
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. The macro HUGE_VALF is a corresponding value for float.

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 Error Handling

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 \texttt{math.h}. It may be set to one of the following constants defined in \texttt{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 \texttt{matherr} function is never called, no warning messages are printed, and \texttt{errno} is never set.

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

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

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

\texttt{log: DOMAIN error}

The library is set to X/Open mode by default.

The aforementioned error reporting is the supported Newlib libm error handling method. However, the majority of the functions are written so as to produce the floating-point exceptions (e.g. "invalid", "divide-by-zero") as required by the C and POSIX standards, for floating-point implementations that support them. Newlib does not provide the floating-point exception access routines defined in the standards for \texttt{fenv.h}, though, which is why they are considered unsupported. It is mentioned in case you have separately-provided access routines so that you are aware that they can be caused.

1.2 Standards Compliance And Portability

Most of the individual function descriptions describe the standards to which each function complies. However, these descriptions are mostly out of date, having been written before C99 was released. One of these days we'll get around to updating the rest of them. (If you'd like to help, please let us know.)

“C99” refers to ISO/IEC 9899:1999, “Programming languages–C”. “POSIX” refers to IEEE Standard 1003.1. POSIX® is a registered trademark of The IEEE.
1.3 acos, acosf—arc cosine

Synopsis

```c
#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.
1.4 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 \texttt{errno} is set to \texttt{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.
1.5 \texttt{asin}, \texttt{asinf}—arc sine

Synopsis

\begin{codeblock}
#include <math.h>

double asin(double \texttt{x});
float asinf(float \texttt{x});
\end{codeblock}

Description

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

\texttt{asinf} is identical to \texttt{asin}, other than taking and returning floats.

You can modify error handling for these routines using \texttt{matherr}.

Returns

\texttt{asin} returns values in radians, in the range of \(-\pi/2\) to \(\pi/2\).

If \texttt{x} is not in the range \(-1\) to \(1\), \texttt{asin} and \texttt{asinf} return NaN (not a number), set the global variable \texttt{errno} to EDOM, and issue a \texttt{DOMAIN} error message.

You can change this error treatment using \texttt{matherr}.
1.6 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

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

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.
1.7 atan, atanf—arc tangent

Synopsis

```
#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.
1.8 atan2, atan2f—arc tangent of $y/x$

Synopsis

```c
#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 $\pi/2$ or $-\pi/2$ (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 $-\pi$ to $\pi$.

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

Portability

`atan2` is ANSI C. `atan2f` is an extension.
1.9 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.
1.10 jN, jNf, yN, yNf—Bessel functions

Synopsis

```c
#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.
1.11 cbbr, cbbrf—cube root

Synopsis

```
#include <math.h>
double cbbr(double x);
float cbbrf(float x);
```

Description

cbbr computes the cube root of the argument.

Returns

The cube root is returned.

Portability

cbbr is in System V release 4. cbbrf is an extension.
1.12 copysign, copysignf—sign of y, magnitude of x

Synopsis

#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).
1.13 cosh, coshf—hyperbolic cosine

Synopsis

```
#include <math.h>
double cosh(double x);
floaht coshf(float x);
```

Description

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

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

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.
1.14 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, $\text{erfc}(x) = 1 - \text{erf}(x)$. `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.
1.15 exp, expf—exponential

Synopsis

#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.
1.16 exp10, exp10f—exponential, base 10

Synopsis

#include <math.h>

double exp10(double x);

float exp10f(float x);

Description

exp10 and exp10f calculate $10^x$, that is, $10^x$

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

Returns

On success, exp10 and exp10f 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

exp10 and exp10f are GNU extensions.
### 1.17 exp2, exp2f—exponential, base 2

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

double exp2(double x);
float exp2f(float x);
```

**Description**

exp2 and exp2f calculate $2^x$, that is, $2^x$.

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

**Returns**

On success, `exp2` and `exp2f` 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**

ANSI C, POSIX.
1.18 expm1, expm1f—exponential minus 1

Synopsis

```c
#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).
1.19 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.
1.20 fdim, fdimf—positive difference

Synopsis

```c
#include <math.h>

double fdim(double x, double y);
float fdimf(float x, float y);
```

Description

The `fdim` functions determine the positive difference between their arguments, returning:

- \( x - y \) if \( x > y \), or
- \( +0 \) if \( x \leq y \), or
- \( \text{NAN} \) if either argument is \( \text{NAN} \).

A range error may occur.

Returns

The `fdim` functions return the positive difference value.

Portability

ANSI C, POSIX.
1.21 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

`floor` and `floorf` find \(\lfloor x \rfloor\), the nearest integer less than or equal to \(x\). `ceil` and `ceilf` find \(\lceil x \rceil\), the nearest integer greater than or equal to \(x\).

