino_t     st_ino;    /*Inode number (N/A in MOD 400)     */
dev_t     st_dev;    /*ID of device containing           */
                     /*a directory entry for this file   */
dev_t     st_rdev;   /*ID of device                      */
                     /*This entry is defined only for    */
                     /*character special or block special
                       files                             */
short     st_nlink;  /*Number of links (N/A in MOD 400)*/
ushort    st_uid;    /*User ID of the file's owner       */
ushort    st_gid;    /*Group ID of the file's group      */
off_t     st_size;   /*File size in characters (N/A)     */
time_t    st_atime;  /*Time of last access               */
time_t    st_mtime;  /*Time of last data modification    */
                     /*Times measured in seconds since
                       00:00:00 GMT, Jan. 1, 1970        */
```

The st_atime member is the date/time when the file was last accessed. It is changed by the functions creat and read.

The st_mtime member is the date/time when the file was last modified. It is changed by the functions creat and write.

The st_ctime member is the date/time when the file was created. It is changed by the functions creat, link, unlink, and write.

Information is not available in the members st_ino, st_nlink, and st_size.

The fstat function fails if:

- The fildes argument is not a valid open file descriptor [EBADF].
- The buf argument points to an invalid address [EFAULT].

RETURN VALUE:

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

RELATED FUNCTIONS:

creat, link, stat, time, unlink.

fwrite

Buffered output.

SYNTAX:

```
# include <stdio.h>

fwrite (buf_ptr, size, nitems, file)
int size;
int nitems;
char *buf_ptr;
FILE *file;
```

ARGUMENTS:

buf_ptr

Buffer address pointer.

size

Item size in characters.

nitems

Number of items to write.

file

File pathname.

DESCRIPTION:

The fwrite function appends at most nitems of size size beginning at buf_ptr to the named output file. It returns the number of items actually written.

RELATED FUNCTIONS:

fopen, fread, gets, printf, putc, puts, read, scanf, write.

gcvt

Output conversion.

SYNTAX:

```
char *gcvt (value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

ARGUMENTS:

value

Value to be converted.

ndigit

Number of significant digits.

buf

Pointer to output string.

DESCRIPTION:

The gcvt function converts the argument value to a null-terminated string pointed to by buf and returns buf. It attempts to produce ndigit significant digits in FORTRAN F-format if possible; otherwise it produces output in E-format, ready for printing. Trailing zeros are suppressed.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

ecvt, fcvt, printf.

getc

Gets character from file.

SYNTAX:

```
# include <stdio.h>

int getc (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The getc function returns the next character from the buffer associated with the named input file. The function obtains a new buffer's worth of characters whenever all the characters have been returned.

DIAGNOSTICS:

This function returns the value -1 when it encounters the end of a file.

NOTE

Because it is a macrocall, getc treats incorrectly a file argument with side effects; for example:

```
getc(*f++);
```

RELATED FUNCTIONS:

ferror, fgetc, fopen, fread, getchar, gets, getw, putc, scanf.

getchar

Gets character from stdin file.

SYNTAX:

```
# include <stdio.h>

int getchar ( )
```

ARGUMENTS:

None.

DESCRIPTION:

The getchar function is identical to getc(stdin). This function is implemented as a macrocall; it cannot be redefined.

DIAGNOSTICS:

This function returns the value -1 when it encounters the end of a file.

RELATED FUNCTIONS:

ferror, fgetc, fopen, fread, getc, gets, getw, putc, scanf.

getcwd

Get current working directory.

SYNTAX:

```
char *getcwd (buf, size)
char *buf;
int size;
```

ARGUMENTS:

buf

Returned current working directory string.

size

Buffer size in characters.

DESCRIPTION:

The getcwd function returns a pointer to the null-terminated character string of the current working directory.

The value of the size argument must be at least one character longer than the pathname to be returned. Under Multics, the maximum length of a directory path is 168 characters.

If the buf argument is a null pointer, getcwd obtains size characters of space using the malloc function. In this case, you can use the returned pointer in a subsequent call to the free function.

If the buf argument is not a null pointer, the string is placed in buf, and the pointer to buf is returned.

DIAGNOSTICS:

If an error occurs, a null pointer is returned.

getenv

Get environment name.

SYNTAX:

```
char *getenv (name)
char *name;
```

ARGUMENTS:

name

Environment name.

DESCRIPTION:

The getenv function searches the environment list for a string of the form name and returns a pointer to that value if such a string is present; otherwise, it returns a null pointer.

getgid

Get real group ID.

SYNTAX:

```
int getgid ( )
```

ARGUMENTS:

None.

DESCRIPTION:

The getgid function returns the real group ID of the calling process.

RELATED FUNCTIONS:

getuid.

getlogin

Get login name.

SYNTAX:

```
char *getlogin ( );
```

ARGUMENTS:

None.

DESCRIPTION:

The getlogin function returns a pointer to a static string containing the login name of the calling process.

DIAGNOSTICS:

This function returns a null pointer if the name is not found.

getopt

Get option letter from argument.

SYNTAX:

```
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;
extern char *optarg;
extern int optind;
```

ARGUMENTS:

argc

Index into *argv.

argv

Input string of options.

optstring

String of valid options.

DESCRIPTION:

The getopt function returns the next option letter in argv that matches a letter in optstring. The argument optstring is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. The pointer optarg is set to point to the start of the option argument on return from getopt.

The getopt function places in optind the argv index of the next argument to be processed. Because optind is external, it is normally initialized to zero automatically before the first call to getopt.

RETURN VALUE:

When all options have been processed (that is, up to the first nonoption argument), getopt returns EOF. The special option minus (-) can be used to delimit the end of the options; EOF is returned, and minus (-) is skipped.

DIAGNOSTICS:

The getopt function displays an error message and returns a question mark (?) when it encounters an option letter not included in optstring.

EXAMPLE:

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the options f and e, both of which require arguments:

```
main (argc, argv)
int argc;
char **argv;
{
     int c;
     extern int optind;
     extern char *optarg;
     .
     .
     .
     while ((c = getopt (argc, argv, "abf:o:")) != EOF)
     switch (c) {

        case "a":

            if (bfg)
                    errfg++;
            else
                    afg++;
            break;
        case "b":

            if (afg)
                    errfg++;
            else
                    bproc();
            break;
        case "f":
            ifile = optarg;
            break;
        case "o";
            ofile = optarg;
            bufsize = 512;
            break;
        case "?":
            errfg++;
      }
   if (errfg) {
        fprintf (stderr, "usage:...");
        exit;
   }
   for ( ; optind < argc; optind++) {
          if (access (argv[optind], 4)) {
          .
          .
          .
          }
   .
   .
   .
   }
}
```

getpid

Get process ID.

SYNTAX:

```
getpid ( )
```

ARGUMENTS:

None.

DESCRIPTION:

The getpid function returns the process ID of the calling process.

gets

Gets string from stdin file.

SYNTAX:

```
# include <stdio.h>

char *gets (s)
char *s;
```

ARGUMENTS:

s

Pointer to buffer that will hold string.

DESCRIPTION:

The gets function reads a string into s from the standard input file stdin. The string is terminated by a newline character, which is replaced in s by a null character. The gets function returns its argument.

DIAGNOSTICS:

The gets function returns a null pointer if it encounters the end of a file or an error.

NOTE

The gets function deletes the newline character ending its input.

RELATED FUNCTIONS:

ferror, fgets, fopen, fread, getc, puts, scan.

getuid

Get real user ID.

SYNTAX:

```
int getuid ( )
```

ARGUMENTS:

None.

DESCRIPTION:

The getuid function returns the real user ID of the calling process.

RELATED FUNCTIONS:

getgid.

getw

Gets word from file.

SYNTAX:

```
# include <stdio.h>

int getw (file)
FILE *file;
```

ARGUMENTS:

file

File pathname.

DESCRIPTION:

The getw function returns the next word from the named input file. It returns the constant EOF when it encounters the end of a file or an error, but since that is a valid integer value, feof and ferror should be used to check the success of getw. The getw function assumes no special alignment in the file.

DIAGNOSTICS:

This function returns the value -1 when it encounters the end of a file.

RELATED FUNCTIONS:

feof, ferror, fgetc, fopen, fread, getc, getchar, gets, putc, putw, scanf.

gmtime

Convert date and time to ASCII.

SYNTAX:

```
struct tm *gmtime (clock)
long *clock;
```

ARGUMENTS:

clock

Military time.

DESCRIPTION:

The gmtime function returns a pointer to a structure containing the components of the time. The gmtime function converts directly to Greenwich Mean Time (GMT).

The structure declaration from the include file is:

```
struct tm   {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday - 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight savings time is in effect.

The external long variable timezone contains the difference, in seconds, between GMT and local standard time (in EST, timezone is 5*60*60); the external variable daylight is nonzero if, and only if, the standard U.S. daylight savings time conversion should be applied.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

asctime, ctime, localtime, time, tzset.

hypot

Euclidean distance.

