This chapter describes how to use the transcendental and auxiliary functions declared in MathLib. This chapter describes the following types of functions:

- comparison
- sign manipulation
- exponential
- logarithmic
- trigonometric
- hyperbolic
- financial
- error and gamma

It shows the declarations of these functions, describes what they do, describes when they raise floating-point exceptions, and gives examples of how to use them. For functions that manipulate the floating-point environment, see Chapter 8, "Environmental Control Functions." For functions that perform conversions, see Chapter 9, "Conversion Functions." For basic arithmetic and comparison operations, see Chapter 6, "Numeric Operations and Functions."

Some transcendental functions have two implementations: double precision and double-double precision. The double-double-precision implementation has the letter *l* appended to the name of the function and performs exactly the same as the double version. This book uses the double-precision implementation's name to mean both of these implementations. All of the transcendental function declarations appear in the file `fp.h`.

Comparison Functions

MathLib provides four functions that perform comparisons between two floating-point arguments:

fdim(x, y)	Returns the positive difference $x - y$ or 0.
fmax(x, y)	Returns the maximum of x or y.
fmin(x, y)	Returns the minimum of x or y.
relation(x, y)	Returns the relationship between x and y.

These functions take advantage of the rule from the IEEE standard that all values except NaNs have an order:

$$-\infty < \text{all negative real numbers} < -0 = +0 < \text{all positive real numbers} < +\infty$$

These functions also make special cases of NaNs so that they raise no floating-point exceptions.

fdim

You can use the `fdim` function to determine the positive difference between two real numbers.

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

`x` Any floating-point number.

`y` Any floating-point number.

DESCRIPTION

The `fdim` function returns the positive difference between its two arguments.

$$\mathrm{fdim}(x, y) = x - y \qquad \text{if } x > y$$
$$\mathrm{fdim}(x, y) = +0 \qquad \text{if } x \le y$$

EXCEPTIONS

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

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

SPECIAL CASES

Table 10-1 shows the results when one of the arguments to the `fdim` function is a zero, a NaN, or an Infinity. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-1 Special cases for the `fdim` function

Operation	Result	Exceptions raised
$\mathrm{fdim}(+0, y)$	+0	None
$\mathrm{fdim}(x, +0)$	x	None
$\mathrm{fdim}(-0, y)$	+0	None
$\mathrm{fdim}(x, -0)$	x	None
$\mathrm{fdim}(\mathrm{NaN}, y)$	NaN*	None†
$\mathrm{fdim}(x, \mathrm{NaN})$	NaN	None†

Table 10-1 Special cases for the `fdim` function (continued)

Operation	Result	Exceptions raised
$\mathrm{fdim}(+\infty, y)$	$+\infty$	None
$\mathrm{fdim}(x, +\infty)$	$+0$	None
$\mathrm{fdim}(-\infty, y)$	$+0$	None
$\mathrm{fdim}(x, -\infty)$	$+\infty$	None

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

EXAMPLES

```
z = fdim(+INFINITY, 300);  /* z = +∞ – 300 = +INFINITY because
                              +∞ > 300 */
z = fdim(300, +INFINITY);  /* z = +0 because 300 ≤ +∞ */
```

fmax

You can use the `fmax` function to find out which is the larger of two real numbers.

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

`x` Any floating-point number.

`y` Any floating-point number.

DESCRIPTION

The `fmax` function determines the larger of its two arguments.

$$\mathrm{fmax}(x, y) = x \quad \text{if } x \geq y$$
$$\mathrm{fmax}(x, y) = y \quad \text{if } x < y$$

If one of the arguments is a NaN, the other argument is returned.

EXCEPTIONS

When x and y are finite and nonzero, the result of $\mathrm{fmax}(x, y)$ is exact.

SPECIAL CASES

Table 10-2 shows the results when one of the arguments to the `fmax` function is a zero, a NaN, or an Infinity. In this table, x is a finite, nonzero floating-point number. (Note that the order of operands for this function does not matter.)

