The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.3.1.7 Real and complex

real type converted to complex

When a value of real type is converted to a complex type, the real part of the complex result value is determined by the rules of conversion to the corresponding real type and the imaginary part of the complex result value is a positive zero or an unsigned zero. 700

Commentary

Prior to the conversion, the value has an arithmetic type and is in the real type domain. After the conversion the value has an arithmetic type and is in the complex type domain.

In mathematical terms all real numbers are part of the complex plane, with a zero imaginary part. It is only when a real type is converted to a complex type that a C implementation needs to start to associate an imaginary value with the complete value.

negative zero

The imaginary part is never a negative zero.

C90

Support for complex types is new in C99.

C++

The constructors for the complex specializations, 26.2.3, take two parameters, corresponding to the real and imaginary part, of the matching floating-point type. The default value for the imaginary part is specified as 0.0.

Other Languages

The Fortran intrinsic function `CMPLX` takes a parameter that specifies the type of conversion that should occur. If no `KIND` is specified, the intrinsic takes the value of the default real type. This intrinsic function also provides a default imaginary value of $0i$ (but does not say anything about the representation of zero).

Coding Guidelines

Support for complex types is new in C99 and there is no experience based on existing usage to draw on. Are there any parallels that can be made with other constructs (with a view to adapting guidelines that have been found to be useful for integer constants)? Some parallels are discussed next; however, no guideline recommendation is made because usage of complex types is not sufficiently great for there to be a likely worthwhile benefit.

Conversions between integer types and real floating types, and conversions between real floating to complex floating, both involve significant changes in developer conception and an implementation's internal representation. However, these parallels do not appear to suggest any worthwhile guideline recommendation.

A value of real type can also be converted to a complex type by adding an imaginary value of zero to it— for instance, `+0.0I` (assuming an implementation supports this form of constant). However, casts are strongly associated with conversion in developers' minds. The binary `+` operation may cause conversions, but this is not its primary association. The possibility that an implementation will not support this literal form of imaginary values might be considered in itself sufficient reason to prefer the use of casts, even without the developer associations.

```
1  #include <complex.h>
2
3  extern double _Complex glob;
4
5  void f(void)
6  {
7  glob = 1.0;                    /* Implicit conversion.      */
8  glob = (double _Complex)1.0;   /* Explicit conversion.      */
9  glob = 1.0 + 0.0I;             /* I might not be supported. */
10 }
```

Conversion of a value having a complex type, whose two components are both constants (the standard does not define the term *complex constant*), to a non-complex type is suspicious for the same reason as are other conversions of constant values.

floating-point
converted to
integer

Are there any issues specifically associated with conversion to or from complex types that do not apply for conversions between other types? It is a change of type domain. At the time if writing there is insufficient experience available, with this new type, to know whether these issues are significant.

---

701 When a value of complex type is converted to a real type, the imaginary part of the complex value is discarded and the value of the real part is converted according to the conversion rules for the corresponding real type.

Commentary

This conversion simply extracts the real part from a complex value. It has the same effect as a call to the `creal` library function. A NaN value in the part being discarded does not affect the value of the result (rather than making the result value a NaN). There are no implicit conversions defined for converting to the type `_Imaginary`. The library function `cimag` has to be called explicitly.

type specifier
syntax

C++

In C++ the conversion has to be explicit. The member functions of the complex specializations (26.2.3) return a value that has the matching floating-point type.

Other Languages

Some languages support implicit conversions while others require an explicit call to a conversion function.

References