Document Number: WG21/N0977 X3J16/96-0159
Date: 11 July 1996
Project: Programming Language C++
Reply to: Library 3 working group

Omnibus clause 23-26 changes

This is a list of WP changes recommended by the Library 3 working˝group. Most are simple cleanups and clarifications, and none are expected to be controversial.

Bitset element access methods

Amend the WP as follows, thus closing issue 23-046:
-- adding the following members to the class bitset synopsis in˝clause 23.2.1:
bool operator [ ] (size_t pos) const;
reference at(size_t pos);
bool at(size_t pos) const;

-- adding the following text to clause 23.2.1.2:
reference at(size_t pos);
Requires: pos is valid
Throws: out_of_range if pos does not correspond to a valid bit˝position.
Returns: reference to the bit at position pos in *this.

bool at(size_t pos) const;
Requires: pos is valid
Throws: out_of_range if pos does not correspond to a valid bit˝position.
Returns: true if the bit at position pos in *this has the value one.

-- adding the following text to clause 23.2.1.3:
bool operator [ ] (size_t pos);
Requires: pos is valid
Returns: reference to the bit at position pos in *this.

bool operator [ ] (size_t pos) const;
Requires: pos is valid
Returns: true if the bit at position pos in *this has the value one.

Clarify descriptions of algorithms

Amend the WP by changing the word "equal" in the second paragraph˝labeled "Complexity" in clause 23.2.2.6 to "at most equal", thus closing issue 23-047.

Amend the WP, closing issue 23-049, by striking the last sentence of˝**`insert`**'s **complexity** description in clause 23.2.5.6 [lib.vector.modifiers] and replacing it with the˝following:
If **`first`** and **`last`** are forward iterators, bidirectional iterators, or random access˝iterators, the complexity is linear in the number of elements in the range **`[first, last)`** plus the distance to the end of the vector. If they are input iterators, the complexity is˝proportional to the number of elements in the range **`[first, last)`** times the distance to the end of the vector.

Amend the WP, closing issue 23-051, by striking unique's **effects** description in clause 23.2.3.7 and replacing it with the following: Eliminates all but the first element˝from every consecutive group of equal elements referred to by the iterator i in the range [first+1, last)˝for which the following corresponding conditions hold:
*i == *(i-1) or pred(*i,*(i-1)) is true.

Amend the WP by striking the **complexity** descriptions for list::unique() in clause 23.2.3.7 and for unique() in clause 25.2.8 and replacing each of them with the˝following:
If the range [first, last) is not empty, exactly (last - first) - 1˝applications of the corresponding predicate; otherwise no applications of the predicate.

Amend the WP by adding the following sentence to **`merge`**'s **effects** clause in clause 23.2.3.7, thus closing issue 23-052:
The list will be sorted in non-decreasing order according to the˝ordering defined by **`comp`**; that is, for every iterator i in **`[first, last)`** other than **`first`**, the condition **`comp(*i, *(i-1))`** will be **`false`**.

Amend the WP by adding the following sentence to the **effects** sections for merge and inplace_merge in clause 25.3.4:
The resulting range will be in non-decreasing order; that is, for˝every iterator i in **`[first, last)`** other than **`first`**, the condition **`*i < *(i-1)`** or **`comp(*i, *(i-1))`** will be **`false`**.

Amend the WP as follows:
-- add the following text to the end of the **Effects** section of clause 25.1.1:
starting from **`first`** and proceeding to **`last - 1`**.

Amend the WP as follows, thus closing issue 25-013:
-- strike the **Requires** section from its present location in clause 25.1.9, and insert it˝before the second **Effects** section in that clause.

Fix vector<bool>

Amend the WP by striking the typedef for const_reference in clause˝23.2.6 and replacing it with typedef bool const_reference;
thus closing issue 23-053.