Table 10-2 Special cases for the `fmax` function

Operation	Result	Exceptions raised
$\mathrm{fmax}(+0, x)$	x if $x > 0$	None
	$+0$ if $x < 0$	
$\mathrm{fmax}(-0, x)$	x if $x > 0$	None
	-0 if $x < 0$	
$\mathrm{fmax}(\pm 0, \pm 0)$	$+0$	None
$\mathrm{fmax}(\mathrm{NaN}, x)$	x*	None†
$\mathrm{fmax}(+\infty, x)$	$+\infty$	None
$\mathrm{fmax}(-\infty, x)$	x	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 = fmax(-INFINITY, -300,000);   /* z = -300,000 because any
                                    integer is greater than -∞ */
z = fmax(NAN, -300,000);   /* z = -300,000 by definition of the
                              function fmax. */
```

fmin

You can use the `fmin` function to determine which is the smaller of two real numbers.

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

`x` Any floating-point number.

`y` Any floating-point number.

DESCRIPTION

The `fmin` function determines the lesser of its two arguments.

$$\text{fmin}(x, y) = x \qquad \text{if } x \le y$$
$$\text{fmin}(x, y) = y \qquad \text{if } y < x$$

If one of the arguments is a NaN, the other argument is returned.

EXCEPTIONS

When x and y are finite and nonzero, the result of $\text{fmin}(x, y)$ is exact.

SPECIAL CASES

Table 10-3 shows the results when one of the arguments to the `fmin` function is a zero, a NaN, or an Infinity. In this table, x is a finite, nonzero floating-point number. (Note that the order of operands for this function does not matter.)

Table 10-3 Special cases for the `fmin` function

Operation	Result	Exceptions raised
$\text{fmin}(+0, x)$	x if $x < 0$	None
	$+0$ if $x > 0$	
$\text{fmin}(-0, x)$	x if $x < 0$	None
	$+0$ if $x > 0$	
$\text{fmin}(\pm 0, \pm 0)$	$+0$	None
$\text{fmin}(\text{NaN}, x)$	x*	None†
$\text{fmin}(+\infty, x)$	x	None
$\text{fmin}(-\infty, x)$	$-\infty$	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 = fmin(–INFINITY, –300,000);   /* z = –INFINITY because –∞ is
                                        smaller than any integer. */
z = fmin(NAN, –300,000);   /* z = –300,000 by definition of the
                                function fmin. */
```

relation

You can use the `relation` function to determine the relationship (less than, greater than, equal to, or unordered) between two real numbers.

```
relop relation (double_t x, double_t y);
```

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

DESCRIPTION

The `relation` function returns the relationship between its two arguments.

The `relation` function is type `relop`, which is an enumerated type. This function returns one of the following values:

if `x` > `y`	`GREATERTHAN`
if `x` < `y`	`LESSTHAN`
if `x` = `y`	`EQUALTO`
if `x` or `y` is a NaN	`UNORDERED`

Programs can use the result of this function in expressions to test for combinations not supported by the comparison operators, such as “less than or unordered.”

EXCEPTIONS

When x and y are finite and nonzero, the result of $\text{relation}(x, y)$ is exact.

SPECIAL CASES

Table 10-4 shows the results when one of the arguments to the `relation` function is a zero, a NaN, or an Infinity. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-4 Special cases for the `relation` function

Operation	Result	Exceptions raised
$\text{relation}(+0, y)$	$<$ if $y > 0$	None
	$>$ if $y < 0$	None
$\text{relation}(x, +0)$	$>$ if $x > 0$	None
	$<$ if $x < 0$	None

Table 10-4 Special cases for the `relation` function (continued)

Operation	Result	Exceptions raised
relation(–0, y)	< if $y > 0$	None
	> if $y < 0$	None
relation(x, –0)	> if $x > 0$	None
	< if $x < 0$	None
relation(+0, –0)	=	None
relation(NaN, y)	Unordered	None*
relation(x, NaN)	Unordered	None*
relation(+∞, y)	>	None
relation(x, +∞)	<	None
relation(+∞, +∞)	=	None
relation(–∞, y)	<	None
relation(x, –∞)	>	None
relation(–∞, –∞)	=	None

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

EXAMPLES

```
r = relation(x, y);
if ((r == LESSTHAN) || (r == UNORDERED))
   printf("No, y is not greater than x.\n");
```

Sign Manipulation Functions

MathLib provides two functions that manipulate the sign bit of the floating-point value:

copysign(x, y)	Copies the sign of y to x.
fabs(x)	Returns the absolute value (positive form) of x.

