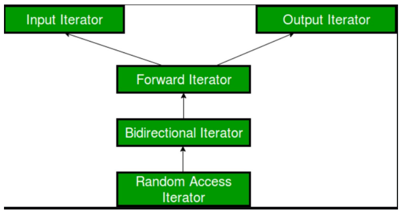
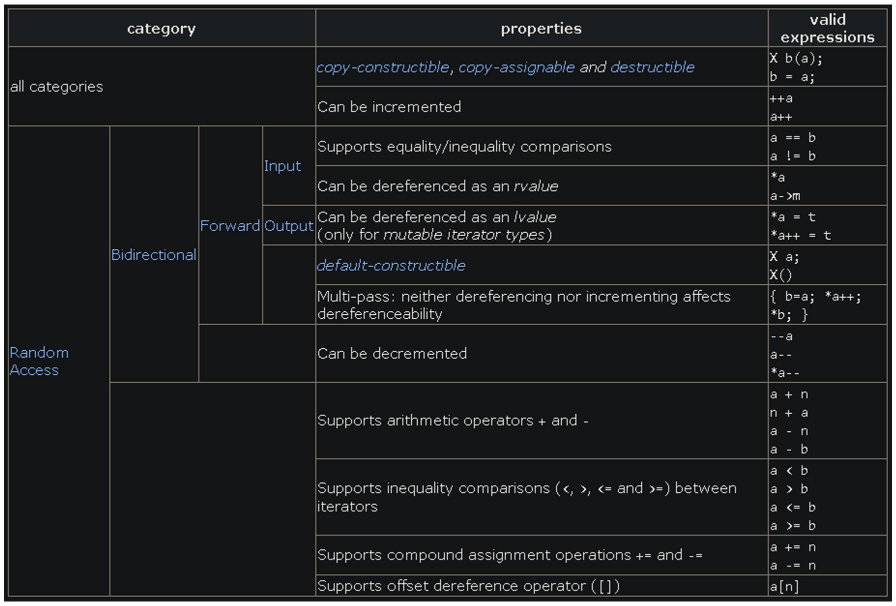
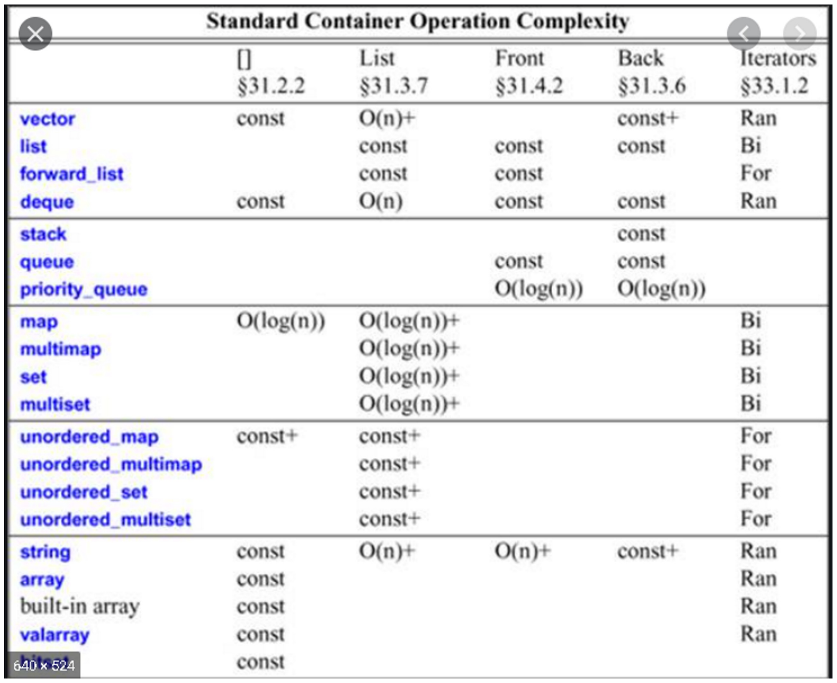
Iterators

09 June 2021 07:07





category				properties	valid expressions
all categories				*copy-constructible*, *copy-assignable* and *destructible*	`X b(a);` `b = a;`
				Can be incremented	`++a` `a++`
Random Access	Bidirectional	Forward	Input	Supports equality/inequality comparisons	`a == b` `a != b`
				Can be dereferenced as an *rvalue*	`*a` `a->m`
			Output	Can be dereferenced as an *lvalue* (only for *mutable iterator types*)	`*a = t` `*a++ = t`
				default-constructible	`X a;` `X()`
				Multi-pass: neither dereferencing nor incrementing affects dereferenceability	`{ b=a; *a++; *b; }`
				Can be decremented	`--a` `a--` `*a--`
				Supports arithmetic operators + and -	`a + n` `n + a` `a - n` `a - b`
				Supports inequality comparisons (<, >, <= and >=) between iterators	`a < b` `a > b` `a <= b` `a >= b`
				Supports compound assignment operations += and -=	`a += n` `a -= n`
				Supports offset dereference operator ([])	`a[n]`



