Iterators

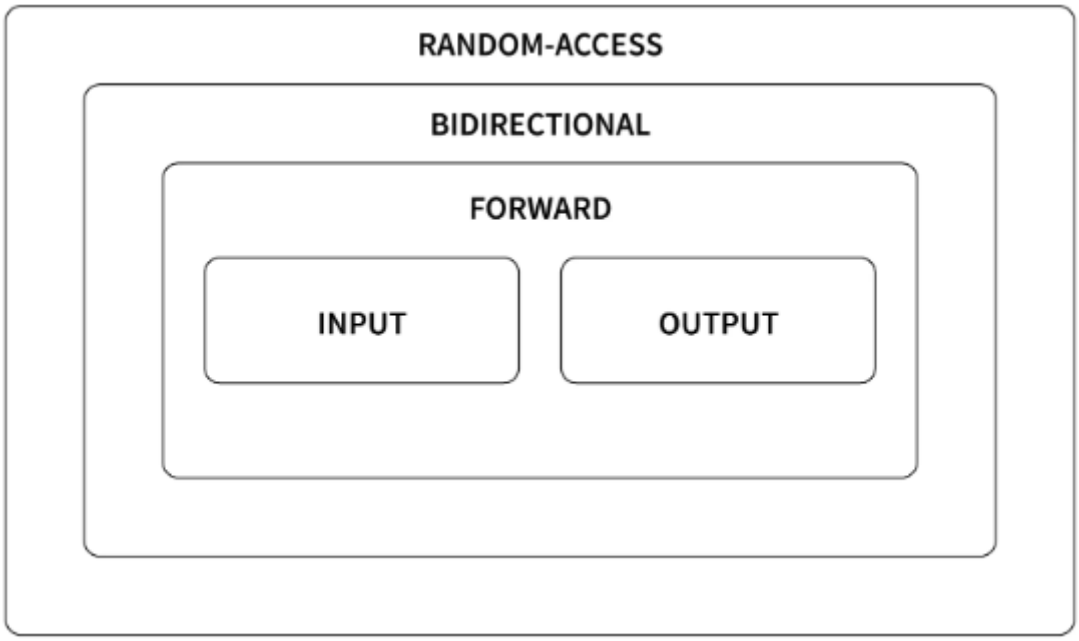
An Iterator is a pointer-like object that points to an element inside some container.

You can use them to iterate over a container which means moving sequentially from one element to another.

Iterators can also help you manipulate the data stored inside a container and connect algorithms to containers.

Iterators provided by STL in C++ are analogous to using the cursor while typing some text. You can shift them to any position you want.

Iterators are classified into five categories depending on the functionalities offered. You can refer to the below diagram. Regarding functionality, the outermost iterator is the most powerful and the innermost the least powerful.



The **common syntax** to declare an iterator:

container_name<object_type>::iterator iterator_name;

The two most commonly used iterators are returned by the following member functions of a container:

begin() -- Returns an iterator to the first element of a container.

end() -- Returns an iterator past the last element of a container.

Iterator Categories

STL in C++ provides the following five iterators:

1. Input Iterator

Input iterator is the weakest and simplest of all.

It has limited functionality and is suitable for single-pass algorithms where you can access the elements in a container sequentially.

It is a unidirectional iterator that only supports increment operation. Thus, it is a read-only iterator.

2. Output Iterator

It is similar to the input iterator but works exactly the opposite.

It is used for assignment operations that help you modify the values inside a container.

It does not allow you to read elements from a container, i.e., it is a write-only iterator.

3. Forward Iterator

Forward iterator is more powerful than input and output iterators.

It combines the functionality of both the iterators (input and output).

It permits you to access (read) and modify (write) the values of a container.

It is also a unidirectional iterator, as the name suggests.

4. Bidirectional Iterator

Bidirectional iterator has all the features of the forward iterator.

It can move in both directions, i.e., backward and forward.

So, you can use both increment (++) and decrement (--) operations.

It does not permit the use of all relational operators.

5. Random Access Iterator

It is the most powerful iterator.

You can use it to access any random element inside a container.

It supports all the functionality of the pointers, like addition and subtraction.

It also gives you relational operator support.