Doc no: N3002=09-0192
Date: 2009-10-23
Reply-To: Gabriel Dos Reis
gdr@cs.tamu.edu

Gaussian Integers in the Standard Library

Gabriel Dos Reis

Texas A&M University

Abstract

This document addresses an often requested and overdue support for complex integers in the Standard Library. The proposed wording is based on LIA-3, the third part of the ISO standard for language independent arithmetic dealing specifically with Gausian integers and floating point complex numbers.

1 Generalities

Replace paragraph 26.4/2 with

> The effect of instantiating the template `complex` for any type other than the arithmetic types (except `bool` and `char`) is unspecified. Each instantiation `complex<T>` is a literal type if the template argument `T` is a literal type.

The LIA-3 standard calls for a datatype for purely imaginary complex numbers. This document does not add one, deferring to the user-defined literal proposal to use the suffix `i` for imaginary complex literals. Please, note the `constexpr` construct make it possible to define such a literal class with the same efficiency as a builtin one.

2 Synopsis

Add the following specializations to the header synopsis 26.4.1

```
template<> class complex<signed char>;
template<> class complex<unsigned char>;
template<> class complex<short>;
template<> class complex<unsigned short>;
template<> class complex<int>;
template<> class complex<unsigned>;
template<> class complex<long>;
template<> class complex<unsigned long>;
template<> class complex<long long>;
template<> class complex<unsigned long long>;
```

The Annex C.3 of LIA-3 does not explicitly mention bindings for the (integer type) template argument `T` of precision less than `int` or greater than `long long`. However, for practical purposes (e.g. embedded systems, DSP, etc.) this document allows `T` to be either `signed char`, `unsigned char`, `short`, `unsigned short`, `long long` and `unsigned long long`.

3 Complex specializations

Add the following explicit specializations to 26.4.3 for each arithmetic type `T` other than `bool` and `char`:

```
template<> class complex<T> {
  typedef T value_type;
  constexpr complex(T re = T(), T im = T());

  constexpr T real();
  void real(T);
  constexpr T imag();
  void imag(T);

  complex<T>& operator=(T);
  complex<T>& operator+=(T);
  complex<T>& operator-=(T);
  complex<T>& operator*=(T);

  template<typename X>
    complex<T>& operator=(const complex<X>&);
  template<typename X>
    complex<T>& operator+=(const complex<X>&);
  template<typename X>
    complex<T>& operator-=(const complex<X>&);
  template<typename X>
    complex<T>& operator*=(const complex<X>&);
};
```

Note: Because of the new narrowing rules, I believe that it is no longer necessary to add explicit constructors to convert from wider precision integers

to narrower precision complex integers.

4 `complex` non-member operations

Add the following paragraph to 26.4.6:

> The effect of instantiating non-member operations on complex integers other than I/O operations, ring operations, comparison operations is unspecified.

Note: This rules out division on complex integers.

5 `complex` value operations

Add the following paragraph to 26.4.7:

> The effect of instantiating `abs`, `arg`, `proj`, and `polar` on complex integers is unspecified.

6 `complex` transcendentals

Add the following paragraph to 26.4.8:

> The effect of instantiating transcendental functions other than `pow(const complex<T>&, const T&)` on complex integers is unspecified.