

STL :: Sort

[https://www.geeksforgeeks.org/sort-c-stl/](https://www.geeksforgeeks.org/sort-c-stl/?ref=lbp)

#include <iostream>

#include <algorithm>

using namespace std;

void show(int a[])

{

for(int i = 0; i < 10; ++i)

cout << a[i] << " ";

}

int main()

{

int a[10]= {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};

cout << "\n The array before sorting is : ";

show(a);

sort(a, a+10);

cout << "\n\n The array after sorting is : ";

show(a);

return 0;

}

## Greater

// C++ program to demonstrate descending order sort using

// greater<>().

#include <bits/stdc++.h>

using namespace std;

int main()

{

int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};

int n = sizeof(arr)/sizeof(arr[0]);

sort(arr, arr+n, greater<int>());

cout << "Array after sorting : \n";

for (int i = 0; i < n; ++i)

cout << arr[i] << " ";

return 0;

}

# Binary Search in C++ Standard Template Library (STL)

// CPP program to implement

// Binary Search in

// Standard Template Library (STL)

#include <algorithm>

#include <iostream>

using namespace std;

void show(int a[], int arraysize)

{

for (int i = 0; i < arraysize; ++i)

cout << a[i] << " ";

}

int main()

{

int a[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };

int asize = sizeof(a) / sizeof(a[0]);

cout << "\n The array is : ";

show(a, asize);

cout << "\n\nLet's say we want to search for 2 in the array";

cout << "\n So, we first sort the array";

sort(a, a + asize);

cout << "\n\n The array after sorting is : ";

show(a, asize);

cout << "\n\nNow, we do the binary search";

if (binary\_search(a, a + 10, 2))

cout << "\nElement found in the array";

else

cout << "\nElement not found in the array";

cout << "\n\nNow, say we want to search for 10";

if (binary\_search(a, a + 10, 10))

cout << "\nElement found in the array";

else

cout << "\nElement not found in the array";

return 0;

}

# Algorithm Library | C++ Magicians STL Algorithm

[geeksforgeeks.org/c-magicians-stl-algorithms/](http://geeksforgeeks.org/c-magicians-stl-algorithms/)

1. [**sort**](https://www.geeksforgeeks.org/sort-c-stl/)**(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
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.
8. [**binary\_search**](http://quiz.geeksforgeeks.org/binary-search-algorithms-the-c-standard-template-library-stl/)**(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’.
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.
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.
15. **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.

/ 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;

}

// 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;

}

// 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;

}

// 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;

}

// 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;

}

// 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;

}

# Array algorithms in C++ STL (all\_of, any\_of, none\_of, copy\_n and iota)

<https://www.geeksforgeeks.org/useful-array-algorithms-in-c-stl/>

**all\_of()**

This function operates on whole range of array elements and can save time to run a loop to check each elements one by one. It checks for a given property on every element and returns true when each element in range satisfies specified property, else returns false.

// C++ code to demonstrate working of all\_of()

#include<iostream>

#include<algorithm> // for all\_of()

using namespace std;

int main()

{

// Initializing array

int ar[6] = {1, 2, 3, 4, 5, -6};

// Checking if all elements are positive

all\_of(ar, ar+6, [](int x) { return x>0; })?

cout << "All are positive elements" :

cout << "All are not positive elements";

return 0;

}

**any\_of()**

This function checks for a given range if there’s even one element satisfying a given property mentioned in function. Returns true if at least one element satisfies the property else returns false.

// C++ code to demonstrate working of any\_of()

#include<iostream>

#include<algorithm> // for any\_of()

using namespace std;

int main()

{

// Initializing array

int ar[6] = {1, 2, 3, 4, 5, -6};

// Checking if any element is negative

any\_of(ar, ar+6, [](int x){ return x<0; })?

cout << "There exists a negative element" :

cout << "All are positive elements";

return 0;

}

**none\_of()**

This function returns true if none of elements satisfies the given condition else returns false.

// C++ code to demonstrate working of none\_of()

#include<iostream>

#include<algorithm> // for none\_of()

using namespace std;

int main()

{

// Initializing array

int ar[6] = {1, 2, 3, 4, 5, 6};

// Checking if no element is negative

none\_of(ar, ar+6, [](int x){ return x<0; })?

cout << "No negative elements" :

cout << "There are negative elements";

return 0;

}

**copy\_n()**

copy\_n() copies one array elements to new array. This type of copy creates a deep copy of array. This function takes 3 arguments, source array name, size of array and the target array name.

