NAME

intro − introduction to subroutines and libraries

SYNOPSIS

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

DESCRIPTION

This section describes functions that may be found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Functions of certain major collections are identified by a letter after the section number at the head of the page:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute library *libc,* which is automatically loaded by the C compiler, *cc*(1). The link editor *ld*(1) searches this library under the **−lc** option. Declarations for some of these functions may be obtained from "include" files indicated on the appropriate pages.

Since this release contains two versions of the C compiler, there are two versions of the C libraries supplied (*libc.a* for *cc*(1) and *liboc.a, liboa.a,* and *liboS.a* for *occ (1)).* The contents of the libraries are identical in interface and function unless otherwise indicated. Any differences are documented as follows: any manual page whose name does not end with :O is in the standard C library. If the routine is not the same in the old library, there will be another version of the manual page suffixed with :O. If the routine exists only in the old version of the library, there will exist only a manual page suffixed with :O.

(3M) These functions constitute the math library, *libm.* They are automatically loaded as needed by the Fortran compiler *f77*(1). The link editor searches this library under the **−lm** option. Declarations for these functions may be obtained from the "include" file **<math.h>**.

(3S) These functions constitute the "standard I/O package," see *stdio*(3S). These functions are in the library *libc*, already mentioned. Declarations for these functions may be obtained from the "include" file **<stdio.h>**.

(3X) Various specialized libraries. The files in which these libraries are found are named on the appropriate pages.

FILES

/lib/libc.a
/lib/liboc.a
/lib/liboa.a
/lib/liboS.a
/lib/libm.a
/lib/libplot.a

SEE ALSO

ar(1), cc(1), occ(1), f77(1), ld(1), nm(1), intro(2), stdio(3S), ostdio(3S), lib7(3X), libil(3X).

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable *errno* (see *intro*(2)) is set to the value EDOM or ERANGE.

ASSEMBLER

In assembly language, these functions may be accessed by simulating the C calling sequence. For example, *ecvt*(3C) might be called as follows:

```
.globl  _ecvt
   :
   :
setd
mov     $sign,−(sp)
mov     $decpt,−(sp)
mov     ndigit,−(sp)
movf    value,−(sp)
jsr     pc,_ecvt
add     $14.,sp
```

NAME

a64l, l64a − convert between long and base-64 ASCII

SYNOPSIS

long a64l (s)
char *s;

char *l64a (l)
long l;

DESCRIPTION

These routines are used to maintain numbers stored in *base-64* ASCII. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are **.** for 0, **/** for 1, **0** through **9** for 2−11, **A** through **Z** for 12−37, and **a** through **z** for 38−63.

A64l takes a pointer to a null-terminated base-64 representation and returns a corresponding **long** value. *L64a* takes a **long** argument and returns a pointer to the corresponding base-64 representation.

BUGS

The value returned by *l64a* is a pointer into a static buffer, the contents of which are overwritten by each call.

NAME

abort − generate an IOT fault

SYNOPSIS

abort ( )

DESCRIPTION

Abort executes the IOT instruction. This is usually considered a program fault by the system and results in termination with a core dump. It is used to generate a core image for debugging.

It is possible for *abort* to return control if **SIGIOT** is caught or ignored.

SEE ALSO

adb(1), signal(2), exit(2)

DIAGNOSTICS

Usually ‘abort − core dumped’ from the Shell.

NAME

abs – integer absolute value

SYNOPSIS

int abs (i)
int i;

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3M).

BUGS

You get what the hardware gives on the largest negative integer.

NAME

alloc − core allocator

SYNOPSIS

char *alloc (size)

DESCRIPTION

Alloc has been made obsolete by malloc(3C). It continues to exist for old programs which may still use it but it calls *malloc* to do all the work. *Alloc* is given a size in bytes; it returns a pointer which is even and hence can hold an object of any type, addressing an area of at least the requested size. A **−1** return indicates failure to allocate.

SEE ALSO

malloc(3C)

NAME

alrm − audible alarm

SYNOPSIS

alrmopen (name,mode)
alrmclos ( )
alrminor (time)
alrmajor ( )
alrmrset ( )
alrm (function)

char *name
int mode, function

DESCRIPTION

The *alrm* subroutines provide an interface the BD04 alarm panel driver. *Name* is the UNIX special file name of the BD04 device; *mode* lists the *open* permissions desired for the BD04 device - write permission must be granted or the interface subroutines will not work.

Alrmopen opens the UNIX file name associated with the BD04 and squirrels away the file descriptor returned by *open*(2). *Alrmopen* must be called first; if it is not then none of the other interface subroutines will work correctly. *Alrmclos* closes the file.

Alrminor causes a minor alarm of *time* seconds duration.

Alrmajor causes a major alarm; it stays on until an *alrmrset* .

Alrmrset turns off all alarms.

The *alrm* subroutine takes as an argument a function code:

0	Reset alarms.
1	Sound a 1 second minor alarm.
3	Sound a major alarm.

SEE ALSO

open(2)

DIAGNOSTICS

A −1 return indicates an error.

