`constexpr` for `<cmath>` and `<cstdlib>`

Document: P0533R0
Date: January 25, 2017
Project: Programming Language C++, Library Working Group
Audience: SG6, LEWG
Reply to: Edward J. Rosten / Oliver J. Rosten <`forename.rosten@gmail.com`>

Abstract

We propose simple criteria for selecting functions in `<cmath>` which should be declared `constexpr`. There is a small degree of overlap with `<cstdlib>`. The aim is to transparently select a sufficiently large portion of `<cmath>` in order to be useful but without placing too much burden on compiler vendors.

CONTENTS

I. INTRODUCTION

This paper seeks to rectify the current absence of `constexpr` in `<cmath>` (and also in `<cstdlib>`), so as to broaden the range of numeric computations that can be performed using standard library facilities. While in principle almost every function in `<cmath>` could be declared `constexpr`, we strike a balance between coverage and onus on compiler/library vendors.

II. MOTIVATION & SCOPE

The introduction of `constexpr` has facilitated intuitive compile-time programming. However, not a single function in `<cmath>` is currently declared `constexpr`, thereby artificially restricting what can be done at compile-time within the standard library. Nevertheless, from casual inspection of `<cmath>`, it may not be immediately obvious precisely which functions should be declared `constexpr`. In this paper, we seek an organizing principle which selects functions which are in a sense no more complicated than the elementary arithmetic operations $(+, -, \times, /)$. This is justified since the latter already support `constexpr`.

To put the application of `constexpr` on a rigorous footing, we propose the following:

Proposal. *A function in* `<cmath>` *shall be declared* `constexpr` *if:*

1. *When taken to act on the set of extended rational numbers, the function is closed;*
2. *The function does not modify any of its arguments which have external visibility.*

By means of a brief illustration, `abs` satisfies both criteria; however, functions such as `exp`, `sqrt`, `cos`, `sin` fall foul of the first criterion and so are excluded as `constexpr` candidates.

It may seem that these criteria are insufficiently restrictive since various functions satisfying both may, under certain conditions, set global flags. Specifically, `errno` may be set and/or the various floating-point exception flags, `FE_DIVBYZERO`, `FE_INVALID`, `FE_OVERFLOW`, `FE_UNDERFLOW` and `FE_INEXACT` may be raised. The latter issue is faced by the standard arithmetic operators but these are nevertheless available for use in constant expressions. The proposed strategy is to mimic the behaviour of the arithmetic operators.

To be precise, functions declared `constexpr`, *when used in a* `constexpr` *context*, should give a compiler error if division by zero, domain errors or overflows occur. When not used in a `constexpr` context, the various global flags should be set as normal. This distinction between these two contexts implies that any implementation cannot be done as a pure library extension. Consequently, there will be some burden on compiler vendors. However, the first criterion above ensures that the functions to be changed are in some sense simple.

III. STATE OF THE ART

Both GCC and clang already support `constexpr` within `<cmath>` to varying extents. Indeed, GCC 5.3.0 declares all functions, with the exception of those taking a pointer argument (cf. the second criterion), as `constexpr`. Therefore, an implementation of the changes to the standard proposed in this paper is available (indeed, this implementation goes beyond what we propose). While clang does not go nearly as far as GCC, it does offer some functions as builtins and is able to use them to perform compile time computations, constant propagation and so on. It is therefore hoped that any burden on compiler vendors implicit in this proposal is minimal.

IV. IMPACT ON THE STANDARD

This proposal impacts the existing headers `<cmath>` and `<cstdlib>` such that the changes do not break existing code and do not degrade performance. However, it is not a pure library extension. The subtlety arises because affected functions must report errors by compilation failure in `constexpr` contexts instead of writing an error code into a global variable.