Returns

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

Portability

`floor` and `ceil` are ANSI. `floorf` and `ceilf` are extensions.
1.22 fma, fmaf—floating multiply add

Synopsis

#include <math.h>

double fma(double x, double y, double z);
float fmaf(float x, float y, float z);

Description

The fma functions compute \((x \times y) + z\), rounded as one ternary operation: they compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS. That is, they are supposed to do this: see below.

Returns

The fma functions return \((x \times y) + z\), rounded as one ternary operation.

Bugs

This implementation does not provide the function that it should, purely returning "\((x \times y) + z;\)" with no attempt at all to provide the simulated infinite precision intermediates which are required. DO NOT USE THEM.

If double has enough more precision than float, then fmaf should provide the expected numeric results, as it does use double for the calculation. But since this is not the case for all platforms, this manual cannot determine if it is so for your case.

Portability

ANSI C, POSIX.
1.23 fmax, fmaxf—maximum

Synopsis

```
#include <math.h>
double fmax(double x, double y);
float fmaxf(float x, float y);
```

Description
The fmax functions determine the maximum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then the fmax functions choose the numeric value.

Returns
The fmax functions return the maximum numeric value of their arguments.

Portability
ANSI C, POSIX.
1.24 fmin, fminf—minimum

Synopsis

#include <math.h>

double fmin(double x, double y);
float fminf(float x, float y);

Description

The fmin functions determine the minimum numeric value of their arguments. NaN arguments are treated as missing data: if one argument is a NaN and the other numeric, then the fmin functions choose the numeric value.

Returns

The fmin functions return the minimum numeric value of their arguments.

Portability

ANSI C, POSIX.
1.25 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 x/y (x modulo y).

Returns

The fmod function returns the value x − i × 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.
1.26 frexp, frexpf—split floating-point number

Synopsis
#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.
1.27 gamma, gammaf, lgamma, lgammaf, gamma_r, gammaf_r, lgamma_r, lgammaf_r, tgamma, and tgammaf—logarithmic and plain gamma functions

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);
double tgamma(double x);
float tgammaf(float x);
```

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).

tgamma and tgammaf are the "true gamma" functions, returning $\Gamma(x)$, the gamma function of $x$–without a logarithm. (They are apparently so named because of the prior existence of the old, poorly-named gamma functions which returned the log of gamma up through BSD 4.2.)

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. It is better not to use either of these; use lgamma or
tgamma instead. lgamma, lgammaf, tgamma, and tgammaf are nominally C standard in terms of the base return values, although the matherr error-handling is not standard, nor is the signgam global for lgamma.
1.28 hypot, hypotf—distance from origin

Synopsis

```c
#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.
1.29 ilogb, ilogbf—get exponent of floating-point number

Synopsis

```c
#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 `FP_ILOGB0`. If `val` is infinite, they return `INT_MAX`. If `val` is NaN, they return `FP_ILOGBNAN`. (`FP_ILOGB0` and `FP_ILOGBNAN` are defined in `math.h`, but in turn are defined as `INT_MIN` or `INT_MAX` from `limits.h`. The value of `FP_ILOGB0` may be either `INT_MIN` or `-INT_MAX`. The value of `FP_ILOGBNAN` may be either `INT_MAX` or `INT_MIN`.)

Portability

C99, POSIX
1.30 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.

Portability

infinity and infinityf are neither standard C nor POSIX. C and POSIX require macros
HUGE_VAL and HUGE_VALF to be defined in math.h, which Newlib defines to be in-
finities corresponding to these archaic infinity() and infinityf() functions in floating-point
implementations which do have infinities.
1.31 isgreater, isgreaterequal, isless, islessequal, islessgreater, and isunordered—comparison macros

Synopsis

```
#include <math.h>
int isgreater(real-floating x, real-floating y);
int isgreaterequal(real-floating x, real-floating y);
int isless(real-floating x, real-floating y);
int islessequal(real-floating x, real-floating y);
int islessgreater(real-floating x, real-floating y);
int isunordered(real-floating x, real-floating y);
```

Description

isgreater, isgreaterequal, isless, islessequal, islessgreater, and isunordered are macros defined for use in comparing floating-point numbers without raising any floating-point exceptions.

The relational operators (i.e. <, >, <=, and >=) support the usual mathematical relationships between numeric values. For any ordered pair of numeric values exactly one of the relationships—less, greater, and equal—is true. Relational operators may raise the "invalid" floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true (i.e., if one or both of the arguments a NaN, the relationship is called unordered). The specified macros are quiet (non-floating-point exception raising) versions of the relational operators, and other comparison macros that facilitate writing efficient code that accounts for NaNs without suffering the "invalid" floating-point exception. In the synopses shown, "real-floating" indicates that the argument is an expression of real floating type.

Please note that saying that the macros do not raise floating-point exceptions, it is referring to the function that they are performing. It is certainly possible to give them an expression which causes an exception. For example:

