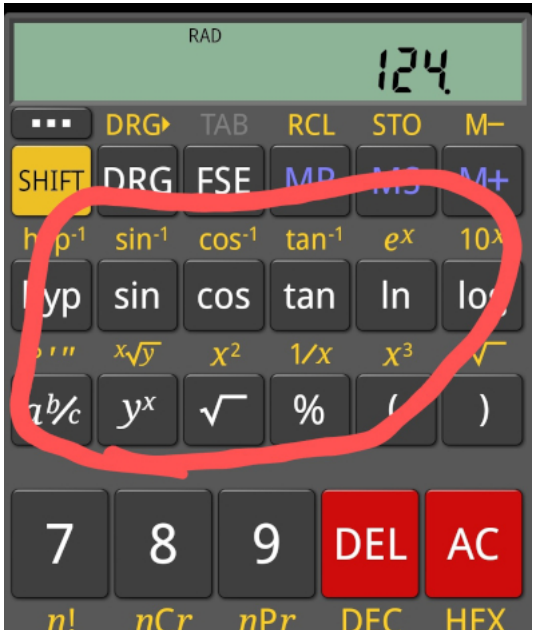
<cmath>

A scientific calculator for C++ - Part 1



Doing math in C++

The math functions from scientific pocket calculators can be found in the header `<cmath>`. But the usage need some knowledge:

Scientific calculator	C++ / `<cmath>`
ln	log
log	log10
e^x	exp(x)
10^x	missing, workaround: pow(10., x)
nth-root	pow(x, 1.0/n)
sin, cos, tan	only RAD, no DEG
π	missing (fixed in C++20, M_PI)

Nothing special, should be found in every documentation, but my favorite one (the C++ standard) will not help.

Motivation (performance) (1/4)

```cpp
#include <cmath>

float foo(float x) {
  return sin(x);
}

float bar(float x) {
  using namespace std;
  return sin(x);
}
```

What's could go possibly wrong with `foo()`?

Motivation (performance) (2/4)

```
Benchmark           Time             CPU   Iterations
----------------------------------------------------
BM_foo/100       8.00 ns         8.00 ns     81838131
BM_bar/100       3.40 ns         3.40 ns    203756132
```

GCC 11.3, Ubuntu 22.04, AMD Ryzen 7 PRO 5850U @ 2.0 GHz

https://quick-bench.com/q/cRPCTW4RdalNq8wlalknPqUHIcc

Motivation (performance) (3/4)

```
#include <cmath>

static void BM_foo(benchmark::State& state) {
    while (state.KeepRunning()) {
        const float x = state.range(0) / 100.0f;
        const float y = sin(x);
        benchmark::DoNotOptimize(y);
    }
}
static void BM_bar(benchmark::State& state) {
    using namespace std;
    while (state.KeepRunning()) {
        const float x = state.range(0) / 100.0f;
        const float y = sin(x);
        benchmark::DoNotOptimize(y);
    }
}
BENCHMARK(BM_foo)->Arg(100);
BENCHMARK(BM_bar)->Arg(100);
```

Motivation (performance) (4/4)

many `sin()` functions

`foo()` calls `sin(double)` from `math.h`

`bar()` calls `std::sin(float)` from `<cmath>`.

The convertions from `float` to `double` and back are more expensive the call of `sin()`.

cppreference (and others) will show the "C++" overloads, but you have to know the overlay between `<math.h>` and `<cmath>` includes.

Motivation (correctness)

```
std::cout << abs(-3.14f) << "\n";
```

Results and performance depends on includes and namespace.

Some backwards references ...

C90 and earlier

1 macro and 22 functions.

- HUGE_VAL (Macro), acos, asin, atan, atan2, cos, sin, tan, cosh, sinh, tanh, exp, frexp, ldexp, log, log10, modf, pow, sqrt, ceil, fabs, floor, fmod,

All function are defined only for **`double`**.

Example:

```
double sin(double x);
```

These set of functions can be found very similar in Python, Java, JavaScript and other languages and also C++.

