The Cygnus C Math Library

`libm` 1.4
May 1993

Steve Chamberlain
Roland Pesch
Cygnus Support

Cygnus Support
sac@cygnus.com
pesch@cygnus.com

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 Cygnus C Support 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 3.40282346638528860e+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: DOMAIN error
```

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

1.2 acos, acosf—arc cosine

```
#include <math.h>
double acos(double x);
float acosf(float x);
```

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.

acos and acosf return values in radians, in the range of 0 to π.

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

```
#include <math.h>
double acosh(double x);
float acoshf(float x);
```

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.

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.

Neither acosh nor acoshf are ANSI C. They are not recommended for portable programs.

1.4 asin, asinf—arc sine

```
#include <math.h>
double asin(double x);
float asinf(float x);
```

`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`.

`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

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

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

$$sign(x) \times ln\Big(|x| + \sqrt{1+x^2}\Big)$$

asinhf is identical, other than taking and returning floats.

asinh and asinhf return the calculated value.

Neither asinh nor asinhf are ANSI C.

1.6 atan, atanf—arc tangent

```
#include <math.h>
double atan(double x);
float atanf(float x);
```

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

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

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

atan is ANSI C. atanf is an extension.

1.7 atan2, atan2f—arc tangent of y/x

```
#include <math.h>
double atan2(double y,double x);
float atan2f(float y,float x);
```

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.

atan2 and atan2f return a value in radians, in the range of $-\pi$ to π.

If both x and y are 0.0, atan2 causes a DOMAIN error.

You can modify error handling for these functions using matherr.

atan2 is ANSI C. atan2f is an extension.

1.8 atanh, atanhf—inverse hyperbolic tangent

```
#include <math.h>
double atanh(double x);
float atanhf(float x);
```

`atanh` calculates the inverse hyperbolic tangent of *x*.

`atanhf` is identical, other than taking and returning `float` values.

`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`.

Neither `atanh` nor `atanhf` are ANSI C.

1.9 jN,jNf,yN,yNf—Bessel functions

```
#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);
```

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

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

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.

The value of each Bessel function at x is returned.

None of the Bessel functions are in ANSI C.

1.10 cbrt, cbrtf—cube root

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

`cbrt` computes the cube root of the argument.

The cube root is returned.

`cbrt` is in System V release 4. `cbrtf` is an extension.

1.11 copysign, copysignf—sign of y, magnitude of x

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

`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.

`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.

`copysign` is not required by either ANSI C or the System V Interface Definition (Issue 2).

1.12 cosh, coshf—hyperbolic cosine

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

`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`.

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`.

`cosh` is ANSI. `coshf` is an extension.

1.13 erf, erff, erfc, erfcf—error function

```
#include <math.h>
double erf(double x);
float erff(float x);
double erfc(double x);
float erfcf(float x);
```

`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)` is `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.

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

None of the variants of `erf` are ANSI C.

1.14 exp, expf—exponential

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

`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.

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`.

`exp` is ANSI C. `expf` is an extension.

1.15 expm1, expm1f—exponential minus 1

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

`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.

e raised to the power x, minus 1.

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)

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

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

The calculated value is returned. No errors are detected.

fabs is ANSI. fabsf is an extension.

1.17 floor, floorf, ceil, ceilf—floor and ceiling

```
#include <math.h>
double floor(double x);
float floorf(float x);
double ceil(double x);
float ceilf(float x);
```

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.

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

floor and ceil are ANSI. floorf and ceilf are extensions.

1.18 fmod, fmodf—floating-point remainder (modulo)

```
#include <math.h>
double fmod(double x, double y)
float fmodf(float x, float y)
```

The `fmod` and `fmodf` functions compute the floating-point remainder of x/y (x modulo y).

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`.

`fmod` is ANSI C. `fmodf` is an extension.

1.19 frexp, frexpf—split floating-point number

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

All non zero, normal numbers can be described as *m* * 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.

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

`frexp` is ANSI. `frexpf` is an extension.

1.20 gamma, gammaf, lgamma, lgammaf, gamma_r,

```
#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);
```

`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).

Do not confuse the function `gamma_r`, which takes an additional argument which is a pointer to an integer, with the function `_gamma_r`, which takes an additional argument which is a pointer to a reentrancy structure.

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`.

Neither `gamma` nor `gammaf` is ANSI C.

1.21 hypot, hypotf—distance from origin

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

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.

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.

hypot and hypotf are not ANSI C.

1.22 `ilogb`, `ilogbf`—get exponent of floating point number

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

All non zero, normal numbers can be described as *m* * 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*.

`ilogb` and `ilogbf` return the power of two used to form the floating point argument. If *val* is `0`, they return `- INT_MAX` (`INT_MAX` is defined in limits.h). If *val* is infinite, or NaN, they return `INT_MAX`.

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

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

`infinity` and `infinityf` return the special number IEEE infinity in double and single precision arithmetic respectivly.

1.24 isnan,isnanf,isinf,isinff,finite,finitef—test for exceptional numbers