V. DESIGN DECISIONS

There is at least one natural candidate in `<cmath>`, namely `abs`, to which `constexpr` should be applied, not least because this has already been proposed for `complex::abs` [P0415R0]. But beyond this function, it is desirable to apply `constexpr` throughout `<cmath>` in a consistent manner. For this purpose, ideally one would like one or more criteria rooted in mathematics. However, the reality is that this is insufficiently restrictive. Nevertheless, mathematics was our starting point; as such, any condition on functions to be `constexpr` must select the basic arithmetic operations, $(+, -, \times, /)$, since these are already `constexpr`.

Mathematically, a field is closed under the elementary operations of addition and multiplication. Numeric types do not form a field; however, since the basic arithmetic operations are already declared `constexpr`, this suggests that it may be possible to utilize a field which captures enough of the properties of numeric types in order to be useful in formulating criteria for the application of `constexpr`. The set of rational numbers is the natural candidate since all valid values of numeric types are elements of this set and, moreover, the rationals close over $(+, -, \times, /)$ (with zero excluded for division).

The subtlety of global flags being set upon encountering floating-point exceptions presents a challenge. If all functions which can set such flags are excluded from the list to tag `constexpr`, then the remaining list is rather sparse. To achieve something more useful suggests expanding the set to include those functions which are 'simple enough'. These considerations lead to the first condition of the proposal. Tables II–V contain the functions in `<cmath>` satisfying this criterion and indicate whether or not they pass the second criterion as well.

To reduce space, the following convention is observed. The functions listed in [c.math] are divided into blocks of closely related functions such as those shown in table I. Note that while the first three functions are overloads, the fourth and fifth have differing names. When classifying those functions which satisfy the first criterion, we will present just the first function in each such block, with the understanding that the others are similar in this regard. Furthermore, we supply various comments in the third column of the tables, observing the following shorthands:

```
int ilogb(float arg)
int ilogb(double arg)
int ilogb(long double arg)
int ilogbf(float arg)
int ilogbl(long double arg)
```

TABLE I. Example of a family of functions which appear as a block in the standard.

1. G: May set global variable;
2. V: Modifies argument with external visibility.

Function	Pass	Comment
`float frexp(float value, int* exp)`	No	V
`int ilogb(float arg)`	Yes	G
`float ldexp(float x, int exp)`	Yes	G
`float logb(float arg)`	Yes	G
`float modf(float value, float* iptr)`	No	V
`float scalbn(float x, int n)`	Yes	G
`float scalbln(float x, long int n)`	Yes	G

TABLE II. Various functions declared in [cmath.syn] which close on the rationals.

Function	Pass
`int abs(int j)`	Yes
`float fabs(float x)`	Yes

TABLE III. Absolute values declared in [cmath.syn] which close on the rationals.

ACKNOWLEDGMENTS

We would like to thank Walter E. Brown, Daniel Krügler and Antony Polukhin for encouragement and advice.

REFERENCES

[P0415R0] Antony Polukhin, Constexpr for std::complex.
[N4618] Richard Smith, ed., Working Draft, Standard for Programming Language C++.

Function	Pass	Comment
`float ceil(float x)`	Yes	G⋆
`float floor(float x)`	Yes	G⋆
`float nearbyint(float x)`	Yes	
`float rint(float x)`	Yes	G
`long int lrint(float x)`	Yes	G
`long long int llrint(float x)`	Yes	G
`float round(float x)`	Yes	G
`float lround(float x)`	Yes	G
`float llround(float x)`	Yes	G
`float trunc(float x)`	Yes	G
`float fmod(float x, float y)`	Yes	G
`float remainder(float x, float y`	Yes	G
`float remquo(float x, float y, int* quo)`	No	V
`float copysign(float x, float y)`	Yes	
`float nextafter(float x, float y)`	Yes	G
`float nexttoward(float x, long double y)`	Yes	G
`float fdim(float x, float y)`	Yes	G
`float fmax(float x, float y)`	Yes	
`float fmin(float x, float y)`	Yes	
`float fma(float x, float y, float z)`	Yes	G

TABLE IV. Additional functions declared in [cmath.syn] which close on the rationals. ⋆ — implementation dependent.