Standard Container Operation Complexity

	[] §31.2.2	List §31.3.7	Front §31.4.2	Back §31.3.6	Iterators §33.1.2
vector	const	O(n)+		const+	Ran
list		const	const	const	Bi
forward_list		const	const		For
deque	const	O(n)	const	const	Ran
stack				const	
queue			const	const	
priority_queue			O(log(n))	O(log(n))	
map	O(log(n))	O(log(n))+			Bi
multimap		O(log(n))+			Bi
set		O(log(n))+			Bi
multiset		O(log(n))+			Bi
unordered_map	const+	const+			For
unordered_multimap		const+			For
unordered_set		const+			For
unordered_multiset		const+			For
string	const	O(n)+	O(n)+	const+	Ran
array	const				Ran
built-in array	const				Ran
valarray	const				Ran
640 × 524	const				

https://www.geeksforgeeks.org/random-access-iterators-in-cpp/
https://www.cplusplus.com/reference/iterator/

1. Random-Access iterator

access elements at any arbitrary offset position by adding offset -> vector, deque
All pointer types are also valid random-access iterators.

2. Bidirectional iterator

list, map, multimap, set and multiset

3. ## Forward iterator

   Forward list

4. ## Input iterator

   Dereferenced as r-value .. Can be read from but not written into.
   Sequential input operations, where each value pointed by the iterator is read-only once and then the iterator is incremented.

5. ## Output iterator

   Dereferenced as l-value.. Can be written into but not read from assigned values in a sequence, but cannot be used to access values,

6. ## For loops to be used with any container

```
for (int number : array)
for (auto number : array) //use auto keyword
for (auto& number : array) //use references since copying each element is expensive
```

   All types that have begin and end member functions or can be used with std::begin and std::end are usable in range-based for-loops.

   C-style fixed arrays can be used with

   ?? Dynamic arrays don't work though, because there is no std::end function for them ???

7. ## Iterator is an object designed to traverse through a container (or) access its elements

   - Operator*
   - Operator++
   - Operator== (* needed)
   - Operator!= (* needed)
   - Operator= (* may be needed)

   End points to element just past end

   1. begin(), end(), cbegin(), cend(),

   Container<type>::iterator provides a read/write iterator ..

```
    std::array<int>::iterator it;
    it = arr.begin();
```

   Container<type>::const_iterator provides a read-only iterator

```
    std::vector<int>::const_iterator it; // declare a read-only iterator
    it = vect.cbegin();                  // assign it to the start of the vector
    while (it != vect.cend())            // while it hasn't reach the end
```

8. ## #include <iterator>

   For std::begin(..) and std::end(..)
   For std::size -> can be used to determine the length of arrays, std::array, std::vector()
   std::distance(itr1, itr2) -> number of elements between itr1 and itr2
   std::advance(itr, 5) -> advance itr by 5 elements
   Nitr = std::next(itr, 5) -> returns nitr which is basically itr advanced by 5
   Pitr = std::prev(itr, 5)

9. ## Iterator adaptors - insert, stream, reverse, move

10. ## Insert iterator -> special output iterator which allows algos to insert elements at specific positions instead of overwrite

   Inserts new elements into container in successive locations starting at the position pointed by it.. Needs container to have insert method.

```
std::list<int> foo;
std::list<int>::iterator it = foo.begin();

std::insert_iterator< std::list<int> > insert_it (foo,it);
std::copy (bar.begin(),bar.end(),insert_it);

operator=
```

   Inserts a new element in the container at the position pointed by the iterator passed on construction, initializing the new element with the argument.

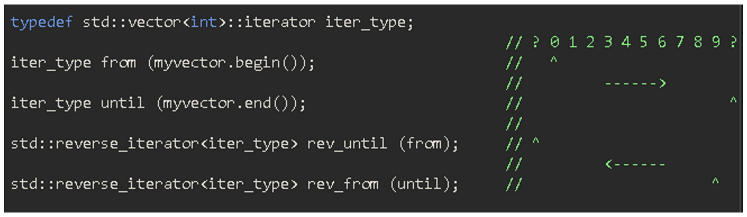
11. ## Inserter

```
std::list<int> foo;
std::list<int>::iterator it = foo.begin();

std::copy (bar.begin(),bar.end(),std::inserter(foo,it));
```

12. ## Reverse iterator -> bidirectional or random-access iterator that reverse iterates

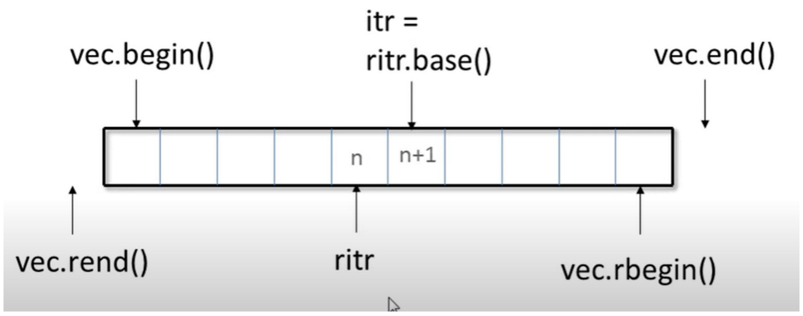
   This class reverses the direction in which a bidirectional or random-access iterator iterates through a range

```
std::reverse_iterator<container's iterator type> revItr (itrInContainer)
```

