NAME

abs — return integer absolute value

SYNOPSIS

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

DESCRIPTION

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

APPLICATION USAGE

In two-complement representation, the absolute value of the negative integer with largest magnitude (INT_MIN) is undefined. Some implementations may catch this as an error but others may ignore it.

SEE ALSO

FLOOR(BA_LIB).

LEVEL

Level 1.

EXCERPTS FROM SVID

BESSEL

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)
int n;
double x;

double y0(x)
double x;

double y1(x)
double x;

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

DESCRIPTION

The functions `j0` and `j1` return Bessel functions of `x` of the first kind of orders 0 and 1 respectively.

The function `jn` returns the Bessel function of `x` of the first kind of order `n`.

The functions `y0` and `y1` return Bessel functions of `x` of the second kind of orders 0 and 1 respectively.

The function `yn` returns the Bessel function of `x` of the second kind of order `n`.

For the functions `y0`, `y1` and `yn`, the argument `x` must be positive.

RETURN VALUE

Non-positive arguments cause `y0`, `y1` and `yn` to return the value −HUGE and to set `errno` to EDOM. In addition, a message indicating argument DOMAIN error is printed on the standard error output.

Arguments too large in magnitude cause the functions `j0`, `j1`, `y0` and `y1` to return zero and to set `errno` to ERANGE. In addition, a message indicating TLOSS error is printed on the standard error output [see MATHERR(BA_LIB)].

APPLICATION USAGE

These error-handling procedures may be changed with the MATHERR(BA_LIB) routine.

APPLICATION USAGE

The pointer returned by `seed48`, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time. Use the pointer to get at and store the last X_i value and then use this

SEE ALSO

RAND(BA_LIB).

LEVEL

Level 1.

ERF

NAME

erf, erfc — error function and complementary error function

SYNOPSIS

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

DESCRIPTION

The function `erf` returns the error function of `x`, defined as follows:

$$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

APPLICATION USAGE

The function `erfc` is provided because of the extreme loss of relative accuracy if `erf(x)` is called for large `x` and the result subtracted from `1.0`.

SEE ALSO

EXP(BA_LIB).

LEVEL

Level 1.

EXP

NAME

exp, log, log10, pow, sqrt — exponential, logarithm, power, square root functions

SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

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

double sqrt(x)
double x;
```

DESCRIPTION

The function `exp` returns e^x.

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

The function `log10` returns the logarithm base ten of `x`. The value of `x` must be positive.

The functions `pow` returns x^y. If `x` is zero, `y` must be positive. If `x` is negative, `y` must be an integer.

The function `sqrt` returns the non-negative square root of `x`. The value of `x` may not be negative.

RETURN VALUE

The function `exp` returns `HUGE` when the correct value would overflow or 0 when the correct value would underflow and sets `errno` to `ERANGE`.

The functions `log` and `log10` return `-HUGE` and set `errno` to `EDOM` when `x` is non-positive. A message indicating `DOMAIN` error (or `SING` error when `x` is 0) is printed on the standard error output.

The function `pow` returns 0 and sets `errno` to `EDOM` when `x` is 0 and `y` is non-positive, or when `x` is negative and `y` is not an integer. In these cases a message indicating `DOMAIN` error is printed on the standard error output. When the correct value for `pow` would overflow or underflow, `pow` returns `±HUGE` or 0 respectively and sets `errno` to `ERANGE`.

The function `sqrt` returns 0 and sets `errno` to `EDOM` when `x` is negative. A message indicating `DOMAIN` error is printed on the standard error output.

APPLICATION USAGE

These error-handling procedures may be changed with the MATHERR(BA_LIB) routine.

SEE ALSO

HYPOT(BA_LIB), MATHERR(BA_LIB), SINH(BA_LIB).

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file. This macro will call a function which will either return $+\infty$ on a system supporting the IEEE P754 standard or +(MAXDOUBLE) on a system that does not support the IEEE P754 standard.

The function `exp` will return `HUGE_VAL` when the correct value overflows.

The functions `log` and `log10` will return `-HUGE_VAL` when `x` is not positive.

The function `sqrt` will return `-0` when the value of `x` is `-0`.

The return value of `pow` will be negative `HUGE_VAL` when an illegal combination of input arguments is passed to `pow`.

LEVEL

Level 1.

FLOOR(BA_LIB)

NAME

floor, ceil, fmod, fabs -- floor, ceiling, remainder, absolute value 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

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

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

The function **fmod** returns the floating-point remainder of the division of **x** by **y**, **x** if **y** is zero or if **x/y** would overflow. Otherwise the number is *f* with the same sign as **x**, such that $x = iy + f$ for some integer *i*, and $|f| < |y|$.

The function **fabs** returns the absolute value of **x**, i.e., $|x|$.

SEE ALSO

ABS(BA_LIB).

LEVEL

Level 1.

FREXP

NAME

frexp, ldexp, modf — manipulate parts of floating-point numbers

SYNOPSIS

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

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

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

DESCRIPTION

Every non-zero number can be written uniquely as $x*2^n$, where the *mantissa* (fraction) x is in the range $0.5 \leq |x| < 1.0$ and the *exponent* n is an integer. The function `frexp` returns the mantissa of a `double value` and stores the exponent indirectly in the location pointed to by `eptr`. If `value` is `0`, both results returned by `frexp` are `0`.

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

The function `modf` returns the fractional part of `value` and stores the integral part indirectly in the location pointed to by `iptr`. Both the fractional and integer parts have the same sign as `value`.

RETURN VALUE

If `ldexp` would cause overflow, ±`HUGE` is returned (according to the sign of `value`) and `errno` is set to `ERANGE`.

If `ldexp` would cause underflow, `0` is returned and `errno` is set to `ERANGE`.

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file This macro will call a function which will either return $+\infty$ on a system supporting the IEEE P754 standard or +[MAXDOUBLE] on a system that does not support the IEEE P754 standard.

The return value of `ldexp` will be ±`HUGE_VAL` (according to the sign of `value`) in case of overflow.

LEVEL

Level 1.

NAME

gamma — log gamma function

SYNOPSIS