Function	Pass	Comment
`int fpclassify(float x);`	Yes	
`int isfinite(float x)`	Yes	
`int isinf(float x)`	Yes	†
`int isnan(float x)`	Yes	†
`int isnormal(float x)`	Yes	
`int signbit(float x)`	Yes	
`int isgreater(float x, float y)`	Yes	
`int isgreaterequal(float x, float y)`	Yes	
`int isless(float x, float y)`	Yes	
`int islessequal(float x, float y)`	Yes	
`int islessgreater(float x, float y)`	Yes	
`int isunordered(float x, float y)`	Yes	

TABLE V. Comparison operators belonging to [cmath.syn] which close on the rationals. † — no utility being declared `constexpr` in of itself, but should be tagged `constexpr` so that it can be incorporated into `constexpr` functions since the latter may be called in non-`constexpr` contexts.

VI. PROPOSED WORDING

The following proposed changes refer to the Working Paper [N4618].

A. Modifications to "Header `<cstdlib>` synopsis" [cstdlib.syn]

```
namespace std{

...

constexpr int abs(int j);
constexpr long int abs(long int j);
constexpr long long int abs(long long int j);
constexpr float abs(float j);
constexpr double abs(double j);
constexpr long double abs(long double j);

constexpr long int labs(long int j);
constexpr long long int llabs(long long int j);

constexpr div_t div(int numer, int denom);
constexpr ldiv_t div(long int numer, long int denom); // see [library.c]
constexpr lldiv_t div(long long int numer, long long int denom); // see [library.c]
constexpr ldiv_t ldiv(long int numer, long int denom);
constexpr lldiv_t lldiv(long long int numer, long long int denom);
}
```