NAME

assert − program verification

SYNOPSIS

#include <assert.h>

assert (expression)

DESCRIPTION

This macro is useful for putting diagnostics into programs. When it is executed, if *expression* is false, it prints "Assertion failed: file *xyz* line *nnn*" on the standard error file and exits. *Xyz* is the source file and *nnn* the source line number of the *assert* statement. Compiling with the option **−DNDEBUG** will cause *assert* to be ignored.

NAME

atof, atoi, atol − convert ASCII to numbers

SYNOPSIS

double atof (nptr)
char *nptr;

atoi (nptr)
char *nptr;

long atol (nptr)
char *nptr;

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional 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.

Atoi and *atol* recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3S)

BUGS

There are no provisions for overflow.

NAME

atof, atoi, atol − convert ASCII to numbers

SYNOPSIS

double atof (nptr)
char *nptr;

atoi (nptr)
char *nptr;

long atoi (nptr)
char *nptr;

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof converts a string to a floating number. *Nptr* should point to a string containing an optional minus sign followed by a string of digits optionally containing one decimal point, then followed optionally by the letter **e**, followed by a signed integer.

Atoi and *atol* recognize an optional string of tabs and spaces, an optional '-' and then an unbroken string of digits.

DIAGNOSTICS

There are none; overflow results in a very large number and garbage characters terminate the scan.

BUGS

Atof should accept initial **+**, initial blanks, and **E** for **e**. Overflow should be signaled.

NAME

j0, j1, jn, y0, y1, yn − bessel functions

SYNOPSIS

```
#include <math.h>

double j0 (x)
double x;

double j1 (x)
double x;

double jn (n, x)
double x;

double y0 (x)
double x;

double y1 (x)
double x;

double yn (n, x)
int n;
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *y0, y1,* and *yn* to return a huge negative value.

NAME

lcall, vcall — create and execute a new process

SYNOPSIS

lcall (name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

vcall (name, argv)
char *name;

DESCRIPTION

The *call* system call has been removed from both the old and new C libraries. For compatibility with existing code, library interfaces to *lcall* and *vcall* have been provided which simply call *fork* and then *execl* or *execv*, respectively, with the appropriate arguments. The process id of the new process is returned from a successful *call.*

NOTE

The use of *call* is discouraged; use *fork* and *exec* instead.

SEE ALSO

fork(2), exec(2)

NAME

calloc, cfree − core memory allocator

SYNOPSIS

```
*calloc (size)
int size;

cfree (ptr)
int *ptr;
```

DESCRIPTION

Calloc and *cfree* provide a simple general-purpose memory allocation package. *Calloc* returns a pointer to a block containing zeros of at least *size* bytes beginning on a word boundary.

The argument to *cfree* is a pointer to an area previously allocated by *calloc*; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by *calloc* is overrun or if some random number is handed to *cfree*.

Calloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls *malloc* to get more core.

SEE ALSO

malloc(3C), break(2)

DIAGNOSTICS

Calloc returns a NULL (0) if there is no available memory.
Exit with the message corrupt arena means you have stored outside the bounds of a block. To get a core dump, use *adb*(1) to plant a breakpoint on *exit*(2).

NAME

clearer − stream error reset

SYNOPSIS

#include <stdio.h>

clearer (stream)

DESCRIPTION

Clearer resets the error indication on the named *stream*.

SEE ALSO

fopen (3S), open (2)

NAME

cnvtime, gtime − convert string to internal time

SYNOPSIS

#include <sys/types.h>

time_t cnvtime (year, month, day, hour, minute, second)
int year, month, day, hour, minute, second;

time_t gtime (str)
char *str;

SYNOPSIS

Cnvtime converts a time specified by *year*, *month*, *day*, *hour*, *minute*and*second* to the system's internal *time_t* form of storing time. *Cnvtime* will correct as required for daylight time and leap years. The time supplied as input must be a local time.

Gtime will also return a *time_t* but expects a string as input with the same format as the string supplied to the *date*(1) command. To reiterate the form of the string is **MMddhhmmyy** where **MM** is the month of the year, **dd** is the day of the month, **hh** is the 24 hour hour of the day, **mm** is the minute of the hour, and the optional **yy** is the last two digits of the year. If **yy** is not supplied the current year is assumed.

SEE ALSO

date(1), ctime(3C)

DIAGNOSTICS

A −1 is returned if the conversion can not be effected because of an invalid specification.

NAME

conns − connect to a remote system

SYNOPSIS