**iot**// C++ code to demonstrate working of copy\_n()

// C++ code to demonstrate working of iota()

#include<iostream>

#include<numeric> // for iota()

using namespace std;

int main()

{

// Initializing array with 0 values

int ar[6] = {0};

// Using iota() to assign values

iota(ar, ar+6, 20);

// Displaying the new array

cout << "The new array after assigning values is : ";

for (int i=0; i<6 ; i++)

cout << ar[i] << " ";

return 0;

}

# std::partition in C++ STL

<https://www.geeksforgeeks.org/stdpartition-in-c-stl/>

**Partition operations** **:**

**1. partition(beg, end, condition)** :- This function is used to **partition the elements** on **basis of condition** mentioned in its arguments.

**2. is\_partitioned(beg, end, condition)** :- This function returns boolean **true if container is partitioned** else returns false.

**3. stable\_partition(beg, end, condition)** :- This function is used to **partition the elements** on **basis of condition** mentioned in its arguments in **such a way that the relative order of the elements is preserved.**.

**4. partition\_point(beg, end, condition)** :- This function **returns an iterator pointing to the partition point** of container i.e. the first element in the partitioned range [beg,end) for which condition is not true. The container should already be partitioned for this function to work.

**5. partition\_copy(beg, end, beg1, beg2, condition)** :- This function **copies the partitioned elements** in the differenet containers mentioned in its arguments. It takes 5 arguments. **Beginning and ending position of container, beginning position of new container where elements have to be copied (elements returning true for condition), beginning position of new container where other elements have to be copied (elements returning false for condition) and the condition**. **Resizing** new containers **is necessary** for this function.

| // C++ code to demonstrate the working of  // partition() and is\_partitioned()  #include<iostream>  #include<algorithm> // for partition algorithm  #include<vector> // for vector  using namespace std;  int main()  {  // Initializing vector  vector<int> vect = { 2, 1, 5, 6, 8, 7 };    // Checking if vector is partitioned  // using is\_partitioned()  is\_partitioned(vect.begin(), vect.end(), [](int x)  {  return x%2==0;    })?    cout << "Vector is partitioned":  cout << "Vector is not partitioned";  cout << endl;    // partitioning vector using partition()  partition(vect.begin(), vect.end(), [](int x)  {  return x%2==0;    });    // Checking if vector is partitioned  // using is\_partitioned()  is\_partitioned(vect.begin(), vect.end(), [](int x)  {  return x%2==0;    })?    cout << "Now, vector is partitioned after partition operation":  cout << "Vector is still not partitioned after partition operation";  cout << endl;    // Displaying partitioned Vector  cout << "The partitioned vector is : ";  for (int &x : vect) cout << x << " ";    return 0;    } |
| --- |

// C++ code to demonstrate the working of

// stable\_partition() and partition\_point()

#include<iostream>

#include<algorithm> // for partition algorithm

#include<vector> // for vector

using namespace std;

int main()

{

// Initializing vector

vector<int> vect = { 2, 1, 5, 6, 8, 7 };

// partitioning vector using stable\_partition()

// in sorted order

stable\_partition(vect.begin(), vect.end(), [](int x)

{

return x%2 == 0;

});

// Displaying partitioned Vector

cout << "The partitioned vector is : ";

for (int &x : vect) cout << x << " ";

cout << endl;

// Declaring iterator

vector<int>::iterator it1;

// using partition\_point() to get ending position of partition

auto it = partition\_point(vect.begin(), vect.end(), [](int x)

{

return x%2==0;

});

// Displaying partitioned Vector

cout << "The vector elements returning true for condition are : ";

for ( it1= vect.begin(); it1!=it; it1++)

cout << \*it1 << " ";

cout << endl;

return 0;

}

// C++ code to demonstrate the working of

// partition\_copy()

#include<iostream>

#include<algorithm> // for partition algorithm

#include<vector> // for vector

using namespace std;

int main()

{

// Initializing vector

vector<int> vect = { 2, 1, 5, 6, 8, 7 };

