Revised suggested TC for CFP DR 13

Submitter: C FP group
Submission Date: 2018-02-11
Source: WG14
Reference Document: N2202
Subject: Type-generic macros for functions that round result to narrower type

Summary

Joseph Myers has pointed out problems with the suggested TC for CFP DR 13, regarding the specification in TS 18661-3 for type-generic macros for functions that round result to narrower type. While investigating, we discovered another problem. Email from Joseph Myers and discussion of all the problems follows below.

Joseph Myers <joseph@codesourcery.com>
(SC22WG14.14879) Floating-point DR#13 and integer arguments to type-generic macros
To: SC22 WG14 sc22wg14@open-std.org

> The proposed resolution to floating-point DR#13 (regarding type-generic macros for functions that round result to narrower type) includes
> "Arguments that have integer type are regarded as having type _Decimal64 if any argument has decimal floating type, and as having type double otherwise.".
>
> This runs into problems if all the arguments to a macro such as d32add are of integer type, because now they are being regarded as of type double, whereas in TS 18661-2 it was clear that it was valid to pass integer arguments to the d32add macro and it would result in d32addd64 being called (and passing such arguments to the d64add macro would result in d64addd128 being called). Is that intended - that these macros should not be valid with only integer type arguments? Or should the logic for what type integer arguments are considered to have be based on the macro prefix in this case?

CFP: The latter: the logic for what type integer arguments are considered to have should be based on the macro prefix.

> That DR resolution also appears to leave results not fully determined in the case of integer arguments. The chosen function is specified by "The unsuffixed name of the function is the name of the macro, and its suffix, if any, corresponds to the parameter type which may be any type with at least the range and precision of the argument types.", but whereas for floating-point arguments the result of the call does not depend on exactly which function gets called, as long as the parameter type has enough range and precision, for integer arguments it *does* matter whether (for example) a call of f32add with two long long arguments ends up calling f32addf64 or f32addf128, because loss of precision when converting such arguments to _Float64.
>
> Possibilities for the not-fully-determined result include: allow it being not fully determined (and maybe add this case to the list in part 5 of the TS of features that

> prevent reproducibility); require the integer arguments to be converted to the type they are considered to have; put in some rules that determine the type more precisely in the case of integer arguments; apply the "any type with at least the range and precision of the argument types" to the original integer types rather than to _Decimal64 / double (which would imply that e.g. calling fadd / dadd with long long arguments is not valid if long double is IEEE binary64, because then long double wouldn't be able to represent all long long values). (Even if the type can represent all values of the integer argument type, you still have issues of decimal exponents depending on what the chosen type is, but that may be less significant.)
>
> --
> Joseph S. Myers
> joseph@codesourcery.com

Joseph Myers <joseph@codesourcery.com>
(SC22WG14.14880) Floating-point DR#13 and integer arguments to type-generic macros
To: SC22 WG14 sc22wg14@open-std.org

On Mon, 6 Nov 2017, Joseph Myers wrote:

> *TS of features that prevent reproducibility); require the integer arguments to be converted to the type they are considered to have; put in*
>
> (Or to the common type of the arguments determined as in DR#9, to make these macros as similar as possible to the other type-generic macros, so that the case of (long long, long long) arguments would convert them to double, or maybe to _Decimal64 for a decimal type-generic macro, but (long long, long double) would convert the long long argument to long double.)

CFP: We agree this is the right approach.

> --
> Joseph S. Myers
> joseph@codesourcery.com

There's another problem here. We say that the function prefix is the same as the macro prefix and we determine the type for generic parameters from the argument types. It may happen that the prefix is for a standard floating type and the parameter type is an interchange or extended floating type, or vice versa. In these cases there is no such function. A simple example is the macro invocation **`fadd(x, y)`** where **`x`** and **`y`** are **`_Float64`**.

To address this problem, we can say that such cases result in undefined behavior. In the example above, the user could invoke **`fadd((double)x, (double)y)`** or **`(float)f32add(x, y)`**, but not **`fadd(x, y)`**. We didn't see a portable way to get the effect of such as **`fadd(x, y)`** if **`x`** is **`_Decimal32x`** and **`y`** is **`long double`**, where the determined parameter type group will differ among implementations.

Alternatively, we considered adding functions to cover all the cases, but the large number of functions required didn't seem to justify the added utility.