```
conns (telno, speed, modes, lname, class)
char *telno;
short speed, modes;
char **lname;
char *class;
```

DESCRIPTION

Conns will allocate the necessary hardware resources and attempt to place a phone call to the telephone number specified. If the telephone number begins with a slash ('/'), *conns* assumes that a hard wired connection is to be made and will not place a phone call. In either case, *conns* will set the line to the speed and mode specified. (*Speed* should be the integer value of the baud desired−e.g., 1200. *Modes* should be the desired initial line modes−see *ioctl*(2).)

If successful, *conns* will return a file desriptor that can be used to read and write from/to the remote system and deposit in **lname* a pointer to the pathname of the line that was selected to establish the connection. *Conns* will not return the file descriptor until carrier is detected.

For dialup calls *conns* will, if there is any equipment available, attempt to place the call twice before giving up. If the user wants to make more or fewer attempts the global integer **_con_try** should be assigned the number of attempts desired.

If *conns* returns a value less than zero, a connection could not be established. The possible error returns and their associated meanings are listed below:

−1	No carrier, busy, or no answer.
−2	All equipment in use.
−3	Bad speed specification.
−4	Bad telephone number.
−5	*Ioctl* failed.
−6	Bad L-devices file. (Max size of L-devices defined MAXDEV
−7	No equipment exists to make desired call.

The *class* argument is used to specify the type of equipment to be used for the call and what special action, if any, should be taken by *conns. Class* is of the form *[type[−flags]]* where *type* is a string that is required to match the first entry in the L-devices file if the entry is to be considered, *flags*, if present, may currently consist only of the letter **I**. If the call is a hard wired call *type* is ignored. If the **−I** flag is present *conns* will not actually place the call but merely determine if equipment to make the call is currently available. In this case the return value is as normal except the file descriptor that is returned for a success indication is not open. By way of example, to initiate a normal acu call *class* should be the string **ACU**. To inquire if such a call could be made without actually making the call *class* should be the string **ACU−I**.

Conns uses the *uucp*(1C) database to find available autodialers and datasets.

SEE ALSO

cu(1C), ct(1C), uucp(1C), cspeed(3C)

FILES

/usr/lib/uucp/L-devices

NAME

toupper, tolower, toascii − character translation

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

Toupper and *tolower* have as domain the range of *getc*: the integers from −1 through 255. If the argument of *toupper* represents a lower-case letter, the result is the corresponding upper-case letter. If the argument of *tolower* represents an upper-case letter, the result is the corresponding lower-case letter. All other arguments in the domain are returned unchanged.

_toupper and *_tolower* are macros that accomplish the same thing as *toupper* and *tolower* but have restricted domains and are faster. *_toupper* requires a lower-case letter as its argument; its result is the corresponding upper-case letter. *_tolower* requires an upper-case letter as its argument; its result is the corresponding lower-case letter. Arguments outside the domain cause garbage results.

Toascii yields its argument with all bits turned off that are not part of a standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

ctype(3C)

NAME

crypt, setkey, encrypt — DES encryption

SYNOPSIS

```
char *crypt (key, salt)
char *key, *salt;

setkey (key)
char *key;

encrypt (block, edflag)
char *block;
int edflag;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the *encrypt* entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3C)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

cspeed − convert baud to speed number

SYNOPSIS

int cspeed (baud)
int baud;

DESCRIPTION

Cspeed will map its integer argument to a speed number that is suitable for use by *ioctl(2)*. Thus, for example, if its argument is 9600, its return value is 13. If the argument cannot be mapped to a legal speed number, a **−1** is returned.

SEE ALSO

ioctl (2)

NAME

ctermid − generate file name for terminal

SYNOPSIS

#include <stdio.h>

char *ctermid (s)
char *s;

DESCRIPTION

Ctermid generates a string that refers to the controlling terminal for the current process when used as a file name.

If (int)*s* is zero, the string is stored in an internal static area, the contents of which are overwritten at the next call to *ctermid*, and the address of which is returned. If (int)*s* is nonzero, then *s* is assumed to point to a character array of at least **L_ctermid** elements; the string is placed in this array and the value of *s* is returned.

NOTES

The difference between *ctermid* and *ttyname* is that *ttyname* must be handed a file descriptor, and returns the actual name of the terminal associated with that file descriptor, where *ctermid* returns a magic string (**/dev/ln**) that will refer to the terminal if used as a file name. Thus *ttyname* is useless unless the process already has at least one file open to a terminal.

SEE ALSO

ttyname(3C)

NAME

ctime, localtime, gmtime, asctime, timezone − convert date and time to ASCII

SYNOPSIS

char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *ftime*(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
/*              @(#)time.h      2.1             */
/*
 * A pointer to this structure is
 * returned by localtime() and gmtime()
 */
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 saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan *timezone(−(60(**4+30), 0)* is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

SEE ALSO
ftime(2)

BUGS
The return values point to static data whose content is overwritten by each call.

NAME

ctime − convert date and time to ASCII

SYNOPSIS

```
char *ctime (tvec)
int tvec[2];

int *localtime (tvec)
int tvec[2];

int *gmtime (tvec)
int tvec[2];
```

DESCRIPTION

Ctime converts a time in the vector *tvec* such as returned by *time*(2) into ASCII and returns a pointer to a character string in the form:

Sun Sep 16 01:03:52 1973

All the fields have constant width.