// Declaring vector1

vector<int> vect1;

// Declaring vector1

vector<int> vect2;

// Resizing vectors to suitable size using count\_if() and resize()

int n = count\_if (vect.begin(), vect.end(), [](int x)

{

return x%2==0;

} );

vect1.resize(n);

vect2.resize(vect.size()-n);

// Using partition\_copy() to copy partitions

partition\_copy(vect.begin(), vect.end(), vect1.begin(),

vect2.begin(), [](int x)

{

return x%2==0;

});

// Displaying partitioned Vector

cout << "The elements that return true for condition are : ";

for (int &x : vect1)

cout << x << " ";

cout << endl;

// Displaying partitioned Vector

cout << "The elements that return false for condition are : ";

for (int &x : vect2)

cout << x << " ";

cout << endl;

return 0;

}

# std:: valarray class in C++

C++98 introduced a special container called valarray to hold and provide mathematical operations on arrays efficiently.

* It supports element-wise mathematical operations and various forms of generalized subscript operators, slicing and indirect access.
* As compare to vectors, valarrays are efficient in certain mathematical operations than vectors also.

**Public member functions in valarray class :**

**1. apply()** :- This function **applies the manipulation** given in its arguments **to all** the valarray elements at once and **returns a new valarray** with manipulated values.

**2. sum()** :- This function **returns the summation** of all the elements of valarrays at once.

| // C++ code to demonstrate the working of  // apply() and sum()  #include<iostream>  #include<valarray> // for valarray functions  using namespace std;  int main()  {  // Initializing valarray  valarray<int> varr = { 10, 2, 20, 1, 30 };    // Declaring new valarray  valarray<int> varr1 ;    // Using apply() to increment all elements by 5  varr1 = varr.apply([](int x){return x=x+5;});    // Displaying new elements value  cout << "The new valarray with manipulated values is : ";  for (int &x: varr1) cout << x << " ";  cout << endl;    // Displaying sum of both old and new valarray  cout << "The sum of old valarray is : ";  cout << varr.sum() << endl;  cout << "The sum of new valarray is : ";  cout << varr1.sum() << endl;    return 0;    } |
| --- |

Output:

The new valarray with manipulated values is : 15 7 25 6 35

The sum of old valarray is : 63

The sum of new valarray is : 88

**3. min()** :- This function returns the **smallest** element of valarray.

**4. max()** :- This function returns the **largest** element of valarray.

| // C++ code to demonstrate the working of  // max() and min()  #include<iostream>  #include<valarray> // for valarray functions  using namespace std;  int main()  {  // Initializing valarray  valarray<int> varr = { 10, 2, 20, 1, 30 };    // Displaying largest element of valarray  cout << "The largest element of valarray is : ";  cout << varr.max() << endl;    // Displaying smallest element of valarray  cout << "The smallest element of valarray is : ";  cout << varr.min() << endl;    return 0;    } |
| --- |

Output:

The largest element of valarray is : 30

The smallest element of valarray is : 1

**5. shift()** :- This function returns the new valarray after **shifting elements by** the **number** mentioned in its argument. If the **number is positive**, **left-shift** is applied, if **number is negative**, **right-shift** is applied.

**6. cshift()** :- This function returns the new valarray after **circularly shifting(rotating)** elements **by** the **number** mentioned in its argument. If the **number is positive, left-circular** **shift** is applied, if **number is negative, right-circular shift** is applied.

| // C++ code to demonstrate the working of  // shift() and cshift()  #include<iostream>  #include<valarray> // for valarray functions  using namespace std;  int main()  {  // Initializing valarray  valarray<int> varr = { 10, 2, 20, 1, 30 };    // Declaring new valarray  valarray<int> varr1;    // using shift() to shift elements to left  // shifts valarray by 2 position  varr1 = varr.shift(2);    // Displaying elements of valarray after shifting  cout << "The new valarray after shifting is : ";  for ( int&x : varr1) cout << x << " ";  cout << endl;    // using cshift() to circulary shift elements to right  // rotates valarray by 3 position  varr1 = varr.cshift(-3);    // Displaying elements of valarray after circular shifting  cout << "The new valarray after circular shifting is : ";  for ( int&x : varr1) cout << x << " ";  cout << endl;    return 0;    } |
| --- |

