
‘libm’ includes software developed at SunPro, a Sun Microsystems, Inc. business. Permission to use, copy, modify, and distribute this software is freely granted, provided that this notice is preserved.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, subject to the terms of the GNU General Public License, which includes the provision that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.
1 Mathematical Functions (‘math.h’)

This chapter groups a wide variety of mathematical functions. The corresponding definitions and declarations are in ‘math.h’. Two definitions from ‘math.h’ are of particular interest.

1. The representation of infinity as a double is defined as HUGE_VAL; this number is returned on overflow by many functions.

2. The structure exception is used when you write customized error handlers for the mathematical functions. You can customize error handling for most of these functions by defining your own version of matherr; see the section on matherr for details.

Since the error handling code calls fputs, the mathematical subroutines require stubs or minimal implementations for the same list of OS subroutines as fputs: close, fstat, isatty, lseek, read, sbrk, write. See Section “System Calls” in The Red Hat newlib C Library, for a discussion and for sample minimal implementations of these support subroutines.

Alternative declarations of the mathematical functions, which exploit specific machine capabilities to operate faster—but generally have less error checking and may reflect additional limitations on some machines—are available when you include ‘fastmath.h’ instead of ‘math.h’.
1.1 Version of library

There are four different versions of the math library routines: IEEE, POSIX, X/Open, or SVID. The version may be selected at runtime by setting the global variable `_LIB_VERSION`, defined in `math.h`. It may be set to one of the following constants defined in `math.h`: `_IEEE_`, `_POSIX_`, `_XOPEN_`, or `_SVID_`. The `_LIB_VERSION` variable is not specific to any thread, and changing it will affect all threads.

The versions of the library differ only in how errors are handled.

In IEEE mode, the `matherr` function is never called, no warning messages are printed, and `errno` is never set.

In POSIX mode, `errno` is set correctly, but the `matherr` function is never called and no warning messages are printed.

In X/Open mode, `errno` is set correctly, and `matherr` is called, but warning message are not printed.

In SVID mode, functions which overflow return \[3.40282346638528860 \times 10^{38}\], the maximum single-precision floating-point value, rather than infinity. Also, `errno` is set correctly, `matherr` is called, and, if `matherr` returns 0, warning messages are printed for some errors. For example, by default `\log(-1.0)` writes this message on standard error output:

\[\log: \text{DOMAIN error}\]

The library is set to X/Open mode by default.
1.2 acos, acosf—arc cosine

Synopsis

#include <math.h>

double acos(double x);
float acosf(float x);

Description

acos computes the inverse cosine (arc cosine) of the input value. Arguments to acos must be in the range $-1$ to $1$.
acosf is identical to acos, except that it performs its calculations on floats.

Returns

acos and acosf return values in radians, in the range of $0$ to $\pi$.
If $x$ is not between $-1$ and $1$, the returned value is NaN (not a number) the global variable errno is set to EDOM, and a DOMAIN error message is sent as standard error output.
You can modify error handling for these functions using matherr.
1.3 acosh, acoshf—inverse hyperbolic cosine

Synopsis

```c
#include <math.h>

double acosh(double x);
float acoshf(float x);
```

Description

`acosh` calculates the inverse hyperbolic cosine of `x`. `acosh` is defined as

\[
\ln\left( x + \sqrt{x^2 - 1} \right)
\]

`x` must be a number greater than or equal to 1.

`acoshf` is identical, other than taking and returning floats.

Returns

`acosh` and `acoshf` return the calculated value. If `x` less than 1, the return value is NaN and `errno` is set to EDOM.

You can change the error-handling behavior with the non-ANSI `matherr` function.

Portability

Neither `acosh` nor `acoshf` are ANSI C. They are not recommended for portable programs.
1.4 asin, asinf—arc sine

Synopsis

#include <math.h>

double asin(double x);
float asinf(float x);

Description

asin computes the inverse sine (arc sine) of the argument x. Arguments to asin must be in the range $-1$ to $1$.

asinf is identical to asin, other than taking and returning floats.
You can modify error handling for these routines using matherr.