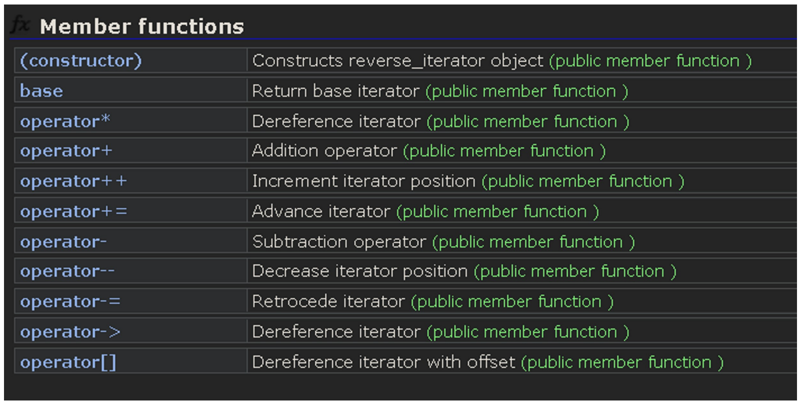


```
typedef std::vector<int>::iterator iter_type;
                                                        // ? 0 1 2 3 4 5 6 7 8 9 ?
iter_type from (myvector.begin());                      //     ^
                                                        //         ------>
iter_type until (myvector.end());                       //                       ^
                                                        //
std::reverse_iterator<iter_type> rev_until (from);      // ^
                                                        //         <------
std::reverse_iterator<iter_type> rev_from (until);      //                     ^
```

```
std::reverse_iterator<std::vector<int>::iterator> ritr;
std::vector<int>::reverse_iterator ritr;

for (ritr = vec.rbegin(); ritr != vec.rend(); ritr++) {}
ritr = std::vector<int>::reverse_iterator(vec.end())
```



Reverse_iterator has always an offset of -1 with respect to its base iterator

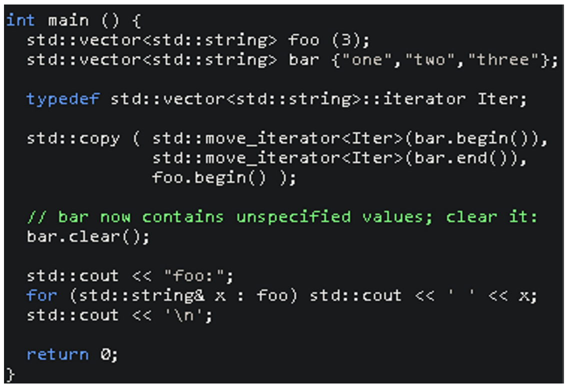


Member functions

(constructor)	Constructs reverse_iterator object (public member function)
base	Return base iterator (public member function)
operator*	Dereference iterator (public member function)
operator+	Addition operator (public member function)
operator++	Increment iterator position (public member function)
operator+=	Advance iterator (public member function)
operator-	Subtraction operator (public member function)
operator--	Decrease iterator position (public member function)
operator-=	Retrocede iterator (public member function)
operator->	Dereference iterator (public member function)
operator[]	Dereference iterator with offset (public member function)

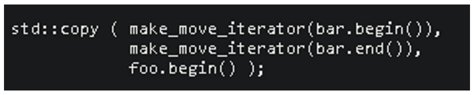
13. Move iterator

This class adapts an iterator so that dereferencing it produces rvalue references (as if std::move was applied), while all other operations behave as in the regular iterator.



```
int main () {
  std::vector<std::string> foo (3);
  std::vector<std::string> bar {"one","two","three"};

  typedef std::vector<std::string>::iterator Iter;

  std::copy ( std::move_iterator<Iter>(bar.begin()),
              std::move_iterator<Iter>(bar.end()),
              foo.begin() );

  // bar now contains unspecified values; clear it:
  bar.clear();

  std::cout << "foo:";
  for (std::string& x : foo) std::cout << ' ' << x;
  std::cout << '\n';

  return 0;
}
```

Constructs a move_iterator object from iterator argument



```
std::copy ( make_move_iterator(bar.begin()),
            make_move_iterator(bar.end()),
            foo.begin() );
```

14. Inserting using iterator

When inserting at position iter, the new element is placed just before the element pointed by iter

The resulting sequence is [begin,iter)[new][iter,end).

15. F
I
16. F
I
17. F
I
18. F
I
19. F
I
20. F
I
21. D
I
22. F
I
23. F
I
24. F
I
25. F
26. F
I
27. F
I
28. F
I
29. F
I
30. F
I
31. F
I
32. D
I
33. F
I
34. F
I
35. F
I
36. F
37. F
I
38. F
I
39. F
I
40. F
I
41. F
I
42. F
I
43. D
I
44. F
I
45. F
I
46. F
I
47. F

48. F
    I
49. F
    I
50. F
    I
51. F
    I
52. F
    I