```
#include <math.h>

double gamma(x)
double x;

extern int signgam;
```

DESCRIPTION

The function `gamma` returns $\ln(|\Gamma(x)|)$, where $\Gamma(x)$ is defined as:

$$\int_0^\infty e^{-t} t^{x-1} dt$$

The sign of $\Gamma(x)$ is returned in the external integer `signgam`. The argument x may not be a non-positive integer.

The following C program fragment might be used to calculate Γ:

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

RETURN VALUE

For non-positive integer arguments `HUGE` is returned, and `errno` is set to `EDOM`. A message indicating `SING` error is printed on the standard error output [see MATHERR(BA_LIB)].

If the correct value would overflow, `gamma` returns `HUGE` and sets `errno` to `ERANGE`.

APPLICATION USAGE

These error-handling procedures may be changed with the MATHERR(BA_LIB) routine.

SEE ALSO

EXP(BA_LIB), MATHERR(BA_LIB).

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file. This macro will call a function which will either return $+\infty$ on a system supporting the IEEE P754 standard or +[MAXDOUBLE] on a system that does not support the IEEE P754 standard.

If the correct value overflows, `gamma` will return `HUGE_VAL`.

LEVEL

Level 1.

NAME

getc, getchar, fgetc, getw — get character or word from a stream

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

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

DESCRIPTION

The function `getc` returns the next character (i.e., byte) from the named input `stream` as an integer. It also moves the file pointer, if defined, ahead one character in `stream`. The macro `getchar` is defined as `getc(stdin)`. Both `getc` and `getchar` are macros.

The function `fgetc` behaves like `getc` but is a function rather than a macro. The function `fgetc` runs more slowly than `getc` but it takes less space per invocation and its name can be passed as an argument to a function.

The function `getw` returns the next word (i.e., integer) from the named input `stream`. The function `getw` increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. The function `getw` assumes no special alignment in the file.

RETURN VALUE

These functions return the constant `EOF` at end-of-file or upon an error. Because `EOF` is a valid integer, the FERROR(BA_OS) routine should be used to detect `getw` errors.

APPLICATION USAGE

If the integer value returned by `getc`, `getchar` or `fgetc` is stored into a character variable and then compared against the integer constant `EOF`, the comparison may never succeed because sign-extension of a character on widening to integer is machine-dependent.

Because of possible differences in word length and byte ordering, files written using `putw` are machine-dependent and may not be read using `getw` on a different processor.

Because it is implemented as a macro, `getc` treats incorrectly a `stream` argument with side effects. In particular, `getc(*f++)` does not work sensibly. The function `fgetc` should be used instead.

SEE ALSO

MALLOC(BA_OS), BSEARCH(BA_LIB), LSEARCH(BA_LIB), STRING(BA_LIB), TSEARCH(BA_LIB).

FUTURE DIRECTIONS

The restriction of having only one hash search table active at any given time will be removed.

LEVEL

Level 1.

NAME

hypot — Euclidean distance function

SYNOPSIS

```
#include <math.h>

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

DESCRIPTION

The function `hypot` returns `sqrt(x * x + y * y)`, taking precautions against unwarranted overflows.