Amend the WP by adding the following declaration to the nested class **`reference`** in clause 23.2.6, thus closing issue 23-054:
`reference& operator=(const reference& x).`

Fix map element access

Amend the WP by striking the text "mapped_type&" in the first line of˝the element access portion of clause 23.3.1 and replacing it with "reference", and by striking the˝text "T&" in clause 23.3.1.5 and replacing it with "reference", thus closing issue 23-055.

Amend the WP by striking the second line of the element access portion˝of clause 23.3.1, which currently reads "const mapped_type& operator [ ](const key_type& x) const;",˝thus closing issue 23-056.

Correct the typedefs of reverse iterators in various containers

Amend the WP as follows, thus closing issues 23-058 and 23-059:
-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.2.2 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer pointer;
typedef reverse_iterator<iterator, value_type, reference, pointer,˝difference_type> reverse_iterator;
typedef const_reverse_iterator<const_iterator, value_type,˝const_reference, const_pointer,
                                    difference_type> reverse_iterator;
```

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.2.3 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_bidirectional_iterator<iterator, value_type,˝reference, pointer, difference_type>
        reverse_iterator;
typedef reverse_bidirectional_iterator<const_iterator, value_type,˝const_reference, const_pointer,
                                        difference_type> const_reverse_iterator;
```

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.2.5 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_iterator<iterator, value_type, reference, pointer,˝difference_type> reverse_iterator;
typedef reverse_iterator<const_iterator, value_type, const_reference,˝const_pointer, difference_type>
        const_reverse_iterator;
```

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.2.6 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_iterator<iterator, value_type, reference, pointer,˝difference_type> reverse_iterator;
typedef reverse_iterator<const_iterator, value_type, const_reference,˝const_pointer, difference_type>
        const_reverse_iterator;
```

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.3.1 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_bidirectional_iterator<iterator, value_type,˝reference, pointer, difference_type>
        reverse_iterator;
typedef reverse_bidirectional_iterator<const_iterator, value_type,˝const_reference, const_pointer,
˝                                       difference_type>˝ const_reverse_iterator;
```

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.3.2 and replacing them with the following:

```
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_bidirectional_iterator<iterator, value_type,˝reference, pointer, difference_type>
        reverse_iterator;
typedef reverse_bidirectional_iterator<const_iterator, value_type,˝const_reference, const_pointer,
```

˝ difference_type>˝ const_reverse_iterator;

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.3.3 and replacing them with the following:
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_bidirectional_iterator<iterator, value_type,˝reference, pointer, difference_type>
reverse_iterator;
typedef reverse_bidirectional_iterator<const_iterator, value_type,˝const_reference, const_pointer,
˝ difference_type>˝ const_reverse_iterator;

-- striking the typedefs for reverse_iterator and˝const_reverse_iterator in clause 23.3.4 and replacing them with the following:
typedef typename Allocator::pointer pointer;
typedef typename Allocator::const_pointer const_pointer;
typedef reverse_bidirectional_iterator<iterator, value_type,˝reference, pointer, difference_type>
reverse_iterator;
typedef reverse_bidirectional_iterator<const_iterator, value_type,˝const_reference, const_pointer,
˝ difference_type>˝ const_reverse_iterator;

fix the postcondition for (&a)->~X()

Amend the WP by striking the text "post: a.size() == 0." from the˝entry for assertion/note for (&a)->~X() in Table 75 in clause 23.1, thus closing issue 23-060.

Correct the introductory text of clause 25

Amend the WP as follows, thus closing issue 23-062:
-- add the following paragraphs to the introductory text of clause 25˝following paragraph 3:

Throughout this clause, the names of template parameters are used to express type requirements. If an algorithm's template parameter is **`InputIterator`**, **`InputIterator1`**, or **`InputIterator2`**, the actual template argument shall satisfy the requirements of an˝input iterator ([lib.input.iterators]). If an algorithm's template˝parameter is **`OutputIterator`**, **`OutputIterator1`**, or **`OutputIterator2`**, the actual template argument shall satisfy the requirements of an output iterator ([lib.output.iterators]). If an˝algorithm's template parameter is **`ForwardIterator`**, **`ForwardIterator1`**, or **`ForwardIterator2`**, the actual template argument shall satisfy the requirements of a forward iterator˝([lib.forward.iterators]). If an algorithm's template parameter is **`BidirectionalIterator`**, **`BidirectionalIterator1`**, or **`BidirectionalIterator2`**, the actual template argument shall satisfy the requirements of a bidirectional iterator˝([lib.bidirectional.iterators]). If an algorithm's template parameter is **`RandomAccessIterator`**, **`RandomAccessIterator1`**, or **`RandomAccessIterator2`**, the actual template argument shall satisfy the requirements of a random access iterator ([lib.random.access.iterators]).

If an algorithm's **effects** section says that a value pointed to by any iterator passed as an argument is modified, then that algorithm has an additional type˝requirement: the type of that argument shall satisfy the requirements of a mutable iterator˝[lib.iterator.requirements]). [Note: this requirement does not affect arguments that are declared as **`OutputIterator`**, **`OutputIterator1`**, or **`OutputIterator2`**, since output iterators must always be mutable.]

-- add the following sentence to the end of paragraph 1 in clause 20.4.2:

The template parameter **`OutputIterator`** is required to satisfy the requirements of an output iterator (lib.output.iterators]).

-- add the following sentences to the end of paragraph 1 in clause˝20.4.4:
In the algorithm **`uninitialized_copy`**, the formal template parameter **`InputIterator`** is required to satisfy the requirements of an input iterator˝(lib.input.iterators). In all of the following algorithms the formal template parameter **`ForwardIterator`** is required to satisfy the requirements of a forward iterator ([lib.forward.iterators]) and to˝satisfy the requirements of a mutable iterator [lib.iterator.requirements]).

-- add the following paragraph at the end of clause 26.4:
The requirements on the types of algorithms' arguments that are˝described in the introduction to clause 25 also apply to the following algorithms.

remove the allocator parameter from container adapters

Amend the WP as follows, thus closing issue 20-025:
-- strike the text
template <class T, class Container = deque<T>, class Allocator =˝allocator<T> > class queue
from clause 23.2.4.1, replacing it with
template <class T, class Container = deque<T> > class queue

-- strike the text
template <class T, class Container = deque<T>, class Allocator =˝allocator<T> > class queue
from the header <queue> synopsis in clause 23.2, replacing it with
template <class T, class Container = deque<T> > class queue

-- strike the text
typedef Allocator allocator_type;
from clause 23.2.4.1, replacing it with
typedef typename Container::allocator_type allocator_type;

-- strike the text
explicit queue(const Allocator& = Allocator());
from clause 23.2.4.1, replacing it with
explicit queue(const allocator_type& = allocator_type());

-- strike the text
```
template <class T, class Container, class Allocator>
  bool operator== (const queue<T, Container, Allocator>& x,
                   const queue<T, Container, Allocator>&˝y);
template <class T, class Container, class Allocator>
  bool operator< (const queue<T, Container, Allocator>& x,
                  const queue<T, Container, Allocator>&˝y);
```
from clause 23.2.4.1, replacing it with
```
template <class T, class Container>
  bool operator== (const queue<T, Container>& x,
                   const queue<T, Container>& y);
template <class T, class Container>
  bool operator< (const queue<T, Container>& x,
                  const queue<T, Container>& y);
```

-- strike the text