SYNTAX:

```
# include <math.h>

double hypot (x, y)
double x, y;
```

ARGUMENTS:

x

Double-precision value.

y

Double-precision value.

DESCRIPTION:

The hypot function returns

$$(x^2 + y^2)$$

taking precautions against unwarranted overflows.

RELATED FUNCTIONS:

sqrt.

ioctl

Control device.

SYNTAX:

ioctl (fildes, request, arg) int fildes, request;

ARGUMENTS:

fildes

A file descriptor.

request

One of the request types described below.

arg

Either a pointer to the termio structure (see below), or an integer, depending on the request type.

DESCRIPTION:

NOTE

The Multics implementation of ioctl is incomplete.

ioctl performs a variety of functions on stdin, stdout, and stderr. Although the mode settings have been translated as closely as possible to Multics mode settings, there may be a difference in the actual actions taken. It is suggested that the user create an intermediate to call the specific I/O module with the desired functionality.

ioctl will fail if one or more of the following are true:

- fildes is not a valid open file descriptor [EBADF].
- fildes is not associated with stdin, stdout and stderr [ENOTTY].
- Request or arg is not valid [EINVAL].

RETURN VALUE:

If an error has occurred, a value of -1 is returned and errno is set to indicate the error.

REQUEST TYPES:

1. The primary ioctl system calls have the form:

   ioctl (fildes, request, arg) struct termio *arg;

   The requests using this form are:

   TCGETA

   Get the parameters associated with the terminal and store in the termio structure referenced by arg.

   TCSETA

   Set the parameters associated with the terminal from the structure referenced by arg. The change is immediate.

   TCSETAW

   Wait for the output to drain before setting the new parameters. Use this form when changing parameters that will affect output.

   TCSETAF

   Wait for the output to drain, then flush the input queue and set the new parameters.

2. Additional ioctl calls have the form:

   ioctl (fildes, request, arg) int arg;

   The requests using this form are:

   TCSBRK

   Wait for the output to drain. If arg is 0, then send a break (zero bits for 0.25 seconds.)

   TCFLSH

   If arg is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

SPECIAL CHARACTERS:

ERASE (#)

Erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@)

Deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (control-d or \F)

May be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF)

The normal line delimiter. It cannot be changed or escaped.

STOP (Control-s or ASCII DC3)

Used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read.

START (Control-q or ASCII DC1)

Used to resume output that has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

TERMIO STRUCTURE:

Several ioctl system calls apply to terminal files. The primary calls use the following structure, defined in <termio.h>:

```
#define   NCC    8
struct    termio {
int                     c_iflag;     /* input modes */
int                     c_oflag;     /* output modes */
int                     c_cflag;     /* control modes */
int                     c_lflag;     /* local modes */
char                    c_line;      /* line discipline */
unsigned    char        c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array c_cc. The relative positions and individual values for each function are as follows:

```
0    VINTR
1    VQUIT
2    VERASE      #
3    VKILL       @
4    VEOF        EOT
5    VEOL
6    reserved
7    switch
```

The c_iflag field describes the basic terminal input control:

```
IGNBRK  0000001  Ignore break condition.
BRKINT  0000002  Signal quit on break.
ISTRIP  0000040  Strip character.
INLCR   0000100  Map NL to CR-NL on input.
IUCLC   0001000  Map uppercase to lowercase on input.
IXON    0002000  Enable start/stop output control.
IXOFF   0010000  Enable start/stop input control.
```

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate a quit signal.

If ISTRIP is set, valid characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is defined by the initial setting for the terminal type on Multics.

The c_oflag field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lowercase to uppercase on output.
ONLCR	0000004	Map NL to CR-NL on output.

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair.

The initial output control value is defined by the initial setting for the terminal type on Multics.

The c_cflag field is unused and set to 0 on Multics.

The c_lflag field of the argument structure is used by the line discipline to control terminal functions:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
ECHO	0000010	Enable echo.
ECHONL	0000100	Echo NL.

If ISIG is set, each input character is checked against the special control characters INTR, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL and EOF. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received. This allows fast bursts of input to be read efficiently while still allowing single character input.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex).

The c_line is unused with a value of '\0' on Multics.

isalnum

Character classification (alphanumeric).

SYNTAX:

```
# include <ctype.h>

int isalnum (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isalnum macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isalnum function is defined only where isascii8 is true and on the single non-ASCII value EOF (see isascii8). The function is nonzero if c is an alphanumeric (letter or digit).

RELATED FUNCTIONS:

isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit.

isalpha

Character classification (alphabetic).

SYNTAX:

```
# include <ctype.h>

int isalpha (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isalpha macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isalpha function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a letter.

RELATED FUNCTIONS:

isalnum, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit.

isascii

Character classification (7-bit ASCII).

SYNTAX:

```
# include <ctype.h>

int isascii (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isascii macrocall classifies 7-bit ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isascii function is defined on all integer values. The function is nonzero if c is a 7-bit ASCII character, that is, a non-negative integer less than hexadecimal 80.

RELATED FUNCTIONS:

isalnum, isalpha, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit.

isatty

Determines if association is to a terminal.

SYNTAX:

```
int isatty (fildes)

int fildes;
```

ARGUMENTS:

fildes

File descriptor.

DESCRIPTION:

The isatty function returns 1 if fildes is associated with a terminal device; otherwise, it returns a 0.

iscntrl

Character classification (control character).

SYNTAX:

```
# include <ctype.h>

int iscntrl (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The iscntrl macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The iscntrl function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a delete character (hexadecimal 7F) or ordinary control character (hexadecimal 0 through 17, 84 through 97, and 9B through 9F).

RELATED FUNCTIONS:

isalnum, isalpha, isascii, isdigit, islower, isprint, ispunct, isspace, isupper, isxdigit.

isdigit

Character classification (digit).

SYNTAX:

```
# include <ctype.h>

int isdigit (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isdigit macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isdigit function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a digit [0 through 9].

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, islower, isprint, ispunct, isspace, isupper, isxdigit.

islower

Character classification (lowercase alphabetic).

SYNTAX:

```
# include <ctype.h>

int islower (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The islower macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The islower function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a lowercase letter. The lowercase letters are hexadecimal 61 through 7A, E0 through F6, and F8 through FF.

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, isprint, ispunct, isspace, isupper, isxdigit.

isprint

Character classification (printing character).

SYNTAX:

```
# include <ctype.h>

int isprint (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isprint macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isprint function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a printing character; that is, hexadecimal 20 (space) through 7E (tilde), or hexadecimal A0 (no-break space) through FF (small letter y with diaeresis).

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, islower, ispunct, isspace, isupper, isxdigit.

ispunct

Character classification (punctuation character).

SYNTAX:

```
# include <ctype.h>

int ispunct (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The ispunct macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The ispunct function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a punctuation character (neither control nor alphanumeric).

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, isspace, isupper, isxdigit.

isspace

Character classification (whitespace character).

SYNTAX:

```
# include <ctype.h>

int isspace (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isspace macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isspace function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is a space, tab, carriage return, newline character, vertical tab, formfeed, or no-break space.

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isupper, isxdigit.

isupper

Character classification (uppercase alphabetic).

SYNTAX:

```
# include <ctype.h>

int isupper (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isupper macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isupper function is defined only where isascii8 is true and on the single non-ASCII value EOF. The function is nonzero if c is an uppercase letter. The uppercase letters are hexadecimal 41 through 5A, C0 through D6, and D8 through DE.

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isxdigit.

isxdigit

Character classification (hexadecimal).

SYNTAX:

```
# include <ctype.h>

int isxdigit (c)
int c;
```

ARGUMENTS:

c

Single-character value.

DESCRIPTION:

The isxdigit macrocall classifies ASCII-coded integer values by table lookup. The macrocall is a predicate returning nonzero for true, zero for false. The isxdigit function is defined only where isascii8 is true and on the single non-ASCII value EOF (see isascii8). The function is nonzero if c is a hexadecimal digit ([0 through 9], [A through F], or [a through f]).

RELATED FUNCTIONS:

isalnum, isalpha, isascii, iscntrl, isdigit, islower, isprint, ispunct, isspace, isupper.

kill

Sends a signal to a process.

SYNTAX:

```
int kill (pid, sig)
int pid, sig;
```

ARGUMENTS:

pid

Process ID to be signaled (ignored).

sig

Signal to be sent.

DESCRIPTION:

The kill function signals the passed signal to the current process according to actions specified by any previous calls to signal. Any signals not defined cause a -1 to be returned. The process id is ignored.

RELATED FUNCTIONS:

signal.

ldexp

Exponential function.

SYNTAX:

```
double ldexp (value, exp)
double value;
int exp;
```

ARGUMENTS:

value

Double-precision value.

exp

Exponent.

DESCRIPTION:

The ldexp function returns the quantity value*2^{exp}.

RELATED FUNCTIONS:

frexp, modf.

link

Link to a file.

SYNTAX:

```
int link (path1, path2)
char *path1, *path2;
```

ARGUMENTS:

$path_1$

Pathname of an existing file.

$path_2$

Pathname of the new directory entry to be created.

DESCRIPTION:

The link function creates a new link (directory entry) for an existing file.

The link function fails and no link is created if:

- A component of either path prefix is not a directory [ENOTDIR].
- A component of either path prefix does not exist [ENOENT].
- A component of either path prefix denies search access [EACCES].
- The file named by $path_1$ does not exist [ENOENT].
- The link named by $path_2$ exists [EEXIST].
- Pointer $path_2$ points to a null pathname [ENOENT].
- The requested link requires writing in a directory without write access [EACCES].

RETURN VALUE:

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

unlink.

localtime

Convert date and time to ASCII.

SYNTAX:

```
# include <time.h>

struct tm *localtime (clock)
long *clock;
```

ARGUMENTS:

clock

Long integer pointer to the time in seconds since Jan. 1, 1970 (such as returned by time).

DESCRIPTION:

The localtime function returns a pointer to a structure containing the components of the time. The localtime function corrects for the time zone and possible daylight savings time.

The structure declaration from the include file is:

```
struct tm   {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday - 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight savings time is in effect.