RETURN VALUE

When the correct value would overflow, `hypot` returns `HUGE` and sets `errno` to `ERANGE`.

These error-handling procedures may be changed with the function defined by the MATHERR(BA_LIB) routine.

SEE ALSO

MATHERR(BA_LIB).

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file. This macro will call a function which will either return $+\infty$ on a system supporting the IEEE P754 standard or +[MAXDOUBLE] on a system that does not support the IEEE P754 standard.

The function `hypot` will return `HUGE_VAL` when the correct value overflows.

LEVEL

Level 1.

MATHERR

NAME

matherr — error-handling function

SYNOPSIS

```
#include <math.h>

int matherr(x)
struct exception *x;
```

DESCRIPTION

The function `matherr` is invoked by math library routines when errors are detected. Users may define their own procedures for handling errors, by including a function named `matherr` in their programs. The function `matherr` must be of the form described above. When an error occurs, a pointer to the `exception` structure `x` will be passed to the user-supplied `matherr` function. This structure, which is defined by the `<math.h>` header file, includes the following members:

```
int type;
char *name;
double arg1, arg2, retval;
```

The element `type` is an integer describing the type of error that has occurred from the following list defined by the `<math.h>` header file:

`DOMAIN`	argument domain error.
`SING`	argument singularity.
`OVERFLOW`	overflow range error.
`UNDERFLOW`	underflow range error.
`TLOSS`	total loss of significance.
`PLOSS`	partial loss of significance.

The element `name` points to a string containing the name of the routine that incurred the error. The elements `arg1` and `arg2` are the first and second arguments with which the routine was invoked.

The element `retval` is set to the default value that will be returned by the routine unless the user's `matherr` function sets it to a different value.

If the user's `matherr` function returns non-zero, no error message will be printed, and `errno` will not be set.

If the function `matherr` is not supplied by the user, the default error-handling procedures, described with the math library routines involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, `errno` is set to `EDOM` or `ERANGE` and the program continues.

ERRORS

DEFAULT ERROR HANDLING PROCEDURES

	Types of Errors					
type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE	ERANGE
BESSEL:	–	–	–	–	M, 0	*
y0, y1, yn (arg < 0)	M, -H	–	–	–	–	–
EXP:	–	–	H	0	–	–
LOG, LOG10:						
(arg < 0)	M, -H	–	–	–	–	–
(arg = 0)	–	M, -H	–	–	–	–
POW:	–	–	±H	0	–	–
neg ** non-int	M, 0	–	–	–	–	–
0 ** non-pos						
SQRT:	M, 0	–	–	–	–	–
GAMMA:	–	M, H	H	–	–	–
HYPOT:	–	–	H	–	–	–
SINH:	–	–	±H	–	–	–
COSH:	–	–	H	–	–	–
SIN, COS, TAN:	–	–	–	–	M, 0	*
ASIN, ACOS, ATAN2:	M, 0	–	–	–	–	–

ABBREVIATIONS

*	As much as possible of the value is returned.
M	Message is printed (EDOM error).
H	HUGE is returned.
-H	-HUGE is returned.
±H	HUGE or -HUGE is returned.
0	0 is returned.

EXAMPLE

```
#include <math.h>

int matherr(x)
register struct exception *x;
{
    switch (x->type) {
    case DOMAIN:
        /* change sqrt to return sqrt(-arg1), not 0 */
        if (!strcmp(x->name, "sqrt")) {
            x->retval = sqrt(-x->arg1);
            return (0);  /* print message and set errno */
        }
    case SING:
        /* SING or other DOMAIN errs, print message and abort */
        fprintf(stderr, "domain error in %s\n", x->name);
        abort();
    case PLOSS:
        /* print detailed error message */
        fprintf(stderr, "loss of significance in %s(%g) = %g\n",
            x->name, x->arg1, x->retval);
        return (1);   /* take no other action */
    }
    return (0); /* all other errors, execute default procedure */
}
```

FUTURE DIRECTIONS

The math functions which return HUGE or ±HUGE on overflow will return HUGE_VAL or ±HUGE_VAL respectively.

