

NAME

intro — introduction to library functions

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. Functions are divided into various libraries distinguished by the section number at the top of the page:

- (3) 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) and the Fortran compiler *f77*(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.
- (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*(3). 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 have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

FILES

/lib/libc.a

/lib/libm.a, /usr/lib/libm.a (one or the other)

SEE ALSO

stdio(3), *nm*(1), *ld*(1), *cc*(1), *f77*(1), *intro*(2)

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. The values of EDOM and ERANGE are defined in the include file *<math.h>*.

ASSEMBLER (PDP-11)

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

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

NAME

abort — generate a fault

DESCRIPTION

Abort executes an instruction which is illegal in user mode. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO

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

DIAGNOSTICS

Usually 'IOT trap — core dumped' from the shell.

ABS (3)

ABS (3)

NAME

abs — integer absolute value

SYNOPSIS

abs(i)

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3) for *fabs*

BUGS

You get what the hardware gives on the smallest integer.

NAME

assert — program verification

SYNOPSIS

#include <assert.h>

assert (*expression*)

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be true at this point in the program. It causes an *exit*(2) with a diagnostic comment on the standard output when *expression* is false (0). Compiling with the *cc*(1) option **-DNDEBUG** effectively deletes *assert* from the program.

DIAGNOSTICS

'Assertion failed: file *f* line *n*.' *F* is the source file and *n* the source line number of the *assert* statement.

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(3)

BUGS

There are no provisions for overflow.

NAME

`crypt`, `setkey`, `encrypt` — DES encryption

SYNOPSIS

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

setkey(key)
char *key;

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

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(3)`

BUGS

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

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 *time*(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:

```
struct tm { /* see ctime(3) */
    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.

CTIME (3)

CTIME (3)

SEE ALSO
time(2)

BUGS

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

NAME

isalpha, *isupper*, *islower*, *isdigit*, *isalnum*, *isspace*, *ispunct*, *isprint*, *isctrl*, *isascii* — character classification

SYNOPSIS

```
#include <ctype.h>
```

```
isalpha(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*(3)).

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 character

isspace *c* is a space, tab, carriage return, newline, or formfeed

ispunct *c* is a punctuation character (neither control nor alphanumeric)

isprint *c* is a printing character, code 040(8) (space) through 0176 (tilde)

isctrl *c* is a delete character (0177) or ordinary control character (less than 040).

isascii *c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

NAME

dbminit, fetch, store, delete, firstkey, nextkey — data base subroutines

SYNOPSIS

```
typedef struct { char *dptr; int dsize; } datum;

dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey();

datum nextkey(key);
datum key;
```

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two filesystem accesses. The functions are obtained with the loader option `-ldb`.

Keys and *contents* are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has `.dir` as its suffix. The second file contains all data and has `.pag` as its suffix.

Before a database can be accessed, it must be opened by *dbminit*. At the time of this call, the files *file.dir* and *file.pag* must exist. (An empty database is created by creating zero-length `.dir` and `.pag` files.)

Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store*. A key (and its associated contents) is deleted by *delete*. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey*. *Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

DIAGNOSTICS

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr*.

BUGS

The `.pag` file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (`cp`, `cat`, `tp`, `tar`, `ar`) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 512 bytes). Moreover all key/content pairs that hash together must fit on a single block. *Store* will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by *firstkey* and *nextkey* depends on a hashing function, not on anything interesting.

NAME

ecvt, *fcvt*, *gcvt* — 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

*Ecv*t 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.

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

*Gcv*t 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(3)`

BUGS

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

END (3)

END (3)

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 resetting the program break by the routines of *brk(2)*, *malloc(3)*, standard input/output (*stdio(3)*), the profile (**-p**) option of *cc(1)*, etc. The current value of the program break is reliably returned by '*sbrk(0)*', see *brk(2)*.

SEE ALSO

brk(2), *malloc(3)*

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(3), *sinh*(3), *intro*(2)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow; *errno* is set to ERANGE. *Pow* returns 0 and sets *errno* to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when x is zero or negative; *errno* is set to EDOM.

Sqrt returns 0 when x is negative; *errno* is set to EDOM.

