Algorithm Library | C++ Magicians STL Algorithm

For all those who aspire to e[illegible]aving a knowledge about containers of STL is of less use till one is not aware what all STL has to offer.

STL has an ocean of algorithms, for all < algorithm > library functions : Refer here.

Some of the most used algorithms on vectors and most useful one's in Competitive Programming are mentioned as follows :

Non-Manipulating Algorithms

1. **sort(first_iterator, last_iterator)** – To sort the given vector.
2. **reverse(first_iterator, last_iterator)** – To reverse a vector.
3. ***max_element (first_iterator, last_iterator)** – To find the maximum element of a vector.
4. ***min_element (first_iterator, last_iterator)** – To find the minimum element of a vector.
5. **accumulate(first_iterator, last_iterator, initial value of sum)** – Does the summation of vector elements

```
// A C++ program to demonstrate working of sort(),
// reverse()
#include <algorithm>
#include <iostream>
#include <vector>
#include <numeric> //For accumulate operation
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23 ,42 , 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is: ";
    for (int i=0; i<n; i++)
```

```
        cout << vect[i] << " ";

    // Sorting the Vector in Ascending order
    sort(vect.begin(), vect.end());

    cout << "\nVector after sorting is: ";
    for (int i=0; i<n; i++)
       cout << vect[i] << " ";

    // Reversing the Vector
    reverse(vect.begin(), vect.end());

    cout << "\nVector after reversing is: ";
    for (int i=0; i<6; i++)
        cout << vect[i] << " ";

    cout << "\nMaximum element of vector is: ";
    cout << *max_element(vect.begin(), vect.end());

    cout << "\nMinimum element of vector is: ";
    cout << *min_element(vect.begin(), vect.end());

    // Starting the summation from 0
    cout << "\nThe summation of vector elements is: ";
    cout << accumulate(vect.begin(), vect.end(), 0);

    return 0;
}
```

Output:

```
Vector before sorting is: 10 20 5 23 42 15
Vector after sorting is: 5 10 15 20 23 42
Vector before reversing is: 5 10 15 20 23 42
Vector after reversing is: 42 23 20 15 10 5
Maximum element of vector is: 42
Minimum element of vector is: 5
The summation of vector elements is: 115
```

6. **count(first_iterator, last_iterator,x)** – To count the occurrences of x in vector.
7. **find(first_iterator, last_iterator, x)** – Points to last address of vector ((name_of_vector).end()) if element is not present in vector.

```
// C++ program to demonstrate working of count()
// and find()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23 ,42, 20, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Occurrences of 20 in vector : ";
```

```
    // Counts the occurrences of 20 from 1st to
    // last element
    cout << count(vect.begin(), vect.end(), 20);

    // find() returns iterator to last address if
    // element not present
    find(vect.begin(), vect.end(),5) != vect.end()?
                          cout << "\nElement found":
                     cout << "\nElement not found";

    return 0;
}
```

Output:

```
Occurrences of 20 in vector: 2
Element found
```

8. **binary_search(first_iterator, last_iterator, x)** – Tests whether x exists in sorted vector or not.
9. **lower_bound(first_iterator, last_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’.
10. **upper_bound(first_iterator, last_iterator, x)** – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

```
// C++ program to demonstrate working of lower_bound()
// and upper_bound().
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Sort the array to make sure that lower_bound()
    // and upper_bound() work.
    sort(vect.begin(), vect.end());

    // Returns the first occurrence of 20
    auto q = lower_bound(vect.begin(), vect.end(), 20);

    // Returns the last occurrence of 20
    auto p = upper_bound(vect.begin(), vect.end(), 20);

    cout << "The lower bound is at position: ";
    cout << q-vect.begin() << endl;

    cout << "The upper bound is at position: ";
    cout << p-vect.begin() << endl;

    return 0;
}
```

Output:

```
The lower bound is at position: 3
The upper bound is at position: 5
```

Some Manipulating Algorithms

11. **arr.erase(position to be deleted)** – This erases selected element in vector and shifts and resizes the vector elements accordingly.
12. **arr.erase(unique(arr.begin(),arr.end()),arr.end())** – This erases the duplicate occurrences in sorted vector in a single line.

```
// C++ program to demonstrate working of erase()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is :";
    for (int i=0; i<6; i++)
        cout << vect[i]<<" ";

    // Delete second element of vector
    vect.erase(vect.begin()+1);

    cout << "\nVector after erasing the element: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // sorting to enable use of unique()
    sort(vect.begin(), vect.end());

    cout << "\nVector before removing duplicate "
             " occurrences: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    // Deletes the duplicate occurrences
    vect.erase(unique(vect.begin(),vect.end()),vect.end());

    cout << "\nVector after deleting duplicates: ";
    for (int i=0; i< vect.size(); i++)
        cout << vect[i] << " ";

    return 0;
}
```

Output:

```
Vector is :5 10 15 20 20 23
Vector after erasing the element: 5 15 20 20 23
```

```
Vector before removing duplicate  occurrences: 5 15 20 20 23
Vector after deleting duplicates: 5 15 20 23 42 45
```

13. **next_permutation(first_iterator, last_iterator)** – This modified the vector to its next permutation.
14. **prev_permutation(first_iterator, last_iterator)** – This modified the vector to its previous permutation.

```
// C++ program to demonstrate working of next_permutation()
// and prev_permutation()
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Given Vector is:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // modifies vector to its next permutation order
    next_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing next permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    prev_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing prev permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    return 0;
}
```

Output:

```
Given Vector is:
5 10 15 20 20 23 42 45
Vector after performing next permutation:
5 10 15 20 20 23 45 42
Vector after performing prev permutation:
5 10 15 20 20 23 42 45
```

14. **distance(first_iterator,desired_position)** – It returns the distance of desired position from the first iterator.This function is very useful while finding the index.

```
// C++ program to demonstrate working of distance()
#include <algorithm>
#include <iostream>
#include <vector>
```

```
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Return distance of first to maximum element
    cout << "Distance between first to max element: ";
    cout << distance(vect.begin(),
                     max_element(vect.begin(), vect.end()));
    return 0;
}
```

Output:

```
Distance between first to max element: 7
```

More – STL Articles

This article is contributed by **Manjeet Singh.** If you like GeeksforGeeks and would like to contribute, you can also write an article and 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:

Prim's algorithm using priority_queue in STL

Dijkstra's shortest path algorithm using set in STL

Convex Hull | Monotone chain algorithm

Minimum enclosing circle | Set 2 - Welzl's algorithm

Print a case where the given sorting algorithm fails

Extended Mo's Algorithm with ≈ O(1) time complexity

Dijkstra's Shortest Path Algorithm using priority_queue of STL

Spanning Tree With Maximum Degree (Using Kruskal's Algorithm)

Sorting objects using In-Place sorting algorithm

Longest Increasing Subsequence using Longest Common Subsequence Algorithm

<numeric> library in C++ STL

<algorithms> library in C++ STL

<iterator> library in C++ STL

<regex> library in C++ STL

The C++ Standard Template Library (STL)

Improved By : auspicious_boy

Article Tags : Competitive Programming cpp-algorithm-library STL

Practice Tags : STL

78

2.6

To-do Done

Based on **78** vote(s)

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