Imaginary Types Via Typedefs?

N408, X3J11/95-009

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

Introduction

"Complex C Extensions" (CCE), Chapter 6 of X3J11's Technical Report, is a specification for complex arithmetic extensions to C that include imaginary types. The benefits of imaginary types are:

1. compatibility with IEEE floating-point arithmetic
2. more efficient storage and algorithms
3. better modeling for complex analysis

Most existing facilities for complex arithmetic follow the traditional Fortran style, which does not include imaginary types. Cray has a C implementation of this sort, but generally complex arithmetic support has been based on other languages. At its December 1994 meeting, X3J11 requested an evaluation of the idea of providing imaginary types via typedefs to complex types. Could typedefs be used to ease the burden of CCE conformance for traditional C implementations (namely Cray's)? Could typedefs facilitate porting between traditional and CCE implementations? This note responds to X3J11's requested.

How it would work

A C implementation without imaginary types would be supplemented with typedefs defining imaginary types to be complex types, for example:

```
typedef float_complex float_imaginary;
typedef double_complex double_imaginary;
typedef long_double_complex long_double_imaginary;
```

To match CCE's notational scheme, the imaginary unit constant `I`, which has imaginary type in CCE, would have complex type with real part equal 0 and imaginary part equal 1.

What would be achieved

The typedefs would not address CCE's goals of IEEE consistency or storage/algorithm efficiency, and some use in modeling the mathematics of complex analysis would be problematic, as noted below. Hence typedefs would not provide a trivially easy way for traditional implementations to conform to CCE. The benefits of the typedefs would be primarily for porting between implementations with and without imaginary types.

327

Note first that CCE amounts to an extension of the traditional complex arithmetic facility, and still supports the traditional style of programming, which declares all variables to be complex. Hence, traditional style programs which don't explicitly name imaginary types largely port without need of the typedefs. (With CCE, because the imaginary unit `I` implicitly introduces imaginary types, even ports of traditional programs benefit from the imaginary types.)

With the typedefs, some code explicitly designating imaginary types could survive on both CCE and traditional implementations. The typedefs would allow implementations without true imaginary types to host some programs that explicitly use imaginary types, and programmers on implementations without true imaginary types could use the typedefs in anticipation of porting to an implementation that had the real thing.

Many programs, for ordinary cases involving only normal numbers, and within the limitations below, would produce similar numerical results (subject to the usual floating-point porting considerations) on the two kinds of implementations.

Limitations

Code that depends on the size of the imaginary types, or on the fact that their representations match the real types, would not port. An example is the use of Standard C I/O with imaginaries, which is guaranteed by CCE.

Size and speed savings due to the smaller size of the imaginary types would not survive a port to an implementation with typedef'd imaginary types.

Only implementations with true imaginary types could guarantee the IEEE consistency benefits of imaginary types.

Of course the typedef'd imaginary types wouldn't really be imaginary. The most basic invariance of imaginary numbers would not be supported: calculated "imaginary" values well might have a non zero real part. Other assumptions guaranteed by CCE would fail on implementations without true imaginary types, e.g. imaginary * imaginary = real and cos(imaginary) = real. Hence use of imaginary typedefs for modeling purposes might prove problematic because expected invariances wouldn't hold.

The limitations described above pertain mostly to porting from CCE implementations to traditional ones.

Conclusions

For traditional implementations, imaginary types via typedefs to complex types would not address the goals of CCE, hence do not provide a mechanism for easy conformance to CCE. With or without the typedefs, most programs should port from traditional to CCE implementations, and even run better there because of the true imaginary types and their implicit use. Programs without explicit use of imaginary types should port from CCE to traditional implementations, with minor limitations. Some programs with explicit use of imaginary types should port from CCE to traditional implementations with typedefs, though with substantial limitations. An implementation without true imaginary types could reasonably provide the typedefs, or just document how the programmer could write them, but the implementation should document their limitations.