LEVEL

Level 1.

NAME

memccpy, memchr, memcmp, memcpy, memset — memory operations

SYNOPSIS

```
#include <memory.h>

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

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

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

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

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

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

The function memccpy copies characters from memory area s2 into s1, stopping after the first occurrence of character c has been copied or after n characters have been copied, whichever comes first. It returns a pointer to the character after the copy of c in s1, or a NULL pointer if c was not found in the first n characters of s2.

The function memchr returns a pointer to the first occurrence of character c in the first n characters of memory area s, or a NULL pointer if c does not occur.

The function memcmp compares its arguments, looking at the first n characters only. It returns an integer less than, equal to or greater than 0, according as s1 is lexicographically less than, equal to or greater than s2.

The function memcpy copies n characters from memory area s2 to s1. It returns s1.

The function memset sets the first n characters in memory area s to the value of character c. It returns s.

NAME

printf, fprintf, sprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>

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

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

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

DESCRIPTION

The function **printf** places output on the standard output stream **stdout**.

The function **fprintf** places output on the named output **stream**.

The function **sprintf** places output, followed by the null character (**\0**) in consecutive bytes starting at ***s**. It is the user's responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the **\0** in the case of **sprintf**) or a negative value if an output error was encountered.

Each of these functions converts, formats and prints its **args** under control of the **format**. The **format** is a character-string that contains three types of objects defined below:

1. plain-characters that are simply copied to the output stream;
2. escape-sequences that represent non-graphic characters; and
3. conversion-specifications.

The following escape-sequences produce the associated action on display devices capable of the action:

\b Backspace.
Moves the printing position to one character before the current position, unless the current position is the start of a line.

\f Form Feed.
Moves the printing position to the initial printing position of the next logical page.

\n New line.
Moves the printing position to the start of the next line.

\r Carriage return.
Moves the printing position to the start of the current line.

\t Horizontal tab.
Moves the printing position to the next implementation defined horizontal tab position on the current line.

\v Vertical tab.
Moves the printing position to the start of the next implementation-defined vertical tab position.

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

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional string of decimal digits to specify a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (−), described below, has been given) to the field width.

A *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions (the field is padded with leading zeros), the number of digits to appear after the decimal point for the **e**, **E** and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion; or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a null digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) to specify that a following **d**, **i**, **o**, **u**, **x** or **X** conversion character applies to a long integer **arg**. An **l** before any other conversion character is ignored.

A conversion character (see below) that indicates the type of conversion to be applied.

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

negative, it will be changed to zero.

The *flag* characters and their meanings are:

–	The result of the conversion will be left-justified within the field.
+	The result of a signed conversion will always begin with a sign (+ or -).
blank	If the first character of a signed conversion is not a sign, a blank will be prepended to the result. This means that if the blank and + flags both appear, the blank flag will be ignored.
#	The value is to be converted to an *alternate form*. For c, d, i, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x or X conversion, a non-zero result will have 0x or 0X prepended to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeroes will *not* be removed from the result as they normally are.

Each conversion character results in fetching zero or more **arg**s. The results are undefined if there are insufficient **arg**s for the format. If the format is exhausted while **arg**s remain, the excess **arg**s are ignored.

The conversion characters and their meanings are:

d,i,o,u,x,X	The integer **arg** is converted to signed decimal (**d** or **i**), **unsigned octal (o)**, unsigned decimal (**u**), or unsigned hexadecimal notation (**x** and **X**). The **x** conversion uses the letters **abcdef** and the **X** conversion uses the letters **ABCDEF**. The *precision* component of **arg** specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it will be expanded with leading zeroes. The default precision is **1**. The result of converting a zero value with a precision of **0** is a null string.
f	The float or double **arg** is converted to decimal notation in the style **[–]ddd.ddd**, where the number of digits after the decimal point is equal to the *precision* specification. If the *precision* is omitted from **arg**, six digits are output; if the *precision* is explicitly **0**, no decimal point appears.
e,E	The float or double **arg** is converted to the style **[–]d.dddde±dd**, where there is one digit before the decimal point and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is **0**, no decimal point appears. The **E** conversion character will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits. However, if the value to be printed is greater than or equal to **1E+100**, additional exponent digits will be printed as necessary.
g,G	The float or double **arg** is printed in style **f** or **e** (or in style **E** in the case of a **G** conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** will be used only if the exponent resulting from the conversion is less than **–4** or greater than the precision. Trailing zeroes are removed from the result. A decimal point appears only if it is followed by a digit.
c	The character **arg** is printed.
s	The **arg** is taken to be a string (character pointer) and characters from the string are printed until a null character (**\0**) is encountered or the number of characters indicated by the *precision* specification of **arg** is reached. If the precision is omitted from **arg**, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for **arg** will yield undefined results.
%	Print a %; no argument is converted.

If the character after the % is not a valid conversion character, the results of the conversion are undefined.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by **printf** and **fprintf** are printed as if the PUTC(BA_LIB) routine had been called.

RETURN VALUE

The functions **printf**, **fprintf**, and **sprintf** return the number of characters transmitted, or return −1 if an error was encountered.

EXAMPLE

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 %i, %d:%.2d",
        weekday, month, day, hour, min);
```