NAME

fclose, fflush — close or flush a stream

SYNOPSIS

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

```
fclose(stream)
```

```
FILE *stream;
```

```
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.

SEE ALSO

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

DIAGNOSTICS

These routines return EOF if *stream* is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

feof, *ferror*, *clearerr*, *fileno* — stream status inquiries

SYNOPSIS

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

```
feof(stream)
```

```
FILE *stream;
```

```
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 an error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clrerr 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(3), *open*(2)

NAME

fabs, *floor*, *ceil* — absolute value, floor, ceiling functions

SYNOPSIS

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

```
double floor(x)
```

```
double x;
```

```
double ceil(x)
```

```
double x;
```

```
double fabs(x)
```

```
double(x);
```

DESCRIPTION

Fabs returns the absolute value $|x|$.

Floor returns the largest integer not greater than x .

Ceil returns the smallest integer not less than x .

SEE ALSO

abs(3)

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)
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

Freopen substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed.

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.

SEE ALSO

`open(2)`, `fclose(3)`

DIAGNOSTICS

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

BUGS

Fdopen is not portable to systems other than UNIX.

NAME

fread, *fwrite* — buffered binary input/output

SYNOPSIS

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

```
fread(ptr, sizeof(*ptr), nitems, stream)
```

```
FILE *stream;
```

```
fwrite(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*(3), *getc*(3), *putc*(3), *gets*(3), *puts*(3), *printf*(3), *scanf*(3)

DIAGNOSTICS

Fread and *fwrite* return 0 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>
```

```
fseek(stream, offset, ptrname)
```

```
FILE *stream;
```

```
long offset;
```

```
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*(3).

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 the only foolproof way to obtain an *offset* for *fseek*.

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

SEE ALSO

lseek(2), *fopen*(3)

DIAGNOSTICS

Fseek returns -1 for improper seeks.

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 might be used to calculate Γ :

```
y = gamma(x);
if (y > 88.)
    error( );
y = exp(y);
if(signgam)
    y = -y;
```

DIAGNOSTICS

A huge value is returned for negative integer arguments.

BUGS

There should be a positive indication of error.

NAME

getarg, iargc
subroutine **getarg**(**argno**, **string**)
integer **argno**
character ***(*)** **string**
iargc()

DESCRIPTION

These procedures permit Fortran programs to access the command arguments. The integer function **iargc** returns the number of command arguments. The subroutine **getarg** stores the *n*th command argument in its second argument. The string is truncated or padded with blanks, in accord with the rules of Fortran character assignment.

The command

go **arg1** **argument2**
will return 2 as the value of **iargc**. If **s** is declared **character*4**, then
 call **getarg**(2, **s**)
will put "argu" in **s**.

SEE ALSO

exec(2)

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 macro; it may be used to save object text.

Getw returns the next word (32-bit integer under UNIX/32V) 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(3)* should be used to check the success of *getw*. *Getw* assumes no special alignment in the file.

SEE ALSO

`fopen(3)`, `putc(3)`, `gets(3)`, `scanf(3)`, `fread(3)`, `ungetc(3)`

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

The end-of-file return from *getchar* is incompatible with that in UNIX editions 1-6.

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

getenv — value for environment name

SYNOPSIS

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

DESCRIPTION

Getenv searches the environment list (see *environ*(5)) for a string of the form *name=value* and returns *value* if such a string is present, otherwise 0 (NULL).

SEE ALSO

environ(5), *exec*(2)

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.

```
struct group { /* see getgrent(3) */
    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 simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

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(3), getpwent(3), 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 first call *getlogin* and if it fails, to call *getpwuid*.

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), 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

getpass — read a password

SYNOPSIS

```
char *getpass(prompt)
char *prompt;
```

DESCRIPTION

Getpass reads a password from the file */dev/tty*, 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/tty

SEE ALSO

crypt(3)

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)
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.

FILES

/etc/passwd

SEE ALSO

getpwent(3), 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 return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
struct passwd { /* see getpwent(3) */
    char    *pw_name;
    char    *pw_passwd;
    int     pw_uid;
    int     pw_gid;
    int     pw_quota;
    char    *pw_comment;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};
```

The fields *pw_quota* and *pw_comment* are unused; the others have meanings described in *passwd(5)*.

Getpwent simply reads the next line while *getpwuid* and *getpwnam* search until a matching *uid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

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(3), getgrent(3), 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;
```

```
FILE *stream;
```

DESCRIPTION

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

Fgets reads *n*−1 characters, or up to a newline character, 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

puts(3), getc(3), scanf(3), fread(3), ferror(3)

DIAGNOSTICS

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

BUGS

Gets deletes a newline, *fgets* keeps it, all in the name of backward compatibility.