The external long variable timezone contains the difference, in seconds, between GMT and local standard time (in EST, timezone is 5*60*60); the external variable daylight is nonzero if and only if the standard U.S. daylight savings time conversion should be applied.

NOTE

The return values point to static data whose contents are overwritten by each call.

RELATED FUNCTIONS:

asctime, ctime, gmtime, time, tzset.

log

Natural logarithm function.

SYNTAX:

```
# include <math.h>

double log (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The log function returns the natural logarithm of x. X must be positive.

DIAGNOSTICS:

The log function returns a huge negative value and sets errno to EDOM when x is nonpositive.

RELATED FUNCTIONS:

exp, hypot, log10, pow, sinh, sqrt.

log10

Common logarithm function.

SYNTAX:

```
# include <math.h>

double log10 (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The log10 function returns the common logarithm of x. X must be positive.

DIAGNOSTICS:

The log function returns a huge negative value and sets errno to EDOM when x is nonpositive.

RELATED FUNCTIONS:

exp, hypot, log, pow, sinh, sqrt.

longjmp

Non-local goto.

SYNTAX:

```
# include <setjmp.h>

void longjmp (env, val)
jmp_buf env;
int val;
```

ARGUMENTS:

env

Pointer to a label structure set by a previous call to setjmp.

val

Value to be returned.

DESCRIPTION:

The longjmp function restores the environment saved by the most recent call to setjmp having env as its argument. It then returns in such a way that execution continues as if the call to setjmp had returned with the value val instead of zero (as is the case with the true return from setjmp). The function that called setjmp must not itself have returned in the interim. If longjmp is invoked with a val argument of zero, it behaves as if 1 had been used instead.

RELATED FUNCTIONS:

kill, setjmp, signal.

malloc

Heaps memory allocator.

SYNTAX:

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

ARGUMENTS:

size

Size of the desired memory block in characters.

DESCRIPTION:

The malloc function is part of a general-purpose heap memory allocation package. The malloc function returns a character pointer to the beginning of a double-word-aligned block of at least size characters. Such blocks are suitable for storing objects of any type.

The heap is managed by the C functions malloc, calloc, realloc, and free.

The heap consists of one or more areas, each consisting of one or more segments. Heap areas are expanded, or new areas are created, as the need arises.

DIAGNOSTICS:

If the heap does not contain enough memory and cannot be sufficiently expanded to meet the request, the variable errno is set to ENOMEM or ENOSPC and a null character pointer is returned.

RELATED FUNCTIONS:

calloc, free, realloc.

memccpy

Memory-to-memory copy.

SYNTAX:

```
# include <memory.h>

unsigned char *memccpy (s1, s2, c, n)
unsigned char *s1, *s2;
unsigned char c;
int n;
```

ARGUMENTS:

s_1

Pointer to target memory area (output).

s_2

Pointer to source memory area (input).

c

Last character to copy (if found in s_2).

n

Number of characters to copy.

DESCRIPTION:

The memccpy function copies characters from memory area s_2 into s_1, stopping after the first occurrence of character c has been copied, or after n characters have been copied, whichever comes first. If n is less than or equal to zero, no characters are copied.

This function operates efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). This function does not check for the overflow of any receiving memory area.

RETURN VALUE:

This function returns a pointer to the character after the copy of c in s_1, or (unsigned char *) 0 if c was not found in the first n characters of s_2.

NOTE

This function is declared in the <memory.h> header file.

RELATED FUNCTIONS:

memchr, memcmp, memcpy, memset, umemchr, umemcmp, umemcpy, umemset.

memchr

Locates character in memory.

SYNTAX:

```
# include <memory.h>

unsigned char *memchr (s, c, n)
unsigned char *s;
unsigned char c;
int n;
```

ARGUMENTS:

s

Pointer to memory area to check.

c

Character to seek.

n

Size of memory area in characters.

DESCRIPTION:

The memchr function returns a pointer to the first occurrence of character c within the first n characters of memory area s, or (unsigned char *) 0 if c does not occur.

This function operates efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character).

NOTE

This function is declared in the <memory.h> header file.

RELATED FUNCTIONS:

memccpy, memcmp, memcpy, memset, umemchr, umemcmp, umemcpy, umemset.

memcmp

Memory-to-memory compare.

SYNTAX:

```
# include <memory.h>

int memcmp (s1, s2, n)
unsigned char *s1, *s2;
int n;
```

ARGUMENTS:

s_1

Pointer to first memory area to be compared.

s_2

Pointer to second memory area to be compared.

n

Size of memory areas in characters.

DESCRIPTION:

The memcmp function compares its arguments, looking at the first n characters only.

This function operates efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). It executes without a stack frame of its own, and it makes use of commercial instructions.

RETURN VALUE:

This function returns an integer less than, equal to, or greater than zero, depending on whether s_1 is less than, equal to, or greater than s_2. If n is less than or equal to zero, equality is indicated.

NOTES

1. This function is declared in the <memory.h> header file.

2. The memcmp function uses 8-bit ASCII comparisons. Comparison proceeds from left to right until an unequal pair of characters is found or until all characters have been compared without finding an unequal pair. If an unequal pair is found, their ordering in the 8-bit ASCII code set determines the ordering of the two operands.

RELATED FUNCTIONS:

memccpy, memchr, memcpy, memset, umemchr, umemcmp, umemcpy, umemset.

memcpy

Memory-to-memory copy.

SYNTAX:

```
# include <memory.h>

unsigned char *memcpy (s1, s2, n)
unsigned char *s1, *s2;
int n;
```

ARGUMENTS:

s_1

Pointer to target memory area (output).

s_2

Pointer to source memory area (input).

n

Number of characters to copy.

DESCRIPTION:

The memcpy function copies n characters from memory area s_2 to s_1.

This function operates efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). This function does not check for the overflow of any receiving memory area. It executes without a stack frame of its own.

RETURN VALUE:

This function returns s_1.

NOTES

1. This function is declared in the <memory.h> header file.

2. The memcpy function produces unspecified results if the memory areas overlap but are not identical.

memset

Initializes memory.

SYNTAX:

```
# include <memory.h>

unsigned char *memset (s, c, n)
unsigned char *s;
unsigned char c;
int n;
```

ARGUMENTS:

s

Pointer to memory area to initialize.

c

Character to fill memory area.

n

Size of memory area in characters.

DESCRIPTION:

The memset function sets the first n characters in memory area s to the value of character c. If n is less than or equal to zero, no characters are set.

This function operates efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). This function does not check for the overflow of any receiving memory area. It executes without a stack frame of its own, and it makes use of commercial instructions.

RETURN VALUE:

This function returns *s.

NOTE

This function is declared in the <memory.h> header file.

mktemp

Makes a unique file name.

SYNTAX:

```
char *mktemp (template)
char *template;
```

ARGUMENTS:

template

Template character string plus six trailing Xs.

DESCRIPTION:

The mktemp function replaces template by a unique file name, and returns the address of the template. The template should look like a file name with six trailing Xs, which will be replaced with a unique string. The letter is chosen so that the resulting name does not duplicate an existing file.

NOTE

It is possible to run out of letters.

RELATED FUNCTIONS:

getpid.

modf

Return fraction part of value.

SYNTAX:

```
double modf (value, iptr)
double value, *iptr;
```

ARGUMENTS:

value

Double-precision value.

iptr

Pointer to integer part of value.

DESCRIPTION:

The modf function returns the signed fractional part of value and stores the integer part indirectly, through iptr.

RELATED FUNCTIONS:

frexp, ldexp.

open

Opens for reading or writing.

SYNTAX:

```
# include <stdio.h>

int open (path, oflag)
char *path;
int oflag;
```

ARGUMENTS:

path

Pathname of file to open.

oflag

Access flag (see below).

DESCRIPTION:

The open function opens a file descriptor for the named file and sets the file status flags according to the value of oflag. The path pointer refers to a pathname naming a file. Oflag values are constructed by performing a logical OR operation on flags from the following list:

O_RDONLY -- Open for reading only.