Because these functions only manipulate the sign bit of the value and do not try to compute the value at all, they raise no floating-point exceptions.

copysign

You can use the `copysign` function to assign to some real number the sign of a second value.

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

`x` Any floating-point number.

`y` Any floating-point number.

DESCRIPTION

The `copysign` function copies the sign of the `y` parameter into the `x` parameter and returns the resulting number.

`copysign(x, 1.0)` is always the absolute value of `x`. The `copysign` function simply manipulates sign bits and hence raises no exception flags.

EXCEPTIONS

When x and y are finite and nonzero, the result of $\text{copysign}(x, y)$ is exact.

SPECIAL CASES

Table 10-5 shows the results when one of the arguments to the `copysign` function is a zero, a NaN, or an Infinity. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-5 Special cases for the `copysign` function

Operation	Result	Exceptions raised
$\text{copysign}(+0, y)$	0 with sign of y	None
$\text{copysign}(x, +0)$	$\lvert x \rvert$	None
$\text{copysign}(-0, y)$	0 with sign of y	None
$\text{copysign}(x, -0)$	$-\lvert x \rvert$	None
$\text{copysign}(\text{NaN}, y)$	NaN with sign of y	None*
$\text{copysign}(x, \text{NaN})$	x with sign of NaN	None*
$\text{copysign}(+\infty, y)$	∞ with sign of y	None
$\text{copysign}(x, +\infty)$	$\lvert x \rvert$	None
$\text{copysign}(-\infty, y)$	∞ with sign of y	None
$\text{copysign}(x, -\infty)$	$-\lvert x \rvert$	None

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

EXAMPLES

```
z = copysign(–1234.567, 1.0);/* z = 1234.567 */
z = copysign(1.0, –1234.567);/* z = –1.0 */
```

fabs

You can use the `fabs` function to determine the absolute value of a real number.

```
double_t fabs (double_t x);
long double fabsl (long double x);
```

`x` Any floating-point number.

DESCRIPTION

The `fabs` function returns the absolute value (positive value) of its argument.

$$\text{fabs}(x) = |x|$$

This function looks only at the sign bit, not the value, of its argument.

EXCEPTIONS

When x is finite and nonzero, the result of $\text{fabs}(x)$ is exact.

SPECIAL CASES

Table 10-6 shows the results when the argument to the `fabs` function is a zero, a NaN, or an Infinity.

Table 10-6 Special cases for the `fabs` function

Operation	Result	Exceptions raised
$\text{fabs}(+0)$	$+0$	None
$\text{fabs}(-0)$	$+0$	None
$\text{fabs}(\text{NaN})$	NaN	None*
$\text{fabs}(+\infty)$	$+\infty$	None
$\text{fabs}(-\infty)$	$+\infty$	None

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

EXAMPLES

```
z = fabs(-1.0);   /* z = 1 */
z = fabs(245.0);  /* z = 245 */
```

Exponential Functions

MathLib provides six exponential functions:

$\exp(x)$	The base e or natural exponential e^x.
$\exp2(x)$	The base 2 exponential 2^x.
$\text{expm1}(x)$	The base e exponential minus 1.
$\text{ldexp}(x, n)$	Returns $x \times 2^n$ (equivalent to scalb).
$\text{pow}(x, y)$	Returns x^y.
$\text{scalb}(x, n)$	Returns $x \times 2^n$.

exp

You can use the `exp` function to raise e to some power.

```
double_t exp (double_t x);
```

`x` Any floating-point number.

