Notes 15.0: Mathematical functions

COMP9021 Principles of Programming

School of Computer Science and Engineering
The University of New South Wales

2013 session 1

The math.h header file

Except for the functions abs(), labs() and llabs(), defined in stdlib.h, all functions described in this set of notes are defined in math.h. They all come in three variants, two of which add f or l to the base name to indicate that they take arguments of type float or long double rather than of type double, respectively, or arguments of type pointer to float or pointer to long double rather than of type pointer to double, respectively (they might also take arguments of another fixed type), and when the base function returns a value of type double, to indicate that they return values of type float or long double, respectively.

Some installations require to pass -lm as argument to the linker when compiling programs that make use of functions or macros defined in math.h.

Constants (1)

The following macros are provided for constants of type double.

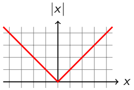
e	M_E (2.718282)
$\log_2(e)$	M_LOG2E (1.442695)
$\log_{10}(e)$	M_LOG10E (0.434294)
$\ln(2)$	M_LN2 (0.693147)
$\ln(10)$	M_LN10 (2.302585)

Constants (2)

π	M_PI (3.141593)
$\pi/2$	M_PI_2 (1.570796)
$\pi/4$	M_PI_4 (0.785398)
$1/\pi$	M_1_PI (0.318310)
$2/\pi$	M_2_PI (2.718282)
$2/\sqrt{\pi}$	M_2_SQRTPI (1.128379)
$\sqrt{2}$	M_SQRT2 (1.414214)
$\sqrt{1/2}$	M_SQRT1_2 (0.707107)

Absolute value

```
double fabs(double);
float fabsf(float);
long double fabsl(long double);
```

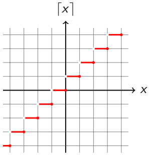


Defined in `stdlib.h`:

```
int abs(int);
long labs(long);
long long llabs(long long);
```

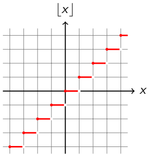
Ceiling

```
double ceil(double);
float ceilf(float);
long double ceill(long double);
```



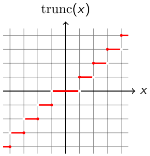
Floor

```
double floor(double);
float floorf(float);
long double floorl(long double);
```



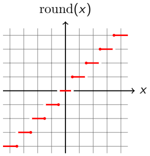
Truncation towards 0

```
double trunc(double);
float truncf(float);
long double truncl(long double);
```



Rounding towards integer larger in absolute value (1)

```
double round(double);
float roundf(float);
long double roundl(long double);
```



Rounding towards integer larger in absolute value (2)

```
long lround(double);
long lroundf(float);
long lroundl(long double);

long long llround(double);
long long llroundf(float);
long long llroundl(long double);
```

Rounding towards nearest integer (1)

```
double nearbyint(double);
float nearbyintf(float);
long double nearbyintl(long double);
```

The header file `fenv.h` defines

- the macros `FE_DOWNWARD`, `FE_UPWARD`, `FE_TONEAREST` and `FE_TOWARDZERO` that correspond to four rounding directions,
- the function `fegetround()` that returns the current rounding direction, and
- the function `fesetround()` that sets the rounding direction.

`nearbyint()` and associated functions compute the nearest integer according to the current value of the rounding direction. The program rounding.c illustrates.

Rounding towards nearest integer (2)

```
double rint(double);
float rintf(float);
long double rintl(long double);

long lrint(double);
long lrintf(float);
long lrintl(long double);

long long llrint(double);
long long llrintf(float);
long long llrintl(long double);
```

`rint()` and associated functions are like `nearbyint()`, but raise an exception (`FE_INEXAC`) when their argument is not an integer. The `fenv.h` header file defines functions to show the value of exception status flags determined by which exceptions have been raised, and to clear exception status flags. This is illustrated in fe_inexact.c.

Floating-point remainder

```
double fmod(double, double);
float fmodf(float, float);
long double fmodl(long double, long double);

double remainder(double, double);
float remainderf(float, float);
long double remainderl(long double, long double);
```

`fmod()` and associated functions, given `x` and `y` as arguments, return `x - trunc(x/y)`, which always has the same sign as `x`.

`remainder()` and associated functions, given `x` and `y` as arguments, return `x - nearbyint(x/y)` with `FE_TONEAREST` as rounding direction, which does not always have the same sign as `x`. (For all integers `n`, `nearbyint(2*n + 0.5)` is equal to `2*n` and `nearbyint(2*n + 1.5)` to `2*n + 2`.) For instance, `remainder(9, 2)` evaluates to `1` while `remainder(11, 2)` evaluates to `-1`.

