Simplification of reverse iterator adaptors

Matthew Austern (`austern@sgi.com`) Angelika Langer (`langer@roguewave.com`)
Alexander Stepanov (`stepanov@mti.sgi.com`)

May 29, 1996

Abstract

Recent language and library changes make it possible to simplify §24.3.1 [lib.reverse.iterators]. It is possible to eliminate the class `reverse_bidirectional_iterator` entirely, without loss of any functionality.

1 Types of reverse iterators

Reverse iterator adaptors, defined in §24.3.1 [lib.reverse.iterators] of the January WP, allow iteration through a range in reverse order. A reverse iterator has some underlying iterator (accessible through the member function `base()`); incrementing a reverse iterator is implemented by decrementing its corresponding base iterator, and *vice versa*.

Decrementing an iterator is not part of the requirements for input iterators, output iterators, or forward iterators, so reverse iterator adaptors are defined only for underlying iterators that are bidirectional iterators or random access iterators. The template class `reverse_bidirectional_iterator` is to be instantiated with a bidirectional iterator type, and the template class `reverse_iterator` is to be instantiated with a random access iterator type.

In fact, the two classes are remarkably similar. The differences are as follows.

- The first template paramater is called `BidirectionalIterator` for `reverse_bidirectional_iterator` and `RandomAccessIterator` for `reverse_iterator`, thus implying (although in fact it is never stated explicitly) that they must satisfy, respectively, the requirements of bidirectional iterators and random access iterators.
- `reverse_bidirectional_iterator` is derived from `iterator<bidirectional_iterator_tag, T, Distance>` (in the January WP it is derived from `bidirectional_iterator<T,Distance>`, but this was changed by the `iterator_traits` proposal), while `reverse_iterator` is derived from `iterator<random_access_iterator_tag, T, Distance>`.
- `reverse_iterator` contains definitions for operators `+`, `+=`, `-`, `-=`, and `[]`, but `reverse_bidirectional_iterator` does not. These operators are supported by random access iterators, but not by bidirectional iterators.

The first of these items is essentially a documentation issue, while the other two are small enough that it is reasonable to merge the two classes into a single class `reverse_iterator`. Its template parameter `Iterator` could be either a bidirectional iterator or a random access iterator, and its base class would be `iterator<iterator_traits<Iterator>::iterator_category, T, Distance>`. The class would contain operators `+`, `+=`, `-`, `-=`, and `[]`.

Since unused member functions of template classes are not instantiated, it would not be an error to instantiate `reverse_iterator` with a bidirectional iterator so long as only those operations supported by bidirectional iterators are used. We therefore propose to eliminate the class `reverse_bidirectional_iterator`, and to allow the template parameter of `reverse_iterator` to be a bidirectional iterator. Note that this is essentially the *status quo* as far as the class `reverse_iterator` is concerned. The only real change that we are proposing is to change its iterator category tag from always being `random_access_iterator_tag` to matching the category tag of the iterator type with which it is instantiated.

2 Simplification of `reverse_iterator`

An entirely separate issue is that `reverse_iterator` does not actually have a single template parameter, but rather five. The complete declaration in the January WP is

```
template<class RandomAccessIterator,
         class T,
         class Reference = T&,
         class Pointer = T*,
         class Distance = ptrdiff_t>.
```

The `iterator_traits` proposal added `iterator_traits<RandomAccessIterator>::value_type` as a default for the template parameter `T`, but made no other changes. In the absence of `iterator_traits`, the additional four template parameters are necessary: they cannot be derived from `RandomAccessIterator`. The `iterator_traits` proposal, however, removes this necessity.

Keeping these four parameters makes `reverse_iterator` more flexible, but we know of no situation where this additional flexibility is actually of any use. We don't know of any reason for defining a reverse iterator whose value, reference, pointer, and distance types are different from those of the underlying iterator. In the interests of simplicity, we therefore propose to change `reverse_iterator` to be a template class parameterized only by the iterator type.

Iterator traits provide `value_type` and `distance_type` but not reference or pointer types; the most sensible way to deal with this problem is simply to add them to `iterator_traits`. This implies that they should also be added to the base struct `iterator`, since the whole purpose of `iterator` is to provide a set of typedefs used by `iterator_traits`.

3 Working paper changes

The basic changes are the elimination of `reverse_bidirectional_iterator`, the elimination of `reverse_iterator`'s four extra template parameters, and the addition of extra typedefs to `iterator_traits` and `iterator`. A few other changes follow mechanically: the WP must be changed in every place where one of those classes is mentioned.

3.1 Elimination of `reverse_bidirectional_iterator`

- Strike §24.3.1.1 [lib.reverse.bidir.iter] and §24.3.1.2 [lib.reverse.bidir.iter.ops]. Remove mention of the class `reverse_bidirectional_iterator` from the library introduction.

3.2 Changes to `iterator_traits` and `iterator`

- Add typedefs `pointer` and `reference` to `iterator_traits`, and to the specialization of `iterator_traits` for pointers.
- Change the declaration of `iterator` from

```
template<class Category, class T, class Distance = ptrdiff_t>
struct iterator
{
  typedef T value_type;
  typedef Distance distance_type;
  typedef Category iterator_category;
};
```

  to

```
template<class Category, class T, class Distance = ptrdiff_t,
         class Pointer = T*, class Reference = T&>
struct iterator
{
  typedef T value_type;
  typedef Distance distance_type;
  typedef Pointer pointer;
  typedef Reference reference;
  typedef Category iterator_category;
};.
```

3.3 Changes to `reverse_iterator`

- Change the declaration of `reverse_iterator` so that it only has a single template parameter, `Iterator`.
- Change `reverse_iterator` so that its base class is

```
iterator<iterator_traits<Iterator>::iterator_category,
         iterator_traits<Iterator>::value_type,
         iterator_traits<Iterator>::distance_type,
         iterator_traits<Iterator>::pointer,
         iterator_traits<Iterator>::reference>.
```

- Throughout §24.3.1.3 [lib.reverse.iterator] and §24.3.1.4 [lib.reverse.iter.ops], replace `RandomAccessIterator` by `Iterator`.

- Add a **requirements** section saying that the template argument Iterator must be a bidirectional iterator or a random access iterator and that, additionally, the operators +, +=, -, -=, and [] have the requirement that Iterator must be a random access iterator.
- In §21.1.1.3 [lib.basic.string], §23.2.2 [lib.deque], §23.2.3 [lib.list], and §23.2.5 [lib.vector], §23.2.6 [lib.vector.bool], §23.3.1 [lib.map], §23.3.2 [lib.multimap], §23.3.3 [lib.set], and §23.3.4 [lib.multiset], change the reverse_iterator and const_reverse_iterator typedefs so that they only take the single template argument iterator and reverse_iterator, respectively.
- In table 62, in §23.1 [lib.container.requirements], change the assertion column entries for X::reverse_iterator and X::const_reverse_iterator to reverse_iterator<iterator> and reverse_iterator<const_iterator>, respectively.

3.4 Changes to other iterators

- In §20.4. [lib.memory] change the base class of raw_storage_iterator from iterator<output_iterator_tag, void, void> to iterator<output_iterator_tag, void, void, void, void>.
- In §24.3.2. [lib.insert.iterators] change the base classes of back_insert_iterator, front_insert_iterator and insert_iterator from iterator<output_iterator_tag, void, void> to iterator<output_iterator_tag, void, void, void, void>.
- In §24.4.2 [lib.ostream.iterator] and §24.4.4 [lib.ostreambuf.iterator], change the base classes of ostream_iterator and ostreambuf_iterator from iterator<output_iterator_tag, void, void> to iterator<output_iterator_tag, void, void, void, void>.
- In §24.4.1 [lib.istream.iterator] change the base class of istream_iterator from iterator<input_iterator_tag, T, Distance> to iterator<input_iterator_tag, T,Distance, const T*, const T&>.
- In §24.4.3 [lib.istreambuf.iterator] change the base class of class istreambuf_iterator from iterator<input_iterator_tag, charT, Distance> to iterator<input_iterator_tag, charT, Distance, charT*, charT&>.