Doc No: X3J16/95-0014
WG21/N0614
Date: 1/30/95
Project: Programming Language C++
Reply To: J. Lawrence Podmolik
Andersen Consulting
`jlp@chi.andersen.com`

Numerous Small STL Changes

Introduction

This paper contains a number of relatively small corrections, modifications and additions to various parts of STL. Most of these changes were identified by Alex Stepanov and Meng Lee, the original authors of STL. The primary goal was to correct obvious errors and omissions; however these changes also bring the WP descriptions of the STL components more in line with the recent versions of STL that Alex and Meng have made publicly available.

To the extent possible, I have tried to separate the substantive issues (such as new components or changes in existing behavior) from editorial issues (typos, minor wording changes, etc.). The clause numbers referenced below are from the pre-Valley Forge version of the WP [Koenig94], which in turn was based on the version of STL described in [Stepanov94]; most of the corrections are from [Stepanov95].

1.0 Substantive Changes

1.1 Clause 17 (Library Introduction)

1.1.1 **Summary:** Add additional container requirements for references, address-of operations, and construction/destruction.

Proposal: Add the following requirements to Clause 17.2.2.4 [lib.container.types], Table 22:

expression	return type	assertion/note pre/post-condition	complexity
`X::reference`	lvalue of `T`		compile time
`X::const_reference`	const lvalue of `T`		compile time
`X::iterator`	iterator type pointing to `X::reference`	an iterator of any iterator category except output iterator	compile time
`X::const_iterator`	iterator type pointing to `X::const_reference`	an constant iterator of any iterator category except output iterator	compile time
`a.swap(b)`	`void`	swap(a,b)	constant

Rationale: The additions/modifications to support the reference types follow directly from corresponding definitions in the allocators. The container-specific `swap()` operation is defined as many containers can provide a more efficient implementation than the generic `swap()` algorithm.

1.1.2 **Summary:** Add a new table to 17.2.2.4 describing requirements for “reversible” containers, i.e. containers whose iterator types belong to the bidirectional or random access iterator categories.

Proposal: Insert the following paragraph and table at the end of Clause 17.2.2.4 [lib.container.types]:

> “If the iterator type of a container belongs to the bidirectional or random access iterator categories, the container is called *reversible* and satisfies the following additional requirements:”

expression	return type	assertion/note pre/post-condition	complexity
X::reverse_iterator	iterator type pointing to X::reference	reverse_iterator<iterator, value_type, reference, difference_type> for random access iterator reverse_bidirectional_iterator<iterator, value_type, reference, difference_type> for bidirectional iterator	compile time
X::const_reverse_iterator	iterator type pointing to X::const_reference	reverse_iterator<const_iterator, value_type, const_reference, difference_type> for random access iterator reverse_bidirectional_iterator<const_iterator, value_type, const_reference, difference_type> for bidirectional iterator	compile time
a.rbegin()	reverse_iterator; const_reverse_iterator for constant a	reverse_iterator(end())	constant
a.rend()	reverse_iterator; const_reverse_iterator for constant a	reverse_iterator(begin())	constant

1.2 Clause 20 (General Utilities Library)

1.2.1 **Summary:** Remove the “empty” tuple from 20.1.2.

Proposal: Replace Clauses 20.1.2 [lib.tuples], 20.1.2.1 [lib.empty], 20.1.2.2 [lib.pair] and 20.1.3 [lib.restrictor] with a single clause 20.1.2 [lib.pair] that reads as follows:

> “The library includes templates for heterogeneous pairs of values. The library also provides a matching template function to simplify their construction.”
>
> <*current contents of 20.1.2.2*>
>
> <*current contents of 20.1.2, excluding the first paragraph*>

Rationale: The `empty` class was used in the original implementation of STL to work around compiler bugs; `restrictor` existed for similar reasons.

1.2.2 **Summary:** Add a new template function called `return_temporary_buffer()` to complement `get_temporary_buffer()`.

Proposal: Create a new section 20.3.3.6 [lib.return.temporary.buffer] that contains the following:

```
template <class T>
void return_temporary_buffer(T* p);
```

`return_temporary_buffer` returns the buffer allocated by `get_temporary_buffer`.

1.3 Clause 23 (Containers Library)

1.3.1 **Summary:** Add the pointer and reference typedefs to the appropriate containers in Clause 23.