HYPOT (3M)

HYPOT (3M)

NAME

hypot, cabs — euclidean distance

SYNOPSIS

```
#include <math.h>

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

double cabs(z)
struct { double x, y;} z;
```

DESCRIPTION

Hypot and *cabs* return

$\sqrt{x*x + y*y}$,

taking precautions against unwarranted overflows.

SEE ALSO

exp(3) for *sqrt*

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)
```

```
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 and set *errno* to EDOM.

NAME

l3tol, *ltol3* — convert between 3-byte integers and long integers

SYNOPSIS

```
l3tol(lp, cp, n)  
long *lp;  
char *cp;  
ltol3(cp, lp, n)  
char *cp;  
long *lp;
```

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

filsys(5)

NAME

`malloc`, `free`, `realloc`, `calloc` — main memory allocator

SYNOPSIS

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

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

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

DESCRIPTION

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

The argument to *free* is a pointer to a block previously allocated by *malloc*; 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 *malloc* is overrun or if some random number is handed to *free*.

Malloc 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 *sbrk* (see *break(2)*) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Realloc also works if *ptr* points to a block freed since the last call of *malloc*, *realloc* or *calloc*; thus sequences of *free*, *malloc* and *realloc* can exploit the search strategy of *malloc* to do storage compaction.

Calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, *realloc* and *calloc* return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. *Malloc* may be recompiled to check the arena very stringently on every transaction; see the source code.

BUGS

When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

NAME

mktemp — make a unique file name

SYNOPSIS

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

DESCRIPTION

Mktemp 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 X's, which will be replaced with the current process id and a unique letter.

SEE ALSO

getpid(2)

NAME

`monitor` — prepare execution profile

SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc) ( ), (*highpc) ( );
short buffer [ ];
```

DESCRIPTION

An executable program created by ‘`cc -p`’ automatically includes calls for *monitor* with default parameters; *monitor* needn’t be called explicitly except to gain fine control over profiling.

Monitor is an interface to *profil*(2). *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. *Monitor* arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of *lowpc* and the highest is just below *highpc*. At most *nfunc* call counts can be kept; only calls of functions compiled with the profiling option `-p` of *cc*(1) are recorded. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see *end*(3).

To stop execution monitoring and write the results on the file *mon.out*, use

```
monitor(0);
```

then *prof*(1) can be used to examine the results.

FILES

mon.out

SEE ALSO

prof(1), *profil*(2), *cc*(1)

NAME

nlist — get entries from name list

SYNOPSIS

```
#include <a.out.h>
nlist(filename, nl)
char *filename;
struct nlist nll[];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See *a.out(5)* for the structure declaration.

This subroutine is useful for examining the system name list kept in the file */unix*. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

NAME

`perror`, `sys_errlist`, `sys_nerr` — system error messages

SYNOPSIS

```
perror(s)  
char *s;  
  
int sys_nerr;  
char *sys_errlist[];
```

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno* (see *intro(2)*), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the newline. *Sys_nerr* is the number of messages 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.

SEE ALSO

intro(2)

NAME

plot: openpl et al. — graphics interface

SYNOPSIS

```

openpl ( )
erase ( )
label(s) char s[ ];
line(x1, y1, x2, y2)
circle(x, y, r)
arc(x, y, x0, y0, x1, y1)
move(x, y)
cont(x, y)
point(x, y)
linemod(s) char s[ ];
space(x0, y0, x1, y1)
closepl ( )

```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See *plot(5)* for a description of their effect. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

String arguments to *label* and *linemod* are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following *ld(1)* options:

- lplot** device-independent graphics stream on standard output for *plot(1)* filters
- l300** GSI 300 terminal
- l300s** GSI 300S terminal
- l450** DASI 450 terminal
- l4014** Tektronix 4014 terminal

SEE ALSO

plot(5), *plot(1)*, *graph(1)*

NAME

popen, *pclose* — initiate I/O to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

DESCRIPTION

The arguments to *popen* are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO

pipe(2), *fopen*(3), *fclose*(3), *system*(3), *wait*(2)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns -1 if *stream* is not associated with a 'popened' command.

BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with *fflush*, see *fclose*(3).

