SPECIAL CASES

Table 10-32 shows the results when the argument to the `atanh` function is a zero, a NaN, or an Infinity, plus other special cases for the `atanh` function.

Table 10-32 Special cases for the `atanh` function

Operation	Result	Exceptions raised
$\text{atanh}(x)$ for $\lvert x \rvert > 1$	NaN	Invalid
$\text{atanh}(-1)$	$-\infty$	None
$\text{atanh}(+1)$	$+\infty$	None
$\text{atanh}(+0)$	+0	None
$\text{atanh}(-0)$	–0	None
$\text{atanh}(\text{NaN})$	NaN	None*
$\text{atanh}(+\infty)$	NaN	Invalid
$\text{atanh}(-\infty)$	NaN	Invalid

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = atanh(1.0);   /* z = +INFINITY */
z = atanh(-1.0);  /* z = -INFINITY */
```

Financial Functions

MathLib provides two functions, `compound` and `annuity`, that can be used to solve various financial or time-value-of-money problems.

compound

You can use the `compound` function to determine the compound interest earned given an interest rate and period.

```
double_t compound (double_t rate, double_t periods);
```

`rate` The interest rate (any positive floating-point number).

`periods` The number of interest periods (any positive floating-point number). This argument might or might not be an integer.

DESCRIPTION

The `compound` function computes the compound interest earned.

$$\text{compound}(r, n) = (1 + r)^n$$

When `rate` is a small number, use the function call `compound(rate,n)` instead of the function call `pow((1 + rate),n)`. The call `compound(rate,n)` produces a more exact result because it avoids the roundoff error that might occur when the expression `1 + rate` is computed.

The `compound` function is directly applicable to computation of present and future values:

$$PV = FV \times (1 + r)^{-n} = \frac{FV}{\text{compound}(r, n)}$$

$$FV = PV \times (1 + r)^n = PV \times \text{compound}(r, n)$$

where *PV* is the amount of money borrowed and *FV* is the total amount that will be paid on the loan.

EXCEPTIONS

When r and n are finite and nonzero, the result of $\text{compound}(r, n)$ might raise one of the following exceptions:

- inexact (for all finite, nonzero values of $r > -1$)
- invalid (if $r < -1$)
- divide-by-zero (if r is -1 and $n < 0$)

SPECIAL CASES

Table 10-33 shows the results when one of the arguments to the `compound` function is a zero, a NaN, or an Infinity, plus other special cases for the `compound` function. In this table, r and n are finite, nonzero floating-point numbers.

Table 10-33 Special cases for the `compound` function

Operation	Result	Exceptions raised
$\text{compound}(r, n)$ for $r < -1$	NaN	Invalid
$\text{compound}(-1, n)$	0 if $n > 0$	None
	$+\infty$ if $n < 0$	Divide-by-zero
$\text{compound}(+0, n)$	1	None
$\text{compound}(r, +0)$	1	None

continued

Table 10-33 Special cases for the `compound` function (continued)

Operation	Result	Exceptions raised
compound(–0, n)	1	None
compound(r, –0)	1	None
compound(±0, ±∞)	NaN	Invalid
compound(NaN, n)	NaN*	None†
compound(r, NaN)	NaN	None†
compound(+∞, n)	+∞ if $n > 0$	None
	0 if $n < 0$	None
compound(r, +∞)	+∞	None
compound(–∞, n)	NaN	Invalid
compound(r, –∞)	0	None

* If both arguments are NaNs, the first NaN is returned.
† If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = compound(-2, 12);   /* z = NAN because a negative interest
                           rate does not make sense. The invalid
                           exception is raised. */
z = compound(-1, -1);   /* z = +INFINITY because a negative
                           interest rate and negative loan period
                           do not make sense. The divide-by-zero
                           exception is raised. */
z = compound(0, INFINITY);/* z = NAN. The invalid exception is
                           raised. */
```

annuity

You can use the `annuity` function to compute the present and future value of annuities.

```
double_t annuity (double_t rate, double_t periods);
```

`rate` The interest rate (any positive floating-point number).

`periods` The number of interest periods (any positive floating-point number). This argument might or might not be an integer.

