<iosfwd>
<iostream>
<istream>
<iterator>
advance
back_insert_iterator Class
back_inserter
begin
bidirectional_iterator_tag Struct
cbegin
cend
checked_array_iterator Class
distance
end
forward_iterator_tag Struct
front_insert_iterator Class
front_inserter
input_iterator_tag Struct
insert_iterator Class
inserter
istream_iterator Class
istreambuf_iterator Class
iterator Struct
iterator_traits Struct
make_checked_array_iterator
make_move_iterator
make_unchecked_array_iterator
move_iterator Class
next
operator!=
operator==
operator<
operator<=
operator>
operator>=
operator+
operator-
ostream_iterator Class
ostreambuf_iterator Class
output_iterator_tag Struct
prev
random_access_iterator_tag Struct
reverse_iterator Class
unchecked_array_iterator Class
<limits>
<list>
<locale>
<map>
<memory>
<mutex>
<new>
<numeric>
<ostream>
<queue>
<random>
<ratio>
<regex>
<scoped_allocator>
<set>
<shared_mutex>
<sstream>
<stack>
<stdexcept>
<streambuf>
<string>

<iterator>

Visual Studio 2015 | Other Versions

Defines the iterator primitives, predefined iterators and stream iterators, as well as several supporting templates. The predefined iterators include insert and reverse adaptors. There are three classes of insert iterator adaptors: front, back, and general. They provide insert semantics rather than the overwrite semantics that the container member function iterators provide.

Any suggestions?
Print
Export (0)
Share

IN THIS ARTICLE
Syntax
Remarks
See Also

Syntax

```
#include <iterator>
```

Remarks

Iterators are a generalization of pointers, abstracting from their requirements in a way that allows a C++ program to work with different data structures in a uniform manner. Iterators act as intermediaries between containers and generic algorithms. Instead of operating on specific data types, algorithms are defined to operate on a range specified by a type of iterator. Any data structure that satisfies the requirements of the iterator may then be operated on by the algorithm. There are five types or categories of iterator, each with its own set of requirements and resulting functionality:

- Output: forward moving, may store but not retrieve values, provided by ostream and inserter.
- Input: forward moving, may retrieve but not store values, provided by istream.
- Forward: forward moving, may store and retrieve values.
- Bidirectional: forward and backward moving, may store and retrieve values, provided by list, set, multiset, map, and multimap.
- Random access: elements accessed in any order, may store and retrieve values, provided by vector, deque, string, and array.

Iterators that have greater requirements and so more powerful access to elements may be used in place of iterators with fewer requirements. For example, if a forward iterator is called for, then a random-access iterator may used instead.

Visual Studio has added extensions to C++ Standard Library iterators to support a variety of debug mode situations for checked and unchecked iterators. For more information, see Safe Libraries: C++ Standard Library.

Functions

advance	Increments an iterator by a specified number of positions.
back_inserter	Creates an iterator that can insert elements at the back of a specified container.
begin	Retrieves an iterator to the first element in a specified container.
cbegin	Retrieves a constant iterator to the first element in a specified container.
cend	Retrieves a constant iterator to the element that follows the last element in the specified container.
distance	Determines the number of increments between the positions addressed by two iterators.
end	Retrieves an iterator to the element that follows the last element in the specified container.
front_inserter	Creates an iterator that can insert elements at the front of a specified container.
inserter	An iterator adaptor that adds a new element to a container at a specified point of insertion.
make_checked_array_iterator	Creates a checked_array_iterator that can be used by other algorithms. **Note** This function is a Microsoft extension of the Standard C++ Library. Code implemented by using this function is not portable to C++ Standard build environments that do not support this Microsoft extension.
make_move_iterator	Returns a move iterator containing the provided iterator as its stored base iterator.
make_unchecked_array_iterator	Creates an unchecked_array_iterator that can be used by other algorithms. **Note** This function is a Microsoft extension of the Standard C++ Library. Code implemented by using this function is not portable to C++ Standard build environments that do not support this Microsoft extension.
next	Iterates a specified number of times and returns the new iterator position.
prev	Iterates in reverse a specified number of times and returns the new iterator position.