The *localtime* and *gmtime* entries return integer vectors to the broken-down time. *Localtime* corrects for the time zone and possible Daylight Savings Time; *gmtime* converts directly to GMT, which is the time UNIX uses. The value is a pointer to an integer array whose components are:

0	seconds
1	minutes
2	hours
3	day of the month (1-31)
4	month (0-11)
5	year 1900
6	day of the week (Sunday = 0)
7	day of the year (0-365)
8	Daylight Saving Time flag if non-zero

The external variable *timezone* contains the difference, in seconds, between GMT and local standard time (in EST, is 5*60*60). The routine knows about Daylight Savings Time in the U.S.A, including the peculiarities of the conversion in 1974 and 1975; if necesary, a table for these years can be extended.

SEE ALSO

time(2)

BUGS

The algorithm fails in Saudi Arabia, which runs on Solar Time.

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii − character classification

SYNOPSIS

#include <ctype.h>

int isalpha (c)
int c;

. . .

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value **EOF** (see *stdio*(3S)).

isalpha	*c* is a letter
isupper	*c* is an upper case letter
islower	*c* is a lower case letter
isdigit	*c* is a digit
isalnum	*c* is an alphanumeric
isspace	*c* is a space, tab, carriage return, new-line, or form-feed
ispunct	*c* is a punctuation character (neither control nor alphanumeric)
isprint	*c* is a printing character, code 040 (space) through 0176 (tilde)
iscntrl	*c* is a delete character (0177(8)) or ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

NAME

cuserid − character user ID

SYNOPSIS

#include <stdio.h>

char *cuserid (s)
char *s;

DESCRIPTION

Cuserid generates a character representation of the user ID of the owner of the current process.

If (int)*s* is zero, this representation is generated in an internal static area, the address of which is returned. If (int)*s* is nonzero, *s* is assumed to point to an array of at least **L_cuserid** characters; the representation is left in this array.

DIAGNOSTICS

If the user ID cannot be found, *cuserid* returns **NULL**. if *s* is nonzero in this case, **\0** will be placed at **s*.

SEE ALSO

getlogin(3C), getpwuid(3C)

NAME

dtol, ltod − double precision integer to floating point conversion

SYNOPSIS

long dotl (d)
double d;

double ltod (t)
long t;

DESCRIPTION

Dtol converts a floating point integer to the equivalent long number. *Ltod* converts a long integer to the equivalent floating point number.

NOTE

These routines have been replaced by the appropriate type casting operations in later versions of the C libraries. Use **(long) t** and **(double) d** instead.

NAME

ecvt, fcvt − output conversion

SYNOPSIS

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

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

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

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to *ecvt*, except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by **ndigit.*

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf.* It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3S)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

end, etext, edata − last locations in program

SYNOPSIS

extern end;
extern etext;
extern edata;

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break coincides with *end*, but the program break may be reset by the routines of *brk*(2), *malloc*(3C), standard input/output (*stdio*(3S)), the profile (**−p**) option of *cc*(1), and so on. Thus, the current value of the program break should be determined by "**sbrk**(0)" (see *brk*(2)).

These symbols are accessible from assembly language if it is remembered that they should be prefixed by _.

SEE ALSO

break(2), malloc(3C)

NAME

exp, log, pow, sqrt − exponential, logarithm, power, square root

SYNOPSIS

#include <math.h>

double exp (x)
double x;

double log (x)
double x;

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

double sqrt (x)
double x;

DESCRIPTION

Exp returns the exponential function of *x*.

Log returns the natural logarithm of *x*.

Pow returns x^y.

Sqrt returns the square root of *x*.

SEE ALSO

hypot(3M), sinh(3M), intro(2)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow.

Log and *pow* return 0 when *x* is zero or negative.

Sqrt returns 0 when *x* is negative.

BUGS

Pow indicates error **ERANGE** (see *intro*(2)) for nonpositive *x* regardless of the value of *y*.

NAME

exprog — perform standard Shell execute sequence

SYNOPSIS

exprog (argv)
char *argv[];

DESCRIPTION

Exprog has been replaced by *execvp*(2) in the newer versions of the compiler. *Exprog* attempts to locate the file specified by *argv*[0] in the current directory. *Argv* should be an argument string in the format required by *execv* (see *exec*(2)). If the file does not exist, *exprog* prepends **/bin/** to *argv*[0] and trys again. Upon failure it further prepends **/usr** and makes one last attempt before returning with an error indication.

If the file is executable but the attempt to execute it fails (see *exec*(2) for reasons for failure) *exprog* passes the file to the shell for interpretation as a command file.

In all cases all arguments given to *exprog* in the argument vector are passed to the program or shell.

DIAGNOSTICS

A -1 is returned if there is no UNIX Shell. Otherwise if *exprog* returns, it returns the global system error number (errno) which describes why the execute was unsuccessful.

BUGS

Exprog uses the default command look-up strategy employed by the shell; however, if you have specified an alternate look-up sequence, *exprog* will continue to use the default strategy. See *sh*(1) for details on the shell look-up. Only 100 arguments may be passed to the shell by *exprog*, a generous but unnecessary restriction.