DESCRIPTION

The `annuity` function computes the present and future values of annuities.

$$\text{annuity}(r, n) = \frac{1 - (1 + r)^{-n}}{r}$$

When `rate` is a small number, use the function call `annuity(rate,n)` instead of the expression:

```
(1 – compound(rate, –n)) / rate
```

The call `annuity(rate,n)` produces a more exact result because it avoids the roundoff errors that might occur when this expression is computed.

This function is directly applicable to the computation of present and future values of ordinary annuities:

$$PV = PMT \times \frac{1 - (1 + r)^{-n}}{r} = PMT \times \text{annuity}(r, n)$$

$$FV = PMT \times \frac{1 - (1 + r)^{n}}{r} = PMT \times (1 + r)n \times \frac{1 - (1 + r)^{-n}}{r}$$
$$= PMT \times \text{compound}(r, n) \times \text{annuity}(r, n)$$

where *PV* is the amount of money borrowed, *FV* is the total amount that will be paid on the loan, and *PMT* is the amount of one periodic payment.

EXCEPTIONS

When r and n are finite and nonzero, the result of $\text{annuity}(r, n)$ might raise one of the following exceptions:

- inexact (for all finite, nonzero values of $r > -1$)
- invalid (if $r < -1$)
- divide-by-zero (if $r = -1$ and $n > 0$)

SPECIAL CASES

Table 10-34 shows the results when one of the arguments to the `annuity` function is a zero, a NaN, or an Infinity, plus other special cases for the `annuity` function. In this table, *r* and *n* are finite, nonzero floating-point numbers.

Table 10-34 Special cases for the `annuity` function

Operation	Result	Exceptions raised
annuity(r, n) for $r < -1$	NaN	Invalid
annuity$(-1, n)$	$+\infty$ if $n > 0$	Divide-by-zero
	-1 if $n < 0$	None
annuity$(+0, n)$	n	None
annuity$(r, +0)$	$+0$	None
annuity$(-0, n)$	n	None
annuity$(r, -0)$	$+0$	None
annuity(NaN, n)	NaN*	None†
annuity(r, NaN)	NaN	None†
annuity$(+\infty, n)$	0 if $n > 0$	None
	$-\infty$ if $n < 0$	None
annuity$(r, +\infty)$	$1/r$	None
annuity$(-\infty, n)$	NaN	Invalid
annuity$(r, -\infty)$	$-\infty$	None

* If both arguments are NaNs, the first NaN is returned.
† If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = annuity(–1, 5);  /* z = +INFINITY. The divide-by-zero
                        exception is raised. */
z = annuity(–2, –2); /* z = NAN. The invalid exception
                        is raised. */
```

Error and Gamma Functions

MathLib provides four error and gamma functions:

erf(x)	Error function
erfc(x)	Complementary error function
gamma(x)	Computes $\Gamma(x)$
lgamma(x)	Computes the natural logarithm of the absolute value of gamma(x)

erf

You can use the `erf` function to perform the error function.

```
double_t erf (double_t x);
```

`x` Any floating-point number.

DESCRIPTION

The `erf` function computes the error function of its argument. This function is antisymmetric.

$$\mathrm{erf}(x) = \frac{2}{\pi}\int_0^x e^{(-t)^2}\,dt$$

EXCEPTIONS

When x is finite and nonzero, either the result of erf(x) is exact or it raises one of the following exceptions:

- inexact (if the result must be rounded or an underflow occurs)
- underflow (if the result is inexact and must be represented as a denormalized number or 0)

SPECIAL CASES

Table 10-35 shows the results when the argument to the `erf` function is a zero, a NaN, or an Infinity.

Table 10-35 Special cases for the `erf` function

Operation	Result	Exceptions raised
erf(+0)	+0	None
erf(–0)	–0	None
erf(NaN)	NaN	None*
erf(+∞)	+1	None
erf(–∞)	–1	None

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = erf(1.0);      /* z ≈ 0.842701. The inexact exception is
                      raised. */
z = erf(–1.0);     /* z ≈ –0.842701. The inexact exception is
                      raised. */
```

erfc

You can use the `erfc` function to perform the complementary error function.

```
double_t erfc (double_t x);
```

`x` Any floating-point number.

