std::pow, std::powf, std::powl

Defined in header <cmath>

`float       pow ( float base, float exp );`
`float       powf( float base, float exp );`
	(1)	
		(since C++11)

`double      pow ( double base, double exp );`	(2)	

`long double pow ( long double base, long double exp );`
`long double powl( long double base, long double exp );`
	(3)	
		(since C++11)

`float       pow ( float base, int iexp );`	(4)	(until C++11)
`double      pow ( double base, int iexp );`	(5)	(until C++11)
`long double pow ( long double base, int iexp );`	(6)	(until C++11)
`Promoted    pow ( Arithmetic1 base, Arithmetic2 exp );`	(7)	(since C++11)

1-6) Computes the value of base raised to the power exp or iexp.

7) A set of overloads or a function template for all combinations of arguments of arithmetic type not covered by 1-3). If any argument has integral type, it is cast to `double`. If any argument is `long double`, then the return type Promoted is also `long double`, otherwise the return type is always `double`.

Parameters

base	-	base as a value of floating-point or integral type
exp	-	exponent as a value of floating-point or integral type
iexp	-	exponent as integer value

Return value

If no errors occur, base raised to the power of exp (or iexp) ($base^{exp}$), is returned.

If a domain error occurs, an implementation-defined value is returned (NaN where supported)

If a pole error or a range error due to overflow occurs, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL is returned.

If a range error occurs due to underflow, the correct result (after rounding) is returned.

Error handling

Errors are reported as specified in math_errhandling.

If base is finite and negative and exp is finite and non-integer, a domain error occurs and a range error may occur.

If base is zero and exp is zero, a domain error may occur.

If base is zero and exp is negative, a domain error or a pole error may occur.

If the implementation supports IEEE floating-point arithmetic (IEC 60559),

- `pow(+0, exp)`, where exp is a negative odd integer, returns +∞ and raises FE_DIVBYZERO
- `pow(-0, exp)`, where exp is a negative odd integer, returns -∞ and raises FE_DIVBYZERO
- `pow(±0, exp)`, where exp is negative, finite, and is an even integer or a non-integer, returns +∞ and raises FE_DIVBYZERO
- `pow(±0, -∞)` returns +∞ and may raise FE_DIVBYZERO
- `pow(+0, exp)`, where exp is a positive odd integer, returns +0
- `pow(-0, exp)`, where exp is a positive odd integer, returns -0
- `pow(±0, exp)`, where exp is positive non-integer or a positive even integer, returns +0
- `pow(-1, ±∞)` returns 1
- `pow(+1, exp)` returns 1 for any exp, even when exp is NaN
- `pow(base, ±0)` returns 1 for any base, even when base is NaN
- `pow(base, exp)` returns NaN and raises FE_INVALID if base is finite and negative and exp is finite and non-integer.
- `pow(base, -∞)` returns +∞ for any |base|<1
- `pow(base, -∞)` returns +0 for any |base|>1

- `pow(base, +∞)` returns +0 for any |base|<1
- `pow(base, +∞)` returns +∞ for any |base|>1
- `pow(-∞, exp)` returns -0 if exp is a negative odd integer
- `pow(-∞, exp)` returns +0 if exp is a negative non-integer or negative even integer
- `pow(-∞, exp)` returns -∞ if exp is a positive odd integer
- `pow(-∞, exp)` returns +∞ if exp is a positive non-integer or positive even integer
- `pow(+∞, exp)` returns +0 for any negative exp
- `pow(+∞, exp)` returns +∞ for any positive exp
- except where specified above, if any argument is NaN, NaN is returned

Notes

pow(float, int) returns `float` until C++11 (per overload 4) but returns `double` since C++11 (per overload 7)

Although std::pow cannot be used to obtain a root of a negative number, std::cbrt is provided for the common case where exp is 1/3

Example

Run this code

```
#include <iostream>
#include <cmath>
#include <cerrno>
#include <cfenv>
#include <cstring>

#pragma STDC FENV_ACCESS ON
int main()
{
    // typical usage
    std::cout << "pow(2, 10) = " << std::pow(2,10) << '\n'
              << "pow(2, 0.5) = " << std::pow(2,0.5) << '\n'
              << "pow(-2, -3) = " << std::pow(-2,-3) << '\n';
    // special values
    std::cout << "pow(-1, NAN) = " << std::pow(-1,NAN) << '\n'
              << "pow(+1, NAN) = " << std::pow(+1,NAN) << '\n'
              << "pow(INFINITY, 2) = " << std::pow(INFINITY, 2) << '\n'
              << "pow(INFINITY, -1) = " << std::pow(INFINITY, -1) << '\n';
    // error handling
    errno = 0;
    std::feclearexcept(FE_ALL_EXCEPT);
    std::cout << "pow(-1, 1/3) = " << std::pow(-1, 1.0/3) << '\n';
    if (errno == EDOM)
        std::cout << "    errno == EDOM " << std::strerror(errno) << '\n';
    if (std::fetestexcept(FE_INVALID))
        std::cout << "    FE_INVALID raised\n";

    std::feclearexcept(FE_ALL_EXCEPT);
    std::cout << "pow(-0, -3) = " << std::pow(-0.0, -3) << '\n';
    if (std::fetestexcept(FE_DIVBYZERO))
        std::cout << "    FE_DIVBYZERO raised\n";
}
```

Possible output:

```
pow(2, 10) = 1024
pow(2, 0.5) = 1.41421
pow(-2, -3) = -0.125
pow(-1, NAN) = nan
pow(+1, NAN) = 1
pow(INFINITY, 2) = inf
pow(INFINITY, -1) = 0
pow(-1, 1/3) = -nan
    errno == EDOM Numerical argument out of domain
    FE_INVALID raised
pow(-0, -3) = -inf
    FE_DIVBYZERO raised
```

See also

sqrt **sqrtf** (C++11) **sqrtl** (C++11)	computes square root ($\sqrt{x}$) (function)
cbrt (C++11) **cbrtf** (C++11) **cbrtl** (C++11)	computes cubic root ($\sqrt[3]{x}$) (function)
hypot (C++11) **hypotf** (C++11) **hypotl** (C++11)	computes square root of the sum of the squares of two or three (C++17) given numbers ($\sqrt{x^2+y^2}$), ($\sqrt{x^2+y^2+z^2}$) (function)
pow(std::complex)	complex power, one or both arguments may be a complex number (function template)
pow(std::valarray)	applies the function **std::pow** to two valarrays or a valarray and a value (function template)
C documentation for **pow**	

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/numeric/math/pow&oldid=138092"