Type-Generic Math Functions

WG14/N639 X3J11/97-002 (Draft 1997-01-03)

Jim Thomas
jthomas@best.com

Synopsis

Type-generic macros allow calling a function whose type is determined by the argument type, as is the case for C operators such as `+` and `*`. For example, with a type-generic `cos` macro, the expression `cos((float)x)` will have type `float`. This feature enables writing more portably efficient code, and alleviates need for awkward casting and suffixing in the process of porting or adjusting precision. This is a proposal for type-generic macros for most of the functions in `<math.h>` and `<complex.h>`. The proposed header `<tgmath.h>` includes the headers `<math.h>` and `<complex.h>` and extends them by defining the type-generic macros.

Background

As recent as December 1995, the floating-point and complex arithmetic specifications included type-generic macros, under the name of overloading macros. Then, in response to the wishes of the C committees, these specifications were redrafted removing the overloading macros, with the intention of later considering them as a separate feature.

At the Amsterdam meeting, the committees accepted in principle the floating-point and complex arithmetic specifications (sans overloading) into the C9X draft; and they requested a newly written proposal for the extracted facility, but under a different name than overloading. Such a proposal (N618) was presented as a Monday paper and discussed in detail at the Toronto meeting.

This document incorporates the input from Toronto, and is intended as a proposal to be voted on (for inclusion into C9X) at the Kona meeting.

Proposal

Rationale given here supplements the rationale in the earlier specification for overloading, and is not intended as a complete rationale for the feature.

In 7, insert this subclause:

7.x Type-generic math <tgmath.h>

The header `<tgmath.h>` includes the headers `<math.h>` and `<complex.h>` and defines several *type-generic macros*.

> [The term *type-generic* was chosen as more specific than the suggested term *intrinsic*, which already is widely used with a more general meaning. The X3J11 Vocabulary Representative concurs with this choice.

The macros are placed in their own header in order not to silently break old programs that include **<math.h>**, for example with **printf("%e", sin(x));** .

The name **tgmath.h** is shorter than the previous **generic_math.h**, which the committee thought was too long.]

7.x.1 Type-generic macros

Of the **<math.h>** and **<complex.h>** functions without an **f** (**float**) or **l** (**long double**) suffix, several have one or more parameters whose corresponding real type is **double** and return a result whose corresponding real type is **double**. For each such function, except for **modf** and **nextafterd**, there is a corresponding *type-generic macro*[1]. The parameters whose corresponding real type is **double** in the function synopsis are *generic parameters*. Use of the macro invokes a function whose corresponding real type and type-domain are determined by the arguments for the generic parameters.

[**modf(double, double*)** is excluded because no way was seen to make it safe without complicating the type resolution.]

Use of the macro invokes a function whose corresponding real type is the wider of the corresponding real types of floating arguments for generic parameters, with integral-type arguments for generic parameters first being converted to **double**.

[This differs from the earlier overloading proposal in that the type is determined solely by the argument, and may be narrower than the type for expression evaluation. This change was made because the performance costs for computing functions with narrow arguments to wide range and precision might be too high, even if the implementation efficiently evaluates basic operations to wide precision.

Also, this differs from earlier proposals in that integral-type arguments are converted to **double** instead of **float**. Although converting to **float** would have been more consistent with the usual arithmetic conversions, converting to **double** has the advantages of preserving the value more often on many systems, and of being more compatible with C90 where unsuffixed calls to math functions with integer arguments were calls to **double** functions.]

For each unsuffixed function in **<math.h>** for which there is a function in **<complex.h>** with the same name except for a **c** prefix, the corresponding type-generic macro (for both functions) has the same name as the function in **<math.h>**. The corresponding type-generic macro for **fabs** and **cabs** is **fabs**.

[1] Like other function-like macros in the Standard libraries, each type-generic macro can be suppressed to make available the corresponding ordinary function.

<math.h> function	**<complex.h>** function	type-generic macro
acos	**cacos**	**acos**
asin	**casin**	**asin**
atan	**catan**	**atan**
acosh	**cacosh**	**acosh**
asinh	**casinh**	**asinh**
atanh	**catanh**	**atanh**
cos	**ccos**	**cos**
sin	**csin**	**sin**
tan	**ctan**	**tan**
cosh	**ccosh**	**cosh**
sinh	**csinh**	**sinh**
tanh	**ctanh**	**tanh**
exp	**cexp**	**exp**
log	**clog**	**log**
pow	**cpow**	**pow**
sqrt	**csqrt**	**sqrt**
fabs	**cabs**	**fabs**

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, use of the macro invokes a real function.