Operators

operator!=	Tests if the iterator object on the left side of the operator is not equal to the iterator object on the right side.
operator==	Tests if the iterator object on the left side of the operator is equal to the iterator object on the right side.
operator<	Tests if the iterator object on the left side of the operator is less than the iterator object on the right side.
operator<=	Tests if the iterator object on the left side of the operator is less than or equal to the iterator object on the right side.
operator>	Tests if the iterator object on the left side of the operator is greater than the iterator object on the right side.
operator>=	Tests if the iterator object on the left side of the operator is greater than or equal to the iterator object on the right side.
operator+	Adds an offset to an iterator and returns the new **reverse_iterator** addressing the inserted element at the new offset position.
operator-	Subtracts one iterator from another and returns the difference.

Classes

back_insert_iterator	The template class describes an output iterator object. It inserts elements into a container of type **Container**, which it accesses through the protected **pointer** object it stores called container.
bidirectional_iterator_tag	A class that provides a return type for an **iterator_category** function that represents a bidirectional iterator.
checked_array_iterator	A class that accesses an array using a random access, checked iterator. **Note** This class is a Microsoft extension of the Standard C++ Library. Code implemented by using this function is not portable to C++ Standard build environments that do not support this Microsoft extension.
forward_iterator_tag	A class that provides a return type for an **iterator_category** function that represents a forward iterator.
front_insert_iterator	The template class describes an output iterator object. It inserts elements into a container of type **Container**, which it accesses through the protected **pointer** object it stores called container.
input_iterator_tag	A class that provides a return type for an **iterator_category** function that represents an input iterator.
insert_iterator	The template class describes an output iterator object. It inserts elements into a container of type **Container**, which it accesses through the protected **pointer** object it stores called container. It also stores the protected **iterator** object, of class **Container::iterator**, called **iter**.
istream_iterator	The template class describes an input iterator object. It extracts objects of class **Ty** from an input stream, which it accesses through an object it stores, of type pointer to **basic_istream**<**Elem**, **Tr**>.
istreambuf_iterator	The template class describes an input iterator object. It inserts elements of class **Elem** into an output stream buffer, which it accesses through an object it stores, of type **pointer** to **basic_streambuf**<**Elem**, **Tr**>.
iterator	The template class is used as a base type for all iterators.
iterator_traits	A template helper class providing critical types that are associated with different iterator types so that they can be referred to in the same way.
move_iterator	A **move_iterator** object stores a random-access iterator of type **RandomIterator**. It behaves like a random-access iterator, except when dereferenced. The result of **operator*** is implicitly cast to `value_type&&`: to make an **rvalue reference**.
ostream_iterator	The template class describes an output iterator object. It inserts objects of class **Type** into an output stream, which it accesses through an object it stores, of type **pointer** to **basic_ostream**<**Elem**, **Tr**>.
ostreambuf_iterator Class	The template class describes an output iterator object. It inserts elements of class **Elem** into an output stream buffer, which it accesses through an object it stores, of type pointer to **basic_streambuf**<**Elem**, **Tr**>.
output_iterator_tag	A class that provides a return type for **iterator_category** function that represents an output iterator.
random_access_iterator_tag	A class that provides a return type for **iterator_category** function that represents a random-access iterator.
reverse_iterator	The template class describes an object that behaves like a random-access iterator, only in reverse.
unchecked_array_iterator	A class that accesses an array using a random access, unchecked iterator. **Note** This class is a Microsoft extension of the Standard C++ Library. Code implemented by using this function is not portable to C++ Standard build environments that do not support this Microsoft extension.

See Also

C++ Standard Library Header Files
Thread Safety in the C++ Standard Library
Standard Template Library