DESCRIPTION

The `erfc` function computes the complementary error of its argument. This function is antisymmetric.

$$\operatorname{erfc}(x) = 1.0 - \operatorname{erf}(x)$$

For large positive numbers (around 10), use the function call `erfc(x)` instead of the expression `1.0 – erf(x)`. The call `erfc(x)` produces a more exact result.

EXCEPTIONS

When x is finite and nonzero, either the result of $\mathrm{erfc}(x)$ is exact or it raises one of the following exceptions:

- inexact (if the result must be rounded or an underflow occurs)
- underflow (if the result is inexact and must be represented as a denormalized number or 0)

SPECIAL CASES

Table 10-36 shows the results when the argument to the `erfc` function is a zero, a NaN, or an Infinity.

Table 10-36 Special cases for the `erfc` function

Operation	Result	Exceptions raised
$\mathrm{erfc}(+0)$	+1	None
$\mathrm{erfc}(-0)$	+1	None
$\mathrm{erfc}(\mathrm{NaN})$	NaN	None*
$\mathrm{erfc}(+\infty)$	+0	None
$\mathrm{erfc}(-\infty)$	+2	None

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = erfc(-INFINITY); /* z = 1 - erf(-∞) = 1 - -1 = +2.0 */
z = erfc(0.0);       /* z = 1 - erf(0) = 1 - 0 = 1.0 */
```

gamma

You can use the `gamma` function to perform $\Gamma(x)$.

```
double_t gamma (double_t x);
```

`x` Any positive floating-point number.

DESCRIPTION

The `gamma` function performs $\Gamma(x)$.

$$\text{gamma}(x) = \Gamma(x) = \int_0^{\infty} e^{-t} t^{x-1} dt$$

The `gamma` function reaches overflow very fast as x approaches +∞. For large values, use the `lgamma` function (described in the next section) instead.

EXCEPTIONS

When x is finite and nonzero, either the result of $\text{gamma}(x)$ is exact or it raises one of the following exceptions:

- inexact (if the result must be rounded or an overflow occurs)
- invalid (if x is a negative integer)
- overflow (if the result is outside the range of the data type)

SPECIAL CASES

Table 10-37 shows the results when the argument to the `gamma` function is a zero, a NaN, or an Infinity, plus other special cases for the `gamma` function.

Table 10-37 Special cases for the `gamma` function

Operation	Result	Exceptions raised
$\text{gamma}(x)$ for negative integer x	NaN	Invalid
$\text{gamma}(+0)$	NaN	Invalid
$\text{gamma}(-0)$	NaN	Invalid
$\text{gamma}(\text{NaN})$	NaN	None*
$\text{gamma}(+\infty)$	+∞	Overflow
$\text{gamma}(-\infty)$	NaN	Invalid

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = gamma(-1.0);  /* z = NAN. The invalid exception is raised. */
z = gamma(6);     /* z = 120 */
```

lgamma

You can use the `lgamma` function to compute the natural logarithm of the absolute value of $\Gamma(x)$.

```
double_t lgamma (double_t x);
```

`x` Any positive floating-point number.

DESCRIPTION

The `lgamma` function computes the natural logarithm of the absolute value of $\Gamma(x)$.

$$\text{lgamma}(x) = \log_e(|\Gamma(x)|) = \ln(|\Gamma(x)|)$$

EXCEPTIONS

When x is finite and nonzero, either the result of $\text{lgamma}(x)$ is exact or it raises one of the following exceptions:

- inexact (if the result must be rounded or an overflow occurs)
- overflow (if the result is outside the range of the data type)
- invalid (if $x \le 0$)

SPECIAL CASES

Table 10-38 shows the results when the argument to the `lgamma` function is a zero, a NaN, or an Infinity, plus other special cases for the `lgamma` function.

