ITERATORS

Iterator definitions:

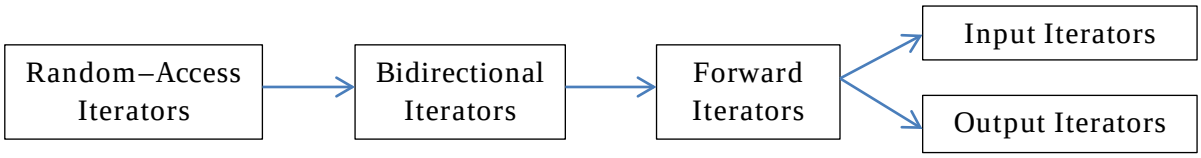
In C++, an iterator is an object that, pointing to some element in a range of elements (such as an array or a container), has the ability to iterate through the elements of that range using a set of operators (at least, the increment (++) and dereference (*) operators).

The most obvious form of an iterator is a pointer: A pointer can point to elements in an array, and can iterate through them using the increment operator (++). But other forms of iterators exist. For example, each container type (such as a vector) has a specific iterator type designed to iterate through its elements in an efficient way.

Notice that while a pointer is a form of iterator, not all iterators have the same functionality a pointer has. To distinguish between the requirements than an iterator has for a specific algorithm, five different iterator categories exist:

Iterator categories:

Iterators are classified in five categories depending on the functionality they implement:



In this graph, each iterator category implements the functionalities of all categories to its right.

Input and output iterators are the most limited types of iterators specialized in performing only sequential input or output operations.

Forward iterators have all the functionality of input and output iterators, although they are limited to one direction in which to iterate through a range.

Bidirectional iterators can iterate in both directions. All standard containers support at least bidirectional iterator types.

Random–access iterators implement all the functionalities of bidirectional iterators, plus, they have the ability to access ranges non-sequentially: offsets can be directly applied to these iterators without iterating through all the elements between. This provides these iterators with the same functionality as standard pointers (pointers are iterators of this category).

The characteristics of each category of iterators are:

category	characteristics	valid expressions
all categories	can be copied and copy-constructed	X b(a)

					b = a
				can be incremented	++a a++ *a++
Random–Access	Bidirectional	Forward	Input	accepts equality/inequality comparisons	a == b a != b
				can be dereferenced as an rvalue	*a a–>m
			Output	can be dereferenced to be the left side of an assignment operation	*a = t *a++ = t
				can be default-constructed	X a X()
				can be decremented	—a a— *a—
				supports arithmetic operators + and –	a + n n + a n – a 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]

Where X is an iterator type, a and b are objects of this iterator type, t is an object of the type pointed by the iterator type, and n is an integer value.

Random–access iterators have all characteristics. Bidirectional iterators have a subset of random–access iterators. Forward iterators have a subset of bidirectional iterators. And input and output iterators have each their own subset of forward iterators.

THE STRING CLASS AND THE VECTOR CONTAINER PROVIDE RANDOM-ACCESS ITERATORS, BUT THE LIST, MAP, AND SET CONTAINERS ONLY PROVIDE BIDIRECTIONAL ITERATORS.

Base:

iterator	iterator base class
iterator_traits	iterator traits

Iterator operations:

advance	advance iterator
distance	return distance between iterators

Inserters:

back_inserter	construct a back insert iterator
front_inserter	constructs a front insert iterator
inserter	construct an insert iterator

Predefined iterators:

reverse_iterator	reverse iterator

Inserter iterators:

back_insert_iterator	back insert iterator
front_insert_iterator	front insert iterator
insert_iterator	insert iterator

Input/output iterators:

istream_iterator	istream iterator
ostream_iterator	ostream iterator
istreambuf_iterator	input stream buffer iterator
ostreambuf_iterator	output stream buffer iterator