NAME

fclose, fflush − close or flush a stream

SYNOPSIS

```
#include <stdio.h>

int fclose (stream)
FILE *stream;

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

DESCRIPTION

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit*(2).

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

These functions return 0 for success, and EOF if any errors were detected.

SEE ALSO

close(2), fopen(3S), setbuf(3S)

NAME

ferror, feof, clearerr, fileno − stream status inquiries

SYNOPSIS

```
#include <stdio.h>

int feof (stream)
FILE *stream;

int ferror (stream)
FILE *stream;

clearerr (stream)
FILE *stream;

fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns non-zero when end of file is read on the named input *stream*, otherwise zero.

Ferror returns non-zero when error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clearerr resets the error indication on the named *stream*.

Fileno returns the integer file descriptor associated with the *stream*, see *open*(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3S), open(2)

NAME

floor, fabs, ceil, fmod − absolute value, floor, ceiling, remainder functions

SYNOPSIS

#include <math.h>

double floor (x)
double x;

double ceil (x)
double x;

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

double fabs (x)
double x;

DESCRIPTION

Fabs returns $|x|$.

Floor returns the largest integer (as a double precision number) not greater than x.

Ceil returns the smallest integer not less than x.

Fmod returns the number f such that $x = iy + f$, for some integer i, and $0 \leqslant f < y$.

SEE ALSO

abs(3C)

NAME

fopen, freopen, fdopen − open a stream

SYNOPSIS

#include <stdio.h>

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

FILE *freopen (filename, type, stream)
char *filename, *type;
FILE *stream;

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

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r"	open for reading
"w"	create for writing
"a"	append; open for writing at end of file, or create for writing
"r+"	open for update (reading and writing)
"w+"	create for update
"a+"	append; open or create for update at end of file

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

Freopen is typically used to attach the preopened constant names, **stdin, stdout, stderr,** to specified files.

Fdopen associates a stream with a file descriptor obtained from *open, dup, creat,* or *pipe*(2). The *type* of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening *fseek* or *rewind*, and input may not be directly followed by output without an intervening *fseek* , *rewind* , or an input operation which encounters end of file.

SEE ALSO

open(2), fclose(3S)

DIAGNOSTICS

Fopen and *freopen* return the pointer NULL if *filename* cannot be accessed.

NAME

fopen, freopen − open a stream

SYNOPSIS

#include <stdio.h>

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

FILE *freopen (filename, type, stream)
char *filename, *type;
FILE *stream;

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r"	open for reading
"w"	create for writing
"a"	append; open for writing at end of file, or create for writing

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

Freopen is typically used to attach the preopened constant names, **stdin, stdout, stderr,** to specified files.

DIAGNOSTICS

Fopen and *freopen* return the pointer NULL if *filename* cannot be accessed.

NAME

fpemul − floating point interpreter

SYNOPSIS

sys **signal; 4; fptrap**

DESCRIPTION

Fpemul is a simulator of the 11/45 FP11-B floating point unit. On a machine equipped with floating point hardware, the module contains a dummy routine which simply re-executes the trapped instruction. On machines without floating point hardware, it contains code to intercept illegal instruction faults and examine the offending operation codes for possible floating point operations, which are then emulated using non-floating instructions.

The emulation routines are automatically loaded only when required by modules using floating point definitions or operations.

SEE ALSO

signal(2), cc(1)

DIAGNOSTICS

A breakpoint trap is given when a real illegal instruction trap occurs.

BUGS

The emulation will not work with 411 (**−i** option) files, since *fpemul* needs to examine the offending instruction.

Rounding mode is not interpreted. The inefficiencies of using illegal instruction traps to emulate floating point seriously compromise speed; *fpemul* is very slow.

NAME

fread, fwrite − buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

int fread ((char *) ptr, sizeof (*ptr), nitems, stream)
FILE *stream;

int fwrite ((char *) ptr, sizeof (*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads, into a block beginning at *ptr, nitems* of data of the type of **ptr* from the named input *stream*. It returns the number of items actually read.

Fwrite appends at most *nitems* of data of the type of **ptr* beginning at *ptr* to the named output *stream*. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S)

DIAGNOSTICS

Fread and *fwrite* return the constant NULL upon end of file or error.

NAME

frexp, ldexp, modf − split into mantissa and exponent

SYNOPSIS

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

double ldexp (value, exp)
double value;

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

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x*, of magnitude less than 1 and stores an integer *n* such that *value* = *x**2***n* indirectly through *eptr*.

Ldexp returns the quantity *value**2***exp*.

