WG14/N310
X3J11/93-057

Why Imaginary Types

X3J11/???, WG14/???

Jim Thomas
Taligent, Inc.
10201 N. DeAnza Blvd.
Cupertino, CA 95014-2233
jim_thomas@taligent.com

This note explains why it is important that C and C++ adopt a complex specification along the lines of "Complex C Extensions" (X3J11.1/93-048), CCE here for short, which includes imaginary types. Although a C specification, it was written with C++ compatibility in mind and has been prototyped in C++.

Imaginary types enable a clean solution to the problem of extending the treatment of special cases from the real to the complex domain. For IEEE implementations this means infinities, NaNs, and signed zeros in complex arithmetic can behave as one reasonably would expect based on their behavior in real arithmetic. A simple example illustrates the problem:

The mathematical product $2.0i * (\infty + 3.0i)$ should be computed with two multiplications and yield $-6.0 + \infty i$. But without an imaginary type, $2.0i$ would have to be represented as $0.0 + 2.0i$, so that

$$\begin{aligned} 2.0i * (\infty + 3.0i) &\Rightarrow (0.0 + 2.0i) * (\infty + 3.0i) \\ &\Rightarrow (0.0*\infty - 2.0*3.0) + (0.0*3.0 + 2.0*\infty)i \\ &\Rightarrow NaN + \infty i \end{aligned}$$

requiring four multiplications and two additions to obtain an undesirable result.

CCE specifies imaginary types to solve these sorts of semantic/efficiency problems and provide consistent treatment of special cases across the real and complex domains.

On the other hand, there is no claim that a specification without imaginary types, or some roughly equivalent mechanism, can provide consistent treatment of special cases. A major achievement of the IEEE floating-point standards was to establish a level of consistency in treatment of special cases for real arithmetic. A complex specification without imaginary types would anchor complex arithmetic a decade back in the past.

The arguments put forth in favor of such an approach are (1) it would be simpler, (2) special cases in the complex domain are so complicated that consistency is not useful, and (3) IEEE style treatment of special cases, though not supported, would not be precluded. There is some truth in all of these, though not enough to support the conclusion.

The imaginary types do complicate the specification to some degree. However, CCE is still straightforward. There has been no claim that imaginary types will be difficult to implement. And, most importantly, a system with consistent treatment of special cases will be less complicated for the user (programmer).

WG14/N310
X3J11/93-057

An explanation aside: CCE makes the imaginary types available for program declarations, though largely to follow the open spirit of C. In the anticipated common programming model, imaginary and complex values will be introduced in the natural mathematical style, `y*I` or `x + y*I`, where `x` and `y` are real and `I` is the predefined imaginary unit constant; variables generally will be declared real or complex.

Special cases for complex arithmetic are more complicated than for real arithmetic. For example, two topologies are commonly used in complex mathematics: the complex plane with its continuum of infinities and the Riemann sphere with its single infinity. The complex plane is better suited for transcendental functions, the Riemann sphere for algebraic functions. CCE uses the natural augmentation of the real and imaginary axes with signed infinities. This provides a useful (though admittedly not perfect) model for the complex plane. CCE prescribes a projection function mapping all infinities to one, which helps model the Riemann sphere. This approach is directly useful in some applications; more generally, it provides the predictability necessary to program without costly preflighting.

It has been pointed out that a specification without imaginary types need not preclude consistent treatment of special cases. Rather, infinities, NaNs, and signed zeros could be regarded as outside the specification's model, which would admit any treatment, including what CCE requires. But this would beg the question, leaving the work of determining consistency to yet another specification, which would have to be retrofitted around the *simple* one.

The C and C++ complex specifications should be as simple as possible—but not more so.