Returns

asin returns values in radians, in the range of $-\pi/2$ to $\pi/2$.
If x is not in the range $-1$ to $1$, asin and asinf return NaN (not a number), set the global variable errno to EDOM, and issue a DOMAIN error message.
You can change this error treatment using matherr.
1.5 asinh, asinhf—inverse hyperbolic sine

Synopsis

```c
#include <math.h>
double asinh(double x);
float asinhf(float x);
```

Description

asinh calculates the inverse hyperbolic sine of x. asinh is defined as

\[
sign(x) \times \ln(|x| + \sqrt{1 + x^2})
\]

asinhf is identical, other than taking and returning floats.

Returns

asinh and asinhf return the calculated value.

Portability

Neither asinh nor asinhf are ANSI C.
1.6 atan, atanf—arc tangent

Synopsis

```c
#include <math.h>

double atan(double x);
float atanf(float x);
```

Description

`atan` computes the inverse tangent (arc tangent) of the input value.

`atanf` is identical to `atan`, save that it operates on `floats`.

Returns

`atan` returns a value in radians, in the range of $-\pi/2$ to $\pi/2$.

Portability

`atan` is ANSI C. `atanf` is an extension.
1.7 atan2, atan2f—arc tangent of y/x

Synopsis

#include <math.h>

double atan2(double y, double x);

float atan2f(float y, float x);

Description

atan2 computes the inverse tangent (arc tangent) of y/x. atan2 produces the correct result even for angles near π/2 or −π/2 (that is, when x is near 0).

atan2f is identical to atan2, save that it takes and returns float.

Returns

atan2 and atan2f return a value in radians, in the range of −π to π.

You can modify error handling for these functions using matherr.

Portability

atan2 is ANSI C. atan2f is an extension.
1.8 atanh, atanhf—inverse hyperbolic tangent

Synopsis

```c
#include <math.h>

double atanh(double x);
float atanhf(float x);
```

Description

`atanh` calculates the inverse hyperbolic tangent of `x`.
`atanhf` is identical, other than taking and returning `float` values.

Returns

`atanh` and `atanhf` return the calculated value.

If |`x`| is greater than 1, the global `errno` is set to `EDOM` and the result is a NaN. A `DOMAIN` error is reported.

If |`x`| is 1, the global `errno` is set to `EDOM`; and the result is infinity with the same sign as `x`. A `SING` error is reported.

You can modify the error handling for these routines using `matherr`.

Portability

Neither `atanh` nor `atanhf` are ANSI C.
1.9 jN, jNf, yN, yNf—Bessel functions

Synopsis
#include <math.h>
double j0(double x);
float j0f(float x);
double j1(double x);
float j1f(float x);
double jn(int n, double x);
float jnf(int n, float x);
double y0(double x);
float y0f(float x);
double y1(double x);
float y1f(float x);
double yn(int n, double x);
float ynf(int n, float x);

Description
The Bessel functions are a family of functions that solve the differential equation

\[ x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - p^2)y = 0 \]

These functions have many applications in engineering and physics.

jn calculates the Bessel function of the first kind of order n. j0 and j1 are special cases for order 0 and order 1 respectively.
Similarly, yn calculates the Bessel function of the second kind of order n, and y0 and y1 are special cases for order 0 and 1.

jnf, j0f, j1f, ynf, y0f, and y1f perform the same calculations, but on float rather than double values.

Returns
The value of each Bessel function at x is returned.

Portability
None of the Bessel functions are in ANSI C.
1.10 cbrt, cbrtf—cube root

Synopsis

```c
#include <math.h>
double cbrt(double x);
float cbrtf(float x);
```

Description

cbrt computes the cube root of the argument.

Returns

The cube root is returned.

Portability

cbrt is in System V release 4. cbrtf is an extension.
1.11 copysign, copysignf—sign of y, magnitude of x

Synopsis

```c
#include <math.h>
double copysign (double x, double y);
float copysignf (float x, float y);
```

Description

copysign constructs a number with the magnitude (absolute value) of its first argument, x, and the sign of its second argument, y.
copysignf does the same thing; the two functions differ only in the type of their arguments and result.

Returns

copysign returns a double with the magnitude of x and the sign of y. copysignf returns a float with the magnitude of x and the sign of y.

Portability

copysign is not required by either ANSI C or the System V Interface Definition (Issue 2).
1.12 cosh, coshf—hyperbolic cosine

Synopsis

```
#include <math.h>

double cosh(double x);
float coshf(float x)
```

Description

cosh computes the hyperbolic cosine of the argument x. cosh(x) is defined as

\[
\cosh(x) = \frac{e^x + e^{-x}}{2}
\]

Angles are specified in radians. coshf is identical, save that it takes and returns float.