DESCRIPTION

The `exp` function performs the exponential function on its argument.

$$\exp(x) = e^x$$

The `log` function performs the inverse operation $(\ln e^x)$.

EXCEPTIONS

When x is finite and nonzero, the result of $\exp(x)$ might raise the following exceptions:

- inexact (for all finite, nonzero values of x)
- 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 10-7 shows the results when the argument to the `exp` function is a zero, a NaN, or an Infinity.

Table 10-7 Special cases for the `exp` function

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

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

EXAMPLES

```
z = exp(0.0);  /* z = e^0 = 1. */
z = exp(1.0);  /* z = e^1 ≈ 2.71828128...The inexact exception is
                  raised. */
```

exp2

You can use the `exp2` function to raise 2 to some power.

```
double_t exp2 (double_t x);
```

`x` Any floating-point number.

DESCRIPTION

The `exp2` function returns the base 2 exponential of its argument.

$$\mathrm{exp2}(x) = 2^x$$

The `log2` function performs the inverse operation $(\log_2 2^x)$.

EXCEPTIONS

When x is finite and nonzero, the result of exp2(x) might raise the following exceptions:

- inexact (for all finite, nonzero values of x)
- 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 10-8 shows the results when the argument to the `exp2` function is a zero, a NaN, or an Infinity.

Table 10-8 Special cases for the `exp2` function

Operation	Result	Exceptions raised
exp2(+0)	+1	None
exp2(–0)	+1	None
exp2(NaN)	NaN	None*
exp2(+∞)	+∞	None
exp2(–∞)	+0	None

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

EXAMPLES

```
z = exp2(2.0); /* z = 2^2 = 4. The inexact exception is raised. */
z = exp2(1.5); /* z = 2^1.5 ≈ 2.82843. The inexact exception is
                  raised. */
```

expm1

You can use the `expm1` function to raise e to some power and subtract 1.

```
double_t expm1 (double_t x);
```

`x` Any floating-point number.

DESCRIPTION

The `expm1` function returns the natural exponential decreased by 1.

$$\text{expm1}(x) = e^x - 1$$

For small numbers, use the function call `expm1(x)` instead of the expression

```
exp(x) - 1
```

The call `expm1(x)` produces a more exact result because it avoids the roundoff error that might occur when the expression is computed.

EXCEPTIONS

When x is finite and nonzero, the result of $\text{expm1}(x)$ might raise the following exceptions:

- inexact (for all finite, nonzero values of x)
- 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 10-9 shows the results when the argument to the `expm1` function is a zero, a NaN, or an Infinity.

Table 10-9 Special cases for the `expm1` function

Operation	Result	Exceptions raised
$\text{expm1}(+0)$	+0	None
$\text{expm1}(-0)$	–0	None
$\text{expm1}(\text{NaN})$	NaN	None*
$\text{expm1}(+\infty)$	+∞	None
$\text{expm1}(-\infty)$	–1	None

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

EXAMPLES

```
z = expm1(-2.1);     /* z = e^-2.1 - 1 = -0.877544. The inexact
                        exception is raised. */
z = expm1(6);        /* z = e^6 - 1 = 402.429. The inexact
                        exception is raised. */
```

ldexp

You can use the `ldexp` function to perform efficient scaling by a power of 2.

```
double_t ldexp (double_t x, int n);
```

`x` Any floating-point number.

`n` An integer representing a power of 2 by which `x` should be multiplied.

DESCRIPTION

The `ldexp` function computes the value $x \times 2^n$ without computing 2^n. This is an ANSI standard C library function.

$$\mathrm{ldexp}(x, n) = x \times 2^n$$

The `scalb` function (described on page 10-19) performs the same operation as this function. The `frexp` function performs the inverse operation; that is, it splits `x` into its fraction field and exponent field.

EXCEPTIONS

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

- inexact (if 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 10-10 shows the results when the floating-point argument to the `ldexp` function is a zero, a NaN, or an Infinity. In this table, n is any integer.