O_WRONLY -- Open for writing only.

O_RDWR -- Open for reading and writing.

O_CREAT -- Create a new file. If the file already exists, this flag has no effect.

O_EXCL -- Only meaningful in combination with O_CREAT; these flags together specify that the file must not already exist.

The file pointer (used to mark the current position within the file) is set to the beginning of the file.

This function also works with dynamic and device files. To open an interactive device file (such as a terminal), use the O_RDWR flag; to open a noninteractive device file (such as a printer), use O_RDONLY or O_WRONLY, as appropriate.

No process can have more than 20 file descriptors open simultaneously.

The open function does not allocate a buffer until it is needed.

RETURN VALUE:

Upon successful completion, a file descriptor (a nonnegative integer) is returned. Otherwise, a value of -1 is returned and the variable errno is set to indicate the error returned from Multics.

RELATED FUNCTIONS:

close, creat, dup, fcntl, read, write.

perror

System error messages.

SYNTAX:

```
void perror (s)
char  *s;

extern int errno;

extern  cahr  *sys_errlist[ ];

extern  int  sys_nerr;
```

ARGUMENTS:

s

A pointer to a message string.

DESCRIPTION:

Perror produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string s is printed first, then the message and a new-line. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable errno, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the array of message strings sys_errlist is provided; errno can be used as an index in this table to get the message string without the new-line. Sys_nerr is the largest number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

pow

Power function.

SYNTAX:

```
# include <math.h>

double pow (x, y)
double x, y;
```

ARGUMENTS:

x, y

Double-precision values.

DESCRIPTION:

The pow function returns x^y. The values of x and y cannot both be zero. If x is less than or equal to zero, y must be an integer.

DIAGNOSTICS:

The pow function returns a huge value when the correct value would overflow. A truly outrageous argument can also result in errno being set to ERANGE.

The pow function returns a huge negative value and sets errno to EDOM when x is nonpositive and y is not an integer, or when x and y are both zero.

RELATED FUNCTIONS:

exp, hypot, log, sinh, sqrt.

printf

Formats output.

SYNTAX:

```
# include <stdio.h>

int printf (format [, arg] ... )
char *format;
```

ARGUMENTS:

format

Format string.

arg

Optional argument to be printed.

DESCRIPTION:

The printf function writes output to the user-out file. It is equivalent to a call to fprintf with the argument stdout inserted before the arguments to fprintf.

For more information on this function, refer to the description of fprint.

RELATED FUNCTIONS:

ecvt, fprintf, putc, scanf, sprintf.

putc

Puts a character on a file.

SYNTAX:

```
# include <stdio.h>

int putc (c, file)
char c;
FILE *file;
```

ARGUMENTS:

c

Character to be appended to the file.

file

File pathname.

DESCRIPTION:

The putc function appends the character c to the buffer associated with the named output file, writing the buffer whenever it is full.

RETURN VALUE:

The putc function returns the character appended.

DIAGNOSTICS:

This function returns the constant EOF when it encounters an error. Since this is a good integer, ferror should be used to detect putw errors.

NOTE

Because it is a macrocall, putc treats incorrectly a file argument with side effects, for example, putc(c, *f++); .

RELATED FUNCTIONS:

ferror, fopen, fputc, fwrite, getc, printf, putchar, puts, putw.

putchar

Puts character on stdout file.

SYNTAX:

```
# include <stdio.h>

putchar (c)
```

ARGUMENTS:

c

Character to be appended to the file.

DESCRIPTION:

The putchar(c) function is defined as putc(c, stdout).

DIAGNOSTICS:

This function returns the constant EOF when it encounters an error. Since this is a good integer, ferror should be used to detect putw errors.

RELATED FUNCTIONS:

ferror, fopen, fputc, fwrite, getc, printf, putc, puts, putw.

puts

Puts string on stdout file.

SYNTAX:

```
# include <stdio.h>

int puts (s)
char *s;
```

ARGUMENTS:

s

String to be written to the file.

DESCRIPTION:

The puts function copies the null-terminated string s to the user-out file and appends a newline character.

This function does not copy the terminating null character.

DIAGNOSTICS:

This function returns EOF on error.

NOTE

The puts function appends a newline character.

RELATED FUNCTIONS:

ferror, fflush, fopen, fputs, fwrite, gets, printf, putc.

putw

Puts a word on a file.

SYNTAX:

```
# include <stdio.h>

putw (w, file)
int w;
FILE *file;
```

ARGUMENTS:

w

Integer to be written to the file.

file

File pathname.

DESCRIPTION:

The putw function appends the integer w to the output file. The putw function neither assumes nor causes special alignment in the file.

DIAGNOSTICS:

This function returns the constant EOF when it encounters an error. Since this is a good integer, ferror should be used to detect putw errors.

RELATED FUNCTIONS:

ferror, fopen, fputc, fwrite, getc, printf, putc, putchar, puts.

rand

Generate random numbers.

SYNTAX:

```
int rand()
```

ARGUMENTS:

None.

DESCRIPTION:

The rand function uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudorandom numbers in the range from 0 to 2^{15} -1.

RELATED FUNCTIONS:

srand.

read

Reads from a file.

SYNTAX:

```
int read (fildes, buf, nchar)
int fildes;
char *buf;
unsigned nchar;
```

ARGUMENTS:

fildes

File descriptor obtained from a creat, open, dup, fcntl, or pipe function call.

buf

Pointer to buffer.

nchar

Number of characters to read.

DESCRIPTION:

The read function attempts to read nchar characters from the file associated with fildes into the buffer pointed to by buf.

Text file end-of-file processing is compatible with a UNIX operating system.

The read function does not allocate a buffer until it is needed.

RETURN VALUE:

Upon successful completion, a nonnegative integer is returned indicating the number of characters actually read and placed in the buffer. A value of -1 is returned when an end of file has been reached. A -1 is returned and the variable errno is set to indicate the error.

RELATED FUNCTIONS:

creat, fcntl, open.

realloc

Reallocates heap memory.

SYNTAX:

```
char *realloc (ptr, size)
char *ptr;
unassigned size;
```

ARGUMENTS:

ptr

Pointer to memory area to be reallocated.

size

New size, in characters.

DESCRIPTION:

The realloc function allocates an area of size and copies the value of the previous block into the new block for the specified size.

DIAGNOSTICS:

If the heap does not contain enough memory, and cannot be sufficiently expanded to meet the request, the variable errno is set to ENOMEN at ENOSPC and a null character pointer is returned. When realloc returns a null pointer, the block pointed to by ptr may have been destroyed.

RELATED FUNCTIONS:

calloc, free, malloc.

sbrk

Changes data segment space allocation.

SYNTAX:

```
char *sbrk (incr)
int incr;
```

ARGUMENTS:

incr

Number of characters to add to brk value.

DESCRIPTION:

sbrk has been converted to operate in the same manner as malloc.

scanf

Formatted input conversion.

SYNTAX:

```
# include <stdio.h>

scanf (format [,pointer]...)
char *format;
```

ARGUMENTS:

format

Control string format.

pointer

Set of arguments indicating where the converted input should be stored.

DESCRIPTION:

The scanf function reads from the standard input file stdin. This function reads characters, interprets them according to a format, and stores the results in its arguments. It requires a control string format and a set of optional pointer arguments indicating where the converted input should be stored.

The scanf function is equivalent to a call to fscanf with the argument stdout inserted before the arguments to scanf.

For more information on this function, refer to the description of the fscanf function.

RELATED FUNCTIONS:

atof, fscanf, getc, printf, sscanf.

setbuf

Assign buffering to a file.

SYNTAX:

```
# include <stdio.h>

setbuf (file, buf)
FILE *file;
char *buf;
```

ARGUMENTS:

file

File pathname.

buf

Pointer to buffer address.

DESCRIPTION:

The setbuf function is used after a file has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer.

A manifest constant BUFSIZ tells how big an array is needed:

```
char buf[BUFSIZ];
```

RELATED FUNCTIONS:

fopen, getc, putc.

setjmp

Non-local goto.

SYNTAX:

```
# include <setjmp.h>

int setjmp (env)
jmp_buf env;
```

ARGUMENTS:

env

Pointer to a label structure for later use by longjmp.

DESCRIPTION:

The setjmp function saves a label structure in env for later use by longjmp.

This routine is useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

RETURN VALUE:

This function returns the value zero.

RELATED FUNCTIONS:

kill, longjmp, signal.

signal

Specifies what to do upon receipt of a signal.

SYNTAX:

```
# include <signal.h>

int (*signal (sig, func))()
int sig;
int (*func)();
```

ARGUMENTS:

sig

Signal to be processed.

func

SIG_DFL, SIG_IGN, or a function address (see below).

DESCRIPTION:

The signal function allows the calling process to choose one of three ways to handle the receipt of a specific signal. The sig argument specifies the signal and the func argument specifies the choice.