Returns

The computed value is returned. When the correct value would create an overflow, cosh returns the value HUGE_VAL with the appropriate sign, and the global value errno is set to ERANGE.

You can modify error handling for these functions using the function matherr.

Portability

cosh is ANSI. coshf is an extension.
1.13 erf, erff, erfc, erfcf—error function

Synopsis

```c
#include <math.h>

double erf(double x);
float erff(float x);

double erfc(double x);
float erfcf(float x);
```

Description

**erf** calculates an approximation to the “error function”, which estimates the probability that an observation will fall within \(x\) standard deviations of the mean (assuming a normal distribution). The error function is defined as

\[
\frac{2}{\sqrt{\pi}} \times \int_0^x e^{-t^2} dt
\]

**erfc** calculates the complementary probability; that is, \(erfc(x) = 1 - erf(x)\). **erfc** is computed directly, so that you can use it to avoid the loss of precision that would result from subtracting large probabilities (on large \(x\)) from 1.

**erff** and **erfcf** differ from **erf** and **erfc** only in the argument and result types.

Returns

For positive arguments, **erf** and all its variants return a probability—a number between 0 and 1.

Portability

None of the variants of **erf** are ANSI C.
1.14 exp, expf—exponential

Synopsis

```c
#include <math.h>
double exp(double x);
float expf(float x);
```

Description

exp and expf calculate the exponential of \( x \), that is, \( e^x \) (where \( e \) is the base of the natural system of logarithms, approximately 2.71828).

You can use the (non-ANSI) function `matherr` to specify error handling for these functions.

Returns

On success, exp and expf return the calculated value. If the result underflows, the returned value is 0. If the result overflows, the returned value is `HUGE_VAL`. In either case, `errno` is set to `ERANGE`.

Portability

exp is ANSI C. expf is an extension.
1.15 expm1, expm1f—exponential minus 1

Synopsis
#include <math.h>
double expm1(double x);
float expm1f(float x);

Description
expm1 and expm1f calculate the exponential of x and subtract 1, that is, $e^x - 1$ (where $e$ is the base of the natural system of logarithms, approximately 2.71828). The result is accurate even for small values of x, where using $\exp(x) - 1$ would lose many significant digits.

Returns
e raised to the power x, minus 1.

Portability
Neither expm1 nor expm1f is required by ANSI C or by the System V Interface Definition (Issue 2).
1.16 fabs, fabsf—absolute value (magnitude)

Synopsis

```c
#include <math.h>
double fabs(double x);
float fabsf(float x);
```

Description

fabs and fabsf calculate \(|x|\), the absolute value (magnitude) of the argument \(x\), by direct manipulation of the bit representation of \(x\).

Returns

The calculated value is returned. No errors are detected.

Portability

fabs is ANSI. fabsf is an extension.
1.17 floor, floorf, ceil, ceilf—floor and ceiling

Synopsis

```c
#include <math.h>

double floor(double x);
float floorf(float x);

double ceil(double x);
float ceilf(float x);
```

Description

c`floor` and `floorf` find ⌊x⌋, the nearest integer less than or equal to x. `ceil` and `ceilf` find ⌈x⌉, the nearest integer greater than or equal to x.

Returns

c`floor` and `ceil` return the integer result as a double. `floorf` and `ceilf` return the integer result as a float.

Portability

c`floor` and `ceil` are ANSI. `floorf` and `ceilf` are extensions.
1.18 fmod, fmodf—floating-point remainder (modulo)

Synopsis

```
#include <math.h>

double fmod(double x, double y)
float fmodf(float x, float y)
```

Description

The `fmod` and `fmodf` functions compute the floating-point remainder of \( \frac{x}{y} \) (\( x \) modulo \( y \)).

Returns

The `fmod` function returns the value \( x - i \times y \), for the largest integer \( i \) such that, if \( y \) is nonzero, the result has the same sign as \( x \) and magnitude less than the magnitude of \( y \).

`fmod(x,0)` returns NaN, and sets `errno` to `EDOM`.

You can modify error treatment for these functions using `matherr`.