To print π to 5 decimal places:

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

SEE ALSO

PUTC(BA_LIB), SCANF(BA_LIB), FOPEN(BA_OS).

FUTURE DIRECTIONS

The function **printf** will make available character string representations for ∞ and "not a number" (NaN: a symbolic entity encoded in floating point format) to support the **IEEE P754** standard.

LEVEL

Level 1.

NAME

scanf, fscanf, sscanf -- convert formatted input

SYNOPSIS

```
#include <stdio.h>

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

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

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

DESCRIPTION

The function **scanf** reads from the standard input stream **stdin**.

The function **fscanf** reads from the named input **stream**.

The function **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. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing the character *, a decimal digit string that specifies an optional numerical maximum field width, an optional letter **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

The conversion characters **d**, **u**, **o**, **x**, and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

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

RETURN VALUE

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

APPLICATION USAGE

Trailing white space (including a new-line) is left unread unless matched in the control string.

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

EXAMPLE

The call to the function **scanf**:

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

with the input line:

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

will assign to **n** the value **3**, to **i** the value **25**, to **x** the value **5.432**, and **name** will contain **thompson\0**.

The call to the function **scanf**:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0-9]", &i, &x, name
```

with the input line:

```
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** [see GETC(BA_LIB)] will return **a**.

SEE ALSO

GETC(BA_LIB), PRINTF(BA_LIB), STRTOD(BA_LIB), STRTOL(BA_LIB).

FUTURE DIRECTIONS

The function **scanf** will make available character string representations for ∞ and "not a number" (NaN: a symbolic entity encoded in floating point format) to support the IEEE P754 standard.

LEVEL

Level 1.

NAME

signal — specify what to do upon receipt of a signal

SYNOPSIS

```
#include <signal.h>

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

DESCRIPTION

The function `signal` allows the calling-process to choose one of three ways in which it is possible to handle the receipt of a specific signal.

The argument `sig` specifies the signal and the argument `func` specifies the choice. The argument `sig` can be assigned any one of the following signals except `SIGKILL`:

`SIGHUP`	hangup
`SIGINT`	interrupt
`SIGQUIT`	quit*
`SIGILL`	illegal instruction (not reset when caught)*
`SIGTRAP`	trace trap (not reset when caught)*
`SIGFPE`	floating point exception*
`SIGKILL`	kill (cannot be caught or ignored)
`SIGSYS`	bad argument to routine*
`SIGPIPE`	write on a pipe with no one to read it
`SIGALRM`	alarm clock
`SIGTERM`	software termination signal
`SIGUSR1`	user-defined signal 1
`SIGUSR2`	user-defined signal 2

For portability, application-programs should use or catch *only* the signals listed above; other signals are hardware and implementation-dependent and may have very different meanings or results across systems (For example, the System V signals `SIGEMT`, `SIGBUS`, `SIGSEGV`, and `SIGIOT` are implementation-dependent and are not listed above). Specific implementations may have other implementation-dependent signals.

* The default action for these signals is an abnormal process termination. See `SIG_DFL`.

The argument `func` is assigned one of three values: `SIG_DFL`, `SIG_IGN`, or an *address* of a signal-catching function. The following actions are prescribed by these values:

`SIG_DFL` Terminate process upon receipt of a signal.

Upon receipt of the signal `sig`, the receiving process is to be terminated with all of the consequences outlined in EXIT(BA_OS). In addition, if `sig` is one of the signals marked with an asterisk above, implementation-dependent abnormal process termination routines, such as a core dump, may be invoked.