[Having a **g** suffix for the generic macros was considered but thought unnecessary.]

For each unsuffixed function in **<math.h>** without a **c**-prefixed counter-part in **<complex.h>**, the corresponding type-generic macro has the same name as the function. These type-generic macros are

atan2	**hypot**	**round**
exp2	**erf**	**trunc**
expm1	**erfc**	**remainder**
frexp	**gamma**	**remquo**
ldexp	**lgamma**	**copysign**
log10	**ceil**	**nextafter**
log1p	**floor**	**fdim**
log2	**fmod**	**fmax**
logb	**nearbyint**	**fmin**
scalb	**rint**	

If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro results in undefined behavior.

[The implementation might, as an extension, endow appropriate ones of these macros with the ability to invoke the complex functions whose names are reserved in 7.14.4.]

For each unsuffixed function in **<complex.h>** that is not a **c**-prefixed counter-part to a function in **<math.h>**, the corresponding type-generic macro has the same name as the function. These type-generic macros are

carg	**cimag**	**creal**
conj	**cproj**	

Use of the macro with any real or complex arguments invokes a **<complex.h>** function.

Examples

With the declarations

```
#include <tgmath.h>
int n;
float f;
double d;
long double ld;
float complex fc;
double complex dc;
long double complex ldc;
```

functions invoked by use of type-generic macros are shown in the table:

macro use	invokes
`exp(n)`	`exp(n)`, the function
`acosh(f)`	`acoshf(f)`
`sin(d)`	`sin(d)`, the function
`atan(ld)`	`atanl(ld)`
`log(fc)`	`clogf(fc)`
`sqrt(dc)`	`csqrt(dc)`
`pow(ldc, f)`	`cpowl(ldc, f)`
`remainder(n, n)`	`remainder(n, n)`, the function
`nextafter(d, f)`	`nextafter(d, f)`, the function
`copysign(n, ld)`	`copysignl(n, ld)`
`ceil(fc)`	undefined behavior
`rint(dc)`	undefined behavior
`fmax(ldc, ld)`	undefined behavior
`carg(n)`	`carg(n)`, the function
`cproj(f)`	`cprojf(f)`
`creal(d)`	`creal(d)`, the function
`cimag(ld)`	`cimagl(ld)`
`cabs(fc)`	`cabsf(fc)`
`carg(dc)`	`carg(dc)`, the function
`cproj(ldc)`	`cprojl(ldc)`

Replace the subclause of 7 specifying the nextafter functions with (leaving footnote 14 as is):

[This is essentially the **`nextafter`** spec prior to the extraction of overloading.]

7.x.11.3 The `nextafter` functions

Synopsis

```
#include <math.h>
double nextafter(double x, long double y);
float nextafterf(float x, float y);
double nextafterd(double x, double y);
long double nextafterl(long double x, long double y);
```

Description

The **nextafter** function determines the next representable value, in the type of the function, after **x** in the direction of **y**, where x and y are first converted to the type of the function.[14] The **nextafter** function returns **y** if **x == y**.

The functions **nextafterf**, **nextafterd**, and **nextafterl** each is similar to the **nextafter** function except the type for both parameters and the returned value is **float**, **double**, or **long double**, respectively.

Returns

The functions **nextafter**, **nextafterf**, **nextafterd**, and **nextafterl** each returns the next representable value in the specified format after **x** in the direction of **y**.

In the Annex Y: IEC 559-Compatible Complex Arithmetic, append the subclause:

Y.7 <tgmath.h>

Type-generic macros that accept complex arguments also accept imaginary arguments. If an argument is imaginary, the macro expands to an expression whose type is real, imaginary, or complex, as appropriate for the particular function: if the argument is imaginary, then the types of **cos**, **cosh**, **fabs**, **carg**, **cimag**, and **creal** are real; the types of **sin**, **tan**, **sinh**, **tanh**, **asin**, **atan**, **asinh**, and **atanh** are imaginary; and the types of the others are complex.

Given an imaginary argument, each of the type-generic macros **cos**, **sin**, **tan**, **cosh**, **sinh**, **tanh**, **asin**, **atan**, **asinh**, and **atanh** is specified by a formula in terms of real functions:

```
cos(i*y)  =  cosh(y)
sin(i*y)  =  i*sinh(y)
tan(i*y)  =  i*tanh(y)
cosh(i*y) =  cos(y)
sinh(i*y) =  i*sin(y)
tanh(i*y) =  i*tan(y)
asin(i*y) =  i*asinh(y)
atan(i*y) =  i*atanh(y)
asinh(i*y) = i*asin(y)
atanh(i*y) = i*atan(y)
```