A signal is generated by some abnormal event, such as a Megabus error, receipt of a kill, or your pressing Break. Normally, all signals terminate the process. The signal function allows a process to ignore a signal or cause an interrupt to a specified location.

The sig argument can be assigned from the following:

SIGHUP	01	Hangup
SIGINT	02	Interrupt
SIGQUIT	03*	Quit
SIGILL	04*	Invalid instruction
SIGTRAP	05*	Trace trap (not reset when caught)
SIGIOT	06*	IOT instruction
SIGEMT	07*	EMT instruction
SEGFPE	08*	Floating-point exception
SIGKILL	09	Kill (cannot be caught or ignored)
SIGBUS	10*	Megabus error
SIGSEGV	11*	Segmentation violation
SIGSYS	12*	Invalid argument to function
SIGALRM	14	Alarm clock

SIGTERM	15	Software termination signal
SIGUSR1	16	User-defined signal 1
SIGUSR2	17	User-defined signal 2
SIGCLD	18	Death of a child (see note)
SIGPWR	19	Power failure recovery (not reset when caught)

The actions prescribed by the sig argument are:

- SIG_DFL -- Pass the signal to the default_error handler (refer to Multics Programmer's Reference Manual).

- SIG_IGN -- The signal sig is to be ignored; the setting of func remains as SIG_IGN.

- function address -- Upon receipt of the signal sig, the receiving process is to execute the signal-catching function pointed to by func.

  Upon return from the signal-catching function, the receiving process resumes from the point where it was when the signal was caught. The value of func for a caught signal is reset to SIG_DFL unless the catching function executes a call to the signal function to set it otherwise.

RETURN VALUE:

Upon successful completion, signal returns the previous value of func for the specified signal sig. Otherwise, a value of -1 is returned and the variable errno is set to indicate the error.

DIAGNOSTICS:

The signal function fails if:

- The argument sig is an illegal signal number, including SIGKILL [EINVAL].

If a signal catcher is invoked while a process is executing a heap management function, and that signal catcher causes a recursive invocation of a heap management function by calling (even indirectly) any heap management function, the heap can be left in an inconsistent state. The heap can also be left in an inconsistent state if such a signal catcher abandons the heap management function using a nonlocal goto. The default signal catcher does neither of these things. (For the purpose of this note, the heap management functions are calloc, malloc, and free.)

RELATED FUNCTIONS:

kill, setjmp.

sin

Sine function.

SYNTAX:

```
# include <math.h>

double sin (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The sin function returns the sine of a radian argument. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

RELATED FUNCTIONS:

acos, asin, atan, atan2, cos, tan.

sinh

Hyperbolic sine function.

SYNTAX:

```
# include <math.h>

double sinh (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The sinh function computes the hyperbolic sine function for real arguments.

DIAGNOSTICS:

The sinh function returns a huge value of appropriate sign when the correct value would overflow.

RELATED FUNCTIONS:

cosh, tanh.

sleep

Suspend execution for interval.

SYNTAX:

```
unsigned sleep (seconds)
unsigned seconds;
```

ARGUMENTS:

seconds

Number of seconds to suspend execution.

DESCRIPTION:

The sleep function suspends the current process from execution for a specified number of seconds. The actual suspension time may be less than that requested for two reasons: because scheduled wakeups occur at fixed 1-second intervals, and because any caught signal terminates the sleep following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by sleep is the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep time, or premature arousal due to a caught signal.

The routine is implemented by setting an alarm signal. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep. If the sleep time exceeds the time until such an alarm signal, the process sleeps only until the alarm signal would have occurred, and the caller's alarm catch routine is executed just before the sleep routine returns. If the sleep time is less than the time until such an alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

RELATED FUNCTIONS:

alarm, signal.

sprintf

Formats output.

SYNTAX:

```
# include <stdio.h>

int sprintf (s, format [, arg] ... )
char *s, format;
```

ARGUMENTS:

format

Format string.

arg

Optional argument to be printed.

s

Address of location to begin output.

DESCRIPTION:

The sprintf function places "output," followed by the null character (\0) in consecutive characters starting at *s; you must ensure that enough storage is available.

This function is equivalent to a call to fprintf, except that the argument s specifies an array into which the generated output is written instead of a file.

For more information on this function, refer to the description of printf.

RELATED FUNCTIONS:

ecvt, fprintf, printf, putc, scanf.

sqrt

Square root function.

SYNTAX:

```
# include <math.h>

double sqrt (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The sqrt function returns the square root of x. X cannot be negative.

DIAGNOSTICS:

The sqrt function returns zero and sets errno to EDOM when x is negative.

RELATED FUNCTIONS:

exp, hypot, log, pow, sinh.

srand

Reset random number generator.

SYNTAX:

```
srand (seed)
unsigned seed;
```

ARGUMENTS:

seed

Seed value.

DESCRIPTION:

The srand function reinitializes the random number generator function. It can be set to a random starting point by calling srand with any argument.

RELATED FUNCTIONS:

rand.

sscanf

Formatted input conversion.

SYNTAX:

```
# include <stdio.h>

sscanf (s, format [,pointer]...)
char *s, *format;
```

ARGUMENTS:

s

Input character string.

format

Control string format.

pointer

Set of arguments indicating where the converted input should be stored.

DESCRIPTION:

The sscanf function reads from the character string s. This function reads characters, interprets them according to a format, and stores the results in its arguments. It requires a control string format and a set of optional pointer arguments indicating where the converted input should be stored.

The sscanf function is equivalent to a call to fscanf, except that the argument s specifies an array from which input is obtained rather than a file.

For more information on this function, refer to the description of fscanf.

RELATED FUNCTIONS:

atof, fscanf, getc, printf, scanf.

stat

Get file status.

SYNTAX:

```
# include <types.h>
# include <stat.h>

int stat (path, buf)
char *path;
struct stat *buf;
```

ARGUMENTS:

path

File pathname. Read, write, or execute access to the named file is not required, but all directories listed in the pathname leading to the file must be searchable.

buf

Pointer to a static structure into which information is placed concerning the file.

DESCRIPTION:

The stat function obtains information about the named file.

The contents of the structure pointed to by buf include the following members:

```
ushort   st_mode;   /*File mode                          */
ino_t    st_ino;    /*Inode number (N/A in MOD 400)      */
dev_t    st_dev;    /*ID of device containing            */
                    /*a directory entry for this file    */
dev_t    st_rdev;   /*ID of device                       */
                    /*This entry is defined only for     */
                    /*character special or block special
                      files                              */
short    st_nlink;  /*Number of links (N/A in MOD 400)   */
ushort   st_uid;    /*User ID of the file's owner        */
ushort   st_gid;    /*Group ID of the file's group       */
off_t    st_size;   /*File size in characters (N/A)      */
time_t   st_atime;  /*Time of last access                */
time_t   st_mtime;  /*Time of last data modification     */
                    /*Time measured in seconds since
                      00:00:00 GMT, Jan. 1, 1970         */
time_t   st_ctime;  /*Time of creation                   */
```

The st_atime member is the date/time when the file was last accessed. It is changed by the functions creat and read.

The st_mtime member is the date/time when the file was last modified. It is changed by the functions creat and write.

The st_ctime member is the date/time when the file was created. It is changed by the following functions: creat, link, unlink, and write.

Information is not available in the members st ino, st_nlink, and st_size.

RETURN VALUE:

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

DIAGNOSTICS:

The stat function fails if:

- A component of the path prefix is not a directory [ENOTDIR].
- The named file does not exist [ENOENT].
- Search access is denied for a component of the path prefix [EACCES].

RELATED FUNCTIONS:

creat, fstat, link, stat, time, unlink.

strcat

Concatenates strings.

SYNTAX:

```
char *strcat (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strcat function appends a copy of string s_2 to the end of string s_1. It returns a pointer to the null-terminated result. This function does not check for overflow of any receiving string.

NOTE

All string movement is performed character by character, starting at the left. Thus, overlapping moves toward the left work as expected, but overlapping moves to the right may not.

RELATED FUNCTIONS:

strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strchr

Finds character in string.

SYNTAX:

```
char *strchr (s, c)
char *s, c;
```

ARGUMENTS:

s

String to search.

c

Character to seek.

DESCRIPTION:

The strchr function returns a pointer to the first occurrence of character c in string s, or NULL if c does not occur in the string. The null character terminating a string is considered to be part of the string.

The strchr function operates on null-terminated strings. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strcmp

Compares strings.

SYNTAX:

```
int strcmp (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strcmp function compares its arguments and returns an integer greater than, equal to, or less than zero, according to whether s_1 is lexicographically greater than, equal to, or less than s_2. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strcpy

Copies string.

SYNTAX:

```
char *strcpy (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strcpy function copies string s_2 to s_1, stopping after the null character has been moved. It returns s_1. This function does not check for overflow of any receiving string.

NOTE

All string movement is performed character by character, starting at the left. Thus, overlapping moves toward the left work as expected, but over- lapping moves to the right may not.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strcspn

Substring operation.