`SIG_IGN` Ignore signal.

The signal `sig` is to be ignored.

NOTE: The signal `SIGKILL` cannot be ignored.

address Catch signal.

Upon receipt of the signal `sig`, the receiving process is to execute the signal-catching function pointed to by `func`. The signal number `sig` will be passed as the only argument to the signal-catching function. Additional arguments may be passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of `func` for the caught signal will be set to `SIG_DFL` unless the signal is `SIGILL`, or `SIGTRAP`.

The function `signal` will not catch an invalid function argument, `func`, and results are undefined when an attempt is made to execute the function at the bad address.

Upon return from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted, except for implementation defined signals where this may not be true.

When a signal to be caught occurs during a non-atomic operation such as a call to a READ(BA_OS), WRITE(BA_OS), OPEN(BA_OS), or IOCTL(BA_OS) routine on a slow device (such as a terminal); or occurs during a PAUSE(BA_OS) routine; or occurs during a WAIT(BA_OS) routine that does not return immediately, the signal-catching function will be executed and then the interrupted routine may return a -1 to the calling-process with `errno` set to `EINTR`.

NOTE: The signal `SIGKILL` cannot be caught.

A call to the function `signal` cancels a pending signal `sig` except for a pending `SIGKILL` signal.

RETURN VALUE

If successful, the function `signal` will return the previous value of the argument `func` for the specified signal `sig`; otherwise, it will return `(int(*)())-1` and `errno` will indicate the error.

ERRORS

The function `signal` will fail and will set `errno` to:

`EINVAL` if `sig` is an illegal signal number or `SIGKILL`.

APPLICATION USAGE

Signals may be sent by the system to an application-program (user-level process) or signals may be sent by one user-level process to another using the KILL(BA_OS) routine. An application-program can catch signals and specify the action to be taken using the SIGNAL(BA_OS) routine. The signals that a portable application-program may *send* are: `SIGKILL`, `SIGTERM`, `SIGUSR1`, and `SIGUSR2`.

For portability, application-programs should use only the symbolic names of signals rather than their values and use only the set of signals defined here. Specific implementations may have additional signals.

SEE ALSO

KILL(BA_OS), PAUSE(BA_OS), WAIT(BA_OS), SETJMP(BA_LIB).

FUTURE DIRECTIONS

`SIGABRT` will be added to the `<signal.h>` header file [see ABORT(BA_OS)].

A macro `SIG_ERR` will be defined by the `<signal.h>` header file to represent the return value `(int(*)())-1` of the function `signal` in case of error.

The end-user level utility KILL(BU_CMD) will be changed to use symbolic signal names rather than numbers.

In keeping with the proposed ANSI X3J11 standard, the argument `func` will be declared as type pointer to a function returning `void`.

The following functions will be added to enhance the signal facility: `sigset`, `sighold`, `sigrelse`, `sigignore` and `sigpause`. These functions will give a calling-process control over the disposition of a specified signal that follows a signal that has been caught. When a signal has been caught, the system will hold (defer) a succeeding signal of the type specified should it occur. Similarly, processes will be able to establish critical regions of code where an incoming-signal is deferred so the critical region can be executed without losing the signal. Finally, a calling process will be able to suspend if a specified signal has not yet occurred.

LEVEL

Level 1.

NAME

sleep — suspend execution for interval

SYNOPSIS

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

DESCRIPTION

The function `sleep` suspends the current-process from execution for the number of seconds specified by the argument `seconds`. The actual suspension-time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals (on the second, according to an internal clock) and (2) because any signal caught will terminate the `sleep` following execution of that signal-catching routine. Also, the suspension-time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system.

The function `sleep` sets an alarm signal and pauses until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling-process may have set up an alarm signal before calling the function `sleep`. If the argument `seconds` exceeds the time until such an alarm signal would occur, the process sleeps only until the alarm signal would have occurred. The alarm signal-catching routine of the calling-process is executed just before the function `sleep` returns. But if the suspension-time is less than the time till such alarm, the prior alarm time remains unchanged.

RETURN VALUE

If successful, the function `sleep` will return the *unslept* amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested suspension-time or premature arousal due to another caught signal; otherwise, the function `sleep` will return `0`.

SEE ALSO

ALARM(BA_OS), PAUSE(BA_OS), SIGNAL(BA_OS).

LEVEL

Level 1.