Table 10-10 Special cases for the `ldexp` function

Operation	Result	Exceptions raised
$\mathrm{ldexp}(+0, n)$	+0	None
$\mathrm{ldexp}(-0, n)$	–0	None
$\mathrm{ldexp}(\mathrm{NaN}, n)$	NaN	None*
$\mathrm{ldexp}(+\infty, n)$	+∞	None
$\mathrm{ldexp}(-\infty, n)$	–∞	None

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

EXAMPLES

```
z = ldexp(3.0, 3);    /* z = 3 × 2³ = 24 */
z = ldexp(0.0, 3);    /* z = 0 × 2³ = 0 */
```

pow

You can use the `pow` function to raise a real number to the power of some other real number.

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

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

DESCRIPTION

The `pow` function computes `x` to the `y` power. This is an ANSI standard C library function.

$$\mathrm{pow}(x, y) = x^y$$

Use the function call `pow(x,y)` instead of the expression

```
exp(y * log(x))
```

The call `pow(x,y)` produces a more exact result.

There are some differences between this implementation and the behavior of the `pow` function in a SANE implementation. For example, in SANE `pow(NAN,0)` returns a NaN, whereas in PowerPC Numerics, `pow(NAN,0)` returns a 1.

EXCEPTIONS

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

- inexact (if y is not an integer or an underflow or overflow occurs)
- invalid (if x is negative and y is not an integer)
- 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 10-11 shows the results when one of the arguments to the `pow` function is a zero, a NaN, or an Infinity, plus other special cases for the `pow` function. In this table, x and y are finite, nonzero floating-point numbers.

Table 10-11 Special cases for the pow function

Operation	Result	Exceptions raised
pow(x, y) for $x < 0$	NaN if y is not integer	Invalid
	x^y if y is integer	None
pow(+0, y)	±0 if y is odd integer > 0	None
	+0 if y > 0 but not odd integer	None
	±∞ if y is odd integer < 0	Divide-by-zero
	+∞ if y < 0 but not odd integer	Divide-by-zero
pow(x, +0)	+1	None
pow(–0, y)	±0 if y is odd integer > 0	None
	+0 if y > 0 but not odd integer	None
	±∞ if y is odd integer < 0	Divide-by-zero
	+∞ if y < 0 but not odd integer	Divide-by-zero
pow(x, –0)	+1	None
pow(NaN, y)	NaN if $y \neq 0$	None*
	+1 if $y = 0$	None*
pow(x, NaN)	NaN	None*
pow(+∞, y)	+∞ if $y > 0$	None
	+0 if $y < 0$	None
	+1 if $y = 0$	None
pow(x, +∞)	+∞ if $\lvert x \rvert > 1$	None
	+0 if $\lvert x \rvert < 1$	None
	NaN if $\lvert x \rvert = 1$	Invalid
pow(–∞, y)	–∞ if y is odd integer > 0	None
	+∞ if y > 0 but not odd integer	None
	–0 if y is odd integer < 0	None
	+0 if y < 0 but not odd integer	None
	+1 if $y = 0$	None
pow(x, –∞)	+0 if $\lvert x \rvert > 1$	None
	+∞ if $\lvert x \rvert < 1$	None
	NaN if $\lvert x \rvert = 1$	Invalid

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

EXAMPLES

```
z = pow(NAN, 0);      /* z = 1 */
```

scalb

You can use the `scalb` function to perform efficient scaling by a power of 2.

```
double_t scalb (double_t x, long int n);
```

`x` Any floating-point number.

`n` An integer representing a power of 2 by which `x` should be multiplied.

DESCRIPTION

The `scalb` function performs efficient scaling of its floating-point argument by a power of 2.

$$\text{scalb}(x, n) = x \times 2^n$$

Using the `scalb` function is more efficient than performing the actual arithmetic.

This function performs the same operation as the `ldexp` transcendental function described on page 10-16.

EXCEPTIONS

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

- inexact (if the result causes an overflow or underflow exception)
- 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 10-12 shows the results when the floating-point argument to the `scalb` function is a zero, a NaN, or an Infinity. In this table, n is any integer.