NAME

printf, fprintf, sprintf — formatted output conversion

SYNOPSIS

```
#include <stdio.h>

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

fprintf(stream, format [, arg ] ... )
FILE *stream;
char *format;

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

DESCRIPTION

Printf places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places 'output' in the string *s*, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg printf*.

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign '-' which specifies *left adjustment* of the converted value in the indicated field;
- an optional digit string specifying a *field width*; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '.' which serves to separate the field width from the next digit string;
- an optional digit string specifying a *precision* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;
- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer *arg*. (A capitalized conversion code accomplishes the same thing.)
- a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer *arg* supplies the field width or precision.

The conversion characters and their meanings are

- d****ox** The integer *arg* is converted to decimal, octal, or hexadecimal notation respectively.
- f** The float or double *arg* is converted to decimal notation in the style '[−]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.
- e** The float or double *arg* is converted in the style '[−]d.ddde±dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.
- g** The float or double *arg* is printed in style **d**, in style **f**, or in style **e**, whichever gives full precision in minimum space.

- c** The character *arg* is printed. Null characters are ignored.
- s** *Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.
- u** The unsigned integer *arg* is converted to decimal and printed (the result will be in the range 0 through MAXUINT, where MAXUINT equals 4294967295 on a VAX-11 and 65536 on a PDP-11),
- %** Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by *putc(3)*.

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:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

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

SEE ALSO

putc(3), *scanf(3)*, *ecvt(3)*

BUGS

Very wide fields (>128 characters) fail.

NAME

`putc`, `putchar`, `fputc`, `putw` — put character or word on a stream

SYNOPSIS

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

```
int putc(c, stream)
```

```
char c;
```

```
FILE *stream;
```

```
putchar(c)
```

```
fputc(c, stream)
```

```
FILE *stream;
```

```
putw(w, stream)
```

```
FILE *stream;
```

DESCRIPTION

Putc appends the character *c* to the named output *stream*. It returns the character written.

Putchar(c) is defined as *putc(c, stdout)*.

Fputc behaves like *putc*, but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. **int**) *w* to the output *stream*. It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

The standard stream *stdout* is normally buffered if and only if the output does not refer to a terminal; this default may be changed by *setbuf(3)*. The standard stream *stderr* is by default unbuffered unconditionally, but use of *freopen* (see *fopen(3)*) will cause it to become buffered; *setbuf*, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. *Flush* (see *fclose(3)*) may be used to force the block out early.

SEE ALSO

fopen(3), *fclose(3)*, *getc(3)*, *puts(3)*, *printf(3)*, *fread(3)*

DIAGNOSTICS

These functions return the constant **EOF** upon error. Since this is a good integer, *ferror(3)* should be used to detect *putw* errors.

BUGS

Because it is implemented as a macro, *putc* treats a *stream* argument with side effects improperly. In particular '*putc(c, *f++);*' doesn't work sensibly.

NAME

`puts`, `fputs` — put a string on a stream

SYNOPSIS

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

```
puts(s)
```

```
char *s;
```

```
fputs(s, stream)
```

```
char *s;
```

```
FILE *stream;
```

DESCRIPTION

Puts copies the null-terminated string *s* to the standard output stream *stdout* and appends a newline character.

Fputs copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminal null character.

SEE ALSO

`fopen(3)`, `gets(3)`, `putc(3)`, `printf(3)`, `ferror(3)`

`fread(3)` for *fwrite*

BUGS

Puts appends a newline, *fputs* does not, all in the name of backward compatibility.

NAME

qsort — quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)();
```

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

NAME

rand, *srand* — random number generator

SYNOPSIS

***srand*(seed)**

int seed;

***rand*()**

DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever you like as argument.

NAME

scanf, fscanf, sscanf — formatted input conversion

SYNOPSIS

```
#include <stdio.h>

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

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

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

DESCRIPTION

Scanf reads from the standard input stream *stdin*. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format*, described below, and a 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 newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
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 non-space 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 legal:

- % 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 will be added. The input field is terminated by a space character or a newline.
- 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 non-space character, try '%ls'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.
- e 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 e followed by

an optionally signed integer.

- [indicates a string 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 circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized 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** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceded by **h** to indicate a pointer to **short** rather than to **int**.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

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

with the input line

```
25 54.32E-1 thompson
```

will assign to *i* the value 25, *x* the value 5.432, and *name* will contain 'thompson\0'. Or,

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

with input

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip '0123', and place the string '56\0' in *name*. The next call to *getchar* will return 'a'.

SEE ALSO

atof(3), getc(3), printf(3)

DIAGNOSTICS

The *scanf* functions return EOF on end of input, and a short count for missing or illegal data items.

BUGS

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

NAME

setbuf — assign buffering to a stream

SYNOPSIS

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