```
#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);
```

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`
Is used to represent number with a zero exponent, but a non zero 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 non zero 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` perform the same operations as their `isnan`, `isinf` and `finite` counterparts, but on single precision floating point numbers.

1.25 ldexp, ldexpf—load exponent

```
#include <math.h>
double ldexp(double val, int exp);
float ldexpf(float val, int exp);
```

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

`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`.

`ldexp` is ANSI, `ldexpf` is an extension.

1.26 log, logf—natural logarithms

```
#include <math.h>
double log(double x);
float logf(float x);
```

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.

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 `-HUGE_VAL` and `errno` is set to `EDOM`. You can control the error behavior via `matherr`.

`log` is ANSI, `logf` is an extension.

1.27 log10, log10f—base 10 logarithms

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

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.

log10 and log10f return the calculated value.

See the description of log for information on errors.

log10 is ANSI C. log10f is an extension.

1.28 log1p, log1pf—log of 1 + x

```
#include <math.h>
double log1p(double x);
float log1pf(float x);
```

`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`.

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

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

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

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.

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 backs 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.

`matherr` is not ANSI C.

1.30 modf, modff—split fractional and integer parts

```
#include <math.h>
double modf(double val, double *ipart);
float modff(float val, float *ipart);
```

`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.

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

`modf` is ANSI C. `modff` is an extension.

1.31 nan, nanf—representation of infinity

```
#include <math.h>
double nan(void);
float nanf(void);
```

nan and nanf return an IEEE NaN (Not a Number) in double and single precision arithmetic respectivly.

1.32 nextafter, nextafterf—get next number

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

`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 the next closest number to *val* in the direction toward *dir*.

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

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

pow and powf calculate x raised to the exp1.0nt y. (That is, x^y.)

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.

pow is ANSI C. powf is an extension.

1.34 rint, rintf, remainder, remainderf—round and remainder

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

rint and rintf returns their argument rounded to the nearest integer. remainder and remainderf find the remainder of x/y; this value is in the range -$y/2$ .. +$y/2$.

rint and remainder return the integer result as a double.

rint and remainder are System V release 4. rintf and remainderf are extensions.

1.35 scalbn, scalbnf—scale by integer

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

`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.

x times 2 to the power n.

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

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

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

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).

`sqrt` is ANSI C. `sqrtf` is an extension.

1.37 sin, sinf, cos, cosf—sine or cosine

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

`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.

The sine or cosine of *x* is returned.

`sin` and `cos` are ANSI C. `sinf` and `cosf` are extensions.

1.38 sinh, sinhf—hyperbolic sine

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

`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.

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`.

`sinh` is ANSI C. `sinhf` is an extension.

1.39 `tan`, `tanf`—tangent

```
#include <math.h>
double tan(double x);
float tanf(float x);
```

`tan` computes the tangent of the argument *x*. Angles are specified in radians.

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

The tangent of *x* is returned.

`tan` is ANSI. `tanf` is an extension.

1.40 tanh, tanhf—hyperbolic tangent

```
#include <math.h>
double tanh(double x);
float tanhf(float x);
```

`tanh` computes the hyperbolic tangent of the argument x. Angles are specified in radians.

`tanh(x)` is defined as

`sinh(x)/cosh(x)`

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

The hyperbolic tangent of x is returned.

`tanh` is ANSI C. `tanhf` is an extension.

2 Reentrancy Properties of libm

When a math function detects an error, it sets the static variable errno. Depending upon the severity of the error, it may also print a message to stderr. None of this behavior is reentrant. When one process detects an error, it sets errno. If another process is testing errno, it detects the change and probably fails. Note that failing system calls can also set errno. This problem can only be fixed by either ignoring errno, or treating it as part of the context of a process and switching it along with the rest of a processor state. In normal debugged programs, there are usually no math subroutine errors—and therefore no matherr calls; in that situation, the math functions behave reentrantly.

As an alternative, you can use the reentrant versions of the mathematical functions: these versions have a different name, normally formed by prepending '_' and appending '_r', and use an extra argument—a pointer to the particular reentrancy structure to use. See section "Reentrancy" in *The Cygnus C Support Library*, for more discussion of this approach to reentrancy.

The reentrancy structure is always an additional first argument; for example, the reentrant version of 'double acos (double *x*)' is 'double _acos_r (void **reent*, double *x*)'.

Here is a list of the names for reentrant versions of the mathematical library functions:

```
_acos_r        _gammaf_r_r     _log10f_r
_acosf_r       _hypot_r        _log_r
_acosh_r       _hypotf_r       _logf_r
_acoshf_r      _j0_r           _pow_r
_asin_r        _j0f_r          _powf_r
_asinf_r       _j1_r           _remainder_r
_atanh_r       _j1f_r          _sinh_r
_atanhf_r      _jn_r           _sinhf_r
_cosh_r        _jnf_r          _sqrt_r
_coshf_r       _ldexp_r        _sqrtf_r
_exp_r         _ldexpf_r       _y0_r
_expf_r        _lgamma_r       _y0f_r
_fmod_r        _lgamma_r_r     _y1_r
_fmodf_r       _lgammaf_r      _y1f_r
_gamma_r       _lgammaf_r_r    _yn_r
_gamma_r_r     _log10_r        _ynf_r
_gammaf_r
```

Index

A

acos ... 3
acosf ... 3
acosh ... 4
acoshf ... 4
asin ... 5
asinf ... 5
asinh ... 6
asinhf ... 6
atan ... 7
atan2 ... 8
atan2f ... 8
atanf ... 7
atanh ... 9
atanhf ... 9

C

cbrt ... 11
cbrtf ... 11
ceil ... 18
ceilf ... 18
copysign ... 12
copysignf ... 12
cos ... 40
cosf ... 40

E

erf ... 14
erfc ... 14
erfcf ... 14
erff ... 14
exp ... 15
expf ... 15
expm1 ... 16
expm1f ... 16

F

fabs ... 17
fabsf ... 17
finite ... 26
finitef ... 26
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 ... 23
hypotf ... 23

I

ilogb ... 24
ilogbf ... 24
infinity ... 25
infinityf ... 25
isinf ... 26
isinff ... 26
isnan ... 26
isnanf ... 26

J

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

L

ldexp ... 27

ldexpf ... 27
lgamma ... 21
lgamma_r ... 21
lgammaf ... 21
lgammaf_r ... 21
log ... 28
log10 ... 29
log10f ... 29
log1p ... 30
log1pf ... 30
logf ... 28

M

matherr ... 31
matherr and reentrancy ... 45
modf ... 33
modff ... 33

N

nan ... 34
nanf ... 34
nextafter ... 35
nextafterf ... 35

O

OS stubs ... 1

P

pow ... 36
powf ... 36

R

reentrancy ... 45
remainder ... 37
remainderf ... 37
rint ... 37
rintf ... 37

S

scalbn ... 38
scalbnf ... 38
sin ... 40
sinf ... 40
sinh ... 41
sinhf ... 41
sqrt ... 39
sqrtf ... 39
stubs ... 1
support subroutines ... 1
system calls ... 1

T

tan ... 42
tanf ... 42
tanh ... 43
tanhf ... 43

Y

y0 ... 10
y0f ... 10
y1 ... 10
y1f ... 10
yn ... 10
ynf ... 10

The body of this manual is set in
cmr10 at 10.95pt,
with headings in
and examples in `cmtt10 at 10.95pt`.
cmti10 at 10.95pt and
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, acosf—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 23
1.22 ilogb, ilogbf—get exponent of floating point number 24
1.23 infinity, infinityf—representation of infinity 25
1.24 isnan,isnanf,isinf,isinff,finite,finitef—test for exceptional numbers 26
1.25 ldexp, ldexpf—load exponent 27
1.26 log, logf—natural logarithms 28
1.27 log10, log10f—base 10 logarithms 29
1.28 log1p, log1pf—log of 1 + x 30
1.29 matherr—modifiable math error handler 31
1.30 modf, modff—split fractional and integer parts 33
1.31 nan, nanf—representation of infinity 34
1.32 nextafter, nextafterf—get next number 35
1.33 pow, powf—x to the power y 36
1.34 rint, rintf, remainder, remainderf—round and remainder 37
1.35 scalbn, scalbnf—scale by integer 38
1.36 sqrt, sqrtf—positive square root 39

1.37 `sin`, `sinf`, `cos`, `cosf`—sine or cosine 40
1.38 `sinh`, `sinhf`—hyperbolic sine.................................... 41
1.39 `tan`, `tanf`—tangent 42
1.40 `tanh`, `tanhf`—hyperbolic tangent 43

2 Reentrancy Properties of `libm` 45

Index 47