Proposal for C2Y
WG14 N3843

Title:	Conflicting requirements for double_t
Author, affiliation:	C FP group
Date:	2026-02-26
Proposal category:	Technical
Reference:	N3983

Background

In [SC22WG14.34201] "FLT_EVAL_METHOD, double_t and Annex H", Joseph Myers pointed out contradictory specification for `double_t`: when `double` and `long double` formats are the same and `FLT_EVAL_METHOD` is 2, the main standard says `double_t` is defined to be `long double`, but Annex H implies `double_t` is defined to be `double`. The relevant sections are:

7.12.1 #3

> ... if `FLT_EVAL_METHOD` equals 2, they [referring to `float_t` and `double_t`] are both `long double`; ...

and

H.11.1 #6

> If `FLT_RADIX` is 2 and `FLT_EVAL_METHOD` (H.3) is nonnegative, then each of the types [referring to the `_t` types] corresponding to a standard or binary floating type is the type whose range and precision are specified by `FLT_EVAL_METHOD` to be used for evaluating operations and literals of that standard or binary floating type.

which refers to

H.3 #2

> [if `FLT_EVAL_METHOD` is 2] evaluate operations and literals, whose semantic type comprises a set of values that is a strict subset of the values of `long double`, to the range and precision of `long double`; evaluate all other operations and literals to the range and precision of the semantic type;

Thus if `double` and `long double` have the same values, Annex H specifies `double_t` to be `double`, contradicting 7.12.1 #3.

As noted in Joseph's message, TS 18661-3 (the base document for Annex H) says

> [if `FLT_EVAL_METHOD` is 2] evaluate operations and constants, whose semantic type has at most the range and precision of `long double`, to the range and precision of `long double`; ...

If `double` and `long double` have the same values, `double` has (at most) the range and precision of `long double`. Thus TS 18661-3 specifies `double_t` to be `long double`, which is consistent with 7.12.1 #3.

Going from TS 18661-3 to Annex H, we effectively modified the criterion for evaluating to the special evaluation type determined by `FLT_EVAL_METHOD` (instead of to the semantic type):

- TS 18661-3 says use the special evaluation type if the values of the semantic type are a subset of (or equivalent to) the values of the evaluation type.

- Annex H says use the special evaluation type if the values of the semantic type are a strict (proper) subset of the values of the evaluation type.

The only cases affected by the criterion modification are where the semantic type of the operation has the same format as the special evaluation type.

The `_t` types are the types `FLT_EVAL_METHOD` specifies to be used for evaluating operations of the semantic type, so they were affected by the criterion modification. Thus, the conflict.

We believe the criterion modification was made without awareness of this problem. Discussion of rationale for the criterion modification is at the end.

The suggested changes below resolve the conflict. They modify the `FLT_EVAL_METHOD` specification in Annex H to be effectively equivalent to TS 18661-3. This makes Annex H a natural extension of the main standard: the practice of using the special evaluation format whenever its values include all the values of the semantic type is extended from to main standard to the types added in the annex.

The suggested changes require implementations that support C23 Annex H to change `_t` types in some cases. However, we expect the impact to be low. Annex H (with the problematic divergence from TS 18661-3) was just introduced in C23. Because the evaluation formats remain unchanged, we believe users would be unlikely to be affected.

We considered two other solutions which are mentioned at the end.

Suggested changes

In H.3 #2:

From:

…

0 evaluate all operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of `float`, to the range and precision of `float`; evaluate all other operations and literals to the range and precision of the semantic type;

1 evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of `double`, to the range and

precision of `double`; evaluate all other operations and literals to the range and precision of the semantic type;

2 evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of `long double`, to the range and precision of `long double`; evaluate all other operations and literals to the range and precision of the semantic type;

N where `_Float`*N* is a supported interchange floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of `_Float`*N*, to the range and precision of `_Float`*N*; evaluate all other operations and literals to the range and precision of the semantic type;

N + 1 where `_Float`*N*`x` is a supported extended floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of `_Float`*N*`x`, to the range and precision of `_Float`*N*`x`; evaluate all other operations and literals to the range and precision of the semantic type.

To:

…

0 evaluate all operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of `float`, to the range and precision of `float`; evaluate all other operations and literals to the range and precision of the semantic type;

1 evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of `double`, to the range and precision of `double`; evaluate all other operations and literals to the range and precision of the semantic type;

2 evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of `long double`, to the range and precision of `long double`; evaluate all other operations and literals to the range and precision of the semantic type;

N where `_Float`*N* is a supported interchange floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of `_Float`*N*, to the range and precision of `_Float`*N*; evaluate all other operations and literals to the range and precision of the semantic type;

N + 1 where `_Float`*N*`x` is a supported extended floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of `_Float`*N*`x`, to the range and precision of

_Float*N*x; evaluate all other operations and literals to the range and precision of the semantic type.

In H.3 #2:

From:

...

