Contents

[Iterators: 2](#_Toc117454031)

[Find(n): 4](#_Toc117454032)

[Binary Search: 5](#_Toc117454033)

# Iterators:

Iterators are used to point at the memory addresses of [STL](https://www.geeksforgeeks.org/the-c-standard-template-library-stl/) containers.

**Types of Iterators :**

* **Input and Output Iterators**: These are the most simplified types of iterators. They are most useful in sequential input-output operations containing single-pass.
* **Forward Iterators**: These are just like input iterators but have a direction i.e. forward direction in the sense they can be used to traverse through a range in the forwarding direction. When forward iterators are not constant they can also be used as output iterators. Most standard STL containers at least support forward iterators.
* **Bidirectional Iterators**: These are similar to forward iterators with the only difference that they are bidirectional. This means that we can use these bidirectional iterators to traverse through the range in the forward as well as backward direction.
* **Random Access Iterators**: Random access iterators are most powerful among all the iterators. These are non-sequential iterators. Random-access iterators allow us to access any random value by applying an offset to the current value without having to pass through each element sequentially. They exhibit similar properties like Pointers in C.

Diagram

Description automatically generated

| **Containers** | **Iterators** |
| --- | --- |
| Stack | No iterator |
| Queue | No iterator |
| Priority Queue | No iterator |
| List | Bidirectional |
| Vector | Random-Access |
| Deque | Random-Access |
| Map | Bidirectional |
| Multimap | Bidirectional |
| Set | Bidirectional |
| Multiset | Bidirectional |

# Find(n):

|  |  |
| --- | --- |
| Syntax | template <class InputIterator, class T>  InputIterator find (InputIterator first, InputIterator last, const T& val); |
| Usage | **Find value in range**  Returns an iterator to the first element in the range [first,last) that compares equal to *val*. If no such element is found, the function returns *last*.  first, last  [Input iterators](https://cplusplus.com/InputIterator) to the initial and final positions in a sequence. The range searched is [first,last), which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.  val  Value to search for in the range. T shall be a type supporting comparisons with the elements pointed by InputIterator using operator== (with the elements as left-hand side operands, and val as right-hand side).  Note: used in vector, set , unordered set,…. |
| Includes | #include <algorithm> |
| Complexity | Up to linear in the [distance](https://cplusplus.com/distance) between first and last: Compares elements until a match is found. |
| Code | #include<iostream>  using namespace std; #include "algorithm" #include "vector" int main() {  int n=3;  vector<int>arr(n);  arr[0]=1;  arr[1]=2;  arr[2]=3;   auto it= find(arr.begin(), arr.end(),2);  if(it!=arr.end())//found  cout<<\*it<<endl;  else cout<<"Not found"<<endl;  it= find(arr.begin(), arr.end(),9);  if(it!=arr.end())//found  cout<<\*it<<endl;  else cout<<"Not found"<<endl; } |
| Output | 2  Not found |

# Binary Search:

/ C++ code to demonstrate the working of binary\_search()

#include <bits/stdc++.h>

**using** **namespace** std;

// Driver's code

**int** main()

{

    // initializing vector of integers

    vector<**int**> arr = { 10, 15, 20, 25, 30, 35 };

    // using binary\_search to check if 15 exists

**if** (binary\_search(arr.begin(), arr.end(), 15))

        cout << "15 exists in vector";

**else**

        cout << "15 does not exist";

    cout << endl;

    // using binary\_search to check if 23 exists

**if** (binary\_search(arr.begin(), arr.end(), 23))

        cout << "23 exists in vector";

**else**

        cout << "23 does not exist";

    cout << endl;

}