Author: Matthew Austern austern@google.com
Revision 1: Alan Talbot alan.talbot@teleatlas.com
Document: N2350=07-0210
Date: 2007-07-19
Reference: N2315 = 07-0175

Container insert/erase and iterator constness (Revision 1)

Ten years ago, just before the standard was finished, Dave Abrahams pointed out that the container requirements were too fussy about constness. Table 82, sequence requirements, says that the expression for single-element `sequence insert` is `a.insert(p, t)`, where `p` stands for "a valid iterator to `a`". The requirement tables say similar things for sequence `erase`, and for associative container `erase` and `insert`. This is too strict, because these requirements confuse iterator constness and container constness. The container is being modified, but the iterator, which is used only for positioning, is not. There is no reason to forbid it from being a const iterator.

Making the iterator argument `const_iterator` does raise another question. Currently the return type for the single-element sequence `insert`, both versions of sequence `erase`, single-element associative container `insert`, and both versions of associative container `erase`, is `iterator`. If we change the arguments without making any other changes, then we will have functions with a `const_iterator` argument type and an `iterator` return type. Strictly speaking, this is not a violation of const correctness. It doesn't turn a `const_iterator` into an `iterator`; all it does is generate an `iterator` given a non-const container, and of course there is nothing extraordinary about that. Still, this member function made some committee members uncomfortable enough to look for an alternative fix: overloading these member functions on `iterator` and `const_iterator`, so that the return type matches the argument type.

This problem was never fixed for the sequence containers (`vector`, `list`, and `deque`) or for the associative containers (`map`, `set`, `multimap`, and `multiset`). It was fixed for the unordered associative containers (`unordered_map`, `unordered_set`, `unordered_multimap`, and `unordered_multiset`). Based on the results of an LWG straw poll, the unordered associative containers use the second alternative: two overloaded member functions.

We ought to fix this problem for the sequences containers and the associative containers. It would be silly to release two standards with this known defect, or to treat this issue differently in the unordered associative containers than in the sequences and the associative containers.

We have two options for this fix: A) apply the same fix to sequences and associative containers as we already did for unordered associative containers, or B) consistently change all of the containers to use a single member function with a `const_iterator` argument and an `iterator` return type.

I prefer option B, because it's simpler: I believe that these overloads are cumbersome and add no const correctness value. The first version of this document

provided wording for both options because option A was more consistent with the straw poll from the 2004 Redmond meeting.

At the 2007 Toronto meeting the LWG discussed this and agreed that Option B was correct after all. The group also strongly believed that list::splice should also have const_iterator arguments. Revision 1 of this document has been updated to eliminate language for Option A and add language for splice.

Proposed Wording

In 23.1.1 [lib.sequence.reqmts] paragraph 3, replace "`p` denotes a valid iterator to `a`, `q` denotes a valid dereferenceable iterator to `a`, `[q1, q2)` denotes a valid range in `a`" with "`p` denotes a valid const iterator to `a`, `q` denotes a valid dereferenceable const iterator to `a`, `[q1, q2)` denotes a valid range of const iterators in `a`"

In 23.1.2 [lib.associative.reqmts] paragraph 7, replace "`p` is a valid iterator to `a`, `q` is a valid dereferenceable iterator to `a`, `[q1, q2)` is a valid range in `a`" with "`p` is a valid const iterator to `a`, `q` is a valid dereferenceable const iterator to `a`, `[q1, q2)` is a valid range of const iterators in `a`".

In the class synopses of 23.2.2 [lib.deque], and 23.2.5 [lib.vector],
and in the member function descriptions in 23.2.2.3 [lib.deque.modifiers], and 23.2.5.4 [lib.vector.modifiers], change the signatures of insert and erase to

```
iterator insert(const_iterator position, const T& x);
void insert(const_iterator position, size_type n, const T& x);
template <class InputIterator>
  void insert(const_iterator position, InputIterator first,
InputIterator last);
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
```

In the class synopsis of 23.2.3 [lib.list], and in the member function description in 23.2.3.3 [lib.list.modifiers], change the signatures of insert, erase and splice to

```
iterator insert(const_iterator position, const T& x);
void insert(const_iterator position, size_type n, const T& x);
template <class InputIterator>
  void insert(const_iterator position, InputIterator first,
InputIterator last);
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
void splice(const_iterator position, list<T,Allocator>&& x );
void splice(const_iterator position,
      list<T,Allocator>&& x, const_iterator i);
void splice(const_iterator position,
      list<T,Allocator>&& x,
      const_iterator first, const_iterator last);
```

In the class synopses in 23.3.1 [lib.map], 23.3.2 [lib.multimap], 23.3.3 [lib.set], and 23.3.4 [lib.multiset], change the signatures

```
iterator insert(iterator position, const value_type& x);
iterator erase(iterator position);
iterator erase(iterator first, iterator last);
```

to

```
iterator insert(const_iterator position, const value_type& x);
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
```

In 23.1.3 [lib.unord.req] paragraph 9, change "`p` and `q2` are valid iterators to `a`, `q` and `q1` are valid dereferenceable iterators to `a`, `[q1, q2)` is a valid range in `a`, `r` and `r1` are valid dereferenceable const iterators to `a`, `r2` is a valid const iterator to `a`, `[r1, r2)` is a valid range in `a`" to "`p` and `q2` are valid const iterators to `a`, `q` and `q1` are valid dereferenceable const iterators to `a`, `[q1, q2)` is a valid range in `a`".

In table 92 (unordered associative container requirements), delete the rows for `a.insert(r, t)`, `a.erase(r)`, and `a.erase(r1, r2)`.

In the class synopses in 23.4.1 [lib.unord.map], 23.4.2 [lib.unord.multimap], 23.4.3 [lib.unord.set], and 23.4.4 [lib.unord.multiset], replace the signatures

```
iterator insert(iterator hint, const value_type& obj);
const_iterator insert(const_iterator hint, const value_type& obj);
iterator erase(iterator position);
const_iterator erase(const_iterator position);
iterator erase(iterator first, iterator last);
const_iterator erase(const_iterator first, const_iterator last);
```

with

```
iterator insert(const_iterator hint, const value_type& obj);
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
```