1 evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of _Decimal64, to the range and precision of _Decimal64; evaluate all other operations and literals to the range and precision of the semantic type;

2 evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of _Decimal128, to the range and precision of _Decimal128; evaluate all other operations and literals to the range and precision of the semantic type;

N where _Decimal*N* is a supported interchange floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of _Decimal*N*, to the range and precision of _Decimal*N*; evaluate all other operations and literals to the range and precision of the semantic type;

N + 1 where _Decimal*N*x is a supported extended floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a ~~strict subset of~~ the values of _Decimal*N*x, to the range and precision of _Decimal*N*x; evaluate all other operations and literals to the range and precision of the semantic type.

To:

...

1 evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of _Decimal64, to the range and precision of _Decimal64; evaluate all other operations and literals to the range and precision of the semantic type;

2 evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of _Decimal128, to the range and precision of _Decimal128; evaluate all other operations and literals to the range and precision of the semantic type;

N where _Decimal*N* is a supported interchange floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of _Decimal*N*, to the range and

precision of _Decimal*N*; evaluate all other operations and literals to the range and precision of the semantic type;

N + 1 where _Decimal*N*x is a supported extended floating type, evaluate operations and literals, whose semantic type comprises a set of values that is a subset of (or equivalent to) the values of _Decimal*N*x, to the range and precision of _Decimal*N*x; evaluate all other operations and literals to the range and precision of the semantic type.

In Table H.7:

From:

	m			
_t type	0	1	2	32
_Float16_t	float	double	long double	_Float32
float_t	float	double	long double	~~float~~
_Float32_t	~~_Float32~~	double	long double	_Float32
double_t	double	double	long double	double
_Float64_t	_Float64	~~_Float64~~	long double	_Float64
long_double_t	long double	long double	long double	long double
_Float128_t	_Float128	_Float128	_Float128	_Float128

	m			
_t type	64	128	33	65
_Float16_t	_Float64	_Float128	_Float32x	_Float64x
float_t	_Float64	_Float128	_Float32x	_Float64x
_Float32_t	_Float64	_Float128	_Float32x	_Float64x
double_t	~~double~~	_Float128	~~double~~	_Float64x
_Float64_t	_Float64	_Float128	~~_Float64~~	_Float64x
long_double_t	long double	_Float128	long double	~~long double~~
_Float128_t	_Float128	_Float128	_Float128	_Float128

To:

	m			
_t type	0	1	2	32
_Float16_t	float	double	long double	_Float32
float_t	float	double	long double	_Float32
_Float32_t	float	double	long double	_Float32
double_t	double	double	long double	double

_Float64_t	_Float64	double	long double	_Float64
long_double_t	long double	long double	long double	long double
_Float128_t	_Float128	_Float128	_Float128	_Float128

_t type	*m*			
	64	128	33	65
_Float16_t	_Float64	_Float128	_Float32x	_Float64x
float_t	_Float64	_Float128	_Float32x	_Float64x
_Float32_t	_Float64	_Float128	_Float32x	_Float64x
double_t	_Float64	_Float128	_Float32x	_Float64x
_Float64_t	_Float64	_Float128	_Float32x	_Float64x
long_double_t	long double	_Float128	long double	_Float64x
_Float128_t	_Float128	_Float128	_Float128	_Float128

Discussion of rationale for the criterion modification from TS 18661-3 to Annex H

Why did we modify the criterion for evaluating to the special evaluation type determined by FLT_EVAL_METHOD. This CFP note suggested the change:

https://cfp-wiki.esi.com.au/pub/CFP/WebHome/Example_for_FLT_EVAL_METHOD-20190815.pdf

Rationale: Explicit use of a type might be expected to result in that type, unless the evaluation format is wider. E.g. if FLT_EVAL_METHOD is 0, one might expect (_Float32)x + (_Float32)y to have type _Float32 instead of float and _Float32_t to be _Float32 instead float.

A counter perspective (reflecting the historical hardware origins of wide evaluation) is that an evaluation type given by the evaluation method is expected to be effective unless the semantic type is wider.

Since Annex H requires both _Float32 and float to have the same format and arithmetic, the result value and side effects are the same regardless of the evaluation type. However, the definition of _Float32_t, which gives the evaluation type, is affected: _Float32 vs float.

Alternate solutions for addressing the conflict

Alternative 1. In Annex H, keep the general specification for FLT_EVAL_METHOD but treat the case of double and long double having the same format specially to match 7.12.1 #3.

Alternative 2. In the main standard, disallow `FLT_EVAL_METHOD` = 2 if `double` and `long double` have the same format and Annex H is supported.

Alternatives 1 and 2 entail adding special cases, further complicating the specification.

Alternatives 1 and 2, except for the one problematic case, preserve the criterion modification from TS 18661-3 to Annex H. The argument for the criterion modification (see discussion of rationale above) is speculative: there is no actual evidence to support it.

Alternative 1 could affect implementations though only if they manifested the incompatibility.

Alternative 2 could affect implementations if they have done what Alternative 2 disallows.