Output:

The new valarray after shifting is : 20 1 30 0 0

The new valarray after circular shifting is : 20 1 30 10 2

**7. swap()** :- This function **swaps** one valarray with other.

| // C++ code to demonstrate the working of  // swap()  #include<iostream>  #include<valarray> // for valarray functions  using namespace std;  int main()  {  // Initializing 1st valarray  valarray<int> varr1 = {1, 2, 3, 4};    // Initializing 2nd valarray  valarray<int> varr2 = {2, 4, 6, 8};    // Displaying valarrays before swapping  cout << "The contents of 1st valarray "  "before swapping are : ";  for (int &x : varr1)  cout << x << " ";  cout << endl;  cout << "The contents of 2nd valarray "  "before swapping are : ";  for (int &x : varr2)  cout << x << " ";  cout << endl;    // Use of swap() to swap the valarrays  varr1.swap(varr2);    // Displaying valarrays after swapping  cout << "The contents of 1st valarray "  "after swapping are : ";  for (int &x : varr1)  cout << x << " ";  cout << endl;    cout << "The contents of 2nd valarray "  "after swapping are : ";  for (int &x : varr2)  cout << x << " ";  cout << endl;    return 0;    } |
| --- |

Output:

The contents of 1st valarray before swapping are : 1 2 3 4

The contents of 2nd valarray before swapping are : 2 4 6 8

The contents of 1st valarray after swapping are : 2 4 6 8

The contents of 2nd valarray after swapping are : 1 2 3 4

Containers

Containers or container classes store objects and data. There are in total seven standard “first-class” container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

* Sequence Containers: implement data structures which can be accessed in a sequential manner.
  + [vector](http://quiz.geeksforgeeks.org/vector-sequence-containers-the-c-standard-template-library-stl-set-1/)
  + [list](http://quiz.geeksforgeeks.org/list-sequence-containers-the-c-standard-template-library-stl/)
  + [deque](http://quiz.geeksforgeeks.org/deque-sequence-containers-the-c-standard-template-library-stl/)
  + [arrays](https://www.geeksforgeeks.org/array-class-c/)
  + [forward\_list](https://www.geeksforgeeks.org/forward-list-c-set-1-introduction-important-functions/)( Introduced in C++11)
* Container Adaptors : provide a different interface for sequential containers.
  + [queue](http://quiz.geeksforgeeks.org/queue-container-adaptors-the-c-standard-template-library-stl/)
  + [priority\_queue](http://quiz.geeksforgeeks.org/priority-queue-container-adaptors-the-c-standard-template-library-stl/)
  + [stack](http://quiz.geeksforgeeks.org/stack-container-adaptors-the-c-standard-template-library-stl/)
* Associative Containers : implement sorted data structures that can be quickly searched (O(log n) complexity).
  + [set](http://quiz.geeksforgeeks.org/set-associative-containers-the-c-standard-template-library-stl/)
  + [multiset](http://quiz.geeksforgeeks.org/multiset-associative-containers-the-c-standard-template-library-stl/)
  + [map](http://quiz.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/)
  + [multimap](http://quiz.geeksforgeeks.org/multimap-associative-containers-the-c-standard-template-library-stl/)
* Unordered Associative Containers : implement unordered data structures that can be quickly searched
  + [unordered\_set](https://www.geeksforgeeks.org/unordered_set-in-cpp-stl/) (Introduced in C++11)
  + [unordered\_multiset](https://www.geeksforgeeks.org/unordered_multiset-and-its-uses/) (Introduced in C++11)
  + [unordered\_map](https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/) (Introduced in C++11)
  + [unordered\_multimap](https://www.geeksforgeeks.org/unordered_multimap-and-its-application/) (Introduced in C++11)

Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes there may be a need of extending the array. Removing the last element takes only constant time because no resizing happens. Inserting and erasing at the beginning or in the middle is linear in time.