```
NaN < 1.0  causes an "invalid" exception,
```

```
isless(NaN, 1.0)
does not, and
```

```
isless(NaN*0., 1.0)
causes an exception due to the "NaN*0.", but not from the resultant reduced
comparison of isless(NaN, 1.0).
```

Returns

No floating-point exceptions are raised for any of the macros.
The isgreater macro returns the value of (x) > (y).
The isgreaterequal macro returns the value of (x) >= (y).
The isless macro returns the value of (x) < (y).
The islessequal macro returns the value of (x) <= (y).
The islessgreater macro returns the value of (x) < (y) || (x) > (y).
The isunordered macro returns 1 if either of its arguments is NaN and 0 otherwise.
**Portability**
C99, POSIX.
1.32 `fpclassify`, `isfinite`, `isinf`, `isnan`, and `isnormal`—floating-point classification macros; `finite`, `finitef`, `isinf`, `isinff`, `isnan`, `isnanf`—test for exceptional numbers

Synopsis

[C99 standard macros:]

```c
#include <math.h>
int fpclassify(real-floating x);
int isfinite(real-floating x);
int isinf(real-floating x);
int isnan(real-floating x);
int isnormal(real-floating x);
```

[Archaic SUSv2 functions:]

```c
#include <math.h>
int isnan(double arg);
int isinf(double arg);
int finite(double arg);
int isnanf(float arg);
int isnanff(float arg);
int finitelf(float arg);
```

Description

`fpclassify`, `isfinite`, `isinf`, `isnan`, and `isnormal` are macros defined for use in classifying floating-point numbers. This is a help because of special "values" like NaN and infinities. In the synopses shown, "real-floating" indicates that the argument is an expression of real floating type. These function-like macros are C99 and POSIX-compliant, and should be used instead of the now-archaic SUSv2 functions.

The `fpclassify` macro classifies its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then classification is based on the type of the argument. The `fpclassify` macro returns the value of the number classification macro appropriate to the value of its argument:

- **FP_INFINITE**: $x$ is either plus or minus infinity;
- **FP_NAN**: $x$ is "Not A Number" (plus or minus);
- **FP_NORMAL**: $x$ is a "normal" number (i.e. is none of the other special forms);
- **FP_SUBNORMAL**: $x$ is too small be stored as a regular normalized number (i.e. loss of precision is likely); or
- **FP_ZERO**: $x$ is 0 (either plus or minus).

The "is" set of macros provide a useful set of shorthand ways for classifying floating-point numbers, providing the following equivalent relations:
isfinite(x) returns non-zero if x is finite. (It is equivalent to (fpclassify(x) != FP_INFINITE && fpclassify(x) != FP_NAN).)

isinf(x) returns non-zero if x is infinite. (It is equivalent to (fpclassify(x) == FP_INFINITE).)

isnan(x) returns non-zero if x is NaN. (It is equivalent to (fpclassify(x) == FP_NAN).)

isnormal(x) returns non-zero if x is normal. (It is equivalent to (fpclassify(x) == FP_NORMAL).)

The archaic SUSv2 functions provide information on the floating-point argument supplied. There are five major number formats ("exponent" referring to the biased exponent in the binary-encoded number):

zero A number which contains all zero bits, excluding the sign bit.

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 functions and as macros in math.h to maintain backward compatibility.

Returns
The fpclassify macro returns the value corresponding to the appropriate FP_. macro.
The isfinite macro returns nonzero if x is finite, else 0.
The isinf macro returns nonzero if x is infinite, else 0.
The isnan macro returns nonzero if x is an NaN, else 0.
The isnormal macro returns nonzero if x has a normal value, else 0.

Portability
math.h macros are C99, POSIX.1-2001.
The functions originate from BSD; isnan was listed in the X/Open Portability Guide and Single Unix Specification, but was dropped when the macro was standardized in POSIX.1-2001.
1.33 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 × 2^{\text{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.
1.34  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...). log and logf are identical save for the return and argument types.

You can use the (non-ANSI) function 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 errno is set to ERANGE. When x is negative, the returned value is NaN (not a number) and errno is set to EDOM. You can control the error behavior via matherr.

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

Synopsis

\begin{verbatim}
#include <math.h>
double log10(double x);
float log10f(float x);
\end{verbatim}

Description

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

Returns

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

Portability

\texttt{log10} is ANSI C. \texttt{log10f} is an extension.
1.36 log1p, log1pf—log of 1 + x

Synopsis

```c
#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).
1.37 \texttt{log2, log2f—base 2 logarithm}

\textbf{Synopsis}

\begin{verbatim}
#include <math.h>
double log2(double x);
float log2f(float x);
\end{verbatim}

\textbf{Description}

The \texttt{log2} functions compute the base-2 logarithm of \(x\). A domain error occurs if the argument is less than zero. A range error occurs if the argument is zero.

The Newlib implementations are not full, intrinsic calculations, but rather are derivatives based on \texttt{log}. (Accuracy might be slightly off from a direct calculation.) In addition to functions, they are also implemented as macros defined in math.h:

\begin{verbatim}
#define log2(x) (log (x) / _M_LN2)
#define log2f(x) (logf (x) / (float) _M_LN2)
\end{verbatim}

To use the functions instead, just undefine the macros first.