B. Modifications to “Header `<cmath>` synopsis” [cmath.syn]

```
...

namespace std{

...

float acos(float x); // see [library.c]
double acos(double x);
long double acos(long double x); // see [library.c]
float acosf(float x);
long double acosl(long double x);

...

float frexp(float value, int* exp); // see [library.c]
double frexp(double value, int* exp);
long double frexp(long double value, int* exp); // see [library.c]
float frexpf(float value, int* exp);
long double frexpl(long double value, int* exp);

constexpr int ilogb(float x); // see [library.c]
constexpr int ilogb(double x);
constexpr int ilogb(long double x); // see [library.c]
constexpr int ilogbf(float x);
constexpr int ilogbl(long double x);

constexpr float ldexp(float x, int exp); // see [library.c]
constexpr double ldexp(double x, int exp);
constexpr long double ldexp(long double x, int exp); // see [library.c]
constexpr float ldexpf(float x, int exp);
constexpr long double ldexpl(long double x, int exp);

float log(float x); // see [library.c]
double log(double x);
long double log(long double x); // see [library.c]
float logf(float x);
long double logl(long double x);

float log10(float x); // see [library.c]
double log10(double x);
long double log10(long double x); // see [library.c]
float log10f(float x);
long double log10l(long double x);

float log1p(float x); // see [library.c]
double log1p(double x);
long double log1p(long double x); // see [library.c]
float log1pf(float x);
long double log1pl(long double x);

float log2(float x); // see [library.c]
double log2(double x);
long double log2(long double x); // see [library.c]
float log2f(float x);
long double log2l(long double x);

constexpr float logb(float x); // see [library.c]
constexpr double logb(double x);
constexpr long double logb(long double x); // see [library.c]
constexpr float logbf(float x);
constexpr long double logbl(long double x);

float modf(float value, float* iptr); // see [library.c]
double modf(double value, double* iptr);
long double modf(long double value, long double* iptr); // see [library.c]
float modff(float value, float* iptr);
long double modfl(long double value, long double* iptr);

constexpr float scalbn(float x, int n); // see [library.c]
constexpr double scalbn(double x, int n);
constexpr long double scalbn(long double x, int n); // see [library.c]
constexpr float scalbnf(float x, int n);
constexpr long double scalbnl(long double x, int n);

constexpr float scalbln(float x, long int n); // see [library.c]
constexpr double scalbln(double x, long int n);
constexpr long double scalbln(long double x, long int n); // see [library.c]
constexpr float scalblnf(float x, long int n);
constexpr long double scalblnl(long double x, long int n);

float cbrt(float x); // see [library.c]
double cbrt(double x);
long double cbrt(long double x); // see [library.c]
float cbrtf(float x);
long double cbrtl(long double x);

// [c.math.abs], absolute values
constexpr int abs(int j);
constexpr long int abs(long int j);
constexpr long long int abs(long long int j);
constexpr float abs(float j);
constexpr double abs(double j);
constexprlong double abs(long double j);

constexpr float fabs(float x); // see [library.c]
constexpr double fabs(double x);
constexpr long double fabs(long double x); // see [library.c]
constexpr float fabsf(float x);
constexpr long double fabsl(long double x);

float hypot(float x, float y); // see [library.c]
double hypot(double x, double y);
long double hypot(double x, double y); // see [library.c]
float hypotf(float x, float y);
long double hypotl(long double x, long double y);

// [c.math.hypot3], three-dimensional hypotenuse
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);

...

constexpr float ceil(float x); // see [library.c]
constexpr double ceil(double x);
constexpr long double ceil(long double x); // see [library.c]
constexpr float ceilf(float x);
constexpr long double ceill(long double x);

constexpr float floor(float x); // see [library.c]
constexpr double floor(double x);
constexpr long double floor(long double x); // see [library.c]
constexpr float floorf(float x);
constexpr long double floorl(long double x);

constexpr float nearbyint(float x); // see [library.c]
constexpr double nearbyint(double x);
constexpr long double nearbyint(long double x); // see [library.c]
constexpr float nearbyintf(float x);
constexpr long double nearbyintl(long double x);

constexpr float rint(float x); // see [library.c]
constexpr double rint(double x);
constexpr long double rint(long double x); // see [library.c]
constexpr float rintf(float x);
constexpr long double rintl(long double x);

constexpr long int lrint(float x); // see [library.c]
constexpr long int lrint(double x);
constexpr long int lrint(long double x); // see [library.c]
constexpr long int lrintf(float x);
constexpr long int lrintl(long double x);

constexpr long long int llrint(float x); // see [library.c]
constexpr long long int llrint(double x);
constexpr long long int llrint(long double x); // see [library.c]
constexpr long long int llrintf(float x);
constexpr long long int llrintl(long double x);

constexpr float round(float x); // see [library.c]
constexpr double round(double x);
constexpr long double round(long double x); // see [library.c]
constexpr float roundf(float x);
constexpr long double roundl(long double x);

constexpr long int lround(float x); // see [library.c]
constexpr long int lround(double x);
constexpr long int lround(long double x); // see [library.c]