SYNTAX:

```
int strcspn (s1, s2)
char *s1, *s2
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strcspn function returns the length of the initial segment of string s_1 which consists entirely of characters not from string s_2. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strlen

Finds length of string.

SYNTAX:

```
int strlen (s)
char *s;
```

ARGUMENTS:

s

Null-terminated string.

DESCRIPTION:

The strlen function returns the number of non-null character in s. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strncat

Concatenates portion of string.

SYNTAX:

```
char *strncat (s1, s2, n)
char *s1, *s2;
int n;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strcat function appends at most n characters of string s_2 to the end of string s_1. It returns a pointer to the null-terminated result. This function does not check for overflow of any receiving string.

NOTE

All string movement is performed character by character, starting at the left. Thus, over-lapping moves toward the left work as expected, but over- lapping moves to the right may not.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncmp, strncpy, strpbrk, strrchr, strspn, strtok.

strncmp

Compares to portion of string.

SYNTAX:

```
int strncmp (s1, s2, n)
char *s1, *s2;
int n;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

n

Number of characters to check.

DESCRIPTION:

The strncmp function looks at up to n characters of string s_1 and compares it to argument s_2, and returns an integer greater than, equal to, or less than zero, according to whether s_1 is lexicographically greater than, equal to, or less than s_2. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncpy, strpbrk, strrchr, strspn, strtok.

strncpy

Copies n characters.

SYNTAX:

```
char *strncpy (s1, s2, n)
char *s1, *s2;
int n;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

n

Number of characters to copy.

DESCRIPTION:

The strncpy function copies exactly n characters of string s_2 to s_1, truncating or null-padding s_2; the target might not be null-terminated if the length of s_2 is n or more. It returns s_1. This function does not check for overflow of any receiving string.

NOTE

All string movement is performed character by character, starting at the left. Thus, overlapping moves toward the left work as expected, but overlapping moves to the right may not.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strpbrk, strrchr, strspn, strtok.

strpbrk

Locates substring.

SYNTAX:

```
char *strpbrk (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strpbrk function returns a pointer to the first occurrence in string s_1 of any character from string s_2, or NULL if no character from s_2 exists in s_1. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strrchr, strspn, strtok.

strrchr

Finds last occurrence of substring.

SYNTAX:

```
char *strrchr (s, c)
char *s, c;
```

ARGUMENTS:

s

Null-terminated string.

c

Character to check for.

DESCRIPTION:

The strrchr function returns a pointer to the last occurrence of character c in string s, or NULL if c does not occur in the string. The null character terminating a string is considered to be part of the string. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strspn, strtok.

strspn

Gets length of substring.

SYNTAX:

```
int strspn (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strspn function returns the length of the initial segment of string s_1 which consists entirely of characters from string s_2. This function does not check for overflow of any receiving string.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strtok.

strtod

Convert string to double-precision number.

SYNTAX:

```
double strtod  (str, ptr)
char  *str,  **ptr;
```

ARGUMENTS:

str

A pointer to a null-terminated string.

ptr

A pointer to the return value.

DESCRIPTION:

strtod returns as a double-precision floating-point number the value represented by the character string pointed to by str. The string is scanned up to the first unrecognized character.

strtod recognizes an optional string of "white-space" characters (as defined by isspace in ctype), then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

if the value of ptr is not NULL, a pointer to the character terminating the scan is returned in the location pointed to by ptr. If no number can be formed, *ptr is set to str, and zero is returned.

strtok

String token operation.

SYNTAX:

```
char *strtok (s1, s2)
char *s1, *s2;
```

ARGUMENTS:

s_1, s_2

Null-terminated strings.

DESCRIPTION:

The strtok function considers the string s_1 to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string s_2. The first call (with pointer s_1 specified) returns a pointer to the first character of the first token, and will have written a NULL character into s_1 immediately following the returned token. Subsequent calls with zero for the first argument work through the string s_1 in this way until no tokens remain. The separator string s_2 may be different from call to call. When no token remains in s_1, a NULL is returned. This function does not check for overflow of any receiving string.

NOTE

All string movement is performed character by character, starting at the left. Thus, overlapping moves toward the left work as expected, but overlapping moves to the right may not.

RELATED FUNCTIONS:

strcat, strchr, strcmp, strcpy, strcspn, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn.

strtol

Convert string to integer.

SYNTAX:

```
long strtol  (str, prt, base)
char  *str  **ptr;
int base;
```

ARGUMENTS:

str

A pointer to a character string.

ptr

A pointer to a return value.

base

Specifies the conversion base.

DESCRIPTION:

strtol returns as a long integer the value represented by the character string pointed to by str. The string is scanned up to the first character inconsistent with the base. Leading "white-space" characters (as defined by isspace in ctype) are ignored.

If the value of ptr is not NULL, a pointer to the character terminating the scan is returned in the location pointed to by ptr. If no integer can be formed, that location is set to str, and zero is returned.

If base is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if base is 16.

If base is zero, the string itself determines the base thusly: after an optional leading sign a leading zero indicates octal conversion, and a leading "0x" or "0X" hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

swab

Swap bytes.

SYNTAX:

```
swab (fr, to, nbytes)
char *fr, *to;
int nbytes;
```

ARGUMENTS:

fr

Pointer to memory area from which bytes are taken.

to

Pointer to memory area in which bytes are placed.

nbytes

Number of bytes to move; argument should be an even number.

DESCRIPTION:

The swab function copies nbytes bytes pointed to by fr to the position specified by to, exchanging adjacent even and odd bytes.

This function is useful on machines where strings of characters are stored from right to left within words and from left to right from word to word, and where words are two characters wide.

system

Issues a Multics command.

SYNTAX:

```
# include <stdio.h>

int system (string)
char *string;
```

ARGUMENTS:

string

Command line.

DESCRIPTION:

The system function causes the string to be given to Multics as input as if the string had been typed as a command at a terminal. The current process waits until the command has completed, then returns the exit status of the command.

DIAGNOSTICS:

An exit status return of -1 is returned if the command processor could not be called successfully.

sys_errlist

System error messages.

SYNTAX:

```
char *sys_errlist [];
```

ARGUMENTS:

None.

DESCRIPTION:

To simplify variant formatting of error messages, the vector of message strings sys_errlist is provided; the variable errno can be used as an index in this table to get the message string without the newline character. The variable sys_nerr is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

RELATED FUNCTIONS:

errno, perror, sys_nerr.

sys_nerr

Number of largest system error message.

SYNTAX:

```
int sys_nerr;
char *sys_errlist [];
```

ARGUMENTS:

None.

DESCRIPTION:

To simplify variant formatting of messages, the vector of message strings sys_errlist is provided; the variable errno can be used as an index in this table to get the message string without the newline character. The variable sys_nerr is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

RELATED FUNCTIONS:

errno, perror, sys_errlist.

tan

Tangent function.

SYNTAX:

```
# include <math.h>

double tan (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The tan function returns the tangent of a radian argument. The caller should check the magnitude of the argument to make sure the result is meaningful.

RELATED FUNCTIONS:

acos, asin, atan, atan2, cos, sin.

tanh

Hyperbolic tangent function.

SYNTAX:

```
# include <math.h>

double tanh (x)
double x;
```

ARGUMENTS:

x

Double-precision value.

DESCRIPTION:

The tanh function computes the hyperbolic tangent function for real arguments.

RELATED FUNCTIONS:

cosh, sinh.

time

Gets time.

SYNTAX:

```
long time ((long *) 0)

long time (tloc)
long *tloc;
```

ARGUMENTS:

tloc

Pointer to memory area in which result is returned.

DESCRIPTION:

The time function returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

If tloc is not null, the return value is also stored in the location to which tloc points.

RETURN VALUE:

Upon successful completion, time returns the value of time. Otherwise, a value of -1 is returned, and the variable errno is set to indicate the error.

DIAGNOSTICS:

The time function fails if tloc points to an invalid address [EFAULT].

times

Get process and child process times.

SYNTAX:

```
#include <sys/types.h>
#include <sys/times.h>

long times (buffer)
struct tms *buffer;
```

ARGUMENTS:

buffer

A pointer to a tms structure (see below).

DESCRIPTION:

Times fills the structure pointed to by buffer with time-accounting information. The following are the contents of this structure:

```
struct  tms {
            time_t  tms_utime;
            time_t  tms_stime;
            time_t  tms_cutime;
            time_t  tms_cstime;
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a wait.

tms_utime is the CPU time used while executing instructions in the user space of the calling process.

tms_stime is the CPU time used by the system on behalf of the calling process. Will be zero.

tms_cutime is the sum of the tms_utimes and tms_cutimes of the child processes. Will be zero.

NOTE

tms_cstime is unavailable on Multics.

times will fail if buffer points to an illegal address [EFAULT].

RETURN VALUE:

Upon successful completion, times returns the elapsed real time, in 60ths (100ths) of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of times to another. If times fails, a -1 is returned and errno is set to indicate the error.

tmpnam

Creates a name for a temporary file.