Modf returns the positive fractional part of *value* and stores the integer part indirectly through *iptr*.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

```
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

long ftell (stream)
FILE *stream;

rewind(stream)
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of *ungetc*(3S).

After *fseek* or *rewind*, the next operation on an update file may be either input or output.

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream*. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and is the only foolproof way to obtain an *offset* for *fseek*.

Rewind(*stream*) is equivalent to *fseek*(*stream*, 0L, 0).

SEE ALSO

lseek(2), fopen(3S)

DIAGNOSTICS

Fseek returns non-zero for improper seeks, otherwise zero.

NAME

gamma − log gamma function

SYNOPSIS

#include <math.h>

double gamma (x)
double x;

DESCRIPTION

Gamma returns $ln|\Gamma(|x|)|$. The sign of $\Gamma(|x|)$ is returned in the external integer *signgam*. The following C program fragment might be used to calculate Γ:

```
y = gamma (x);
if (y > 88.0)
        error ();
y = exp (y) * signgam;
```

DIAGNOSTICS

A huge value is returned for negative integer arguments.

BUGS

There should be a positive indication of error.

NAME

getc, getchar, fgetc, getw − get character or word from stream

SYNOPSIS

#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar ( )

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;

DESCRIPTION

Getc returns the next character from the named input *stream*.

Getchar() is identical to *getc(stdin)*.

Fgetc behaves like *getc*, but is a genuine function, not a a macro; it may be used as an argument, or to save on object text.

Getw returns the next word from the named input *stream*. It returns the constant **EOF** upon end of file or error, but since that is a good integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*. *Getw* assumes no special alignment in the file.

SEE ALSO

fopen(3S), putc(3S), gets(3S), scanf(3S), fread(3S), ferror(3S)

DIAGNOSTICS

These functions return the integer constant **EOF** at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by *fopen*.

BUGS

Getc and its variant *getchar* return **EOF** on end of file; this is wiser than, but incompatible with, the older *getchar*(3S).
Because it is implemented as a macro, *getc* treats a *stream* argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

NAME

getc − buffered input

SYNOPSIS

fopen (filename, iobuf)
char *filename;
struct buf *iobuf;

getc (iobuf)
struct buf *iobuf;

getw (iobuf)
struct buf *iobuf;

DESCRIPTION

These routines are early versions of the standard I/O routines; they provide a buffered input facility. *Iobuf* is the address of a buffer area whose contents are maintained by these routines. Its format is:

```
struct buf {
        int fildes;             /* file descriptor
        int nleft;              /* characters left in buffer
        char *nextp;            /* pointer to next character
        char buffer[512];       /* the buffer
};
```

Fopen may be called initially to open the file. −1 is returned if the open failed. If *fopen* is never called, *getc* and *getw* will read from the standard input file.

Getc returns the next byte from the file; −1 is returned on end-of-file or error.

Getw returns the next word. *Getc* and *getw* may be used alternately; there are no odd/even problems.

Iobuf must be provided by the user; it must begin on a word boundary.

To reuse the same buffer for another file, it is sufficient to close the original file and call *fopen* again.

SEE ALSO

open(2), read(2), putc(3C)

DIAGNOSTICS

Negative return indicates error or EOF.

NAME
getchar − read character

SYNOPSIS
getchar ( )

DESCRIPTION
Getchar is a simple means of reading characters from the standard input. It remains in current versions of the C library (however, see note below). *Getchar* returns successive characters until end-of-file, when it returns "\0".

Associated with this routine is an external variable called *fin*, which is a structure containing a buffer such as described under *getc:o*(3C).

Generally speaking, *getchar* should be used only for the simplest applications; *getc* is better when there are multiple input files.

SEE ALSO
getc(3C)

DIAGNOSTICS
Null character returned on EOF or error.

BUGS
−1 should be returned on EOF; null is a legitimate character.

NOTE
In the *occ* version of the standard I/O library as well as later versions of the C libraries, *getchar* has been changed to return −1 on end-of-file.

NAME

getenv − value for environment name

SYNOPSIS

char *getenv (name)
char *name;

DESCRIPTION

Getenv searches the environment list (see *environ*(7)) for a string of the form *name*=*value* and returns *value* if such a string is present, otherwise 0 (NULL).

SEE ALSO

environ (7)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent − get group file entry

SYNOPSIS

#include <grp.h>

struct group *getgrent ( )

struct group *getgrgid (gid)
int gid;

struct group *getgrnam (name)
char *name;

int setgrent ( );

int endgrent ( );

DESCRIPTION

Getgrent, getgrgid and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
/*          @(#)grp.h      2.1          */
struct      group {
            char           *gr_name;
            char           *gr_passwd;
            int            gr_gid;
            char           **gr_mem;
};
```

The members of this structure are:

gr_name	The name of the group.
gr_passwd	The encrypted password of the group.
gr_gid	The numerical group-ID.
gr_mem	Null-terminated vector of pointers to the individual member names.

Getgrent reads the next line of the file, so successive calls may be used to search the entire file. *Getgrgid* and *getgrnam* search from the beginning of the file until a matching *gid* or *name* is found, or EOF is encountered.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

FILES

/etc/group

SEE ALSO

getlogin(3C), getpwent(3C), group(5)

DIAGNOSTICS

A null pointer (0) is returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getlogin − get login name

SYNOPSIS

char *getlogin ( )

DESCRIPTION

Getlogin returns a pointer to the login name as found in **/etc/utmp**. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same userid is shared by several login names.