Portability

`fmod` is ANSI C. `fmodf` is an extension.
1.19 frexp, frexpf—split floating-point number

Synopsis

```c
#include <math.h>
double frexp(double val, int *exp);
float frexpf(float val, int *exp);
```

Description

All nonzero, normal numbers can be described as $m \times 2^p$. `frexp` represents the double `val` as a mantissa $m$ and a power of two $p$. The resulting mantissa will always be greater than or equal to 0.5, and less than 1.0 (as long as `val` is nonzero). The power of two will be stored in `*exp`. $m$ and $p$ are calculated so that $val = m \times 2^p$.

`frexpf` is identical, other than taking and returning floats rather than doubles.

Returns

`frexp` returns the mantissa $m$. If `val` is 0, infinity, or Nan, `frexp` will set `*exp` to 0 and return `val`.

Portability

`frexp` is ANSI. `frexpf` is an extension.
1.20 gamma, gammaf, lgamma, lgammaf, gamma_r,

Synopsis

```
#include <math.h>

double gamma(double x);
float gammaf(float x);
double lgamma(double x);
float lgammaf(float x);
double gamma_r(double x, int *signgamp);
float gammaf_r(float x, int *signgamp);
double lgamma_r(double x, int *signgamp);
float lgammaf_r(float x, int *signgamp);
```

Description

gamma calculates \( \ln(\Gamma(x)) \), the natural logarithm of the gamma function of \( x \). The gamma function (\( \exp(\gamma(x)) \)) is a generalization of factorial, and retains the property that \( \Gamma(N) \equiv N \times \Gamma(N - 1) \). Accordingly, the results of the gamma function itself grow very quickly. gamma is defined as \( \ln(\Gamma(x)) \) rather than simply \( \Gamma(x) \) to extend the useful range of results representable.

The sign of the result is returned in the global variable signgam, which is declared in math.h. gammaf performs the same calculation as gamma, but uses and returns float values.

lgamma and lgammaf are alternate names for gamma and gammaf. The use of lgamma instead of gamma is a reminder that these functions compute the log of the gamma function, rather than the gamma function itself.

The functions gamma_r, gammaf_r, lgamma_r, and lgammaf_r are just like gamma, gammaf, lgamma, and lgammaf, respectively, but take an additional argument. This additional argument is a pointer to an integer. This additional argument is used to return the sign of the result, and the global variable signgam is not used. These functions may be used for reentrant calls (but they will still set the global variable errno if an error occurs).

Returns

Normally, the computed result is returned.

When \( x \) is a nonpositive integer, gamma returns HUGE_VAL and errno is set to EDOM. If the result overflows, gamma returns HUGE_VAL and errno is set to ERANGE.

You can modify this error treatment using matherr.

Portability

Neither gamma nor gammaf is ANSI C.
1.21 hypot, hypotf—distance from origin

Synopsis

#include <math.h>

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

Description

hypot calculates the Euclidean distance $\sqrt{x^2 + y^2}$ between the origin (0,0) and a point represented by the Cartesian coordinates (x,y). hypotf differs only in the type of its arguments and result.

Returns

Normally, the distance value is returned. On overflow, hypot returns HUGE_VAL and sets errno to ERANGE.

You can change the error treatment with matherr.

Portability

hypot and hypotf are not ANSI C.
1.22 ilogb, ilogbf—get exponent of floating-point number

Synopsis

```
#include <math.h>
int ilogb(double val);
int ilogbf(float val);
```

Description

All nonzero, normal numbers can be described as $m \times 2^p$. `ilogb` and `ilogbf` examine the argument `val`, and return $p$. The functions `frexp` and `frexpf` are similar to `ilogb` and `ilogbf`, but also return $m$.

Returns

`ilogb` and `ilogbf` return the power of two used to form the floating-point argument. If `val` is 0, they return $-\text{INT\_MAX}$ (\text{INT\_MAX} is defined in \text{limits.h}). If `val` is infinite, or NaN, they return \text{INT\_MAX}.

Portability

Neither `ilogb` nor `ilogbf` is required by ANSI C or by the System V Interface Definition (Issue 2).
1.23 infinity, infinityf—representation of infinity

Synopsis

#include <math.h>
double infinity(void);
float infinityf(void);

Description

infinity and infinityf return the special number IEEE infinity in double- and single-precision arithmetic respectively.
1.24 isnan, isnanf, isinf, isinff, finite, finitef—test for exceptional numbers

Synopsis
#include <ieeefp.h>
int isnan(double arg);
int isinf(double arg);
int finite(double arg);
int isnanf(float arg);
int isinff(float arg);
int finitef(float arg);

Description
These functions provide information on the floating-point argument supplied.
There are five major number formats:

zero A number which contains all zero bits.

subnormal A number with a zero exponent but a nonzero fraction.

normal A number with an exponent and a fraction.

infinity A number with an all 1’s exponent and a zero fraction.