Proposal: For each of the following containers in Clause 23 [lib.containers], add typedefs for `reference` and `const_reference` in accordance with the requirements described in §1.1.2:

- `vector` (Clause 23.1.5 [lib.vector])
- `vector<bool>` (Clause 23.1.6 [lib.vector.bool])
- `list` (Clause 23.1.7 [lib.list])
- `deque` (Clause 23.1.8 [lib.deque])
- `set` (Clause 23.2.1 [lib.set])
- `multiset` (Clause 23.2.2 [lib.multiset])
- `map` (Clause 23.2.3 [lib.map])
- `multimap` (Clause 23.2.4 [lib.multimap])

For each container `X`, the typedefs and functions look as follows; note that these typedefs are as described in 94-0161/N0548 ("Runtime-Variable Allocators for STL"):

```
typedef Allocator::types<T>::reference          reference;
typedef Allocator::types<T>::const_reference    const_reference;
```

Rationale: This follows directly given 94-0161/N0548 and assuming that the proposal in §1.1.1 is accepted.

1.3.2 **Summary:** Integrate `X::reference` and `X::const_reference` into each of the container declarations in Clause 23.

Proposal: For each of the container declarations in Clause 23, replace each occurrence of `T&` with the typedef `reference`, and replace each occurrent of `const T&` with the typedef `const_reference`.

Rationale: This follows directly assuming that the proposal in §1.1.1 is accepted.

1.3.3 **Summary:** Add `swap()` member functions to the appropriate containers in Clause 23.

Proposal: For each of the following containers in Clause 23 [lib.containers], add a `swap()` member function defined in accordance with the requirements described in §1.1.1:

- `vector` (Clause 23.1.5 [lib.vector])
- `vector<bool>` (Clause 23.1.6 [lib.vector.bool])
- `list` (Clause 23.1.7 [lib.list])
- `deque` (Clause 23.1.8 [lib.deque])

- `set` (Clause 23.2.1 [lib.set])
- `multiset` (Clause 23.2.2 [lib.multiset])
- `map` (Clause 23.2.3 [lib.map])
- `multimap` (Clause 23.2.4 [lib.multimap])

For each container `X`, the functions should have the form

```
void swap(X& x);
```

In addition, add the following function definition to 23.1.5 [lib.vector]:

```
void swap(vector<bool, allocator>::reference x,
          vector<bool, allocator>::reference::y);
```

Rationale: This follows directly assuming that the proposal in §1.1.1 is accepted.

1.3.4 **Summary:** Add the reverse iterator typedefs and `rbegin()/rend()` functions to the appropriate containers in Clause 23.

Proposal: For each of the following containers in Clause 23 [lib.containers], add typedefs for `reverse_iterator` and `const_reverse_iterator`, and declarations for `rbegin()` and `rend()` in accordance with the requirements described in §1.1.2:

- `vector` (Clause 23.1.5 [lib.vector])
- `vector<bool>` (Clause 23.1.6 [lib.vector.bool])
- `list` (Clause 23.1.7 [lib.list])
- `deque` (Clause 23.1.8 [lib.deque])
- `set` (Clause 23.2.1 [lib.set])
- `multiset` (Clause 23.2.2 [lib.multiset])
- `map` (Clause 23.2.3 [lib.map])
- `multimap` (Clause 23.2.4 [lib.multimap])

For each container `X`, the typedefs and functions look as follows:

```
typedef ? reverse_iterator;
typedef ? const_reverse_iterator;

reverse_iterator        rbegin();
const_reverse_iterator  rbegin() const;
reverse_iterator        rend();
const_reverse_iterator  rend() const;
```

Rationale: This follows directly assuming that the proposal in §1.1.2 is accepted.

1.3.5 **Summary:** Remove a default parameter for an overloading of `insert()` defined for sequence containers.

Proposal: In clauses 23.1.5 [lib.vector], 23.1.6 [lib.vector.bool], 23.1.7 [lib.list] and 23.1.8 [lib.deque], remove the default argument `x` for the member function `insert(iterator position, size_type n, const T& x).`

Rationale: The presence of the default argument causes an ambiguity when the sequence is instantiated with `T` == `size_type`.

1.3.6 **Summary:** Remove the implementation restrictions on the `deque` member functions `insert()/push()` and `erase()/pop()` when using and/or popping elements from either end.

Proposal: Replace the first two sentences of 23.1.8.2 [lib.deque.members], paragraph 1 with the following:

> "`insert` and `push` invalidate all the iterators and references to the deque."

Also, replace the first two sentences of 23.1.8.2 [lib.deque.members], paragraph 2 with the following:

> "`erase` and `pop` invalidate all the iterators and references to the deque."