SYNTAX:

```
# include <stdio.h>

char *tmpnam (s)
char *s;
```

ARGUMENTS:

s

Address of array to receive result.

DESCRIPTION:

The tmpnam function generates a file name that can safely be used for a temporary file. If (int)s is zero, tmpnam leaves its result in an internal static area and returns a pointer to that area. The next call to tmpnam destroys the contents of the area. If (int)s is nonzero, s is assumed to be the address of an array of at least L_tmpnam characters, where L_tmpnam is a constant defined in stdio.h; tmpnam places its result in that array and returns s as its value.

The tmpnam function generates a different file name each time it is called.

Files created using tmpnam and either fopen or creat are only temporary in the sense that they reside in a directory intended for temporary use, and their names are unique. You must use unlink to remove the file when its use is ended.

RELATED FUNCTIONS:

create, unlink, fopen, mktemp.

toascii

Character translation.

SYNTAX:

```
# include <ctype.h>

int toascii (c)
int c;
```

ARGUMENTS:

c

Character to translate.

DESCRIPTION:

The toascii function translates a character into 7-bit ASCII.

The toascii function yields its argument with all bits turned off that are not part of a standard 7-bit ASCII character; it is intended for compatibility with other systems.

RELATED FUNCTIONS:

ctype, getc, tolower, toupper.

tolower

Character translation.

SYNTAX:

```
# include <ctype.h>

int tolower (c)
int c;
```

ARGUMENTS:

c

Character to translate.

DESCRIPTION:

The tolower function has as a domain all 8-bit ASCII codes (hexadecimal 0 through FF). If the argument represents an uppercase letter, the result is the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

RELATED FUNCTIONS:

ctype, getc, toascii, toupper.

_tolower

Character translation.

SYNTAX:

```
# include <ctype.h>

int _tolower (c)
int c;
```

ARGUMENTS:

c

Character to translate.

DESCRIPTION:

The _tolower macrocall takes as an argument an uppercase letter. The result is the corresponding lowercase letter. All other arguments cause unspecified results.

RELATED FUNCTIONS:

ctype, getc, toascii, toupper.

toupper

Character translation.

SYNTAX:

```
# include <ctype.h>

int toupper (c)
int c;
```

ARGUMENTS:

c

Character to translate.

DESCRIPTION:

The toupper function has as a domain all 8-bit ASCII codes (hexadecimal 0 through FF). If the argument represents a lowercase letter that has a corresponding uppercase letter, the result is that uppercase letter. All other arguments in the domain are returned unchanged.

RELATED FUNCTIONS:

ctype, getc, toascii, tolower.

_toupper

Character translation.

SYNTAX:

```
# include <ctype.h>

int _toupper (c)
int c;
```

ARGUMENTS:

c

Character to translate.

DESCRIPTION:

The _toupper macrocall takes as an argument a lowercase letter that has a corrseponding uppercase letter. The result is the corresponding uppercase letter. All other arguments in the domain cause unspecified results.

RELATED FUNCTIONS:

ctype, getc, toascii, tolower.

tzset

Set time zone.

SYNTAX:

```
void tzset ()
```

DESCRIPTION:

The tzset function sets the external variables timezone, daylight, and tzname, using either the external variable TZ (if present) or the system time zone. It is called by the asctime function, but you can also call it directly.

The value of TZ must be a time zone acronym, a time offset, and an optional daylight-savings time zone acronym.

- The time zone acronym is up to four characters long.
- The time offset represents the difference between local time in the designated time zone and GMT. The difference is represented by a string of digits with an optional leading minus sign (for locations east of Greenwich, England) and with an optional trailing .5 (for locations some odd number of half-hours from Greenwich).
- The optional daylight savings time zone acronym is up to four characters long.

For example, the setting for Boston would be EST5EDT.

RELATED FUNCTIONS:

asctime, ctime, gmtime, localtime, time.

ulimit

Get and set user limits.

SYNTAX:

```
long ulimit (cmd, newlimit)
int cmd;
long newlimit;
```

ARGUMENTS:

cmd

The command to execute. The cmd values available are:

1 -- Get the file size limit of the process.

2 -- (On Multics the maximum segment size is set by system defaults).

3 -- Get the maximum possible allocation size.

newlimit

The new size.

DESCRIPTION:

This function provides for control over process limits.

RETURN VALUE:

Upon successful completion, a non-negative value is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

ungetc

Pushes character back into input file.

SYNTAX:

```
int ungetc (c, file)
char c;
FILE *file;
```

ARGUMENTS:

c

Character to push.

file

Pathname of input file.

DESCRIPTION:

The ungetc function pushes the character c back on an input file. That character is returned by the next getc call on that file. The ungetc function returns c.

One character of pushback is guaranteed provided something has been read from the file and the file is actually buffered. Attempts to push EOF are rejected.

DIAGNOSTICS:

The ungetc function returns EOF if it cannot push a character back.

RELATED FUNCTIONS:

getc, setbuf.

unlink

Removes directory entry.

SYNTAX:

```
int unlink (path)
char *path;
```

ARGUMENTS:

path

Pathname of directory entry.

DESCRIPTION:

The unlink function deletes the file entry named by the path argument. If path is a link, the link is removed. If path is a file, the file is deleted.

RETURN VALUE:

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the variable errno is set to indicate the error.

DIAGNOSTICS:

The unlink function fails if:

- The volume is write protected [EROFS].

RELATED FUNCTIONS:

close, link, open.

utime

Set file access and modification times.

SYNTAX:

```
#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;
```

ARGUMENTS:

path

Path points to a path name naming a file. utime sets the access and modification times of the named file.

times

On Multics utime can only change the access times to the current time.

DESCRIPTION:

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

```
struct    utimbuf  {
          time_t  actime;          /* access time */
          time_t  modtime;         /* modification time */
};
```

utime will fail if one or more of the following are true:

- The named file does not exist [ENOENT].
- A component of the path prefix is not a directory [ENOTDIR].
- Search permission is denied by a component of the path prefix [EACCES].

RETURN VALUE:

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

varargs

Handle variable argument list.

SYNTAX:

```
#include <varargs.h>

va_alist

va_dcl

    va_list get_arg(), last_arg();
    va_list pvar;

    pvar = get_arg;

    void va_start(pvar)
    va_list pvar;

    type va_arg(pvar, type)
    va_list pvar;

    void va_end(pvar)
    va list pvar;
```

DESCRIPTION:

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as printf) but do not use varargs are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for va_alist. No semicolon should follow va_dcl.

va_list is a type defined for the variable used to traverse the list.

get_arg is a routine that returns a va_list pointer to the required argument. Due to the argument list structure on Multics a direct relationship between an arguments address and its position in the argument list does not exist. get_arg is used to return the required va_list pointer to the argument. In the call pvar = get_arg; get_arg will return a va_list pointer to the third argument in the routines argument list.

last_arg returns a va_list pointer to the last argument being passed. This will usually be the return argument. last_arg is used by va_end to test for the last argument.

va_start is called to initialize pvar to the beginning of the list. If the argument is the first argument then va_start can be used to get a va_list pointer to the first argument. If an argument in another position is required as the starting position then get_arg must be used.

va_arg will return the next argument int he list pointed to by pvar. Type is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by va_start or get_arg ... va_end, are possible.

The following example is a possible implementation of execl:

```
#include <varargs.h>
#define MAXARGS     100

/*        execl is called by
                  execl(file, arg1, arg2, ..., (char *)0);
/*
execl(va_alist)
va_dcl
{

   va_list ap;
   char *file;
   char *args[MAXARGS];
   int argno = 0;

   va_start(ap);
   file = *va_arg(ap, char *);
   while ((args[argno++] = *va_arg(ap, char *)) != (char *)0)
            ;
   va_end(ap;
   return execv(file, args);
}
```

or execl could be done as follows using get_arg:

```
#include <varargs.h>
#define MAXARGS 100

/* execl is called by
          execl (file, arg1, arg2, .... NULL);
*/

execl (file, va_alist)
char *file;
va_dcl
{

  va_list ap;
  char *args [MAXARGS];
  int  argno = 0;

  ap = get_arg;  /* returns a va_list pointer to the
                          second argument */

  while ((args [argno++] = *va_arg (ap, char *)) != NULL)
          ;
  va_end(ap);
  return execv (file,args);
}
```

vprintf, vfprintf, vsprintf

Print formatted output of a varargs argument list.

SYNTAX:

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

int vprintf (format, ap)
char *format;
va_list ap;

int vfprintf  (stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf (s, format, ap)
char *s, *format;
va_list ap;
```

ARGUMENTS:

stream

A file pointer.

format

A pointer to a null-terminated string.

ap

A pointer to a varying argument list.

s

A pointer to the return value.

DESCRIPTION:

vprintf, vfprintf, and vsprintf are the same as printf, fprintf, and sprintf respectively, except that instead of being called with a variable number or arguments, they are called with an argument list as defined by varargs.

EXAMPLE:

The following demonstrates how vfprintf could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
.
.
.
/*
*  error should be called like
*       error (function_name, format, arg1, arg2...);
*/
/*VARARGS0*/
void
error(va_alist)
/* Note that the function_name and format arguments cannot be
*   separately declared because of the definition of varargs.
*/
va_dcl
{
   va_list args;
   char *fmt;

   va_start(args);
   /* print out name of function causing error */
  (void)fprint(stderr, "ERROR in %s ", va_arg(args, char *));
  fmt = va_arg(args, char *);
   /* print out remainder of message */
   (void)vfprintf(fmt, args);
   va_end(args);
   (void)abort ( );
}
```

write

Writes on a file.