Table 10-38 Special cases for the `lgamma` function

Operation	Result	Exceptions raised
$\text{lgamma}(x)$ for $x < 0$	NaN	Invalid
$\text{lgamma}(+0)$	NaN	Invalid
$\text{lgamma}(-0)$	NaN	Invalid
$\text{lgamma}(\text{NaN})$	NaN	None*
$\text{lgamma}(+\infty)$	$+\infty$	Overflow
$\text{lgamma}(-\infty)$	NaN	Invalid

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = lgamma(–1.0);    /* z = NAN. The invalid exception is
                        raised. */
z = lgamma(3.41);    /* z = 1.10304. The inexact exception is
                        raised. */
```

Miscellaneous Functions

There are three remaining MathLib transcendental functions:

nextafter(x, y)	Returns next representable number after x in direction of y.
hypot(x)	Computes hypotenuse of a right triangle.
randomx(x)	A pseudorandom number generator.

nextafter

You can use the **nextafter functions** to find out the next value that can be represented after a given value in a particular floating-point type.

```
float       nextafterf (float x, float y);
double      nextafterd (double x, double y);
```

`x`	Any floating-point number.
`y`	Any floating-point number.

DESCRIPTION

The nextafter functions (one for each data type) generate the next representable neighbor of `x` in the direction of `y` in the proper format.

The floating-point values representable in single and double formats constitute a finite set of real numbers. The nextafter functions illustrate this fact by returning the next representable value.

If $x = y$, nextafter(x, y) returns x if x and y are not signed zeros.

EXCEPTIONS

When x and y are finite and nonzero, either the result of nextafter(x, y) is exact or it raises one of the following exceptions:

- inexact (if an overflow or underflow exception occurs)
- overflow (if x is finite and the result is infinite)
- underflow (if the result is inexact, must be represented as a denormalized number or 0, and $x \neq y$)

SPECIAL CASES

Table 10-39 shows the results when one of the arguments to a nextafter function is a zero, a NaN, or an Infinity. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-39 Special cases for the nextafter functions

Operation	Result	Exceptions raised
nextafter(+0, y)	Next representable number in direction of y	Underflow
nextafter(x, +0)	Next representable number in direction of 0	None
nextafter(–0, y)	Next representable number in direction of y	Underflow
nextafter(–0, +0)	+0	None
nextafter(x, –0)	Next representable number in direction of 0	None
nextafter(+0, –0)	–0	None
nextafter(NaN, y)	NaN*	None†
nextafter(x, NaN)	NaN	None†
nextafter(+∞, y)	Largest respresentable number	None
nextafter(x, +∞)	Next representable number greater than x	None
nextafter(–∞, y)	Smallest representable number	None
nextafter(x, –∞)	Next representable number smaller than x	None

* If both arguments are NaNs, the value of the first NaN is returned.
† If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = nextafterf(1.0, +∞);/* z = 1.00000000000000000000001₂
                              ≈ 1.000000119209289551 */
z = nextafterd(1.0, +∞);/* z = 1.00000000…000000000000000001₂
                              ≈ 1.000000000000000222 */
```

hypot

You can use the `hypot` function to compute the length of the hypotenuse of a right triangle.

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

x	Any floating-point number.
y	Any floating-point number.

DESCRIPTION

The `hypot` function computes the square root of the sum of the squares of its arguments. This is an ANSI standard C library function.

$$\text{hypot}(x, y) = \sqrt{x^2 + y^2}$$

The function `hypot` performs its computation without undeserved overflow or underflow. For example, if $x^2 + y^2$ is greater than the maximum representable value of the data type but the square root of $x^2 + y^2$ is not, then no overflow occurs.

EXCEPTIONS

When x and y are finite and nonzero, either the result of $\text{hypot}(x, y)$ is exact or it raises one of the following exceptions:

- inexact (if the result must be rounded or an overflow or underflow occurs)
- overflow (if the result is outside the range of the data type)
- underflow (if the result is inexact and must be represented as a denormalized number or 0)

SPECIAL CASES

Table 10-40 shows the results when one of the arguments to the `hypot` function is a zero, a NaN, or an Infinity. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-40 Special cases for the `hypot` function

Operation	Result	Exceptions raised
hypot(+0, y)	$\lvert y \rvert$	None
hypot(x, +0)	$\lvert x \rvert$	None
hypot(–0, y)	$\lvert y \rvert$	None
hypot(x, –0)	$\lvert x \rvert$	None
hypot(NaN, y)	NaN	None*
hypot(x, NaN)	NaN	None*
hypot(NaN, ±∞)	∞	None
hypot(±∞, NaN)	∞	None
hypot(+∞, y)	+∞	None
hypot(x, +∞)	+∞	None
hypot(–∞, y)	+∞	None
hypot(x, –∞)	+∞	None

* If the NaN is a signaling NaN, the invalid exception is raised.