Rationale: Guaranteeing the validity of iterators when manipulating either end of a deque precludes several reasonable implementation techniques.

1.3.7 **Summary:** Impose constraints on the insert() and erase() member functions for associative containers.

Proposal: Modify paragraph 6 of 17.2.2.4.2 [lib.associative.containers] to read as follows (changes underlined):

> "`iterator` of an associative container is of the bidirectional iterator category. <u>`insert` does not affect the validity of iterators and references to the container, and `erase` invalidates only the iterators and references to the erased elements</u>."

Rationale: These constraints are met by well-known implementations of associative containers, and provide additional benefit to users.

1.3.8 **Summary:** Replace implicit conversion functions with alternate names for both reverse iterator adaptors.

Proposal: Replace the conversion functions in In 24.2.1.1 [lib.reverse.bidir.iter] and 24.2.1.2 [lib.reverse.iterator] with member functions called `base()`, defined as follows:

For `reverse_bidirectional_iterator`:

```
BidirectionalIterator base() { return current; }
```

and for `reverse_iterator`:

```
RandomAccessIterator base() { return current; }
```

Rationale: Replacing the conversion functions helps prevent unwanted implicit conversions on the reverse iterator adaptor types.

1.4 Clause 25 (Algorithms Library)

1.4.1 **Summary:** Change the return type of `for_each()` from `void` to `Function`.

Proposal: In 25.1.1 [lib.alg.foreach], change the return type of the `for_each()` function from `void` to `Function`.

Rationale: This change allows the caller to retain any state kept in the function object after the call to `for_each()`.

1.4.2 **Summary:** Add a function `iter_swap()` to exchange the values pointed to by two iterators.

Proposal: Add the following to the end of 25.2.2.1 [lib.swap]:

```
template <class ForwardIterator1, ForwardIterator2>
void iter_swap(ForwardIterator1 a, ForwardIterator2 b);
```

`iter_swap` exchanged values pointed to by the two iterators `a` and `b`.

Rationale: Simply a convenience.

1.4.3 **Summary:** Modify the iterator category for `rotate()` and fix the function specification.

Proposal: In 25.2.10.1 [lib.rotate], change the iterator type to ForwardIterator

```
template <class ForwardIterator>
void rotate(ForwardIterator first, ForwardIterator middle,
            ForwardIterator last);
```

and modify the function specification in paragraph 1 to read as follows (changes underlined):

> "For each non-negative integer `i < (last - first)`, rotate places the element from the position `first + i` into position `first + (i + `<u>`(last - middle)`</u>`) % (last - first)`. <u>`[first, middle)` and `[middle, last)` are valid ranges.</u>"

Rationale: A forward iterator is sufficient for this algorithm; and the current WP description is wrong.

2.0 Editorial Changes

2.1 Clause 17 (Library Introduction)

2.1.1 **Proposal:** Change the first sentence of paragraph 5 of 17.2.2.2 [lib.iterator.types] to read (changes underlined):

> "An iterator `j` is called *reachable* from an iterator `i` <u>if and only if</u> there is a finite sequence of applications of `operator++` to `i` that makes `i == j`."

2.1.2 **Proposal:** Change Table 17, row 1, column 4 in 17.2.2.2.2 [lib.output.iterators] to read:

> "`*a = t` is equivalent to `*X(a) = t`. note: a destructor is assumed"

2.1.3 **Proposal:** Remove the pre- and post-conditions from rows 4 and 4 of Table 17 in 17.2.2.2.2 [lib.output.iterators]. Also change the last condition in Table 17, row 4, column 4 in 17.2.2.2.2 [lib.output.iterators] to read:

> "`&r == &++r`"

2.1.4 **Proposal:** Change the last two conditions in Table 18, row 9, column 4 in 17.2.2.2.3 [lib.forward.iterators] to read:

> "`r == s` and `r` is dereferenceable implies `++r == ++s`. `&r == &++r`"

2.1.5 **Proposal:** In 17.2.2.2.5 [lib.random.access.iterators], Table 20, remove the operational semantics for `b - a`. The current description is incorrect. Place an editorial box noting that correct semantics must be established for this expression.

2.1.6 **Proposal:** In 17.2.2.3 [lib.allocator.types], Table 21 change the note for X::pointer to read:

> “the result of `operator*` of values of `X::pointer` is of `reference`”

Also, change the note for a.max_size() to read:

> “the largest positive value of `X::difference_type`.”