NAN A number with an all 1’s exponent and a nonzero fraction.

isnan returns 1 if the argument is a nan. isinf returns 1 if the argument is infinity. finite returns 1 if the argument is zero, subnormal or normal. The isnanf, isinff and finitef functions perform the same operations as their isnan, isinf and finite counterparts, but on single-precision floating-point numbers.

It should be noted that the C99 standard dictates that isnan and isinf are macros that operate on multiple types of floating-point. The SUSv2 standard declares isnan as a function taking double. Newlib has decided to declare them both as macros in math.h and as functions in ieeefp.h.
1.25 ldexp, ldexpf—load exponent

Synopsis

```c
#include <math.h>

double ldexp(double val, int exp);
float ldexpf(float val, int exp);
```

Description

`ldexp` calculates the value `val \times 2^{exp}`. `ldexpf` is identical, save that it takes and returns `float` rather than `double` values.

Returns

`ldexp` returns the calculated value.

Underflow and overflow both set `errno` to `ERANGE`. On underflow, `ldexp` and `ldexpf` return 0.0. On overflow, `ldexp` returns plus or minus `HUGE_VAL`.

Portability

`ldexp` is ANSI. `ldexpf` is an extension.
1.26  log, logf—natural logarithms

Synopsis

#include <math.h>

double log(double x);
float logf(float x);

Description

Return the natural logarithm of \( x \), that is, its logarithm base \( e \) (where \( e \) is the base of the natural system of logarithms, \( 2.71828... \)). \texttt{log} and \texttt{logf} are identical save for the return and argument types.

You can use the (non-ANSI) function \texttt{matherr} to specify error handling for these functions.

Returns

Normally, returns the calculated value. When \( x \) is zero, the returned value is \(-HUGE VAL\) and \texttt{errno} is set to \texttt{ERANGE}. When \( x \) is negative, the returned value is NaN (not a number) and \texttt{errno} is set to \texttt{EDOM}. You can control the error behavior via \texttt{matherr}.

Portability

\texttt{log} is ANSI. \texttt{logf} is an extension.
1.27 log10, log10f—base 10 logarithms

Synopsis

```c
#include <math.h>
double log10(double x);
float log10f(float x);
```

Description

log10 returns the base 10 logarithm of x. It is implemented as \( \log(x) / \log(10) \).
log10f is identical, save that it takes and returns float values.

Returns

log10 and log10f return the calculated value.
See the description of log for information on errors.

Portability

log10 is ANSI C. log10f is an extension.
1.28 log1p, log1pf—log of 1 + x

Synopsis

#include <math.h>

double log1p(double x);
float log1pf(float x);

Description

log1p calculates $\ln(1 + x)$, the natural logarithm of $1+x$. You can use log1p rather than ‘log(1+x)’ for greater precision when $x$ is very small.

log1pf calculates the same thing, but accepts and returns float values rather than double.

Returns

log1p returns a double, the natural log of $1+x$. log1pf returns a float, the natural log of $1+x$.

Portability

Neither log1p nor log1pf is required by ANSI C or by the System V Interface Definition (Issue 2).
1.29 matherr—modifiable math error handler

Synopsis

```c
#include <math.h>
int matherr(struct exception *e);
```

Description

matherr is called whenever a math library function generates an error. You can replace matherr by your own subroutine to customize error treatment. The customized matherr must return 0 if it fails to resolve the error, and non-zero if the error is resolved.

When matherr returns a nonzero value, no error message is printed and the value of errno is not modified. You can accomplish either or both of these things in your own matherr using the information passed in the structure *e.

This is the exception structure (defined in ‘math.h’):

```c
struct exception {
    int type;
    char *name;
    double arg1, arg2, retval;
    int err;
};
```

The members of the exception structure have the following meanings:

- **type**: The type of mathematical error that occurred; macros encoding error types are also defined in ‘math.h’.
- **name**: A pointer to a null-terminated string holding the name of the math library function where the error occurred.
- **arg1, arg2**: The arguments which caused the error.
- **retval**: The error return value (what the calling function will return).
- **err**: If set to be non-zero, this is the new value assigned to errno.