C++-98

- reference to "ISO/IEC 9899:1990, Programming languages - C"
- same math functions as C90,
- new header **`<cmath>`**
- overloads for all functions with **`float`**, **`double`**, **`long double`** in the namespace **`std`**.

26.8 C Library [c.math] (C++-98)

- **1.** ...
- **2.** The contents of these headers (**<cmath>** and **<cstdlib>**) are the same as the Standard C library headers **<math.h>** and **<stdlib.h>** respectively, with the following changes:
- **3.** & **4.** stdlib overloads for **int**, **long** and **long long** (**abs**, **div**)
- **5.** In addition to the double versions of the math functions in **<cmath>**, C++ adds float and long double overloaded versions of these functions, with the same semantics.
- **6.** list of function with additional overloads ("added signatures")
  - abs, sin, trunc, ...
  - "SEE ALSO: ISO C subclauses 7.5, 7.10.2, 7.10.6."

dual exists of `std::abs`

`<cmath>`	`<cstdlib>`
float std::abs(float);	int std::abs(int);
double std::abs(double);	long std::abs(long);
long double std::abs(long double);	long std::labs(long);
float std::fabs(float);	int abs(int);
double std::fabs(double);	long labs(long);
long double std::fabs(long double);	
double fabs(double);	

abs example

```cpp
//#include <cmath>
#include <cstdlib>
#include <iostream>

int main()
{
    //using namespace std;
    std::cout << abs(-3.14f) << "\n";
}
```

include	namespace	result
<cstdlib>	global	3
<cstdlib>	std	compile error
<cmath>	global	3.0 (double)
<cmath>	std	3.0f (float)

C99

- 2 types: **`double_t`**, **`float_t`**
- new macros: **`HUGE_VALF`**, **`HUGE_VALL`**, **`INFINITY`**, **`NAN`** and more
- number classification macros: FP_INFINITE, FP_NAN, FP_NORMAL, FP_SUBNORMAL, FP_ZERO, isXXX
- updated error handling,
- 33 new functions: acosh, **asinh**, atanh, exp2, expm1, ilogb, log1p, log2, logb, scalbn, scalbln, cbrt, **hypot**, erf, erfc, lgamma, tgamma, nearbyint, rint, lrint, llrint, round, trunc, remainder, remquo, copysign, nan, nextafter, nexttoward, fdim, fmax, fmin, fma,
- Comparison macros: isgreater, isgreaterequal, isless, islessequal, islessgreater, isunordered,
- new "overloads" for double functions with new names: **`sinf`**, **`sinl`**,... (**`33*3 + 22*3`** => 165 math functions)

12 long years between 1999 and 2011

Compilers like GCC includes latest **`math.h`** (C99 version) in **`<cmath>`**, but in **`namespace std`** supports only C98 function set.

Because of:
"The contents of these headers (**`<cmath>`** and **`<cstdlib>`**) are the same as the Standard C library headers ..."

Example:

```
#include <cmath>
using namespace std;
float x = ...;
float t1 = sin(x);       // float std::sin(float); part of C++
float result = asinh(t1); // double asinh(double); part of C99
```

(Side note: TR1 already addressed this issue.)

C++11

(reference to C99)

includes new math functions from C99, with same overloads as before.

all ...f and ...l functions are part of **<cmath>**, but in global namespace.

3 Pages with a list of math functions with added signatures. (**double** functions are not listed)

Comparision macros from C99 as functions, the list of function is including the **double** overloads.

(take over proposal from TR1, but without the special math functions)

no changes in this context.

(C++14 still points to C99)

C++-17

(reference to C11)

Renamed header "29.9 Mathematical functions for floating-point types" (was "C Library")

hypot Three-dimensional hypotenuse (**`hypot`** with 3 parameters)

first math function with "real" description (Returns: ...) in the C++ standard.

```
float hypot(float x, float y, float z);
double hypot(double x, double y, double z);
long double hypot(long double x, long double y, long double z);
floating-point-type hypot(floating-point-type x, floating-point-type
```

The 2 parameter overload from the C standard is only listed in the list of functions without description.