```
template <class T, class Container, class Allocator>
  bool operator== (const queue<T, Container, Allocator>& x,
                   const queue<T, Container, Allocator>& y);
template <class T, class Container, class Allocator>
  bool operator< (const queue<T, Container, Allocator>& x,
                  const queue<T, Container, Allocator>& y);
```
from the header <queue> synopsis in clause 23.2, replacing it with
```
template <class T, class Container>
  bool operator== (const queue<T, Container>& x,
                   const queue<T, Container>& y);
template <class T, class Container>
  bool operator< (const queue<T, Container>& x,
                  const queue<T, Container>& y);
```

-- strike the text
```
template <class T, class Container = vector<T>,
          class Compare = less<Container::value_type>,
          class Allocator = allocator<T> >
class priority_queue
```
from clause 23.2.4.2, replacing it with
```
template <class T, class Container = vector<T>,
          class Compare = less<Container::value_type> >
class priority_queue
```

-- strike the text
```
template <class T, class Container = vector<T>,
          class Compare = less<Container::value_type>,
          class Allocator = allocator<T> >
class priority_queue
```
from the header <queue> synopsis in clause 23.2, replacing it with
```
template <class T, class Container = vector<T>,
          class Compare = less<Container::value_type> >
class priority_queue
```

-- strike the text
typedef Allocator allocator_type;
from clause 23.2.4.2, replacing it with
typedef typename Container::allocator_type allocator_type;

-- strike the text
explicit priority_queue(const Compare& x = Compare(), const˝Allocator& = Allocator());
from clause 23.2.4.2, replacing it with
explicit priority_queue(const Compare& x = Compare(), const˝allocator_type& = allocator_type());

-- strike the text
template <class T, class Container = deque<T>, class Allocator =˝allocator<T> > class stack
from clause 23.2.4.3, replacing it with
template <class T, class Container = deque<T> > class stack

-- strike the text
template <class T, class Container = deque<T>, class Allocator =˝allocator<T> > class stack
from the header <stack> synopsis in clause 23.2, replacing it with
template <class T, class Container = deque<T> > class stack

-- strike the text

typedef Allocator allocator_type;
from clause 23.2.4.3, replacing it with
typedef typename Container::allocator_type allocator_type;

-- strike the text
explicit stack(const Allocator& = Allocator());
from clause 23.2.4.3, replacing it with
explicit stack(const allocator_type& = allocator_type());

Remove unapproved algorithms

Amend the WP as follows:
-- strike the following text from the transcendentals section of˝clause 26.2:
template <class T> complex<T> acos(const complex<T>&);
template <class T> complex<T> asin(const complex<T>&);
template <class T> complex<T> atan(const complex<T>&);
template <class T> complex<T> atan2(const complex<T>&, const˝complex<T>&);
template <class T> complex<T> atan2(const complex<T>&, T);
template <class T> complex<T> atan2(T, const complex<T>&);

Specify branch cuts and ranges for transcendental complex functions

Amend the WP by striking the contents of clause 26.2.7 and replacing˝it with the following text, thus closing issues 26-016 and 26-051:
template <class T> complex<T> cos (const complex<T>& x);
Returns: the complex cosine of x.
template <class T> complex<T> cosh (const complex<T>& x);
Returns: the complex hyperbolic cosine of x.
template <class T> complex<T> exp (const complex<T>& x);
Returns: the complex base e exponential of x.
template <class T> complex<T> log (const complex<T>& x);
Notes: the branch cuts are along the negative real axis.
Returns: the complex natural (base e) logarithm of x, in the range˝of a strip mathematically unbounded along the real axis and in the interval [-i*pi,i*pi] along the˝imaginary axis. When x is a negative real number, imag(log(x)) is pi.
template <class T> complex<T> log10 (const complex<T>& x);
Notes: the branch cuts are along the negative real axis.
Returns: the common (base 10) logarithm of x, defined as˝log(x)/log(10).
template <class T> complex<T> pow (const complex<T>& x, const˝complex<T>& y);
template <class T> complex<T> pow (const complex<T>& x, T y);