The error types defined in ‘math.h’ represent possible mathematical errors as follows:

- **DOMAIN**: An argument was not in the domain of the function; e.g. \( \log(-1.0) \).
- **SING**: The requested calculation would result in a singularity; e.g. \( \text{pow}(0.0, -2.0) \).
- **OVERFLOW**: A calculation would produce a result too large to represent; e.g. \( \exp(1000.0) \).
- **UNDERFLOW**: A calculation would produce a result too small to represent; e.g. \( \exp(-1000.0) \).
- **TLOSS**: Total loss of precision. The result would have no significant digits; e.g. \( \sin(10\text{e}70) \).
- **PLOSS**: Partial loss of precision.

Returns

The library definition for matherr returns 0 in all cases.
You can change the calling function’s result from a customized `matherr` by modifying `e->retval`, which propagates back to the caller.

If `matherr` returns 0 (indicating that it was not able to resolve the error) the caller sets `errno` to an appropriate value, and prints an error message.

**Portability**

`matherr` is not ANSI C.
1.30 modf, modff—split fractional and integer parts

Synopsis

```c
#include <math.h>

double modf(double val, double *ipart);
fldt modff(float val, float *ipart);
```

Description

modf splits the double `val` apart into an integer part and a fractional part, returning the fractional part and storing the integer part in `*ipart`. No rounding whatsoever is done; the sum of the integer and fractional parts is guaranteed to be exactly equal to `val`. That is, if `realpart = modf(val, &ipart);` then `{realpart+intpart}` is the same as `val`. modff is identical, save that it takes and returns `float` rather than `double` values.

Returns

The fractional part is returned. Each result has the same sign as the supplied argument `val`.

Portability

modf is ANSI C. modff is an extension.
1.31 nan, nanf—representation of “Not a Number”

Synopsis

```c
#include <math.h>

double nan(const char *);
float nanf(const char *);
```

Description

`nan` and `nanf` return an IEEE NaN (Not a Number) in double- and single-precision arithmetic respectively. The argument is currently disregarded.
1.32 nextafter, nextafterf—get next number

Synopsis

#include <math.h>
double nextafter(double val, double dir);
float nextafterf(float val, float dir);

Description

nextafter returns the double-precision floating-point number closest to val in the direction toward dir. nextafterf performs the same operation in single precision. For example, nextafter(0.0,1.0) returns the smallest positive number which is representable in double precision.

Returns

Returns the next closest number to val in the direction toward dir.

Portability

Neither nextafter nor nextafterf is required by ANSI C or by the System V Interface Definition (Issue 2).
1.33 pow, powf—x to the power y

Synopsis

```c
#include <math.h>

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

Description

`pow` and `powf` calculate `$x^y$`. (That is, $x^y$.)

Returns

On success, `pow` and `powf` return the value calculated.

When the argument values would produce overflow, `pow` returns `HUGE_VAL` and set `errno` to `ERANGE`. If the argument `x` passed to `pow` or `powf` is a negative noninteger, and `y` is also not an integer, then `errno` is set to `EDOM`. If `x` and `y` are both 0, then `pow` and `powf` return 1.

You can modify error handling for these functions using `matherr`.

Portability

`pow` is ANSI C. `powf` is an extension.
1.34 remainder, remainderf—round and remainder

Synopsis

```c
#include <math.h>
double remainder(double x, double y);
float remainderf(float x, float y);
```

Description

`remainder` and `remainderf` find the remainder of `x/y`; this value is in the range `-y/2 .. +y/2`.

Returns

`remainder` returns the integer result as a double.

Portability

`remainder` is a System V release 4. `remainderf` is an extension.
1.35 scalbn, scalbnf—scale by power of two

Synopsis

```c
#include <math.h>
double scalbn(double x, int y);
float scalbnf(float x, int y);
```

Description

`scalbn` and `scalbnf` scale `x` by `n`, returning `x` times 2 to the power `n`. The result is computed by manipulating the exponent, rather than by actually performing an exponentiation or multiplication.

Returns

`x` times 2 to the power `n`.

Portability

Neither `scalbn` nor `scalbnf` is required by ANSI C or by the System V Interface Definition (Issue 2).
1.36 sqrt, sqrtf—positive square root

Synopsis