C++-17

- list of math functions now contain also **`float`** (e.g. **`sinf`**) and **`long double`** (e.g. **`sinl`**) variant in the namespace std:

```
namespace std {
  ...
  float sin(float x); // see 20.2
  double sin(double x);
  long double sin(long double x); // see 20.2
  float sinf(float x);
  long double sinl(long double x);
  ...
}
```

(cppreference: **`std::sinf()`** and **`std::sinl()`** since C++11)

...f and ...l functions

C99 introduced "overloads" for **`float`** and **`double`** for the math functions.

Are these functions (...f and ...l) part of the namespace std in C++?

```
float std::sinf(float);
long double std::sinl(long double);
```

reference	version
IMHO	since C++17
cppreference	since C++11
GCC	not implemented
Clang	not implemented
MSVC	since C++14 earlier?

GCC and Clang tested on godbolt and local computer in differt versions.

MSVC v19.22 tested on godbolt. (earlier version unknown)

C++-17 absolut values

`std::abs` with overloads for **`int`**, **`long`**, **`long long`**, **`float`**, **`double`**, **`long double`** are defined in both headers **`<cmath>`** and **`<cstdlib>`**.

`fabs` with overloads for **`float`**, **`double`**, **`long double`** are only defined in **`<cmath>`**.

In the global namespace the floating-point (**`<math.h>`**) and integer (**`<stdlib.h>`**) **`abs`** functions are C-Style "overloads".

C++-17 special functions

- Mathematical special functions from TR1

  assoc_laguerre, assoc_legendre, beta, comp_ellint_1, comp_ellint_2, comp_ellint_3, cyl_bessel_i, cyl_bessel_j, cyl_bessel_k, cyl_neumann, ellint_1, ellint_2, ellint_3, expint, hermite, laguerre, legendre, riemann_zeta, sph_bessel, sph_legendre, sph_neumann.

  All function als C-Style overloads `...`, `...f` and `...l` .

no changes in this context.

C++-20

(reference to C18)

- Linear interpolation

```
constexpr floating-point-type lerp(floating-point-type a,
                                   floating-point-type b,
                                   floating-point-type t) noexcept;
```

- New header **<numbers>** with constants like **std::pi_v<T>**, ... in the namespace **std::numbers**
  - "template constants": e_v, log2e_v, log10e_v, pi_v, inv_pi_v, inv_sqrtpi_v, ln2_v, ln10_v, sqrt2_v, sqrt3_v, inv_sqrt3_v, egamma_v, phi_v
  - double specialisations: e, log2e, log10e, pi, inv_pi, inv_sqrtpi, ln2, ln10, sqrt2, sqrt3, inv_sqrt3, egamma, phi
  - Example **const double myPi = std::numbers::pi;**

New, more compact list format for the list of the math function:

```
floating-point-type sin(floating-point-type x);
float sinf(float x);
long double sinl(long double x);
```

C23

Work in progress (AFIK), (information based on n3088.pdf)

New "decimal floating-point types": **`_Decimal32`**, **`_Decimal64`**, **`_Decimal128`**, result in new math functions:

```c
#include <math.h>
double sin(double x);
float sinf(float x);
long double sinl(long double x);
#ifdef __STDC_IEC_60559_DFP__
_Decimal32 sind32(_Decimal32 x);
_Decimal64 sind64(_Decimal64 x);
_Decimal128 sind128(_Decimal128 x);
#endif
```

C23 pi-functions

Trigonometric functions based on **`2.0`** is a full circle.