SEE ALSO

SIGNAL(BA_OS).

LEVEL

Level 1.

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

The functions `sinh`, `cosh`, and `tanh` return, respectively, the hyperbolic sine, cosine and tangent of their argument.

RETURN VALUE

The functions `sinh` and `cosh` return `HUGE`, and `sinh` may return `-HUGE` for negative `x`, when the correct value would overflow and set `errno` to `ERANGE`.

APPLICATION USAGE

These error-handling procedures may be changed with the MATHERR(BA_LIB) routine.

SEE ALSO

MATHERR(BA_LIB).

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file. This macro will call a function which will either return $+\infty$ on a system supporting the IEEE P754 standard or +(MAXDOUBLE) on a system that does not support the IEEE P754 standard.

The functions `sinh` and `cosh` will return `HUGE_VAL` (`sinh` will return `-HUGE_VAL` for negative `n`) when the correct value overflows.

LEVEL

Level 1.

APPLICATION USAGE

All these functions are declared by the `<string.h>` header file.

Both `strcmp` and `strncmp` use native character comparison. The sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

SEE ALSO

MEMORY(BA_LIB).

FUTURE DIRECTIONS

The type of argument `n` to `strncat`, `strncmp` and `strncpy` and the type of value returned by `strlen` will be declared through the `typedef` facility in a header file as `size_t`.

LEVEL

Level 1.

NAME

strtod, atof — convert string to double-precision number

SYNOPSIS

```
double strtod(str, ptr)
char *str, **ptr;

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

DESCRIPTION

The function `strtod` returns as a double-precision floating-point number the value represented by the character string pointed to by `str`. The string is scanned up to the first unrecognized character.

The function `strtod` recognizes an optional string of *white-space* characters [as defined by `isspace` in CTYPE(BA_LIB)], then an optional sign, then a string of digits optionally containing a decimal point, then an optional `e` or `E` followed by an optional sign, followed by an integer.

If the value of `ptr` is not `((char **)0)`, a pointer to the character terminating the scan is returned in the location pointed to by `ptr`. If no number can be formed, `*ptr` is set to `str`, and 0 is returned.

The function call `atof(str)` is equivalent to:

```
strtod(str, (char **)0)
```

RETURN VALUE

If the correct value would cause overflow, ±`HUGE` is returned (according to the sign of the value) and `errno` is set to `ERANGE`.

If the correct value would cause underflow, zero is returned and `errno` is set to `ERANGE`.

APPLICATION USAGE

The function `strtod` was added to System V in System V Release 2.0.

SEE ALSO

CTYPE(BA_LIB), SCANF(BA_LIB), STRTOL(BA_LIB).

FUTURE DIRECTIONS

A macro `HUGE_VAL` will be defined by the `<math.h>` header file. This macro will call a function which will either return $+\infty$ on a system that supports the IEEE P754 standard or +(MAXDOUBLE) on a system that does not support the IEEE P754 standard.

If the correct value overflows, ±`HUGE_VAL` will be returned (according to the sign of the value).

LEVEL

Level 1.

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  tan(x)
double  x;

double  asin(x)
double  x;

double  acos(x)
double  x;

double  atan(x)
double  x;

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

DESCRIPTION

The functions `sin`, `cos` and `tan` return respectively the sine, cosine and tangent of their argument, `x`, measured in radians.

The function `asin` returns the arcsine of the argument `x` in the range $-\pi/2$ to $\pi/2$.

The function `acos` returns the arccosine of the argument `x` in the range 0 to π.

The function `atan` returns the arctangent of the argument `x` in the range $-\pi/2$ to $\pi/2$.

The function `atan2` returns the arctangent of `y/x` in the range $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUE

Both `sin` and `cos` lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return zero when there would otherwise be a complete loss of significance. In this case a message indicating `TLOSS` error is printed on the standard error output [see MATHERR(BA_LIB)]. For less extreme arguments causing partial loss of significance, a `PLOSS` error is generated but no message is printed. In both cases, `errno` is set to `ERANGE`.

If the magnitude of the argument of `asin` or `acos` is greater than one, or if both arguments of `atan2` are zero, zero is returned and `errno` is set to `EDOM`. In addition, a message indicating `DOMAIN` error is printed on the standard error output.

APPLICATION USAGE

These error-handling procedures may be changed with the MATHERR(BA_LIB) routine.

SEE ALSO

MATHERR(BA_LIB).

LEVEL

Level 1.