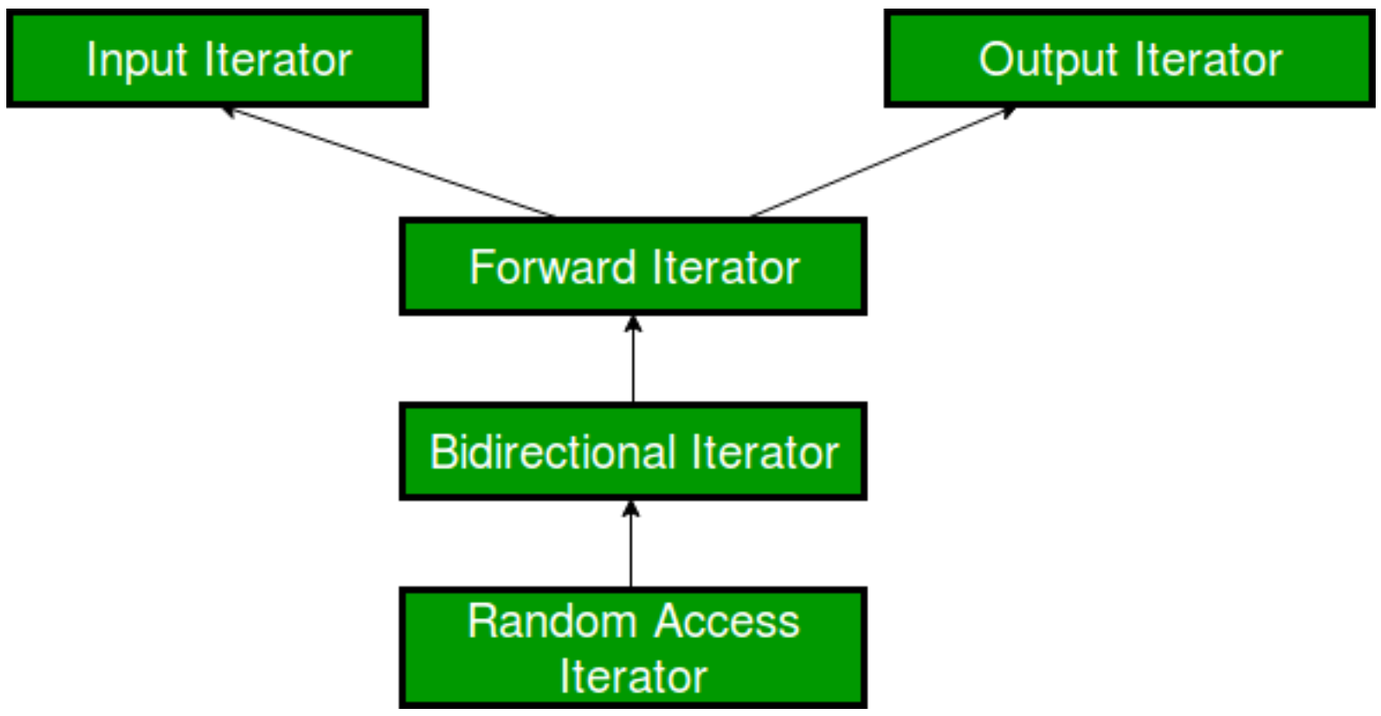
Output Iterators in C++

After going through the template definition of various STL algorithms like **std::copy**, **std::move**, **std::transform**, you must have found their template definition consisting of objects of type **Output Iterator**. So what are they and why are they used ?

Output iterators are one of the five main types of iterators present in C++ Standard Library, others being **Input iterators**, **Forward iterator**, **Bidirectional iterator** and **Random – access iterators**.

Output iterators are considered to be the **exact opposite** of input iterators, as they perform the opposite function of input iterators. They can be **assigned values in a sequence**, but cannot be used to access values, unlike input iterators which do the reverse of accessing values and cannot be assigned values. So, we can say that **input and output iterators are complementary** to each other.



One important thing to be kept in mind is that **forward**, **bidirectional** and **random-access** iterators are also valid output iterators, as shown in the iterator hierarchy above.

Salient Features

1. **Usability:** Just like input iterators, Output iterators can be used only with **single-pass algorithms**, i.e., algorithms in which we can go to all the locations in the range at most once, such that these locations can be dereferenced or assigned value only once.
2. **Equality / Inequality Comparison:** Unlike input iterators, output iterators cannot be compared for equality with another iterator.
   So, the following two expressions are invalid if A and B are output iterators:

```
A == B  // Invalid - Checking for equality
A != B  // Invalid - Checking for inequality
```

3. **Dereferencing:** An input iterator can be dereferenced as an rvalue, using operator * and ->, whereas an **output iterator can be dereferenced as an lvalue** to provide the location to store the value.
   So, the following two expressions are valid if A is an output iterator:

```
*A = 1      // Dereferencing using *
A -> m = 7   // Assigning a member element m
```

4. **Incrementable:** An output iterator can be incremented, so that it refers to the next element in sequence, using operator ++().
   So, the following two expressions are valid if A is an output iterator:

```
A++   // Using post increment operator
++A   // Using pre increment operator
```

5. **Swappable:** The value pointed to by these iterators can be exchanged or swapped.