Certain functions associated with the vector are:

**Iterators**

1. [begin()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the first element in the vector
2. [end()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/) – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3. [rbegin()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4. [rend()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/) – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5. [cbegin()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the first element in the vector.
6. [cend()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/) – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
7. [crbegin()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8. [crend()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/) – Returns a constant reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

| // C++ program to illustrate the  // iterators in vector  #include <iostream>  #include <vector>    using namespace std;    int main()  {  vector<int> g1;    for (int i = 1; i <= 5; i++)  g1.push\_back(i);    cout << "Output of begin and end: ";  for (auto i = g1.begin(); i != g1.end(); ++i)  cout << \*i << " ";    cout << "\nOutput of cbegin and cend: ";  for (auto i = g1.cbegin(); i != g1.cend(); ++i)  cout << \*i << " ";    cout << "\nOutput of rbegin and rend: ";  for (auto ir = g1.rbegin(); ir != g1.rend(); ++ir)  cout << \*ir << " ";    cout << "\nOutput of crbegin and crend : ";  for (auto ir = g1.crbegin(); ir != g1.crend(); ++ir)  cout << \*ir << " ";    return 0;  } |
| --- |

**Output:**

Output of begin and end: 1 2 3 4 5

Output of cbegin and cend: 1 2 3 4 5

Output of rbegin and rend: 5 4 3 2 1

Output of crbegin and crend : 5 4 3 2 1

**Capacity**

1. [size()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/) – Returns the number of elements in the vector.
2. [max\_size()](https://www.geeksforgeeks.org/vector-max_size-function-in-c-stl/) – Returns the maximum number of elements that the vector can hold.
3. [capacity()](https://www.geeksforgeeks.org/vector-capacity-function-in-c-stl/) – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
4. [resize(n)](https://www.geeksforgeeks.org/vector-resize-c-stl/) – Resizes the container so that it contains ‘n’ elements.
5. [empty()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/) – Returns whether the container is empty.
6. [shrink\_to\_fit()](https://www.geeksforgeeks.org/vector-shrink_to_fit-function-in-c-stl/) – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
7. [reserve()](https://www.geeksforgeeks.org/using-stdvectorreserve-whenever-possible/) – Requests that the vector capacity be at least enough to contain n elements.

| // C++ program to illustrate the  // capacity function in vector  #include <iostream>  #include <vector>    using namespace std;    int main()  {  vector<int> g1;    for (int i = 1; i <= 5; i++)  g1.push\_back(i);    cout << "Size : " << g1.size();  cout << "\nCapacity : " << g1.capacity();  cout << "\nMax\_Size : " << g1.max\_size();    // resizes the vector size to 4  g1.resize(4);    // prints the vector size after resize()  cout << "\nSize : " << g1.size();    // checks if the vector is empty or not  if (g1.empty() == false)  cout << "\nVector is not empty";  else  cout << "\nVector is empty";    // Shrinks the vector  g1.shrink\_to\_fit();  cout << "\nVector elements are: ";  for (auto it = g1.begin(); it != g1.end(); it++)  cout << \*it << " ";    return 0;  } |
| --- |

**Output:**

Size : 5

Capacity : 8

Max\_Size : 4611686018427387903

Size : 4

Vector is not empty

Vector elements are: 1 2 3 4

**Element access:**

1. [reference operator [g]](https://www.geeksforgeeks.org/vectoroperator-vectoroperator-c-stl/) – Returns a reference to the element at position ‘g’ in the vector
2. [at(g)](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/) – Returns a reference to the element at position ‘g’ in the vector
3. [front()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/) – Returns a reference to the first element in the vector
4. [back()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/) – Returns a reference to the last element in the vector
5. [data()](https://www.geeksforgeeks.org/vector-data-function-in-c-stl/) – Returns a direct pointer to the memory array used internally by the vector to store its owned elements.

| // C++ program to illustrate the  // element accesser in vector  #include <bits/stdc++.h>  using namespace std;    int main()  {  vector<int> g1;    for (int i = 1; i <= 10; i++)  g1.push\_back(i \* 10);    cout << "\nReference operator [g] : g1[2] = " << g1[2];    cout << "\nat : g1.at(4) = " << g1.at(4);    cout << "\nfront() : g1.front() = " << g1.front();    cout << "\nback() : g1.back() = " << g1.back();    // pointer to the first element  int\* pos = g1.data();    cout << "\nThe first element is " << \*pos;  return 0;  } |
| --- |