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

\textbf{Returns}

The \texttt{log2} functions return \(\log_2(x)\) on success. When \(x\) is zero, the returned value is \(-\texttt{HUGE}_\texttt{VAL}\) 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}.

\textbf{Portability}

C99, POSIX, System V Interface Definition (Issue 6).
1.38 logb, logbf—get exponent of floating-point number

Synopsis

```c
#include <math.h>

double logb(double x);
float logbf(float x);
```

Description

The `logb` functions extract the exponent of `x`, as a signed integer value in floating-point format. If `x` is subnormal it is treated as though it were normalized; thus, for positive finite `x`, \(1 \leq (x \cdot \text{FLT}_{\text{RADIX}}^{-\log_{\text{FLT}_{\text{RADIX}}}(x)}) < \text{FLT}_{\text{RADIX}}\). A domain error may occur if the argument is zero. In this floating-point implementation, FLT_RADIX is 2. Which also means that for finite `x`, \(\logb(x) = \text{floor}(\log_{\text{FLT}_{\text{RADIX}}}(\text{fabs}(x)))\).

All nonzero, normal numbers can be described as \(m \cdot 2^p\), where \(1.0 \leq m < 2.0\). The `logb` functions examine the argument `x`, and return `p`. The `frexp` functions are similar to the `logb` functions, but returning `m` adjusted to the interval \([.5, 1)\) or 0, and `p+1`.

Returns

When `x` is:
- `+inf` or `-inf`, `+inf` is returned;
- NaN, NaN is returned;
- 0, `-inf` is returned, and the divide-by-zero exception is raised;
- otherwise, the `logb` functions return the signed exponent of `x`.

Portability

ANSI C, POSIX

See Also

`frexp`, `ilogb`
1.39 \texttt{lrint, lrintf, llrint, llrintf}—round to integer

Synopsis

\begin{verbatim}
#include <math.h>
long int lrint(double x);
long int lrintf(float x);
long long int llrint(double x);
long long int llrintf(float x);
\end{verbatim}

Description

The \texttt{lrint} and \texttt{llrint} functions round their argument to the nearest integer value, using the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified. A range error may occur if the magnitude of $x$ is too large. The "inexact" floating-point exception is raised in implementations that support it when the result differs in value from the argument (i.e., when a fraction actually has been truncated).

Returns

$x$ rounded to an integral value, using the current rounding direction.

See Also

\texttt{lround}

Portability

ANSI C, POSIX
1.40 lround, lroundf, llround, llroundf—round to integer, to nearest

Synopsis

```c
#include <math.h>
long int lround(double x);
long int lroundf(float x);
long long int llround(double x);
long long int llroundf(float x);
```

Description

The `lround` and `llround` functions round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified (depending upon the floating-point implementation, not the library). A range error may occur if the magnitude of x is too large.

Returns

x rounded to an integral value as an integer.

See Also

See the `round` functions for the return being the same floating-point type as the argument. `lrint`, `llrint`.

Portability

ANSI C, POSIX
1.41 matherr—modifiable math error handler

Synopsis

    #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’):

    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 occured; 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. 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(10e70).
    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.
1.42 modf, modff—split fractional and integer parts

Synopsis

```c
#include <math.h>
double modf(double val, double *ipart);
float 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, &intpart)`; 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.
1.43 nan, nanf—representation of “Not a Number”

Synopsis

```c
#include <math.h>
double nan(const char *unused);
float nanf(const char *unused);
```

Description

`nan` and `nanf` return an IEEE NaN (Not a Number) in double- and single-precision arithmetic respectively. The argument is currently disregarded.
1.44 nearbyint, nearbyintf—round to integer

Synopsis

```c
#include <math.h>
double nearbyint(double x);
float nearbyintf(float x);
```

Description

The `nearbyint` functions round their argument to an integer value in floating-point format, using the current rounding direction and (supposedly) without raising the "inexact" floating-point exception. See the `rint` functions for the same function with the "inexact" floating-point exception being raised when appropriate.

Bugs

Newlib does not support the floating-point exception model, so that the floating-point exception control is not present and thereby what may be seen will be compiler and hardware dependent in this regard. The Newlib `nearbyint` functions are identical to the `rint` functions with respect to the floating-point exception behavior, and will cause the "inexact" exception to be raised for most targets.

Returns

`x` rounded to an integral value, using the current rounding direction.

Portability

ANSI C, POSIX

See Also

`rint`, `round`
1.45 nextafter, nextafterf—get next number

Synopsis

```c
#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).
1.46 pow, powf—x to the power y

Synopsis

```
#include <math.h>

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

Description

pow and powf calculate x raised to the exponent 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.
### 1.47 pow10, pow10f—base 10 power functions

**Synopsis**

