N350
94-035

June 3, 1994

An Alternative to Imaginary Types

Tom MacDonald
Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55121
tam@cray.com

The primary example used to demonstrate the need to add imaginary types to C is that the evaluation:

```
2i * (∞ + 3i) → (-6 + ∞i)
```

is preferable to the evaluation specified by:

```
2i * (∞ + 3i) → (NaN + ∞i)
```

(See **WG14/N339, X3J11/94-024,** *Issues Regarding Imaginary Types for C and C++.*) However, there is a technique programmers can use when they want to ensure this kind of behavior. Given a real number x and complex number z, the following macro produces the desired result for the mathematical expression $xi * z$.

```
#define IXC(x,z) CMPLX(-(x)*cimag(z), (x)*creal(z))
```

and only uses two multiplications. In the rare instances when the ∞ must be preserved, a programmer can use the above expression to guarantee a certain behavior. There is also the added advantage that the programmer is clear about where the program depends upon specific behavior involving ∞ values.

In conclusion, the above technique allows programmers to control expression evaluation in an efficient way without introducing new types to C (or new classes to C++). It continues to be my belief that the benefits gained from adding the new types does not justify the expense of adding the new types. There are easy to use alternatives for programmers when they need to preserve certain exceptional values.

41