2.2 Clause 20 (General Utilities Library)

2.2.1 **Proposal:** In 20.2 [lib.function.objects], remove the fourth parameter "`b.end()`" from the `transform` example directly following paragraph 3.

2.2.2 **Proposal:** In 20.2.1 [lib.base], remove the inheritance from `empty` for both `unary_function` and `binary_function`.

2.2.3 **Proposal:** In 20.2.5 [lib.negators], remove the inheritance from `restrictor` for both `unary_negate` and `binary_negate`.

2.2.4 **Proposal:** In 20.3.3.4 [lib.destroy], change the implementation of the first destroy function to read:

```
pointer->~T();
```

2.3 Clause 23 (Containers Library)

2.3.1 **Proposal:** In Clause 23.1.7.2 [lib.list.members], modify the descriptions of the splice() member functions (paragraphs 5, 6 and 7) to read as follows (changes underlined):

> “`void splice(iterator position, list<T, Allocator>& x)` inserts the contents of `x` before `position` and `x` becomes empty. It takes constant time. The result is undefined if `&x == this`.”

> “`void splice(iterator position, list<T, Allocator>& x, iterator i)` inserts an element pointed to by `i` from list `x` before `position` and removes the element from `x`. It takes constant time. `i` is a valid dereferenceable iterator of `x`. The result is unchanged if `position == i` or `position == ++i`.”

> “`void splice(iterator position, list<T, Allocator>& x, iterator first, iterator last)` inserts elements in the range `[first, last)` before `position` and removes the elements from `x`. It takes constant time if `&x == this`; otherwise, it takes linear time. `[first, last)` is a valid range in `x`. The result is undefined if `position` is an iterator in the range `[first, last)`.”

2.3.2 **Proposal:** In 23.2.3.1 [lib.map.typedefs], change paragraph 2 so that is reads “`const_iterator` is a constant...”

2.4 Clause 24 (Iterators Library)

2.4.1 **Proposal:** In 24.1.1 [lib.examples], in the example following paragraph 5 change the type of the argument to `iterator_category()` from `T*` to `const T*`.

2.4.2 **Proposal:** In 24.1.2.1 [lib.std.iterator.tags], remove the inheritance from `empty` for each of the iterator tag types.

2.4.3 **Proposal:** In 24.1.2.2 [lib.basic.iterators], remove the inheritance from `empty` for each of the basic iterator templates.

2.4.4 **Proposal:** In 24.1.3 [lib.iterator.operations], change the second to last sentence to read "advance takes a negative argument..."

2.4.5 **Proposal:** In 24.2.1.2 [lib.reverse.iterator], change the second function signature

```
operator-(Distance n) const
```

to read

```
operator-=(Distance n) const.
```

2.4.6 **Proposal:** Add a friend declaration to class `istream_iterator` in 24.3.1 [lib.istream.iterator] as follows:

```
friend bool operator==(const istream_iterator<T, Distance>& x,
                       const istream_iterator<T, Distance>& y);
```

2.4.7 **Proposal:** In 24.3.1 [lib.istream.iterator], modify `istream_iterator` to publicly inherit from its base class (i.e., add missing `public` keyword).

2.4.8 **Proposal:** In 24.3.2 [lib.ostream.iterator], remove the constructor with the signature

```
ostream_iterator(const char* delimiter);
```

2.4.9 **Proposal:** In 24.3.2 [lib.ostream.iterator], change the return type of `operator++(int)` from `ostream_iterator<T>` to `ostream_iterator<T>&`. This is clearly just a typo.

2.4.10 **Proposal:** Change the return types of `operator++(int)` in 24.2.2.1 [lib.back.insert.iterator], 24.2.2.2 [lib.front.insert.iterator] and 24.2.2.3 [lib.insert.iterator] from values to references.

2.4.11 **Proposal:** In 23.1.9 [lib.stack], add a friend declaration for `operator<()` to `stack` as follows:

```
friend bool operator<(const stack<Container>& x,
                      const stack<Container>& y);
```

and add the following definition:

```
template <class Container>
bool operator<(const stack<Container>& x,
               const stack<Container>& y) {
    return x.c < y.c;
}
```

Similarly, add the following declaration to `queue` as follows:

```
friend bool operator<(const queue<Container>& x,
                      const queue<Container>& y);
```

and add the following definition:

```
template <class Container>
bool operator<(const queue<Container>& x,
               const queue<Container>& y) {
    return x.c < y.c;
}
```

2.4.12 **Proposal:** In 24.2.1.1 [lib.reverse.bidir.iter] and 24.2.1.2 [lib.reverse.iterator], add an additional template argument that specifies the return type of `operator*` for these iterator adaptors:

```
template <class BidirectionalIterator, class T,
          class Reference = T&, class Distance = ptrdiff_t>
class reverse_bidirectional_iterator : /*...*/ {
    ...
};

template <class RandomAccesIterator, class T,
          class Reference = T&, class Distance = ptrdiff_t>
class reverse_iterator : /*...*/ {
    ...
};
```

Clearly, this change must be reflected anywhere the class type itself is written, in addition to in the definitions of `operator*()` for each adaptor.