```c
#include <math.h>
double sqrt(double x);
float  sqrtf(float  x);
```

Description

`sqrt` computes the positive square root of the argument. You can modify error handling for this function with `matherr`.

Returns

On success, the square root is returned. If `x` is real and positive, then the result is positive. If `x` is real and negative, the global value `errno` is set to `EDOM` (domain error).

Portability

`sqrt` is ANSI C. `sqrtf` is an extension.
1.37 **sin, sinf, cos, cosf—sine or cosine**

**Synopsis**

```c
#include <math.h>
double sin(double x);
float sinf(float x);
double cos(double x);
float cosf(float x);
```

**Description**

* sin and *cos compute (respectively) the sine and cosine of the argument *x*. Angles are specified in radians.

* sinf and *cosf are identical, save that they take and return *float values.*

**Returns**

The sine or cosine of *x* is returned.

**Portability**

* sin and *cos are ANSI C. *sinf and *cosf are extensions.
1.38 sinh, sinhf—hyperbolic sine

Synopsis

```
#include <math.h>
  double sinh(double x);
  float sinhf(float x);
```

Description

`sinh` computes the hyperbolic sine of the argument `x`. Angles are specified in radians. `sinh(x)` is defined as

\[ \frac{e^x - e^{-x}}{2} \]

`sinhf` is identical, save that it takes and returns `float` values.

Returns

The hyperbolic sine of `x` is returned.

When the correct result is too large to be representable (an overflow), `sinh` returns `HUGE_VAL` with the appropriate sign, and sets the global value `errno` to `ERANGE`.

You can modify error handling for these functions with `matherr`.

Portability

`sinh` is ANSI C. `sinhf` is an extension.
1.39 tan, tanf—tangent

Synopsis

```c
#include <math.h>

double tan(double x);
float tanf(float x);
```

Description
tan computes the tangent of the argument x. Angles are specified in radians.
tanf is identical, save that it takes and returns float values.

Returns
The tangent of x is returned.

Portability
tan is ANSI. tanf is an extension.
1.40 tanh, tanhf—hyperbolic tangent

Synopsis