Decomposition of floating-point numbers (1)

```
double frexp(double, int *);
float frexpf(float, int *);
long double frexpl(long double, int *);
```

`frexp()` and associated functions,

- given `0` and `&n` as arguments, return `0` and store `0` at location `&n`;
- given nonzero `x` and `&n` as arguments, compute the unique f and n such that $0.5 \leq |f| < 1$ and $f \times 2^n$ is equal to `x`, return f and store n at location `&n`.

For instance, `frexp(2, &n)` returns `0.5` and stores `2` at location `&n`, `frexp(3, &n)` returns `0.75` and stores `2` at location `&n`, and `frexp(4, &n)` returns `0.5` and stores `3` at location `&n`.

Decomposition of floating-point numbers (2)

```
double ldexp(double, int);
float ldexpf(float, int);
long double ldexpl(long double, int);
```

`ldexp()` and associated functions are the inverse of `frexp()` and associated functions, in that given a real f and an integer n as arguments, they return $f \times 2^n$.

Decomposition of floating-point numbers (3)

```
double modf(double, double *);
float modff(float, float *);
long double modfl(long double, long double *);
```

`modf()` and associated functions, given `x` and `&n` as arguments, compute the unique f and n such that $|f| < 1$, f and n have the same sign as `x` and $f + n$ is equal to `x`, return f and store n at location `&n`. For instance, `modf(2.6, &n)` returns `0.6` and stores `2` at location `&n` and `modf(-2.6, &n)` returns `-0.6` and stores `-2` at location `&n`.

Scaling of floating-point numbers

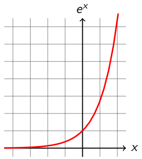
```
double scalbn(double, int);
float scalbnf(float, int);
long double scalbnl(long double, int);

double scalbln(double, long);
float scalblnf(float, long);
long double scalblnl(long double, long);
```

`scalbn()` and associated functions, given `x` and `n` as arguments, return the product of `x` with `FLT_RADIX` raised to the power `n`. The macro `FLT_RADIX` is defined in the header file `float.h` and typically has the value `2`.

Calling `scalbn(x, n)` is meant to be more efficient than executing `x * pow(FLT_RADIX, n)`.

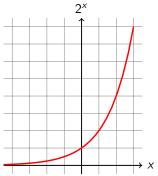
Exponential functions (1)

```
double exp(double x);
float expf(float);
long double expl(long double);
```



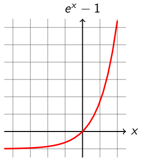
Exponential functions (2)

```
double exp2(double);
float exp2f(float);
long double exp2l(long double);
```



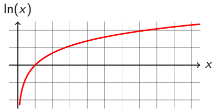
Exponential functions (3)

```
double expm1(double);
float expm1f(float);
long double expm1l(long double);
```



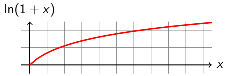
Logarithmic functions (1)

```
double log(double);
float logf(float);
long double logl(long double);
```



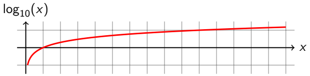
Logarithmic functions (2)

```
double log1p(double);
float log1pf(float);
long double log1pl(long double);
```



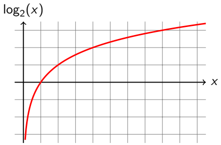
Logarithmic functions (3)

```
double log10(double);
float log10f(float);
long double log10l(long double);
```



Logarithmic functions (4)

```
double log2(double);
float log2f(float);
long double log2l(long double);
```



Logarithmic functions (5)

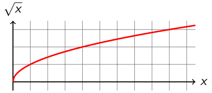
```
double logb(double);
float logbf(float);
long double logbl(long double);
int ilogb(double);
int ilogbf(float);
int ilogbl(long double);
```

logb(), ilogb() and associated functions, given nonzero x as argument, return the exponent of the representation of x as a floating-point number with FLT_RADIX used for the radix. The macro FLT_RADIX is defined in the header file float.h and typically has the value 2, so for instance logb(0.5) and ilogb(0.5) return -1, and logb(20), ilogb(20), logb(25) and ilogb(25) return 4.

When given 0 as argument, logb() and related functions return -inf while ilogb() and related functions return the value of the macro FP_ILOGB0 (which on some machines is equal to -2^{32}).