- **`acospi`**, **`asinpi`**, **`atanpi`**, **`atan2pi`**,
- **`cospi`**, **`sinpi`**, **`tanpi`**

with the "overloads" **`sinpif`**, **`sinpil`**, (and **`sinpid32`**, **`sinpid64`**, **`sinpid128`**)

(not implemented in GCC 12.2 & Clang 15.0 on godbolt.org)

Example 7.12.4.11 The atan2pi functions

Synopsis

```
#include <math.h>
double atan2pi(double y, double x);
float atan2pif(float y, float x);
long double atan2pil(long double y, long double x);
#ifdef __STDC_IEC_60559_DFP__
...
#endif
```

Description

The **atan2pi** functions compute the angle, measured in half-revolutions, subtended at the origin by the point (**x**,**y**) and the positive x-axis. Thus, the **atan2pi** functions compute `arctan( x/y )/π`, in the range `[-1,+1]`. A domain error may occur if both arguments are zero. A range error occurs if **x** is positive and nonzero `x/y` is too close to zero.

Returns

The **atan2pi** functions return the computed angle, in the interval `[-1,+1]`.

Summery of the history

- C++ depends on the headers and functions from C
- The C++ standard descripe these part only in a very reduced form.
- The reader needs to be familar with "ISO/IEC 9899:2018 Programming languages — C" and maybe also "ISO/IEC/IEEE 60559:2020" / "IEEE 754 (IEEE Standard for Floating-Point Arithmetic)"
- Over time, C++ gets better to keepup with updates in C.
- Later version of C++ makes misuse harder
- `float`: use always std::math_function or math_function**f**
- Since C is a living languages, C language updates can influence C++ programs.

CMath content (1/x)

Types

- float_t // most efficient floating-point type at least as wide as float
- double_t // most efficient floating-point type at least as wide as double

Basic operations	Exponential functions	Power functions
abs	exp	pow
fabs	log	sqrt
fmod	log10	cbrt
remainder	exp2	hypot
remquo	expm1	
fma	log2	
fmax	log1p	
fmin		
fdim		
nan		

CMath content (2/x)

Trigonometric functions	Hyperbolic functions	Error and gamma functions
sin	sinh	erf
cos	cosh	erfc
tan	tanh	tgamma
asin	asinh	lgamma
acos	acosh	
atan	atanh	
atan2		

CMath content (3/x)

Nearest integer	Floating point manipulation functions	Classification and comparison
ceil	frexp	fpclassify
floor	ldexp	isfinite
trunc	modf	isinf
round	scalbn	isnan
lround	scalbln	isnormal
llround	ilogb	signbit
nearbyint	logb	isgreater
rint	nextafter	isgreaterequal
lrint	nexttoward	isless
llrint	copysign	islessequal
		islessgreater
		isunordered

Pitfalls

```
double result = sqrt(x) + asin(y) + u / v;
if (std::isnan(result))
    std::cout << "invalid result\n";
else
    std::cout << "result: " << result << "\n";
```

Is the check for **`isnan`** enough?

Pitfalls

```
double result = sqrt(x) + asin(y) + u / v;
if (!std::isfinite(result))
    std::cout << "invalid result\n";
else
    std::cout << "result: " << result << "\n";
```

check	purpose	others
isnan	check for NaN	false for Inf, -Inf
isinf	checks for Inf and -Inf	false for NaN
isnormal	checks for floating-point is normal	false for null / zero, subnormal (small numbers), NaN, Inf, ...
isfinite	checks for normal, subnormal, zero	false for NaN or Inf

some advice

- prefer C++ `<cmath>` over C `<math.h>`
  (Example: comparison function vs. comparison function)
- use latest C++ version (at least C++17)
- if you want to use ...f- and ...l-functions: write a conform portabilltiy header.

physicist view (code should look like a math formular)

- always include `<cmath>`
- write `using namespace std;` below the includes

C++ programmer

- explict specify functions with the namespace std. (Always!)
- RTFM (all of them)
- make yourset familar with the floating-point datatyps

What's missing

- Error handling
  - math_errhandling, MATH_ERRNO
- what's new in C99 and similar
- Floating-point environment

Not part of this presentation

- similar set of functions for `std::complex` and `std::valarray`