Title: Open Issues for Numeric Libraries (Chapter 26)
Author: Judy Ward
Document Number: X3J16/96-0026
WG21/N0844

Work Group: Library
Issue Number: 26/007
Title: cleanup of Chapter 26
Section: 26 New
Status: active
Description:

Editorial changes:

Section 26.5
The added signatures list at the end is incomplete.
It only includes the the float ones, not the long
double functions mentioned in the previous sentence.

Why does the double abs(double) have a comment that says fabs?
I'm not sure what this comment or the labs() or ldiv() comments
mean.

Shouldn't the last two prototypes be:
float abs(float);
float pow(float, int);

Resolution:
Requestor: Judy Ward
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
Papers: (committee documents that discuss this issue)
. . . . . .
Work Group: Library
Issue Number: 26/008
Title: algebraic structures as traits
Section: 26 New
Status: active
Description:

From: Takanori Adachi <taka@miwa.co.jp>
Date: 30-Jun-1995 2324 -0500" 30-JUN-1995 23:22:47.04
CC: taka@miwa.co.jp
Subj: field_traits

To: C++ libraries mailing list
Message c++std-lib-3819

The following is an idea to specify algebraic structures by the traits
parameterized with the underlying set T.

```
  template <class T> struct field_traits {}; // abstraction of field structure

  struct field_traits<float> { // a specialization for 'float'
  typedef float scalar_type;

  // fundamental field operators
  static bool eq(const scalar_type s1, const scalar_type s2)
     { return (s1 == s2); }
  static scalar_type plus(const scalar_type s1, const scalar_type s2)
     { return s1 + s2; }
```

```
  ...
  // and special functions
  static scalar_type sin_f(const scalar s) { return sin(s); }
  ...
  // and constants
  static scalar_type pi() { return 3.141592; }
  ...
  };

  // fundamental field operators
  static bool eq(const scalar_type s1, const scalar_type s2)
     { return (s1 == s2); }
  static scalar_type plus(const scalar_type s1, const scalar_type s2)
     { return s1 + s2; }
  ...
  // and special functions
  static scalar_type sin_f(const scalar s) { return sin(s); }
  ...
  // and constants
  static scalar_type pi() { return 3.14159265358979; }
    // more precise than that of float;
  ...
  };
```

Using these structure specifying traits, we can define several numeric operators.

```
  template<class T, class traits> complex<T,traits>
    operator+(const complex<T,traits>& c1, const complex<T,traits>& c2) {
      return complex(
        traits:plus(c1.real(), c2.real()),
        traits:plus(c1.imag(), c2.imag()) );
    }


  template<class T, class traits> complex<T,traits>
    exp(const complex<T,traits>& c) {
      // use Euler’s theorem and traits::exp_f, sin_f, cos_f.
    }
```

The classes complex and valarray over an field T are also defined as:

```
  template<class T, class traits = field_traits<T> >
  class complex { ... };
  template<class T, class traits = field_traits<T> > class valarray;
```

Resolution:
Requestor: Takanori Adachi
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
c++std-lib-3820
c++std-lib-3821
c++std-lib-3832
c++std-lib-3835
Papers: (committee documents that discuss this issue)
. . . . . .
Work Group: Library
Issue Number: 26/009
Title: valarray usefulness
Section: 26 New
Status: active
Description:

From: ncm@netcom.com (Nathan Myers)
Date: 24-Jul-1995 1608 -0500" 24-JUL-1995 16:07:47.61

CC: alv@roguewave.com,keffer@roguewave.com,vandevod@cs.rpi.edu
Subj: valarray

To: C++ libraries mailing list
Message c++std-lib-3881

In ANSI public comment T29, David Vandevoorde <vandevod@cs.rpi.edu> says:

> Comments on the proposed <valarray> header
> ...
> Probably the simplest way to address the above concerns is to simply
> abandon the standardization of a numerical array.

I would like to take this alternative seriously.