constexpr long int lroundf(float x);
constexpr long int lroundl(long double x);

constexpr long long int llround(float x); // see [library.c]
constexpr long long int llround(double x);
constexpr long long int llround(long double x); // see [library.c]
constexpr long long int llroundf(float x);
constexpr long long int llroundl(long double x);

constexpr float trunc(float x); // see [library.c]
constexpr double trunc(double x);
constexpr long double trunc(long double x); // see [library.c]
constexpr float truncf(float x);
constexpr long double truncl(long double x);

constexpr float fmod(float x, float y); // see [library.c]
constexpr double fmod(double x, double y);
constexpr long double fmod(long double x, long double y); // see [library.c]
constexpr float fmodf(float x, float y);
constexpr long double fmodl(long double x, long double y);

constexpr float remainder(float x, float y); // see [library.c]
constexpr double remainder(double x, double y);
constexpr long double remainder(long double x, long double y); // see [library.c]
constexpr float remainderf(float x, float y);
constexpr long double remainderl(long double x, long double y);

float remquo(float x, float y, int* quo); // see [library.c]
double remquo(double x, double y, int* quo);
long double remquo(long double x, long double y, int* quo); // see [library.c]
float remquof(float x, float y, int* quo);
long double remquol(long double x, long double y, int* quo);

constexpr float copysign(float x, float y); // see [library.c]
constexpr double copysign(double x, double y);
constexpr long double copysign(long double x, long double y); // see [library.c]
constexpr float copysignf(float x, float y);
constexpr long double copysignl(long double x, long double y);

double nan(const char* tagp);
float nanf(const char* tagp);
long double nanl(const char* tagp);

constexpr float nextafter(float x, float y); // see [library.c]
constexpr double nextafter(double x, double y);
constexpr long double nextafter(long double x, long double y); // see [library.c]
constexpr float nextafterf(float x, float y);
constexpr long double nextafterl(long double x, long double y);

constexpr float nexttoward(float x, long double y); // see [library.c]
constexpr double nexttoward(double x, long double y);
constexpr long double nexttoward(long double x, long double y); // see [library.c]
constexpr float nexttowardf(float x, long double y);
constexpr long double nexttowardl(long double x, long double y);

constexpr float fdim(float x, float y); // see [library.c]
constexpr double fdim(double x, double y);
constexpr long double fdim(long double x, long double y); // see [library.c]
constexpr float fdimf(float x, float y);
constexpr long double fdiml(long double x, long double y);

constexpr float fmax(float x, float y); // see [library.c]
constexpr double fmax(double x, double y);
constexpr long double fmax(long double x, long double y); // see [library.c]
constexpr float fmaxf(float x, float y);
constexpr long double fmaxl(long double x, long double y);

constexpr float fmin(float x, float y); // see [library.c]
constexpr double fmin(double x, double y);
constexpr long double fmin(long double x, long double y); // see [library.c]
constexpr float fminf(float x, float y);
constexpr long double fminl(long double x, long double y);

constexpr float fma(float x, float y, float z); // see [library.c]
constexpr double fma(double x, double y, double z);
constexpr long double fma(long double x, long double y, long double z); // see [library.c]
constexpr float fmaf(float x, float y, float z);
constexpr long double fmal(long double x, long double y, long double z);

// [c.math.fpclass], classification / comparison functions:
constexpr int fpclassify(float x);
constexpr int fpclassify(double x);
constexpr int fpclassify(long double x);
constexpr int isfinite(float x);
constexpr int isfinite(double x);
constexpr int isfinite(long double x);

constexpr int isinf(float x);
constexpr int isinf(double x);
constexpr int isinf(long double x);

constexpr int isnan(float x);
constexpr int isnan(double x);
constexpr int isnan(long double x);

constexpr int isnormal(float x);
constexpr int isnormal(double x);
constexpr int isnormal(long double x);

constexpr int signbit(float x);
constexpr int signbit(double x);
constexpr int signbit(long double x);

constexpr int isgreater(float x, float y);
constexpr int isgreater(double x, double y);
constexpr int isgreater(long double x, long double y);

constexpr int isgreaterequal(float x, float y);
constexpr int isgreaterequal(double x, double y);
constexpr int isgreaterequal(long double x, long double y);

constexpr int isless(float x, float y);
constexpr int isless(double x, double y);
constexpr int isless(long double x, long double y);

constexpr int islessequal(float x, float y);
constexpr int islessequal(double x, double y);
constexpr int islessequal(long double x, long double y);

constexpr int islessgreater(float x, float y);
constexpr int islessgreater(double x, double y);
constexpr int islessgreater(long double x, long double y);

constexpr int isunordered(float x, float y);
constexpr int isunordered(double x, double y);
constexpr int isunordered(long double x, long double y);
```

C. Modifications to “Absolute Values” [c.math.abs]

...

```
constexpr int abs(int j);
constexpr long int abs(long int j);
constexpr long long int abs(long long int j);
constexpr float abs(float j);
constexpr double abs(double j);
constexpr long double abs(long double j);
```