**Output:**

Reference operator [g] : g1[2] = 30

at : g1.at(4) = 50

front() : g1.front() = 10

back() : g1.back() = 100

The first element is 10

**Modifiers:**

1. [assign()](https://www.geeksforgeeks.org/vector-assign-in-c-stl/) – It assigns new value to the vector elements by replacing old ones
2. [push\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/) – It push the elements into a vector from the back
3. [pop\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/) – It is used to pop or remove elements from a vector from the back.
4. [insert()](https://www.geeksforgeeks.org/vector-insert-function-in-c-stl/) – It inserts new elements before the element at the specified position
5. [erase()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/) – It is used to remove elements from a container from the specified position or range.
6. [swap()](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/) – It is used to swap the contents of one vector with another vector of same type. Sizes may differ.
7. [clear()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/) – It is used to remove all the elements of the vector container
8. [emplace()](https://www.geeksforgeeks.org/vector-emplace-function-in-c-stl/) – It extends the container by inserting new element at position
9. [emplace\_back()](https://www.geeksforgeeks.org/vectoremplace_back-c-stl/) – It is used to insert a new element into the vector container, the new element is added to the end of the vector

| // C++ program to illustrate the  // Modifiers in vector  #include <bits/stdc++.h>  #include <vector>  using namespace std;    int main()  {  // Assign vector  vector<int> v;    // fill the array with 10 five times  v.assign(5, 10);    cout << "The vector elements are: ";  for (int i = 0; i < v.size(); i++)  cout << v[i] << " ";    // inserts 15 to the last position  v.push\_back(15);  int n = v.size();  cout << "\nThe last element is: " << v[n - 1];    // removes last element  v.pop\_back();    // prints the vector  cout << "\nThe vector elements are: ";  for (int i = 0; i < v.size(); i++)  cout << v[i] << " ";    // inserts 5 at the beginning  v.insert(v.begin(), 5);    cout << "\nThe first element is: " << v[0];    // removes the first element  v.erase(v.begin());    cout << "\nThe first element is: " << v[0];    // inserts at the beginning  v.emplace(v.begin(), 5);  cout << "\nThe first element is: " << v[0];    // Inserts 20 at the end  v.emplace\_back(20);  n = v.size();  cout << "\nThe last element is: " << v[n - 1];    // erases the vector  v.clear();  cout << "\nVector size after erase(): " << v.size();    // two vector to perform swap  vector<int> v1, v2;  v1.push\_back(1);  v1.push\_back(2);  v2.push\_back(3);  v2.push\_back(4);    cout << "\n\nVector 1: ";  for (int i = 0; i < v1.size(); i++)  cout << v1[i] << " ";    cout << "\nVector 2: ";  for (int i = 0; i < v2.size(); i++)  cout << v2[i] << " ";    // Swaps v1 and v2  v1.swap(v2);    cout << "\nAfter Swap \nVector 1: ";  for (int i = 0; i < v1.size(); i++)  cout << v1[i] << " ";    cout << "\nVector 2: ";  for (int i = 0; i < v2.size(); i++)  cout << v2[i] << " ";  } |
| --- |