With the advent of Todd Veldhuizen's work on Expression Templates, it is far from clear that valarray<> is the appropriate vehicle to aid in optimizing numeric array processing in C++. (For those who have not read Veldhuizen's work in C++ Report, a copy may be found at <http://www.roguewave.com/>.) His work implies that using even a vendor-optimized/compiler-supported valarray<> may cost a factor of two or more in speed compared to using another library based on portable language facilities. This brings into question the value of the valarray<> template; the original argument in its favor was that it provided the hooks to permit optimal implementation "under the hood" (that's "under the bonnet" for you Brits).

This is not a formal proposal to eliminate valarray<>, yet; it is instead a request for comments. I would like particularly to hear from ISO representatives whose vote might be forced to change if it is removed.

Nathan Myers
myersn@roguewave.com

Resolution:

Requestor: Nathan Myers
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
c++std-lib-3880
c++std-lib-3883
c++std-lib-3886
c++std-lib-3887
c++std-lib-3889
c++std-lib-3897
c++std-lib-3900
c++std-lib-3906
c++std-lib-3908
c++std-lib-3909
c++std-lib-3910
c++std-lib-3914
c++std-lib-3918
c++std-lib-3920
c++std-lib-3925

Papers: (committee documents that discuss this issue)
. . . . . .
Work Group: Library
Issue Number: 26/010
Title: basic_complex
Section: 26 New
Status: active

Description:
From: Dag Bruck <dag@dynasim.se>
Date: 16-AUG-1995 07:47:07.55
Subj: Complex datatype

To: C++ libraries mailing list
Message c++std-lib-3962

I support

```
        template <class scalar> class basic_complex {
                // .....
        };

        typedef basic_complex<double> complex;
```

(1) This is similar to basic_string.
(2) It maintains compatibility with current implementations.
(3) It satisfies 98% of all use of complex, i.e., I claim
    that other complex types are rare in practice.
(4) There is no generally known prior art for naming other
    complex types that I'm aware of.

-- Dag

Resolution:
Requestor: Dag Bruck
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
c++std-lib-3963
c++std-lib-3964
c++std-lib-3965
c++std-lib-3966
c++std-lib-3970
c++std-lib-3971
c++std-lib-3977
c++std-lib-3978
c++std-lib-3981
c++std-lib-3987
c++std-lib-3988
c++std-lib-3922
c++std-lib-4004
c++std-lib-4006
c++std-lib-4009
c++std-lib-4051
c++std-lib-4077

Papers: (committee documents that discuss this issue)
. . . . . .
Work Group: Library
Issue Number: 26/011
Title: ambiguity with assignment ops in complex
Section: 26.2.2 New
Status: active
Description:

```
/*
I think this code shows a problem the complex library.
If you run this code you will see
that the compiler gives an ambiguity error:

"t.cxx", line 32: error: more than one operator "=" matches these operands:
            function "complex<double>::operator=(const complex<float> &)"
            function "complex<double>::operator=(const complex<double> &)"
        d = 0.0;
          ^
```

```
1 error detected in the compilation of "t.cxx".

So you can't assign a scalar of type double
to a complex<double>. I think the intention
is that users should be able to do this, right?
Our non-templatized complex library allows it.
Is there any way to fix the complex library to allow
this?

Judy Ward
j_ward@decc.enet.dec.com
*/

// simplified version of class complex
template <class T>
class complex {};

struct complex<float>
{
    complex(const float& re_arg = 0.0f, const float& imag_arg= 0.0f ) {;}
};

struct complex<double>
{
    complex(const double& re_arg = 0.0, const double& imag_arg = 0.0) { ; }
    complex<double>& operator=(const complex<float>&) { return *this; }
    complex<double>& operator=(const complex<double>&) { return *this; }
};

int main() {
        complex<double> d;
        d = 0.0;
        return 0;
}


Resolution:

template <class X> complex<float>& operator=(const X&);
template <class X> complex<double>& operator=(const X&);
template <class X> complex<long double>& operator=(const X&);

to each of the complex specializations.

Requestor: Judy Ward
Owner:         Judy Ward
Emails: (email reflector messages that discuss this issue)
Papers: (committee documents that discuss this issue)
. . . . . .
Work Group:     Library
Issue Number:   26/012
Title: Complex class operators
Section:        26.2.1 New
Status:         active
Description:

To: C++ libraries mailing list
Message c++std-lib-4384

There is an inconsistency in clause 26.2.1 where
template class complex operators are specified.
The following example shows this.

 #include <complex>
 int main ()
 {
```

```
   complex<float> c(2.0, 2.0);

   c = c + 1.0;
   // This will work because of the
   // declaration of operator +(const complex<T>&, T);
   //

   c += 1.0;
   // This will fail because operator +=(const complex<X>&)
   // is not overloaded.

   c += (complex<float>)1.0;
   // This will work because of the
   // declaration of operator +=(const complex<X>&)

   // Yet they should all have the same ultimate
   // behavior.  Either they should all work, or
   // they should all fail.

   return 0;
 }
```