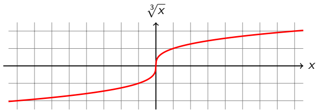
Square root

```
double sqrt(double);
float sqrtf(float);
long double sqrt2l(long double);
```



Cube root

```
double cbrt(double);
float cbrtf(float);
long double cbrt2l(long double);
```



Power and hypotenuse functions

```
double pow(double, double);
float powf(float, float);
long double powl(long double, long double);

double hypot(double, double);
float hypotf(float, float);
long double hypotl(long double, long double);
```

pow() and associated functions, given x and y as arguments, return the value of x raised to the power y. If x is equal to 0 then y has to be strictly positive; if x is negative then y has to be an integer.

hypot() and associated functions, given x and y as arguments, return the value of sqrt(x*x + y*y) without causing the undue overflow that the latter expression could cause.

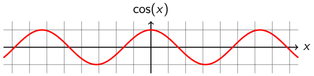
Multiply and add

```
double fma(double, double, double);
float fmaf(float, float, float);
long double fmal(long double,
                 long double,
                 long double);
```

`fma()` and associated functions, given `x`, `y` and `z` as arguments, return the value of `x * y + z`, with rounding to the precision of the return type done only to the final result, not to intermediate computations.

Cosine

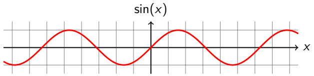
```
double cos(double);
float cosf(float);
long double cosl(long double);
```



The argument is taken in radian, and will yield a result of little significance if too large.

Sine

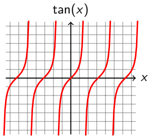
```
double sin(double);
float sinf(float);
long double sinl(long double);
```



The argument is taken in radian, and will yield a result of little significance if too large.

Tangent

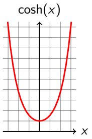
```
double tan(double);
float tanf(float);
long double tanl(long double);
```



The argument is taken in radian, and will yield a result of little significance if too large.

Hyperbolic cosine

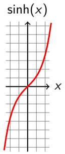
```
double cosh(double);
float coshf(float);
long double coshl(long double);
```



The argument is taken in radian.

Hyperbolic sine

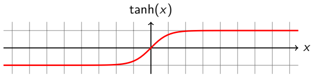
```
double sinh(double);
float sinhf(float);
long double sinhl(long double);
```



The argument is taken in radian.

Hyperbolic tangent

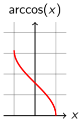
```
double tanh(double);
float tanhf(float);
long double tanhl(long double);
```



The argument is taken in radian.

Arc cosine

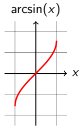
```
double acos(double);
float acosf(float);
long double acosl(long double);
```



The returned value is in radian.

Arc sine

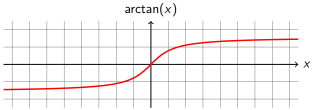
```
double asin(double);
float asinf(float);
long double asinl(long double);
```



The returned value is in radian.

Arc tangent

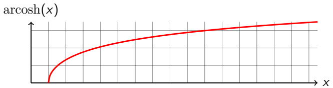
```
double atan(double);
float atanf(float);
long double atanl(long double);
```



The returned value is in radian.

Area hyperbolic cosine

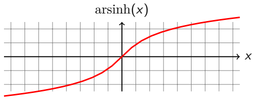
```
double acosh(double);
float acoshf(float);
long double acoshl(long double);
```



The returned value is in radian.

Area hyperbolic sine

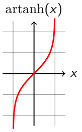
```
double asinh(double);
float asinhf(float);
long double asinhl(long double);
```



The returned value is in radian.

Area hyperbolic tangent

```
double atanh(double);
float atanhf(float);
long double atanhl(long double);
```



The returned value is in radian.

Difference, max and min

```
double fdim(double, double);
float fdimf(float, float);
long double fdiml(long double, long double);
double fmax(double, double);
float fmaxf(float, float);
long double fmaxl(long double, long double);
double fmin(double, double);
float fmif(float, float);
long double fminl(long double, long double);
```

`fdim()` and associated functions, given `x` and `y` as arguments, return `x - y` if `x` is greater than `y`, and `0` otherwise.

`fmax()` and associated functions, given `x` and `y` as arguments, return the maximum of `x` and `y`.

`fmin()` and associated functions, given `x` and `y` as arguments, return the minimum of `x` and `y`.