```c
#include <math.h>

double tanh(double x);
float tanhf(float x);
```

Description

`tanh` computes the hyperbolic tangent of the argument `x`. Angles are specified in radians.
`tanh(x)` is defined as

\[
\frac{\sinh(x)}{\cosh(x)}
\]

`tanhf` is identical, save that it takes and returns `float` values.

Returns

The hyperbolic tangent of `x` is returned.

Portability

`tanh` is ANSI C. `tanhf` is an extension.
2 Reentrancy Properties of libm

When a libm function detects an exceptional case, \texttt{errno} may be set, the \texttt{matherr} function may be called, and a error message may be written to the standard error stream. This behavior may not be reentrant.

With reentrant C libraries like the Red Hat newlib C library, \texttt{errno} is a macro which expands to the per-thread error value. This makes it thread safe.

When the user provides his own \texttt{matherr} function it must be reentrant for the math library as a whole to be reentrant.

In normal debugged programs, there are usually no math subroutine errors—and therefore no assignments to \texttt{errno} and no \texttt{matherr} calls; in that situation, the math functions behave reentrantly.
Library Index

A
acos ................................. 3
cosf .................................. 3
acosh .................................. 4
coshf .................................. 4
asin .................................. 5
asin .................................. 5
asinh .................................. 6
asinhf .................................. 6
atan .................................. 7
atan .................................. 7
atan2 .................................. 8
atan2f .................................. 8
atan ................................. 7
atanh .................................. 9
atanhf .................................. 9

C
cbrt .................................. 11
cbrtf .................................. 11
ceil .................................. 18
ceilf .................................. 18
copysign ............................... 12
copysignf .............................. 12
cos .................................. 39
cosf .................................. 39

e
erf .................................. 14
erfc .................................. 14
erfcf .................................. 14
erff .................................. 14
exp .................................. 15
expf .................................. 15
expm1 .................................. 16
expmlf .................................. 16

F
fabs .................................. 17
fabsf .................................. 17
finite .................................. 25
finitef .................................. 25
floor .................................. 18
floorf .................................. 18
fmod .................................. 19
fmodf .................................. 19
frexp .................................. 20
frexpf .................................. 20

g
gamma .................................. 21
gamma_r ................................. 21
gammaf ................................. 21
gammaf_r ................................. 21

H
hypot .................................. 22
hypotf .................................. 22

I
ilogb .................................. 23
ilogbf .................................. 23
infinity .................................. 24
infinityf .................................. 24
isinf .................................. 25
isinff .................................. 25
isnan .................................. 25
isnanf .................................. 25

J
j0 .................................. 10
j0f .................................. 10
j1 .................................. 10
j1f .................................. 10
jn .................................. 10
jnf .................................. 10

L
ldexp .................................. 26
ldexpf .................................. 26
lgamma .................................. 21
lgamma_r ................................. 21
lgammaf ................................. 21
lgammaf_r ................................. 21
log .................................. 27
log10 .................................. 28
log10f .................................. 28
log1p .................................. 29
log1pf .................................. 29
logf .................................. 27

M
matherr .................................. 30
matherr and reentrancy ................. 43
mod .................................. 32
modf .................................. 32

N
nan .................................. 33
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nanf</td>
<td>NaN check functions</td>
<td>33</td>
</tr>
<tr>
<td>nextafter</td>
<td>Next value</td>
<td>34</td>
</tr>
<tr>
<td>nextafterf</td>
<td>Next value</td>
<td>34</td>
</tr>
<tr>
<td>O</td>
<td>OS stubs</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
The body of this manual is set in
\texttt{cmr10} at 10.95pt,
with headings in \texttt{cmb10 at 10.95pt}
and examples in \texttt{cmtt10 at 10.95pt}.
\texttt{cmti10 at 10.95pt} and
\texttt{cmsl10 at 10.95pt}
are used for emphasis.
Table of Contents

1 Mathematical Functions (‘math.h’) .......... 1
  1.1 Version of library ........................................ 2
  1.2 acos, acosl—arc cosine ................................ 3
  1.3 acosh, acoshf—inverse hyperbolic cosine ............. 4
  1.4 asin, asinf—arc sine ..................................... 5
  1.5 asinh, asinhf—inverse hyperbolic sine ................. 6
  1.6 atan, atanf—arc tangent ................................. 7
  1.7 atan2, atan2f—arc tangent of y/x ...................... 8
  1.8 atanh, atanhf—inverse hyperbolic tangent .......... 9
  1.9 jN, jNf, yN, yNf—Bessel functions .................. 10
  1.10 cbrt, cbrtf—cube root .................................. 11
  1.11 copysign, copysignf—sign of y, magnitude of x ...... 12
  1.12 cosh, coshf—hyperbolic cosine ....................... 13
  1.13 erf, erff, erfc, erfcf—error function ............... 14
  1.14 exp, expf—exponential .................................. 15
  1.15 expm1, expm1f—exponential minus 1 ................. 16
  1.16 fabs, fabsf—absolute value (magnitude) ............. 17
  1.17 floor, floorf, ceil, ceilf—floor and ceiling ....... 18
  1.18 fmod, fmodf—floating-point remainder (modulo) .... 19
  1.19 frexp, frexpf—split floating-point number .......... 20
  1.20 gamma, gammaf, lgamma, lgammaf, gamma_r .......... 21
  1.21 hypot, hypotf—distance from origin .................. 22
  1.22 ilogb, ilogbf—get exponent of floating-point number 23
  1.23 infinity, infinityf—representation of infinity .... 24
  1.24 isnan, isnanf, isinf, isninf, finite, finitef—test for 25
    exceptional numbers ...................................... 26
  1.25 ldexp, ldexpf—load exponent .......................... 26
  1.26 log, logf—natural logarithms ......................... 27
  1.27 log10, log10f—base 10 logarithms .................. 28
  1.28 log1p, log1pf—log of 1 + x ......................... 29
  1.29 matherr—modifiable math error handler .............. 30
  1.30 modf, modff—split fractional and integer parts .... 32
  1.31 nan, nanf—representation of “Not a Number” ....... 33
  1.32 nextafter, nextafterf—get next number ............ 34
  1.33 pow, powf—x to the power y .......................... 35
  1.34 remainder, remainderf—round and remainder .......... 36
  1.35 scalbn, scalbnf—scale by power of two ............ 37
  1.36 sqrt, sqrtf—positive square root .................... 38
  1.37 sin, sinf, cos, cosf—sine or cosine ................. 39
  1.38 sinh, sinhf—hyperbolic sine ........................... 40
  1.39 tan, tanf—tangent ...................................... 41
  1.40 tanh, tanhf—hyperbolic tangent ..................... 42
2  Reentrancy Properties of libm ............... 43

Library Index .................................................. 45