template <class T> complex<T> pow (T x, const complex<T>& y);
Notes: the branch cut for x is along the negative real axis.
Returns: the complex power of base x raised to the y-th power,˝defined as exp(y*log(x)). The value returned for pow(0,0) is implementation-defined.
template <class T> complex<T> sin (const complex<T>& x);
Returns: the complex sine of x.
template <class T> complex<T> sinh (const complex<T>& x);
Returns: the complex hyperbolic sine of x.
template <class T> complex<T> sqrt (const complex<T>& x);
Notes: the branch cuts are along the negative real axis.

Returns: the complex square root of x, in the range of the right˝half-plane. If the argument is a negative real number, the value returned lies on the positive imaginary axis.
template <class T> complex<T> tan (const complex<T>& x);
Returns: the complex tangent of x.
template <class T> complex<T> tanh (const complex<T>& x);
Returns: the complex hyperbolic tangent of x.

Small valarray fixes

Amend the WP by changing valarray<int> in paragraph 1 of clause 26.3.8˝to valarray<size_t>, thus closing issue 26-022.

Amend the WP as follows, thus closing issue 26-023 and 26-043:
-- add the following text to clause 26.3.1.7:
T min() const;
This function returns the minimum value contained in *this.
The value returned for an array of length 0 is undefined. for an array˝of length 1, the value of element 0 is returned. For all other array lengths, the determination is made using˝operator <.
T max() const;
This function returns the maximum value contained in *this.
The value returned for an array of length 0 is undefined. for an array˝of length 1, the value of element 0 is returned. For all other array lengths, the determination is made using˝operator <.
-- strike all of clause 26.3.2.3

Amend the WP striking the text
size_t length() const;
from both clauses 26.3.1 and 26.3.1.7, replacing it in both cases˝with
size_t size() const;
thus closing issue 26-028:

Amend the WP by changing the return types of operator || and operator˝&& from valarray<T> to valarray<bool>, thus closing issue 26-029.

Amend the WP by striking the text
valarray<T> operator!() const;
from both clauses 26.3.1 and 26.3.1.5, replacing it in both cases˝with
valarray<bool>, thus closing issue 26-032.

Amend the WP as follows, thus closing issue 26-039:
-- strike the lines
operator T*();
and
operator const T *();
from clause 26.3.1.
-- strike the lines
operator T*();
and
operator const T *();
and the corresponding descriptive text from clause 26.3.1.7.

Amend the WP as follows, thus closing issues 26-040 and 26-042:
-- Move the last two sentences of the Effects section for˝valarray() to Notes in clause 26.3.1.1.

-- Move the last sentence of the description of valarray(const T*,˝size_t) to Notes in clause 26.3.1.1.
-- Move the last two sentences of the description of valarray(const˝valarray<T>&) to Notes in clause 26.3.1.1.

Amend the WP by adding the following text to the end of clause˝26.3.1.1, thus closing issue 26-041:
The destructor is applied to every element of *this; all allocated˝memory is returned.

Small fixes to complex template

Amend the WP by inserting the following text after the first sentence˝of clause 26.2, thus closing issue 26-035:
The effect of instantiating the template complex for any type other˝than float, double, or long double is unspecified.

Amend the WP as follows, thus closing issue 26-036:
-- change every parameter of type T to type const T& throughout˝clauses 26.2, 26.2.1, 26.2.5, and 26.2.7.

Amend the WP as follows, thus closing issue 26-038:
-- strike the text
the phase angle of x.
from the Returns: section for arg in clause 26.2.6, replacing it with˝the following:
the phase angle of x, or atan2(imag(x), real(x)).

Amend the WP by inserting the text "If the result of a function is not˝mathematically defined or not in the range of representable values for its type, the behavior is˝undefined." after the first paragraph of clause 26.2, thus closing issue 26-049.

Other clause 26 fixes

Amend the WP by striking the text "Assigns to every iterator i" in˝clause 26.4.3 and replacing it with "Assigns to every element referred to by iterator i", thus closing˝issue 26-045.

Amend the WP by removing boxes 97, 98, 99, and 100, thus closing˝issues 26-046, 26-047, 26-048, and 26-050.