```c
#include <math.h>

double pow10(double x);
float pow10f(float x);
```

**Description**

`pow10` and `pow10f` calculate $10^x$, that is, $10^x$.

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

**Returns**

On success, `pow10` and `pow10f` 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**

`pow10` and `pow10f` are GNU extensions.
1.48 remainder, remainderf—round and remainder

Synopsis

#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.
1.49 remquo, remquof—remainder and part of quotient

Synopsis

```c
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
```

Description

The `remquo` functions compute the same remainder as the `remainder` functions; this value is in the range \(-y/2 \ldots +y/2\). In the object pointed to by `quo` they store a value whose sign is the sign of \(x/y\) and whose magnitude is congruent modulo \(2^n\) to the magnitude of the integral quotient of \(x/y\). (That is, `quo` is given the \(n\) lsbs of the quotient, not counting the sign.) This implementation uses \(n=31\) if int is 32 bits or more, otherwise, \(n\) is 1 less than the width of int.

For example:

```c
remquo(-29.0, 3.0, &quo)
```
returns -1.0 and sets `quo`=10, and

```c
remquo(-98307.0, 3.0, &quo)
```
returns -0.0 and sets `quo`=-32769, although for 16-bit int, `quo`=-1. In the latter case, the actual quotient of -(32769=0x8001) is reduced to -1 because of the 15-bit limitation for the quotient.

Returns

When either argument is NaN, NaN is returned. If \(y\) is 0 or \(x\) is infinite (and neither is NaN), a domain error occurs (i.e. the "invalid" floating point exception is raised or errno is set to EDOM), and NaN is returned. Otherwise, the `remquo` functions return \(x \text{ REM } y\).

Bugs

IEEE754-2008 calls for `remquo(subnormal, inf)` to cause the "underflow" floating-point exception. This implementation does not.

Portability

C99, POSIX.
1.50 rint, rintf—round to integer

Synopsis

#include <math.h>

double rint(double x);
float rintf(float x);

Description
The rint functions round their argument to an integer value in floating-point format, using the current rounding direction. They raise the "inexact" floating-point exception if the result differs in value from the argument. See the nearbyint functions for the same function with the "inexact" floating-point exception never being raised. Newlib does not directly support floating-point exceptions. The rint functions are written so that the "inexact" exception is raised in hardware implementations that support it, even though Newlib does not provide access.

Returns
x rounded to an integral value, using the current rounding direction.

Portability
ANSI C, POSIX

See Also
nearbyint, round
1.51 round, roundf—round to integer, to nearest

Synopsis

```c
#include <math.h>

double round(double x);
float roundf(float x);
```

Description

The `round` functions round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction. (While the "inexact" floating-point exception behavior is unspecified by the C standard, the `round` functions are written so that "inexact" is not raised if the result does not equal the argument, which behavior is as recommended by IEEE 754 for its related functions.)

Returns

x rounded to an integral value.

Portability

ANSI C, POSIX

See Also

`nearbyint`, `rint`
1.52 scalbn, scalbnf, scalbln, scalblnf—scale by power of FLT_RADIX (=2)

Synopsis

```c
#include <math.h>
double scalbn(double x, int n);
float scalbnf(float x, int n);
double scalbln(double x, long int n);
float scalblnf(float x, long int n);
```

Description

The `scalbn` and `scalbln` functions compute \( x \cdot FLT\_RADIX^n \) efficiently. The result is computed by manipulating the exponent, rather than by actually performing an exponentiation or multiplication. In this floating-point implementation FLT_RADIX=2, which makes the `scalbn` functions equivalent to the `ldexp` functions.

Returns

\( x \) times 2 to the power \( n \). A range error may occur.

Portability

ANSI C, POSIX

See Also

`ldexp`
1.53 `signbit`—Does floating-point number have negative sign?

Synopsis

```c
#include <math.h>
int signbit(real-floating x);
```

Description

The `signbit` macro determines whether the sign of its argument value is negative. The macro reports the sign of all values, including infinities, zeros, and NaNs. If zero is unsigned, it is treated as positive. As shown in the synopsis, the argument is "real-floating," meaning that any of the real floating-point types (float, double, etc.) may be given to it.

Note that because of the possibilities of signed 0 and NaNs, the expression "x < 0.0" does not give the same result as `signbit` in all cases.

Returns

The `signbit` macro returns a nonzero value if and only if the sign of its argument value is negative.

Portability

C99, POSIX.
1.54 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.
1.55 sinh, sinhf—hyperbolic sine

Synopsis

```c
#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.
1.56 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.
### 1.57 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.
1.58 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.
1.59 trunc, truncf—round to integer, towards zero

Synopsis

```c
#include <math.h>

double trunc(double x);
float truncf(float x);
```

Description

The `trunc` functions round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument, regardless of the current rounding direction. (While the "inexact" floating-point exception behavior is unspecified by the C standard, the `trunc` functions are written so that "inexact" is not raised if the result does not equal the argument, which behavior is as recommended by IEEE 754 for its related functions.)