2.4.13 **Proposal:** Add the following friend declaration to 24.2.1.1 [lib.reverse.bidir.iter] for class `reverse_bidirectional_iterator`:

```
friend bool operator==(
          const reverse_bidirectional_iterator<
                BidirectionalIterator, T,
                Reference, Distance>& x,
          const reverse_bidirectional_iterator<
                BidirectionalIterator, T,
                Reference, Distance>& y);
```

and modify the definition accordingly. Also, add the following friend declarations to 24.2.1.2 [lib.reverse.iter] for `reverse_iterator`:

```
friend bool operator<(
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& x,
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& y);

friend bool operator==(
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& x,
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& y);

friend Distance operator-(
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& x,
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& y);

friend reverse_iterator<RandomAccessIterator, T,
                  Reference, Distance> operator+(
            Distance n,
            const reverse_iterator<
                  RandomAccessIterator, T,
                  Reference, Distance>& x);
```

2.5 Clause 25 (Algorithms Library)

2.5.1 Proposal: Change paragraph 2 of 25.1.3 [lib.alg.adjacent.find] to read as follows:

> "Complexity: Exactly `find(first, last, value) - first` applications of the corresponding predicate are done."

2.5.2 Proposal: Add the following complexity clause to 25.1.7 [lib.alg.search]:

> "Complexity: At most `(last1 - first1) * (last2 - first2)` applications of the corresponding predicate are done. The quadratic behavior, however, is extremely unlikely."

2.5.3 Proposal: Add the following to the last sentence of paragraph 3 of 25.2.11 [lib.alg.random.shuffle]:

> "...such that rand <u>takes a positive argument n of distance type of the `RandomAccessIterator` and</u> returns a..."

2.5.4 Proposal: Change the return type of function `stable_partition()` in 25.2.12.1 [lib.stable.partition] from `ForwardIterator` to `BidirectionalIterator`.

2.5.5 **Proposal:** Change the description of the number of comparisons in a `stable_sort()` in 25.3.1.2 [lib.stable.sort], paragraph 2 from `Nlog2N` to `N(logN)`^{2}.

2.5.6 **Proposal:** Change paragraph 2 of 25.3.3.3 [lib.equal.range] to read:

> "Complexity: At most `2 * log(last - first) + 1` comparisons are done."

2.5.7 **Proposal:** Change paragraph 2 of 25.3.3.4 [lib.binary.search] to read:

> "Complexity: At most `log(last - first) + 2`comparisons are done."

2.5.8 **Proposal:** Change paragraph 2 of 25.3.4.2 [lib.inplace.merge] to read:

> "Complexity: When enough additional memory is available, at most `(last - first) - 1` comparisons are performed. If no additional memory is available, an algorithm with `O(NlogN)` complexity may be used."

2.5.9 **Proposal:** Modify the return values of `min()` and `max()` in 25.3.7.1 [lib.min] and 25.3.7.2 [lib.max], to return `const T&` instead of `T`.

2.5.10 **Proposal:** The following sections of the WP contain incorrect implementations of one or more functions. For corrected versions of these functions see [Stepanov95]. The relevant sections of the WP are:

- 20.3.2 [lib.storage.iterator]
- 20.2.5 [lib.negators]
- 20.2.6.1 [lib.binders]
- 20.2.7 [lib.function.pointer.adaptors]
- 20.3.4 [lib.specialized.algorithms]

References

[Koenig94] Koenig, A (ed.), *Working Paper for Draft Proposed International Standard for Information Systems — Programming Language C++*. X3J16/94-0158=WG21/N0545.

[Stepanov94] Stepanov, A. and M. Lee. *The Standard Template Library.* X3J16/94-0140=WG21/N0527.

[Stepanov95] Stepanov, A. and M. Lee. *The Standard Template Library*, Hewlett-Packard Technical Report HPL-95-11.