Practical implementation

After understanding its features and deficiencies, it is very important to learn about its practical implementation as well. As told earlier, output iterators are used only when we want to assign elements and not when we have to access elements. The following two STL algorithms can show this fact:

- **std::move:** As the name suggests, this algorithm is used to move elements in a range into another range. Now, as far as accessing elements are concerned, input iterators are fine, **but as soon as we have to assign elements in another container, then we cannot use these input iterators** for this purpose, that is why here using output iterators becomes a compulsion.

```
// Definition of std::move()
template
OutputIterator move (InputIterator first, InputIterator last,
                     OutputIterator result)
{
    while (first!=last)
    {
        *result = std::move(*first);
        ++result;
        ++first;
    }
    return result;
}
```

  Here, since the result is the iterator to the resultant container, to which elements are assigned, so for this, we cannot use input iterators and have made use of output iterators at their place, whereas for accessing elements, input iterators are used which only needs to be incremented and accessed.

- **std::find:** As we know this algorithm is used to find the presence of an element inside a container and doesn't involve the use of output iterators.

```
// Definition of std::find()
template
InputIterator find (InputIterator first, InputIterator last,
                    const T& val)
{
    while (first!=last)
    {
        if (*first==val) return first;
        ++first;
    }
    return last;
}
```

  So, since here, there was no need for assigning values to iterators and only we needed to access and compare the iterators, so there was no need of output iterator and we have, therefore, used only the input iterators only.

So, the two above examples very well show when, where, why and how output iterators are used practically.

Limitations

After studying about the salient features, one must also know the deficiencies of output iterators as well, which are mentioned in the following points:

1. **Only assigning, no accessing:** One of the biggest deficiency is that **we cannot access the output iterators as rvalue.** So, an output iterator can only modify the element to which it points by being used as the target for an assignment.

```
// C++ program to demonstrate output iterator
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector<int>v1 = {1, 2, 3, 4, 5};

    // Declaring an iterator
    vector<int>::iterator i1;

    for (i1=v1.begin();i1!=v1.end();++i1)
    {
        // Assigning elements using iterator
        *i1 = 1;
    }
    // v1 becomes 1 1 1 1 1
    return 0;
}
```

The above is an example of assigning elements using output iterator, however, if we do something like:

```
a = *i1 ; // where a is a variable
```

So, this is not allowed for output iterator, as they can only be the target in assignment. However, if you try this for above code, it will work, because vectors return iterators higher in hierarchy than output iterators.

This big deficiency is the reason why many algorithms like std::find, which requires to access the elements in a range and check for equality cannot use output iterators for doing so, because we can't access values using it, so instead we make use of input iterators.

2. **Cannot be decremented:** Just like we can use operator ++() with output iterators for incrementing them, we cannot decrement them.

```
If A is an output iterator,then

A--    // Not allowed with output iterators
```

3. **Use in multi-pass algorithms:** Since, it is unidirectional and can only move forward, therefore, such iterators cannot be used in multi-pass algorithms, in which we need to move through the container multiple times.
4. **Relational Operators:** Just like output iterators cannot be used with equality operators (==), it also can not be used with other relational operators like =.

```
If A and B are output iterators, then

A == B     // Not Allowed
A <= B     // Not Allowed
```

5. **Arithmetic Operators:** Similar to relational operators, they also can't be used with arithmetic operators like +, – and so on. This means that output operators can only move in one direction that too forward and that too sequentially.

```
If A and B are output iterators, then

A + 1     // Not allowed
B - 2     // Not allowed
```

This article is contributed by **Mrigendra Singh**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Iterators in C++ STL

Input Iterators in C++

Forward Iterators in C++

Bidirectional Iterators in C++

Introduction to Iterators in C++

Random-access Iterators in C++

Output of C++ programs | Set 22

Output of C Program | Set 29

Output of C++ programs | Set 37

Output of C++ programs | Set 36

Output of C++ Program | Set 10

Output of C++ Program | Set 16

Output of C++ Program | Set 14

Output of C++ Program | Set 18

Output of C++ programs | Set 25 (Macros)

Improved By : Akanksha_Rai

Article Tags : C++ cpp-iterator STL

Practice Tags : STL CPP

1

5

Based on **1** vote(s)

To-do Done

Feedback/ Suggest Improvement Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments