

Reverse Iterator

Author: David Abrahams, Jeremy Siek, Thomas Witt
Contact: dave@boost-consulting.com, jsiek@osl.iu.edu, witt@ive.uni-hannover.de
Organization: Boost Consulting, Indiana University Open Systems Lab, University of Hanover [Institute for Transport Railway Operation and Construction](#)
Date: 2004-11-01
Copyright: Copyright David Abrahams, Jeremy Siek, and Thomas Witt 2003.

abstract: The reverse iterator adaptor iterates through the adapted iterator range in the opposite direction.

Table of Contents

[reverse_iterator synopsis](#)
[reverse_iterator requirements](#)
[reverse_iterator models](#)
[reverse_iterator operations](#)
[Example](#)

reverse_iterator synopsis

```
template <class Iterator>
class reverse_iterator
{
public:
    typedef iterator_traits<Iterator>::value_type value_type;
    typedef iterator_traits<Iterator>::reference reference;
    typedef iterator_traits<Iterator>::pointer pointer;
    typedef iterator_traits<Iterator>::difference_type difference_type;
    typedef /* see below */ iterator_category;

    reverse_iterator() {}
    explicit reverse_iterator(Iterator x) ;

    template<class OtherIterator>
    reverse_iterator(
        reverse_iterator<OtherIterator> const& r
        , typename enable_if_convertible<OtherIterator, Iterator>::type* = 0 // exposition
    );
    Iterator const& base() const;
```

```

    reference operator*() const;
    reverse_iterator& operator++();
    reverse_iterator& operator--();
private:
    Iterator m_iterator; // exposition
};

```

If `Iterator` models Random Access Traversal Iterator and Readable Lvalue Iterator, then `iterator_category` is convertible to `random_access_iterator_tag`. Otherwise, if `Iterator` models Bidirectional Traversal Iterator and Readable Lvalue Iterator, then `iterator_category` is convertible to `bidirectional_iterator_tag`. Otherwise, `iterator_category` is convertible to `input_iterator_tag`.

reverse_iterator requirements

`Iterator` must be a model of Bidirectional Traversal Iterator. The type `iterator_traits<Iterator>::reference` must be the type of `*i`, where `i` is an object of type `Iterator`.

reverse_iterator models

A specialization of `reverse_iterator` models the same iterator traversal and iterator access concepts modeled by its `Iterator` argument. In addition, it may model old iterator concepts specified in the following table:

| If I models | then reverse_iterator<I> models |
|--|---------------------------------|
| Readable Lvalue Iterator, Bidirectional Traversal Iterator | Bidirectional Iterator |
| Writable Lvalue Iterator, Bidirectional Traversal Iterator | Mutable Bidirectional Iterator |
| Readable Lvalue Iterator, Random Access Traversal Iterator | Random Access Iterator |
| Writable Lvalue Iterator, Random Access Traversal Iterator | Mutable Random Access Iterator |

`reverse_iterator<X>` is interoperable with `reverse_iterator<Y>` if and only if `X` is interoperable with `Y`.

reverse_iterator operations

In addition to the operations required by the concepts modeled by `reverse_iterator`, `reverse_iterator` provides the following operations.

```
reverse_iterator();
```

Requires: `Iterator` must be Default Constructible.

Effects: Constructs an instance of `reverse_iterator` with `m_iterator` default constructed.

```
explicit reverse_iterator(Iterator x);
```

Effects: Constructs an instance of `reverse_iterator` with `m_iterator` copy constructed from `x`.

```
template<class OtherIterator>
reverse_iterator(
```

```

    reverse_iterator<OtherIterator> const& r
    , typename enable_if_convertible<OtherIterator, Itera-
tor>::type* = 0 // exposition
);

```

Requires: OtherIterator is implicitly convertible to Iterator.

Effects: Constructs instance of reverse_iterator whose m_iterator subobject is constructed from y.base().

```
Iterator const& base() const;
```

Returns: m_iterator

```
reference operator*() const;
```

Effects:

```
Iterator tmp = m_iterator;
return *--tmp;
```

```
reverse_iterator& operator++();
```

Effects: --m_iterator

Returns: *this

```
reverse_iterator& operator--();
```

Effects: ++m_iterator

Returns: *this

```
template <class BidirectionalIterator>
reverse_iterator<BidirectionalIterator>n
make_reverse_iterator(BidirectionalIterator x);
```

Returns: An instance of reverse_iterator<BidirectionalIterator> with a current constructed from x.

Example

The following example prints an array of characters in reverse order using reverse_iterator.

```

char letters_[] = "hello world!";
const int N = sizeof(letters_)/sizeof(char) - 1;
typedef char* base_iterator;
base_iterator letters(letters_);
std::cout << "original sequence of letters:\t\t\t" << let-
ters_ << std::endl;

boost::reverse_iterator<base_iterator>
    reverse_letters_first(letters + N),
    reverse_letters_last(letters);

std::cout << "sequence in reverse order:\t\t\t";
std::copy(reverse_letters_first, reverse_letters_last,

```

```
        std::ostream_iterator<char>(std::cout));
std::cout << std::endl;

std::cout << "sequence in double-reversed (normal) order:\t";
std::copy(boost::make_reverse_iterator(reverse_letters_last),
          boost::make_reverse_iterator(reverse_letters_first),
          std::ostream_iterator<char>(std::cout));
std::cout << std::endl;
```

The output is:

```
original sequence of letters:          hello world!
sequence in reverse order:            !dlrow olleh
sequence in double-reversed (normal) order:  hello world!
```

The source code for this example can be found [here](#).