We also considered a scheme where **`float`** and **`double`** are regarded as **`_Float32`** and **`_Float64`** to avoid type group mismatches, but it became exceedingly complicated and **`long double`** was still a problem.

The following is a suggested TC to replace the one in DR 13. The change addresses all the problems discussed above.

Suggested Technical Corrigendum

In clause 15, after the change to 7.25#6, add:

Change 7.25#6a from:

> [6a] The functions that round result to a narrower type have type-generic macros whose names are obtained by omitting any suffix from the function names. Thus, the macros with **f** or **d** prefix are:
>
> **fadd fmul ffma**
> **dadd dmul dfma**
> **fsub fdiv fsqrt**
> **dsub ddiv dsqrt**
>
> and the macros with **d32** or **d64** prefix are:
>
> **d32add d32mul d32fma**
> **d64add d64mul d64fma**
> **d32sub d32div d32sqrt**
> **d64sub d64div d64sqrt**
>
> All arguments are generic. If any argument is not real, use of the macro results in undefined behavior. If the macro prefix is **f** or **d**, use of an argument of decimal floating type results in undefined behavior. If the macro prefix is **d32** or **d64**, use of an argument of standard floating type results in undefined behavior. The function invoked is determined as follows:
>
> — If any argument has type **_Decimal128**, or if the macro prefix is **d64**, the function invoked has the name of the macro, with a **d128** suffix.
>
> — Otherwise, if the macro prefix is **d32**, the function invoked has the name of the macro, with a **d64** suffix.
>
> — Otherwise, if any argument has type **long double**, or if the macro prefix is **d**, the function invoked has the name of the macro, with an **l** suffix.

— Otherwise, the function invoked has the name of the macro (with no suffix).

to:

[6a] The functions that round result to a narrower type have type-generic macros whose names are obtained by omitting any suffix from the function names. Thus, the macros with **f** or **d** prefix are:

fadd	**fmul**	**ffma**
dadd	**dmul**	**dfma**
fsub	**fdiv**	**fsqrt**
dsub	**ddiv**	**dsqrt**

and the macros with **f***M*, **f***M***x**, **d***M*, or **d***M***x** prefix are:

f*M***add**	**f***M***xmul**	**d***M***fma**
f*M***sub**	**f***M***xdiv**	**d***M***sqrt**
f*M***mul**	**f***M***xfma**	**d***M***xadd**
f*M***div**	**f***M***xsqrt**	**d***M***xsub**
f*M***fma**	**d***M***add**	**d***M***xmul**
f*M***sqrt**	**d***M***sub**	**d***M***xdiv**
f*M***xadd**	**d***M***mul**	**d***M***xfma**
f*M***xsub**	**d***M***div**	**d***M***xsqrt**

All arguments are generic. If any argument is not real, use of the macro results in undefined behavior. If the macro prefix is **f** or **d**, use of an argument of interchange or extended floating type results in undefined behavior. If the macro prefix is **f***M*, or **f***M***x**, use of an argument of standard or decimal floating type results in undefined behavior. If the macro prefix is **d**M or **d***M***x**, use of an argument of standard or binary floating type results in undefined behavior. The function invoked is determined as follows:

— Arguments that have integer type are regarded as having type **double** if the macro prefix is **f** or **d**, as having type **_Float64** if the macro prefix is **f***M* or **f***M***x**, and as having type **_Decimal64** if the macro prefix is **d**M or **d***M***x**.

— If the function has exactly one generic parameter, the type determined is the type of the argument.

— If the function has exactly two generic parameters, the type determined is the type determined by the usual arithmetic conversions (6.3.1.8) applied to the arguments.

— If the function has three generic parameters, the type determined is the

type determined by applying the usual arithmetic conversions twice, first to the first two arguments, then to that result type and the third argument.

In clause 15, at the end of the text appended to the table in 7.25#7, further append:

fsub(d, ld)	**fsubl**
f32add(f64x, f64)	**f32addf64x**
d32xsqrt(n)	**d32xsqrtd64**
f32mul(f128, f32x)	**f32mulf128** if **_Float128** is at least as wide as **_Float32x**, or **f32mulf32x** if **_Float32x** is wider than **_Float128**
f32fma(f32x, n, f32x)	**f32fmaf64** if **_Float64** is at least as wide as **_Float32x**, or **f32fmaf32x** if **_Float32x** is wider than **_Float64**
ddiv(ld, f128)	undefined
f32fma(f64, d, f64)	undefined
fmul(dc, d)	undefined