Returns

*x* truncated to an integral value.

Portability

ANSI C, POSIX
2 Mathematical Complex Functions (complex.h)

This chapter groups the complex mathematical functions. The corresponding definitions and declarations are in complex.h. Functions and documentations are taken from NetBSD.
2.1 cabs, cabsf, cabsl—complex absolute-value

Synopsis

#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);

Description

These functions compute the complex absolute value (also called norm, modulus, or magnitude) of \( z \).

cabsf is identical to cabs, except that it performs its calculations on float complex.
cabsl is identical to cabs, except that it performs its calculations on long double complex.

Returns

The cabs* functions return the complex absolute value.

Portability

cabs, cabsf and cabsl are ISO C99
2.2 cacos, cacosf—complex arc cosine

Synopsis

```c
#include <complex.h>
double complex cacos(double complex z);
float complex cacosf(float complex z);
```

Description

These functions compute the complex arc cosine of \( z \), with branch cuts outside the interval \([-1, +1]\) along the real axis.

`cacosf` is identical to `cacos`, except that it performs its calculations on `floats complex`.

Returns

These functions return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval \([0, \pi]\) along the real axis.

Portability

`cacos` and `cacosf` are ISO C99
2.3 cacosh, cacoshf—complex arc hyperbolic cosine

Synopsis

```
#include <complex.h>

double complex cacosh(double complex z);
float complex cacoshf(float complex z);
```

Description
These functions compute the complex arc hyperbolic cosine of \( z \), with a branch cut at values less than 1 along the real axis.

\texttt{cacoshf} is identical to \texttt{cacosh}, except that it performs its calculations on \texttt{floats complex}.

Returns
These functions return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval \([-i\pi, +i\pi]\) along the imaginary axis.

Portability
\texttt{cacosh} and \texttt{cacoshf} are ISO C99
2.4 carg, cargf—argument (phase angle)

Synopsis

```c
#include <complex.h>
double carg(double complex z);
float cargf(float complex z);
```

Description

These functions compute the argument (also called phase angle) of z, with a branch cut along the negative real axis.

cargf is identical to carg, except that it performs its calculations on floats complex.

Returns

The carg functions return the value of the argument in the interval \([-\pi, +\pi]\)

Portability

carg and cargf are ISO C99
2.5 casin, casinf—complex arc sine

Synopsis

#include <complex.h>

double complex casin(double complex z);
float complex casinf(float complex z);

Description

These functions compute the complex arc sine of \( z \), with branch cuts outside the interval \([-1, +1]\) along the real axis.

casinf is identical to casin, except that it performs its calculations on floats complex.

Returns

These functions return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval \([-\pi/2, +\pi/2]\) along the real axis.

Portability

casin and casinf are ISO C99
2.6 casinh, casinhf—complex arc hyperbolic sine

Synopsis

#include <complex.h>

double complex casinh(double complex z);
float complex casinhf(float complex z);

Description

These functions compute the complex arc hyperbolic sine of \( z \), with branch cuts outside the interval \([-i, +i]\) along the imaginary axis.

\texttt{casinhf} is identical to \texttt{casinh}, except that it performs its calculations on \texttt{floats complex}.

Returns

These functions return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval \([-i\pi/2, +i\pi/2]\) along the imaginary axis.

Portability

\texttt{casinh} and \texttt{casinhf} are ISO C99
2.7 catan, catanf—complex arc tangent

Synopsis

```c
#include <complex.h>

double complex catan(double complex z);
float complex catanf(float complex z);
```

Description

These functions compute the complex arc tangent of \( z \), with branch cuts outside the interval \([-i, +i]\) along the imaginary axis.

`catanf` is identical to `catan`, except that it performs its calculations on floats complex.

Returns

These functions return the complex arc tangent, in the range of a strip mathematically unbounded along the imaginary axis and in the interval \([-\pi/2, +\pi/2]\) along the real axis.

Portability

`catan` and `catanf` are ISO C99.
2.8 `catanh`, `catanhf`—complex arc hyperbolic tangent

Synopsis

```c
#include <complex.h>

double complex catanh(double complex z);
float complex catanhf(float complex z);
```

Description

These functions compute the complex arc hyperbolic tan of `z`, with branch cuts outside the interval \([-1, +1]\) along the real axis.

`catanhf` is identical to `catanh`, except that it performs its calculations on `floats complex`.

Returns

These functions return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval \([-i\pi/2, +i\pi/2]\) along the imaginary axis.

Portability

`catanh` and `catanhf` are ISO C99
2.9 ccos, ccosf—complex cosine

Synopsis

```
#include <complex.h>

double complex ccos(double complex z);
float complex ccosf(float complex z);
```

Description

These functions compute the complex cosine of z.

ccosf is identical to ccos, except that it performs its calculations on floats complex.

Returns

These functions return the complex cosine value.