EXAMPLES

```
z = hypot(2.0, 2.0); /* z = sqrt(8.0) ≈ 2.82843. The inexact
                        exception is raised. */
```

randomx

You can use the `randomx` function to generate a random number.

```
double_t randomx (double_t * x);
```

`x` The address of an integer in the range $1 \le x \le 2^{31} - 2$ stored as a floating-point number.

DESCRIPTION

The `randomx` function is a pseudorandom number generator. The function `randomx` returns a pseudorandom number in the range of its argument. It uses the iteration formula

$$x \leftarrow (75 \times x)\mathrm{mod}(2^{31} - 1)$$

If seed values of `x` are not integers or are outside the range specified for `x`, then results are unspecified. A pseudorandom rectangular distribution on the interval (0, 1) can be obtained by dividing the results from `randomx` by

$$2^{31} - 1 = \mathrm{scalb}(31, 1) - 1$$

EXCEPTIONS

The results are unspecified if the value of x is a noninteger or is outside of the range $1 \le x \le 2^{31} - 2$

SPECIAL CASES

If x is a zero, NaN, or Infinity, the results are unspecified.

EXAMPLES

$\mathrm{randomx}(1)$ = any value in the range $1 \le x \le 2^{31} - 2$.

Transcendental Functions Summary

This section summarizes the transcendental functions declared in the MathLib header file `fp.h` and the constants and data types that they use.

C Summary

Constants

```
extern const double_t pi;
```

Data Types

```
typedef short relop;

enum
{
   GREATERTHAN = ((relop) (0)),
   LESSTHAN,
   EQUALTO,
   UNORDERED
};
```

Transcendental Functions

Comparison Functions

```
double_t fdim               (double_t x, double_t y);
double_t fmax               (double_t x, double_t y);
double_t fmin               (double_t x, double_t y);
relop relation              (double_t x, double_t y);
```

Sign Manipulation Functions

```
double_t copysign           (double_t x, double_t y);
double_t fabs               (double_t x);
long double copysignl       (long double x, long double y);
long double fabsl           (long double x);
```

Exponential Functions

```
double_t exp                (double_t x);
double_t exp2               (double_t x);
double_t expm1              (double_t x);
double_t ldexp              (double_t x, int n);
double_t pow                (double_t x, double_t y);
double_t scalb              (double_t x, long int n);
```

Logarithmic Functions

```
double_t frexp              (double_t x, int *exponent);
double_t log                (double_t x);
double_t log10              (double_t x);
double_t log1p              (double_t x);
double_t log2               (double_t x);
double_t logb               (double_t x);
float modff                 (float x, float *iptrf);
double modf                 (double x, double *iptr);
```

Trigonometric Functions

```
double_t cos                (double_t x);
double_t sin                (double_t x);
double_t tan                (double_t x);
double_t acos               (double_t x);
double_t asin               (double_t x);
double_t atan               (double_t x);
double_t atan2              (double_t y, double_t x);
```

Hyperbolic Functions

```
double_t cosh               (double_t x);
double_t sinh               (double_t x);
double_t tanh               (double_t x);
double_t acosh              (double_t x);
double_t asinh              (double_t x);
double_t atanh              (double_t x);
```

Financial Functions

```
double_t compound          (double_t rate, double_t periods);
double_t annuity           (double_t rate, double_t periods);
```

Error and Gamma Functions

```
double_t erf               (double_t x);
double_t erfc              (double_t x);
double_t gamma             (double_t x);
double_t lgamma            (double_t x);
```

Nextafter Functions

```
float nextafterf           (float x, float y);
double nextafterd          (double x, double y);
```

Hypotenuse Function

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

Random Number Generator Function

```
double_t randomx           (double_t * x);
```