```
setbuf(stream, buf)
```

```
FILE *stream;
```

```
char *buf;
```

DESCRIPTION

Setbuf is used after a stream 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. If *buf* is the constant pointer NULL, input/output will be completely unbuffered.

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

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

A buffer is normally obtained from *malloc*(3) upon the first *getc* or *putc*(3) on the file, except that output streams directed to terminals, and the standard error stream *stderr* are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

NAME

`setjmp`, `longjmp` — non-local goto

SYNOPSIS

```
#include <setjmp.h>
```

```
setjmp(env)
```

```
jmp_buf env;
```

```
longjmp(env, val)
```

```
jmp_buf env;
```

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp*. It returns value 0.

Longjmp restores the environment saved by the last call of *setjmp*. It then returns in such a way that execution continues as if the call of *setjmp* had just returned the value *val* to the function that invoked *setjmp*, which must not itself have returned in the interim. All accessible data have values as of the time *longjmp* was called.

SEE ALSO

`signal(2)`

NAME

sin, *cos*, *tan*, *asin*, *acos*, *atan*, *atan2* — trigonometric functions

SYNOPSIS

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

```
double sin(x)
```

```
double x;
```

```
double cos(x)
```

```
double x;
```

```
double asin(x)
```

```
double x;
```

```
double acos(x)
```

```
double x;
```

```
double atan(x)
```

```
double x;
```

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

```
double x, y;
```

DESCRIPTION

Sin, *cos* and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π .

Atan returns the arc tangent of *x* in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of *x/y* in the range $-\pi$ to π .

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to EDOM. The value of *tan* at its singular points is a huge number, and *errno* is set to ERANGE.

BUGS

The value of *tan* for arguments greater than about 2^{31} is garbage.

NAME

sinh, cosh, tanh — hyperbolic functions

SYNOPSIS

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

```
double sinh(x)
```

```
double x;
```

```
double cosh(x)
```

```
double x;
```

```
double tanh(x)
```

```
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow.

SLEEP (3)

SLEEP (3)

NAME

sleep — suspend execution for interval

SYNOPSIS

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

alarm(2), pause(2)

NAME

stdio — standard buffered input/output package

SYNOPSIS

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

```
FILE *stdin;
```

```
FILE *stdout;
```

```
FILE *stderr;
```

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros *getc* and *putc(3)* handle characters quickly. The higher level routines *gets*, *fgets*, *scanf*, *fscanf*, *fread*, *puts*, *fputs*, *printf*, *fprintf*, *fwrite* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. *Fopen(3)* creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant 'pointer' **NULL** (0) designates no stream at all.

An integer constant **EOF** (−1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file *<stdio.h>* of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, *fileno*.

SEE ALSO

open(2), *close(2)*, *read(2)*, *write(2)*

DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with *fopen*, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

NAME

strcat, *strcatn*, *strcmp*, *strcmpn*, *strcpy*, *strcpyn*, *strlen*, *index*, *rindex* — string operations

SYNOPSIS

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

char *strcatn(s1, s2, n)
char *s1, *s2;

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

strcmpn(s1, s2, n)
char *s1, *s2;

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

char *strcpyn(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

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

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strcatn* copies at most *n* characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*. *Strcmpn* makes the same comparison but looks at at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been moved. *Strcpyn* copies exactly *n* characters, truncating or null-padding *s2*; the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*.

Strlen returns the number of non-null characters in *s*.

Index (*rindex*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or zero if *c* does not occur in the string.

BUGS

Strcmp uses native character comparison, which is signed on PDP11's and VAX-11's, unsigned on other machines.

NAME

swab — swap bytes

SYNOPSIS

swab(from, to, nbytes)
char *from, *to;

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. *Nbytes* should be even.

NAME

`system` — issue a shell command

SYNOPSIS

```
system(string)  
char *string;
```

DESCRIPTION

System causes the *string* to be given to *sh*(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

`popen`(3), `exec`(2), `wait`(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

ttyname, *isatty*, *ttyslot* — find name of a terminal

SYNOPSIS

char **ttyname*(*fildes*)

***isatty*(*fildes*)**

***ttyslot*()**

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes*.

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the *ttys*(5) file for the control terminal of the current process.

FILES

/dev/*

/etc/ttys

SEE ALSO

ioctl(2), *ttys*(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if *fildes* does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

BUGS

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

NAME

`ungetc` — push character back into input stream

SYNOPSIS

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

```
ungetc(c, stream)
```

```
FILE *stream;
```

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c*.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed back characters.

SEE ALSO

`getc`(3), `setbuf`(3), `fseek`(3)

DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.