If *getlogin* is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to call *cuserid*, or to call *getlogin* and if it fails, to call *getpwuid*.

FILES

/etc/utmp

SEE ALSO

cuserid(3S), getpwent(3C), getgrent(3C), utmp(5)

DIAGNOSTICS

Returns NULL (0) if name not found.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getopt − get option letter from argv

SYNOPSIS

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

DESCRIPTION

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

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

When all options have been processed (i.e., up to the first non-option argument), *getopt* returns **EOF**. The special option "− −" may be used to delimit the end of the options; **EOF** will be returned, and "− −" will be skipped.

DIAGNOSTICS

Getopt prints an error message on *stderr* 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 **o**, 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 (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;

                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc();
                        break;

                case 'f':
                        ifile = optarg;
                        break;

                case 'o':
                        ofile = optarg;
                        bufsiza = 512;
                        break;

                case '?':
                        errflg++;
                }
        if (errflg) {
                fprintf (stderr, "usage: . . . ");
                exit (2);
        }
        for( ; optind < argc; optind++) {
                if (access (argv[optind], 4)) {
        :
        :
}
```

NAME

getpass − read a password

SYNOPSIS

char *getpass (prompt)
char *prompt;

DESCRIPTION

Getpass reads a password from the file **/dev/ln**, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES

/dev/ln

SEE ALSO

crypt(3C)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getpw − get name from UID

SYNOPSIS

getpw (uid, buf)
int uid;
char *buf;

DESCRIPTION

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

This routine is included only for compatibility with prior systems and should not be used; see *getpwent*(3C) for routines to use instead.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), passwd(5)

DIAGNOSTICS

Non-zero return on error.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent − get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent ( )

struct passwd *getpwuid (uid)
int uid;

struct passwd *getpwnam (name)
char *name;

int setpwent ( )

int endpwent ( )
```

DESCRIPTION

Getpwent, *getpwuid*, and *getpwnam* each returns a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
/*              @(#)pwd.h       2.2             */
struct          passwd
{
                char            *pw_name;
                char            *pw_passwd;
                int             pw_uid;
                int             pw_gid;
                char            *pw_age;
                char            *pw_comment;
                char            *pw_gecos;
                char            *pw_dir;
                char            *pw_shell;
};
```

The *pw_comment* field is unused; the others have meanings described in *passwd*(5).

Getpwent reads the next line in the file, so successive calls can be used to search the entire file. *Getpwuid* and *getpwnam* search from the beginning of the file until a matching *uid* or *name* is found, or EOF is encountered.

A call to *setpwent* has the effect of rewinding the password file to allow repeated searches. *Endpwent* may be called to close the password file when processing is complete.

FILES

/etc/passwd

SEE ALSO

getlogin(3C), getgrent(3C), passwd(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

gets, fgets − get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets (s)
char *s;

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

DESCRIPTION

Gets reads a string into *s* from the standard input stream **stdin.** The string is terminated by a new-line character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads *n*−1 characters, or up to a new-line character (which is retained), whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), getc(3S), puts(3S), scanf(3S).

DIAGNOSTICS

Gets and *fgets* return the constant pointer NULL upon end-of-file or error.

NOTE

Gets deletes the new-line ending its input, but *fgets* keeps it.

NAME

getutent, getutid, getutline, pututline, setutent, endutent, utmpname − access utmp file entry

SYNOPSIS

```
#include <utmp.h>

struct utmp *getutent()

struct utmp *getutid(id)
struct utmp *id ;

struct utmp *getutline(line)
struct utmp *line ;

pututline(utmp)
struct utmp *utmp ;

setutent()

endutent()

utmpname(file)
char *file ;
```

DESCRIPTION

Getutent, *getutid*, and *getutline* each return a pointer to a structure of the following type:

```
/*              @(#)utmp.h      3.2             */

/*              <sys/types.h> must be included.                         */

#define         UTMP_FILE       "/etc/utmp"
#define         WTMP_FILE       "/etc/wtmp"

struct utmp
  {
                char ut_user[8] ;               /* User login name */
                char ut_id[2] ;                 /* /etc/lines id(usually line #) */
                char ut_line[12] ;              /* device name (console, lnxx) */
                short ut_pid ;                  /* process id */
                struct exit_status
                  {
                    char e_termination ;        /* Process termination status */
                    char e_exit ;               /* Process exit status */
                  }
                ut_exit ;                       /* The exit status of a process
                                                 * marked as DEAD_PROCESS.
                                                 */
                short ut_type ;                 /* type of entry */
                time_t ut_time ;                /* time entry was made */
  } ;

