WG14 N3215

Title:	**Background regarding complex and imaginary types**
Authors:	**Jim Thomas and Jerome Coonen**
Date:	**2024-01-18**
Reference:	**N3206**

This note is intended as background for discussion about C2Y changes to complex and imaginary types and Annex G, including questions raised in N3206.

Why imaginary types?

The real and imaginary parts of values in a C complex type have the values of the corresponding real type. For IEC 60559 real types, the values include infinities, NaNs, and signed zeros, which therefore appear in complex data and operations. Annex G in C99 introduced imaginary types to accommodate these special values with consistent semantics and good performance.

An example. With imaginary types, $2i \times (\infty + 3i)$ requires just 2 real multiplications to produce the result $-6 + \infty i$. Without imaginary types, $2i$ would be taken to be the complex number $0 + 2i$ and the product

$$(0 + 2i) \times (\infty + 3i) = (0 \times \infty - 2 \times 3) + (0 \times 3 + 2 \times \infty)i$$
$$= NaN + \infty i$$

would take 4 multiplications and 2 adds to produce an unsatisfactory result (and an "invalid" floating-point exception).

Similarly, $2 \times (\infty + 3i)$, which does not involve imaginary types, would require 4 multiplications and 2 adds and produce a NaN component if the real operand were required to be promoted to complex.

Imaginary types and Annex G operator definitions also help preserve the sign of zero which is an important design feature of the floating-point standard. As noted in C23 7.3 and discussed in detail in the UC Berkeley Technical Report noted below, the sign of zero can be helpful for computations around branch cuts in the complex plane. The need to preserve (and properly propagate) the sign of zero components of complex values was an important motivation for defining an imaginary type.

More background informing the design of complex with imaginary in C99 is in:

"Augmenting a Programming Language with Complex Arithmetic", by William Kahan and J. W. Thomas, UC Berkeley/EECS Technical Report UCB/CSD-92-667, 1991, at
https://wiki.edg.com/pub/CFP/WebHome/wk_jt_augmenting_NCEG_nov91.pdf

and

"Issues Regarding Imaginary Types for C and C++", by Jim Thomas and Jerome T. Coonen, *The Journal of C Language Translation*, Volume 5, Number 3, March 1994, at
https://wiki.edg.com/pub/CFP/WebHome/imaginary%20types%20in%20C--Thomas-Coonen.pdf

Traditional complex programming

Beginning well before IEEE 754 or C99, Fortran and other facilities for complex have represented imaginary and real values as complex values. Many applications don't encounter special values and programmers can work around them with varying effort.

The **CMPLX** macros in **<complex.h>** support this traditional style of programming. They can be used to create all complex values, and the programmer can avoid imaginary types entirely.

If the Annex G imaginary types are supported, a programmer who wishes can easily avoid them. A program that uses the **I** macro will get semantic and efficiency benefits of imaginary types even without explicitly mentioning them. Or, a program can use the **I** macro and still avoid imaginary types altogether by including

```
#undef I
#define I _Complex_I
```

The imaginary types in C accommodate IEC 60559 special values with consistent semantics and good performance, while not impeding traditional complex programming.

Implementation experience

Hewlett-Packard's HP-UX C/C++ included full support of Annex G and author Thomas played an active role in the implementation. He doesn't recall any significant hurdles. The convenience and efficiency were as anticipated.

Possible reorganization of features

The main body of C does not fully specify the semantics of complex arithmetic. The usual arithmetic conversions determine a common corresponding real type (instead of a common type), which is needed for Annex G semantics. However, without Annex G, the implementation might just promote the real operand to complex as a simplification, and thus introduce a problematic zero-valued imaginary component. Also, without Annex G, behavior of special values (signed zeros, infinities, and NaNs) in complex arithmetic is unspecified.

If imaginary types remain an optional feature for complex and implementations are reluctant to support it, some reorganization of features should be considered. Annex G includes useful specification for complex arithmetic for implementations that support Annex F, whether or not imaginary types are supported. And some Annex G specification for imaginary types is applicable whether or not Annex F is supported. CFP could write a proposal for possible changes in C2Y to allow separate conformance for imaginary types and IEC 60559 compatible complex.

***i** suffix for floating constants*

An **i** or **j** suffix for floating constants was considered for C99. Despite cosmetic appeal, it was not included because its functionality duplicated and was more limited than ***I**, which can be used for variables (e.g. **y*I**), macros (e.g. **INFINITY*I**), and expressions, as well as constants. If such a suffix were added to C2Y, it should be regarded as a cosmetic feature and its semantics should match ***I**, unless a significant reason to do otherwise is known.