SYNTAX:

```
int write (fildes, buf, nchars)
int fildes;
char *buf;
unsigned nchars;
```

ARGUMENTS:

fildes

File descriptor obtained from a creat, dup, open, or pipe function.

buf

Address of buffer containing characters to be written.

nchars

Number of characters to write.

DESCRIPTION:

The write function attempts to write nchars characters from the buffer pointed to by buf to the file associated with the file descriptor fildes.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from write, the file pointer is incremented by the number of characters actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is unspecified.

If the O APPEND file status flag is set, the file pointer is set to the end of the file before each write.

If a write requests that more characters be written than there is room for (ULIMIT or the physical end of a medium), only as many characters as there is room for will be written. For example, if there is space for 20 characters more in a file reaching a limit, a write of 512 characters returns 20. The next write of a nonzero number of characters gives a failure return (except as noted below).

The write function does not allocate a buffer until it is needed.

RETURN VALUE:

Upon successful completion, the number of characters actually written is returned. Otherwise, -1 is returned and the variable errno is set to indicate the error.

DIAGNOSTICS:

The write function fails and the file pointer is unchanged if:

- The fildes argument is not a valid file descriptor open for writing [EBADF].

RELATED FUNCTIONS:

creat, dup, open, pipe.

Appendix A
C COMPILER DIAGNOSTIC MESSAGES

This appendix lists the C compiler diagnostic messages in alphabetical order. In messages, [---] indicates a variable.

Table A-1 lists the C compiler diagnostic messages. These messages are written to the error-out file.

Table A-1. C Compiler Diagnostic Messages
(Sheet 1 of 3)

Message	Class
[name] evaluation order undefined	Warning
[name] may be used before set	Warning
[name] redefinition hides earlier one	Error
[name] set but not used in function [name]	Warning
[name] undefined	Error
bad structure offset	Error
[name] unused in function [name]	Warning
=<[character] illegal	Error
=>[character] illegal	Error
BCD constant exceeds 6 characters	Error
a function is declared as an argument	Warning
ambiguous assignment: simple assign, unary op assumed	Warning
argument [name] unused in function [name]	Warning
array of functions is illegal	Error
assignment of different structures	Error
bad ASM construction	Error
bad scalar initialization	Error
can't take & of [name]	Error
cannot initialize extern or union	Error
case not in switch	Error
comparison of unsigned with negative constant,	Warning
constant argument to NOT	Warning
constant expected	Error
constant in conditional context	Warning
constant too big for cross-compiler	Error
conversion from long may lose accuracy	Warning
conversion to long may sign-extend incorrectly	Warning
declared argument [name] is missing	Error
default not inside switch	Error
degenerate unsigned comparison	Warning
division by 0	Error
duplicate case in switch [number]	Error
duplicate default in switch	Error
empty array declaration	Warning
empty character constant	Error
enumeration type clash, operator [operator]	Warning
field outside of structure	Error
field too big	Error
fortran declaration must apply to function	Error
fortran function has wrong type	Error
fortran keyword nonportable	Warning
function [name] has return(e); and return;,	Warning
function declaration in bad content	Error

Table A-1. C Compiler Diagnostic Messages
(Sheet 2 of 3)

Message	Class
function has illegal storage class	Error
function illegal in structure or union	Error
function returns illegal type	Error
gcos BCD constant illegal	Error
illegal array size combination	Warning
illegal break	Error
illegal character: [number] (octal)	Error
illegal class	Error
illegal combination of pointer and integer, op [name]	Warning
illegal comparison of enums	Error
illegal continue	Error
illegal field size	Error
illegal field type	Error
illegal function	Error
illegal hex constant	Error
illegal indirection	Error
illegal initialization	Error
illegal lhs of assignment operator	Error
illegal member use: [name]	Error
illegal member use: [name]	Warning
illegal member use: perhaps [name].[name]	Warning
illegal pointer combination	Warning
illegal pointer subtraction	Error
illegal register declaration	Error
illegal structure pointer combination	Warning
illegal type combination	Error
illegal type in :	Error
illegal use of field	Error
illegal zero sized structure member: [name]	Warning
illegal {	Error
loop not entered at top	Warning
member of structure or union required	Error
newline in BCD constant	Error
newline in string or char constant	Error
no automatic aggregate initializer	Error
non-constant case expression	Error
non-null byte ignored in string initialization	Warning
nonportable character comparison	Warning
nonportable field type	Error
nonunique name demands struct/union or struct/union pointer	Error
null dimenstion	Error
null effect	Warning
old-fashioned assignment operator	Warning
old-fashioned initialization use =	Warning
operands of [operator] have incompatible types,	Error
pointer required	Error

Table A-2. C Compiler Diagnostic Messages
(Sheet 3 of 3)

Message	Class
possible pointer alignment problem	Warning
precedence confusion possible: parenthesize!	Warning
precision lost in assignment to (sign-extended?) field	Warning
precision lost in field assignment	Warning
questionable conversion of function pointer	Error
redeclaration of [name]	Error
redeclaration of formal parameter, [name]	Error
pointer casts may be troublesome	Warning
size of returns value less than or equal to zero	Warning
statement not reached	Warning
static variable [name] unused	Warning
struct/union [name] never defined	Warning
struct/union or struct/union pointer required	Warning
structure [name] never defined	Werror
structure reference must be addressable	Error
structure typed union member must be named	Warning
too many characters in character constant	Error
too many initializers	Error
type clash in conditional	Error
unacceptable operand of &	Error
undeclared initializer name [name]	Warning
undefined structure or union	Error
unexpected EOF	Error
unknown size	Error
unsigned comparison with 0?	Warning
void function [name] cannot return value	Error
void type for [name]	Uerror
void type illegal in expression	Error
zero or negative subscript	Warning
zero size field	Error
zero sized structure	Error
} expected	Error
long in case or switch statement may be truncated	Warning
bad octal digit [digit]	Warning
floating point constant folding causes exception	Error
old style assign-op causes syntax error	Warning
main() returns random value to invocation environment	Warning
'[name]' may be indistinguishable from '[name]' due to internal name truncation	Warning

Appendix B
C ENVIRONMENT SUPPORT COMMANDS

The following commands were ported from UNIX System V. They are available to aid in the porting process.

touch

Update access and modification times of a file.

SYNTAX:

touch [-amc] files

DESCRIPTION:

Touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. The current time is used. The -a and -m options cause touch to update only the access or modification times respectively (default is -am). The -c option silently prevents touch from creating the file if it did not previously exist.

The return code from touch is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

env

Set environment for command execution.

SYNTAX:

```
env [ - ] [ name=value ] ... [ command args ]
```

DESCRIPTION:

env obtains the current environment, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form name=value are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

GLOSSARY

byte

A Multics byte is nine bits long. In this manual, the terms byte and character are synonymous.

character

In this manual, the terms character and byte are synonymous.

character array

A sequence of characters.

file

File names consisting of up to 32 characters are allowed. The Multics file naming conventions are listed in the Multics Programmer's Reference Manual.

file access

File access is controlled in accordance with standard Multics conventions as described in the Multics Programmer's Reference Manual.

file descriptor

An integer from 0 to 19 that designates a file to be processed by low-level I/O. See low-level I/O.

heap

The area in which all memory allocation takes place, including all global and C static variables, but not including local variables.

high-level I/O

Functions (such as fopen and fprint) that return a pointer to a file. See low-level I/O.

low-level I/O

Functions (such as close, open, read, and write) that use file descriptors. See high-level I/O.

null character (NUL)

The ASCII character 00. In C, it is represented as \0.

null pointer

The value obtained by casting 0 into a pointer. This value never matches any legitimate pointer, so many functions that return pointers will return a null pointer to indicate an error.

search rules

Search rules are described in the Multics Programmer's Reference Manual.

string

A sequence of characters ending with a null character.

INDEX

HONEYWELL BULL
Technical Publications Remarks Form

TITLE **MULTICS C USER'S GUIDE**

ORDER NO. **HH07-01**

DATED **November 1987**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

PLEASE FILL IN COMPLETE ADDRESS BELOW.

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U.S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL BULL
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell Bull

Honeywell Bull

Corporate Headquarters:
3800 West 80th St., Minneapolis, MN 55431
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., North York, ON M2H 3P9
Mexico: Av. Constituyentes 900, 11950 Mexico, D.F. Mexico
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

49314, 1087, Printed in U.S.A.

HH07-01