Table 10-12 Special cases for the `scalb` function

Operation	Result	Exceptions raised
scalb(+0, n)	+0	None
scalb(–0, n)	–0	None
scalb(NaN, n)	NaN	None*
scalb(+∞, n)	+∞	None
scalb(–∞, n)	–∞	None

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

EXAMPLES

```
z = scalb(1, 3);  /* z = 1 × 2^3 = 8 */
```

Logarithmic Functions

MathLib provides seven logarithmic functions:

frexp(x, exp)	Splits x into fraction and exponent fields.
log(x)	Base e or natural logarithm.
log10(x)	Base 10 logarithm.
log1p(x)	Computes $\log(1+x)$.
log2(x)	Base 2 logarithm.
logb(x)	Returns exponent part of x.
modf(x, $iptr$)	Splits x into an integer and a fraction.

frexp

You can use the `frexp` function to find out the values of a floating-point number's fraction field and exponent field.

```
double_t frexp (double_t x, int *exponent);
```

`x`	Any floating-point number.
`exponent`	A pointer to an integer in which the value of the exponent can be returned.

DESCRIPTION

The `frexp` function splits its first argument into a fraction part and a base 2 exponent part. This is an ANSI standard C library function.

$$\text{frexp}(x, n) = f \quad \text{such that } x = f \times 2^n$$

or

$$\text{frexp}(x, n) = f \quad \text{such that} \quad n = (1 + \text{logb}(x)) \quad \text{and} \quad f = \text{scalb}(x, -n)$$

The return value of `frexp` is the value of the fraction field of the argument `x`. The exponent field of `x` is stored in the address pointed to by the `exponent` argument.

For finite nonzero inputs, `frexp` returns either 0.0 or a value whose magnitude is between 0.5 and 1.0.

The `ldexp` and `scalb` functions perform the inverse operation (compute $f \times 2^n$).

EXCEPTIONS

If x is finite and nonzero, the result of $\text{frexp}(x, n)$ is exact.

SPECIAL CASES

Table 10-13 shows the results when the input argument to the `frexp` function is a zero, a NaN, or an Infinity.

Table 10-13 Special cases for the `frexp` function

Operation	Result	Exceptions raised
$\text{frexp}(+0, n)$	$+0\ (n = 0)$	None
$\text{frexp}(-0, n)$	$-0\ (n = 0)$	None
$\text{frexp}(\text{NaN}, n)$	NaN (n is undefined)	None*
$\text{frexp}(+\infty, n)$	$+\infty$ (n is undefined)	None
$\text{frexp}(-\infty, n)$	$-\infty$ (n is undefined)	None

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

EXAMPLES

```
z = frexp(2E300, n);    /* z ≈ 0.746611 and n = 998. In other
                           words, 2 × 10^300 ≈ 0.746611 × 2^998. */
```

log

You can use the `log` function to compute the natural logarithm of a real number.

```
double_t log (double_t x);
```

`x` Any positive floating-point number.

DESCRIPTION

The `log` function returns the natural (base *e*) logarithm of its argument.

$$\log(x) = \log_e x = \ln x = y \quad \text{such that } x = e^y$$

The `exp` function performs the inverse (exponential) operation.

EXCEPTIONS

When x is finite and nonzero, the result of $\log(x)$ might raise one of the following exceptions:

- inexact (for all finite, nonzero values of x other than +1)
- invalid (if x is negative)

SPECIAL CASES

Table 10-14 shows the results when the argument to the `log` function is a zero, a NaN, or an Infinity, plus other special cases for the `log` function.

Table 10-14 Special cases for the `log` function

Operation	Result	Exceptions raised
$\log(x)$ for $x < 0$	NaN	Invalid
$\log(+1)$	+0	None
$\log(+0)$	–∞	Divide-by-zero
$\log(-0)$	–∞	Divide-by-zero
$\log(\text{NaN})$	NaN	None*
$\log(+\infty)$	+∞	None
$\log(-\infty)$	NaN	Invalid

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