Portability

ccos and ccosf are ISO C99
2.10 ccosh, ccoshf—complex hyperbolic cosine

Synopsis

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
```

Description

These functions compute the complex hyperbolic cosine of z.

ccoshf is identical to ccosh, except that it performs its calculations on floats complex.

Returns

These functions return the complex hyperbolic cosine value.

Portability

ccosh and ccoshf are ISO C99
2.11 cexp, cexpf—complex base-e exponential

Synopsis
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);

Description
These functions compute the complex base-e exponential of z.

`cexpf` is identical to `cexp`, except that it performs its calculations on floats complex.

Returns
The cexp functions return the complex base-e exponential value.

Portability
`cexp` and `cexpf` are ISO C99
2.12 cimag, cimagf, cimagl—imaginary part

Synopsis

```c
#include <complex.h>

double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

Description

These functions compute the imaginary part of z.

cimagf is identical to cimag, except that it performs its calculations on float complex.
cimagl is identical to cimag, except that it performs its calculations on long double complex.

Returns

The cimag* functions return the imaginary part value (as a real).

Portability

cimag, cimagf and cimagl are ISO C99
2.13  clog, clogf—complex base-e logarithm

Synopsis
#include <complex.h>
double complex clog(double complex z);
float complex clogf(float complex z);

Description
These functions compute the complex natural (base-e) logarithm of z, with a branch cut along the negative real axis.
clogf is identical to clog, except that it performs its calculations on floats complex.

Returns
The clog functions return the complex natural logarithm value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

Portability
clog and clogf are ISO C99
2.14 clog10, clog10f—complex base-10 logarithm

Synopsis

```c
#define _GNU_SOURCE
#include <complex.h>

double complex clog10(double complex z);
float complex clog10f(float complex z);
```

Description

These functions compute the complex base-10 logarithm of z. The `clog10` is equivalent to `clog(z)/log(10)`.
`clog10f` is identical to `clog10`, except that it performs its calculations on floats complex.

Returns

The clog10 functions return the complex base-10 logarithm value.

Portability

`clog10` and `clog10f` are GNU extensions.
2.15 conj, conjf—complex conjugate

Synopsis

```c
#include <complex.h>
double complex conj(double complex z);
float complex conjf(float complex z);
```

Description
These functions compute the complex conjugate of \( z \), by reversing the sign of its imaginary part.

`conjf` is identical to `conj`, except that it performs its calculations on floats complex.

Returns
The `conj` functions return the complex conjugate value.

Portability
`conj` and `conjf` are ISO C99
2.16 cpow, cpowf—complex power

Synopsis

```c
#include <complex.h>

double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
```

Description

The cpow functions compute the complex power function \( x^y \) power, with a branch cut for the first parameter along the negative real axis.

cpowf is identical to cpow, except that it performs its calculations on floats complex.

Returns

The cpow functions return the complex power function value.

Portability

cpow and cpowf are ISO C99
2.17 cproj, cprojf— Riemann sphere projection

Synopsis

```
#include <complex.h>

double complex cproj(double complex z);
float complex cprojf(float complex z);
```

Description
These functions compute a projection of z onto the Riemann sphere: z projects to z except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If z has an infinite part, then cproj(z) is equivalent to INFINITY + I * copysign(0.0, cimag(z))
cprojf is identical to cproj, except that it performs its calculations on floats complex.

Returns
The cproj functions return the value of the projection onto the Riemann sphere.

Portability
Cproj and cprojf are ISO C99
2.18 creal, crealf, creall—real part

Synopsis

```c
#include <complex.h>
double creal(double complex z);
float crealf(float complex z);
double long creall(long double complex z);
```

Description
These functions compute the real part of z.

crealf is identical to creal, except that it performs its calculations on float complex.

creall is identical to creal, except that it performs its calculations on long double complex.

Returns
The creal* functions return the real part value.

Portability
creal, crealf and creall are ISO C99
2.19  csin, csinf—complex sine

Synopsis

    #include <complex.h>
    double complex csin(double complex z);
    float complex csinf(float complex z);

Description
These functions compute the complex sine of z.

    csinf is identical to csin, except that it performs its calculations on floats complex.

Returns
These functions return the complex sine value.

Portability
    csin and csinf are ISO C99
2.20 csinh, csinhf—complex hyperbolic sine

Synopsis

#include <complex.h>

double complex csinh(double complex z);
float complex csinhf(float complex z);

Description

These functions compute the complex hyperbolic sine of z.

ccoshf is identical to ccosh, except that it performs its calculations on floats complex.

Returns

These functions return the complex hyperbolic sine value.

Portability

csinh and csinhf are ISO C99
2.21 csqrt, csqrtf—complex square root

Synopsis

```
#include <complex.h>

double complex csqrt(double complex z);
float complex csqrtf(float complex z);
```

Description

These functions compute the complex square root of z, with a branch cut along the negative real axis.

`csqrtf` is identical to `csqrt`, except that it performs its calculations on `floats complex`.