/*              Definitions for ut_type                                 */

#define         EMPTY           0
#define         RUN_LVL         1
#define         BOOT_TIME       2
#define         OLD_TIME        3
#define         NEW_TIME        4
#define         INIT_PROCESS    5               /* Process spawned by "init" */
#define         LOGIN_PROCESS 6                 /* A "getty" process waiting for login */
#define         USER_PROCESS    7               /* A user process */
#define         DEAD_PROCESS    8

#define         UTMAXTYPE   DEAD_PROCESS        /* Largest legal value of ut_type */
```

```
/*        Special strings or formats used in the "ut_line" field when        */
/*        accounting for something other than a process.                     */
/*        ** Note ** each message is such that is takes exactly 11           */
/*        spaces + a null, so that it fills the "ut_line" array.             */

#define   RUNLVL_MSG  "run_level_%c"
#define   BOOT_MSG    "system_boot"
#define   OTIME_MSG   "old_time   "
#define   NTIME_MSG   "new_time   "
```

Getutent reads in the next entry from a *utmp* like file. If the file is not already open, it opens it. If it reaches the end of the file, it fails.

Getutid searches forward from the current point in the *utmp* file until it finds an entry with a *ut_type* matching *id−>ut_type* if the type specified is **RUN_LVL**, **BOOT_TIME**, **OLD_TIME**, or **NEW_TIME**. If the type specified in *id* is **INIT_PROCESS**, **LOGIN_PROCESS**, **USER_PROCESS**, or **DEAD_PROCESS**, then *getutid* will return a pointer to the first entry whose type is one of these four and whose *ut_id* field matches *id−>ut_id.* If the end of file is reached without a match, it fails.

Getutline searches forward from the current point in the utmp file until it finds an entry of the type **LOGIN_PROCESS** or **USER_PROCESS** which also has a *ut_line* string matching *line−>ut_line* string. If the end of file is reached without a match, it fails.

Pututline writes out the supplied *utmp* structure into the utmp file. It uses *getutid* to search forward for the proper place if it finds that it is not already at the proper place. It is expected that normally the user of *pututline* will have searched for the proper entry using one of the *get* routines. If so, *pututline* will not search. If *pututline* does not find a matching slot for the new entry, it will add a new entry to the end of the file.

Setutent resets the input stream to the beginning of the file. This should be done inbetween each search for a new entry if it is desired that the entire file be examined.

Endutent closes the currently open file.

Utmpname allows the user to change the name of the file examined from **/etc/utmp** to any other file. It is most often expected that this other file will be **/etc/wtmp.** If the file doesn't exist, this will not be apparent until the first attempt to reference the file is made. *Utmpname* does not open the file. It just closes the old file if it is currently open and saves the new file name.

FILES

/etc/utmp,
/etc/wtmp

SEE ALSO

utmp(5)

DIAGNOSTICS

A **NULL** pointer is returned upon failure to read, whether for permissions or having reached the end of file, or upon failure to write.

COMMENTS

The most current entry is saved in a static structure. Multiple accesses require that it be copied before further accesses are made. Each call to either *getutid* or *getutline* sees the routine examine the static structure before performing more io. If the contents of the static structure match what it is searching for, it looks no further. For this reason to use *getutline* to search for multiple occurances, it would be necessary to zero out the static after each success, or *getutline* would just return the same pointer over and over again. There is one exception to the rule about removing the structure before further reads are done. The implicit read done by *pututline* if it finds that it isn't already at the correct place in the file will not hurt the contents of the static structure returned by the *getutent*, *getutid*, or *getutline* routines, if the user has just modified those contents and passed the pointer back to *pututline*.

These routines use buffered standand io for input, but *pututline* uses an unbuffered non-standard write to avoid race conditions between processes trying to modify the *utmp* and *wtmp* files.

NAME

hmul − high-order product

SYNOPSIS

hmul (x, y)

DESCRIPTION

Hmul returns the high-order 16 bits of the product of **x** and **y**. (The binary multiplication operator generates the low-order 16 bits of a product.)

NOTE

This routine has been deleted from later versions of the library. Use **long** variables instead.

NAME

hypot − euclidean distance

SYNOPSIS

#include <math.h>

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

DESCRIPTION

Hypot returns

sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

SEE ALSO

sqrt(3M)

NAME

itol − integer to long integer conversion

SYNOPSIS

long itol(hi, lo)
int hi, lo;

DESCRIPTION

Itol combines the two integers *hi* and *lo* to form a long integer. This allows integers to be converted to long integers without sign extension.

SEE ALSO

ltoi(3C)

NAME

l3tol, ltol3 − convert between 3-byte integers and long integers

SYNOPSIS

```
l3tol (lp, cp, n)
long *lp;
char *cp;
int n;

ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;
```

DESCRIPTION

L3tol converts a list of *n* three-byte integers packed into a character string pointed to by *cp* into a list of long integers pointed to by *lp*.

Ltol3 performs the reverse conversion from long integers (*lp*) to three-byte integers (*cp*).

These functions are useful for file-system maintenance where the i-numbers are three bytes long.

SEE ALSO

fs(5)