Resolution:

John Max Skaller proposes that:

This is what you get when you mess around with conversions.

There's a simple rule for such types as complex:
DO NOT OVERLOAD METHODS ON RELATED TYPES.

There should be ONE, and ONLY ONE, place in which
an embedding such as float --> complex is represented
and that is in the constructor of complex.

All the other operations should rely on that embedding.

Requestor: Annie Groeninger and Marlene J. Hart
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
c++std-lib-4397
Papers: (committee documents that discuss this issue)

. . . . . .
Work Group: Library
Issue Number: 26/013
Title: sqrt() function in complex lib -- which root does it return?
Section: 26.2 New
Status: active
Description:

To: C++ libraries mailing list
Message c++std-lib-4427

I see we have a sqrt(complex) that returns a complex (of the
right type). However, doesn't a complex have MANY square roots?

Are anyone conducting a review of the match library?

- Bjarne

To: C++ libraries mailing list
Message c++std-lib-4430

Bjarne Stroustrup writes:

> I see we have a sqrt(complex) that returns a complex (of the
> right type). However, doesn't a complex have MANY square roots?

Well, it has two. If x is a root, then -x is also a root. By
widespread convention, the root with phase angle [-pi/2, pi/2)
(as I recall) is preferred as the return value for sqrt.

> Are anyone conducting a review of the match library?

We've had some useful feedback from the heavy hitters in the C
math library community.

P.J. Plauger

Resolution:

We should document the "widespread convention" of
returning the root with phase angle [-pi/2, pi/2] in Section
26.2.7 of the draft.

Requestor: Bjarne Stroustrup
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)
Papers: (committee documents that discuss this issue)

. . . . . .
Work Group: Library
Issue Number: 26/014
Title: Friends operators within class complex
Section: 26.2.1 New
Status: active
Description:

From: "Marlene Hart" <hart@roguewave.com>
Date: 12-DEC-1995 20:50:16.29
To: @proxy.research.att.com:rmeyers@decc.enet.dec.com,
@proxy.research.att.com:mglenn@primenet.com
CC: groening@roguewave.com
Subj: Friends operators within class complex

To: C++ libraries mailing list
Message c++std-lib-4383

Is it necessary for the operators +, -, *, /, ==,
and != to be friends of the template class complex,
Section 26.2.1?

This declaration implies an implementation that requires
these operators to access private data. However, the
class complex provides public real and imaginary
data accessors, thus implying that it would not be
necessary to access any private data.

Annie Groeninger and Marlene Hart
Software Developers
Rogue Wave Software
Corvallis, OR
groening@roguewave.com
hart@roguewave.com

Resolution:
Requestor: Annie Groeninger and Marlene Hart
Owner: Judy Ward
Emails: (email reflector messages that discuss this issue)

Papers: (committee documents that discuss this issue)