Returns

The `csqrt` functions return the complex square root value, in the range of the right halfplane (including the imaginary axis).

Portability

`csqrt` and `csqrtf` are ISO C99
2.22 \texttt{ctan}, \texttt{ctanf}—complex tangent

Synopsis

\begin{verbatim}
#include <complex.h>
double complex ctan(double complex z);
float complex ctanf(float complex z);
\end{verbatim}

Description
These functions compute the complex tangent of \( z \).
\texttt{ctanf} is identical to \texttt{ctan}, except that it performs its calculations on floats complex.

Returns
These functions return the complex tangent value.

Portability
\texttt{ctan} and \texttt{ctanf} are ISO C99
2.23  ctanh, ctanf—complex hyperbolic tangent

Synopsis
   
   #include <complex.h>
   double complex ctanh(double complex z);
   float complex ctanhf(float complex z);

Description
These functions compute the complex hyperbolic tangent of z.

ctanhf is identical to ctanh, except that it performs its calculations on floats complex.

Returns
These functions return the complex hyperbolic tangent value.

Portability
ctanh and ctanhf are ISO C99
3 Reentrancy Properties of libm

When a libm function detects an exceptional case, errno may be set, the 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, errno is a macro which expands to the per-thread error value. This makes it thread safe.

When the user provides his own 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 errno and no matherr calls; in that situation, the math functions behave reentrantly.
4 The long double function support of libm

Currently, the full set of long double math functions is only provided on platforms where long double equals double. For such platforms, the long double math functions are implemented as calls to the double versions.
Document Index

M
matherr and reentrancy......................... 89

O
OS stubs......................................... 1

R
reentrancy...................................... 89

S
stubs............................................. 1
support subroutines........................... 1
system calls................................... 1
The body of this manual is set in cmr10,
with headings in cmbx10
and examples in cmtt10.
cmti10 and
cmsl10
are used for emphasis.
### Table of Contents

1 Mathematical Functions (math.h) .................. 1

1.1 Error Handling ...................................... 2
1.2 Standards Compliance And Portability ................ 2
1.3 acos, acosf—arc cosine .................................. 3
1.4 acosh, acoshf—inverse hyperbolic cosine ............... 4
1.5 asin, asinf—arc sine .................................... 5
1.6 asinh, asinhf—inverse hyperbolic sine ................ 6
1.7 atan, atanf—arc tangent .................................. 7
1.8 atan2, atan2f—arc tangent of y/x ...................... 8
1.9 atanh, atanhf—inverse hyperbolic tangent ............. 9
1.10 jN, jNf, yN, yNf—Bessel functions .................... 10
1.11 cbrt, cbrtf—cube root ................................... 11
1.12 copysign, copysignf—sign of y, magnitude of x ....... 12
1.13 cosh, coshf—hyperbolic cosine .......................... 13
1.14 erf, erff, erfc, erfcf—error function .................. 14
1.15 exp, expf—exponential .................................... 15
1.16 exp10, exp10f—exponential, base 10 .................... 16
1.17 exp2, exp2f—exponential, base 2 ....................... 17
1.18 expm1, expm1f—exponential minus 1 .................... 18
1.19 fabs, fabsf—absolute value (magnitude) ............... 19
1.20 fdiv, fdivf—positive difference ....................... 20
1.21 floor, floorf, ceil, ceilf—floor and ceiling .......... 21
1.22 fma, fmaf—floating multiply add ....................... 22
1.23 fmax, fmaxf—maximum ................................... 23
1.24 fmin, fminf—minimum .................................... 24
1.25 fmod, fmodf—floating-point remainder (modulo) ....... 25
1.26 frexp, frexf—split floating-point number .............. 26
1.27 gamma, gammaf, lgamma, lgammaf, gamma_r, lgammaf_r, 
    lgamma_r, lgammaf_r, tgamma, and tgammaf—logarithmic and 
    plain gamma functions .................................. 27
1.28 hypot, hypotf—distance from origin ..................... 29
1.29 ilogb, ilogbf—get exponent of floating-point number ...... 30
1.30 infinity, infinityf—representation of infinity ........ 31
1.31 isgreater, isgreateorequal, isless, islessequal, islessgreater, 
    and isunordered—comparison macros .................... 32
1.32 fclassify, isfinite, isinf, isnan, and isnormal—floating-point 
    classification macros; finite, finitef, isinf, isnaff, isnan, 
    isnanf—test for exceptional numbers .................... 34
1.33 ldexp, ldexpf—load exponent ......................... 36
1.34 log, logf—natural logarithms .......................... 37
1.35 log10, log10f—base 10 logarithms .................... 38
1.36 log1p, log1pf—log of 1 + x ........................... 39
1.37 log2, log2f—base 2 logarithm ........................ 40
1.38 logb, logbf—get exponent of floating-point number .... 41
2 Mathematical Complex Functions (complex.h)
3 Reentrancy Properties of libm ............... 89
4 The long double function support of libm .... 91

Document Index ........................................... 93