Floating-Point Items

WG14/N826 J11/98-025 (Draft 1998-05-29)

Jim Thomas
jthomas@best.com

This document proposes minor changes to the C9X floating-point arithmetic specification. Section numbers refer to C9X Working Draft N794.

1. NaN results: The current C9x specification requires that (quiet) NaN arguments generally pass through unchanged to results, i.e. the result NaN is the same NaN as the argument. IEC 559 requires that NaN arguments generally cause NaN results, but only recommends, and does not require, that NaN arguments pass through unchanged. A library with a stricter specification than the arithmetic wouldn't as a result be more useful; but it might be more difficult to implement, as the library code would have to test all inputs for NaNs if the arithmetic didn't preserve NaNs. *Proposal*: loosen the C9x specification to follow the approach in IEC 559, and specify strict NaN preservation only as recommended practice.

In F.9, replace [10] with:

Functions with a NaN argument return a NaN result and raise no exception, except where stated otherwise.

To F.9, append:

Recommended practice

If a function with a NaN argument returns a NaN result, then the result is the same as an argument NaN (converted to its parameter type), except perhaps for the sign.

In F.9.1.4 (atan2), remove the first bullet.

In F.9.3.4 (frexp), change the third bullet to:

- **`frexp(x, exp)`** stores an unspecified value in **`*exp`** (and returns a NaN) when **`x`** is a NaN.

In F.9.3.11 (modf), change the third bullet to:

- **`modf`** of a NaN argument stores a NaN in **`*iptr`** (and returns a NaN).

In F.9.3.12 (scalbn), remove "or a NaN" *from the first bullet.*

In F.9.4.2 (hypot), append to the second bullet ", even if **`y`** is a NaN", *and delete the third bullet.*

In F.9.4.3 (pow), append to the first bullet ", even a NaN", *and remove the twelfth bullet (about NaNs).*

In F.9.7.1 (fmod), remove the first bullet.

In F.9.8.3 (nextafter), remove the first bullet.

In F.9.9.1 (fdim), remove the first bullet.

In F.9.9.2 (fmax), change the first bullet to:

— If just one argument is a NaN then **`fmax`** returns the other argument (and if both arguments are NaNs then **`fmax`** returns a NaN).

In F.9.9.3 (fmin), change the first bullet to:

— If just one argument is a NaN then **`fmin`** returns the other argument (and if both arguments are NaNs then **`fmin`** returns a NaN).

In F.9.10.1 (fma), remove the second bullet.

2. Extension exception and rounding macros: The intention from the start has been to allow an implementation to add its own exception and rounding direction macros, with the **`FE_`** prefix. However, the words have been lost in the evolution of documents. UK Comment 130 highlighted the omission. *Proposal*: add words to the intended effect, following the model for the signal handling macros in 7.11.

In 7.6 (fenv.h), after the first sentence in [5], add the sentence:

The implementation may also specify additional supported floating-point exceptions, with macro definitions beginning with **`FE_`** and an upper case letter.

In 7.6 (fenv.h), after the first sentence in [7], add the sentence:

"The implementation may also specify additional supported rounding directions, with macro definitions beginning with **`FE_`** and an upper case letter."

3. `CX_LIMITED_RANGE` for `cabs`: The **`CX_LIMITED_RANGE`** macro currently applies to complex multiplication and division, allowing faster algorithms where the programmer can determine that extreme-range values need not be considered. The same issues pertain to the **`cabs`** function. Speed is important. **`cabs`** is problematic because of the product z*conj(z), hence the range considerations are commensurate with those for multiply. The ordinary mathematical formula for complex absolute value, like multiply and divide, fails the specification for infinities with a NaN part. *Proposal*: allow the ususal formula for **`cabs`** where the state for **`CX_LIMITED_RANGE`** is on.

In 7.8.1 (CX_LIMITED_RANGE), change the first two sentences to:

The usual mathematical formulas for complex multiply, divide, and absolute value are problematic because of their treatment of infinities and because of undue overflow and underflow. The **`CX_LIMITED_RANGE`** pragma can be used to inform the implementation that (where the state is on) the usual mathematical formulas for complex multiply, divide, and absolute value are acceptable.

In footnote 187, add the formula: cabs(x+y*i) = sqrt(x*x+y*y)

4. Single Unix: This item follows up on the incompatibilities between C9x and Single Unix pointed out in recent email. The incompatibilities in the treatment of errno were due to inconsistencies in the C9x draft and have been addressed through editorial changes. These remain:

1.1 Single UNIX requires **`hypot`** return a NaN whenever one of its arguments is a NaN. C9x (with **`__STDC_IEC_559__`**) requires **`hypot`** return infinity if one of its arguments is infinite, even if the other argument is a NaN. The principle behind the C9x specification is the following: if a function takes a certain value independent of the numerical value of a parameter, then the function takes the common value even if the value of the parameter is a NaN. This approach is more useful (i.e. gets a usable result for more data) than the more conservative approach of returning a NaN. Also, C9x defines **`cabs(x+I*y)`** to be equivalent to **`hypot(x,y)`**, and **`cabs`**'s treatment of infinities (including inf+iNaN and the like) is integral to the specification for complex special cases. *Proposal*: retain the current C9x specification and recommend that X/Open allow the C9x behavior.

1.2 Single UNIX requires **`pow(+0.0,y)`** and **`pow(-0.0,y)`** to be **`-HUGE_VAL`** if **`y`** is negative. C9x (with **`__STDC_IEC_559__`**) requires it be plus or minus infinity depending on **`y`** and the sign of zero, e.g. **`pow(+0.0, -2)`** and **`pow(-0.0, -2)`** return +inf, and **`pow(-0.0,-1)`** returns -inf. The C9x specification is a better match to mathematics, which says, for example, $(+0)^y$ and $(-0)^y$ are +inf if y is an even negative integer. (For **`__STDC_IEC_559__`**, **`HUGE_VAL`** is an infinity.) *Proposal*: retain the current C9x specification and recommend that X/Open allow the C9x behavior.

1.3 Single UNIX defines **`gamma`** to be equivalent to **`lgamma`** (log gamma). C9x defines it to be a true gamma function. (In draft versions of XPG4.2, **`gamma`** was marked to be withdrawn, but this was retracted in the final version and the misnomer persists.) The demand for a standard true gamma function may not be large enough to justify introducing the incompatibility. *Proposal*: remove the **`gamma`** function from the C9x draft.