**Output:**

The vector elements are: 10 10 10 10 10

The last element is: 15

The vector elements are: 10 10 10 10 10

The first element is: 5

The first element is: 10

The first element is: 5

The last element is: 20

Vector size after erase(): 0

Vector 1: 1 2

Vector 2: 3 4

After Swap

Vector 1: 3 4

Vector 2: 1 2

**All Vector Functions :**

* [vector::begin() and vector::end()](https://www.geeksforgeeks.org/vectorbegin-vectorend-c-stl/)
* [vector rbegin() and rend()](https://www.geeksforgeeks.org/vector-rbegin-and-rend-function-in-c-stl/)
* [vector::cbegin() and vector::cend()](https://www.geeksforgeeks.org/vector-cbegin-vector-cend-c-stl/)
* [vector::crend() and vector::crbegin()](https://www.geeksforgeeks.org/vectorcrend-vectorcrbegin-examples/)
* [vector::assign()](https://www.geeksforgeeks.org/vector-assign-in-c-stl/)
* [vector::at()](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/)
* [vector::back()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/)
* [vector::capacity()](https://www.geeksforgeeks.org/vector-capacity-function-in-c-stl/)
* [vector::clear()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/)
* [vector::push\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/)
* [vector::pop\_back()](https://www.geeksforgeeks.org/vectorpush_back-vectorpop_back-c-stl/)
* [vector::empty()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/)
* [vector::erase()](https://www.geeksforgeeks.org/vectorclear-vectorerase-c-stl/)
* [vector::size()](https://www.geeksforgeeks.org/vectorempty-vectorsize-c-stl/)
* [vector::swap()](https://www.geeksforgeeks.org/vectorat-vectorswap-c-stl/)
* [vector::reserve()](https://www.geeksforgeeks.org/using-stdvectorreserve-whenever-possible/)
* [vector::resize()](https://www.geeksforgeeks.org/vector-resize-c-stl/)
* [vector::shrink\_to\_fit()](https://www.geeksforgeeks.org/vector-shrink_to_fit-function-in-c-stl/)
* [vector::operator=](https://www.geeksforgeeks.org/vectoroperator-vectoroperator-c-stl/)
* [vector::operator[]](https://www.geeksforgeeks.org/vectoroperator-vectoroperator-c-stl/)
* [vector::front()](https://www.geeksforgeeks.org/vectorfront-vectorback-c-stl/)
* [vector::data()](https://www.geeksforgeeks.org/vector-data-function-in-c-stl/)
* [vector::emplace\_back()](https://www.geeksforgeeks.org/vectoremplace_back-c-stl/)
* [vector::emplace()](https://www.geeksforgeeks.org/vector-emplace-function-in-c-stl/)
* [vector::max\_size()](https://www.geeksforgeeks.org/vector-max_size-function-in-c-stl/)
* [vector::insert()](https://www.geeksforgeeks.org/vector-insert-function-in-c-stl/)

# List in C++ Standard Template Library (STL)

Lists are sequence containers that allow non-contiguous memory allocation. As compared to vector, list has slow traversal, but once a position has been found, insertion and deletion are quick. Normally, when we say a List, we talk about doubly linked list. For implementing a singly linked list, we use forward list.

Below is the program to show the working of some functions of List:

| #include <iostream>  #include <list>  #include <iterator>  using namespace std;    //function for printing the elements in a list  void showlist(list <int> g)  {  list <int> :: iterator it;  for(it = g.begin(); it != g.end(); ++it)  cout << '\t' << \*it;  cout << '\n';  }    int main()  {    list <int> gqlist1, gqlist2;      for (int i = 0; i < 10; ++i)  {  gqlist1.push\_back(i \* 2);  gqlist2.push\_front(i \* 3);  }  cout << "\nList 1 (gqlist1) is : ";  showlist(gqlist1);    cout << "\nList 2 (gqlist2) is : ";  showlist(gqlist2);    cout << "\ngqlist1.front() : " << gqlist1.front();  cout << "\ngqlist1.back() : " << gqlist1.back();    cout << "\ngqlist1.pop\_front() : ";  gqlist1.pop\_front();  showlist(gqlist1);    cout << "\ngqlist2.pop\_back() : ";  gqlist2.pop\_back();  showlist(gqlist2);    cout << "\ngqlist1.reverse() : ";  gqlist1.reverse();  showlist(gqlist1);    cout << "\ngqlist2.sort(): ";  gqlist2.sort();  showlist(gqlist2);    return 0;    } |
| --- |

The output of the above program is :

List 1 (gqlist1) is : 0 2 4 6

8 10 12 14 16 18

List 2 (gqlist2) is : 27 24 21 18

15 12 9 6 3 0

gqlist1.front() : 0

gqlist1.back() : 18

gqlist1.pop\_front() : 2 4 6 8

10 12 14 16 18

gqlist2.pop\_back() : 27 24 21 18

15 12 9 6 3

gqlist1.reverse() : 18 16 14 12

10 8 6 4 2

gqlist2.sort(): 3 6 9 12

15 18 21 24 27

Deque in C++ Standard Template Library (STL)

Double ended queues are sequence containers with the feature of expansion and contraction on both the ends.

They are similar to vectors, but are more efficient in case of insertion and deletion of elements. Unlike vectors, contiguous storage allocation may not be guaranteed.

Double Ended Queues are basically an implementation of the data structure double ended queue. A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back. Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends.

The functions for deque are same as [vector](http://quiz.geeksforgeeks.org/vector-sequence-containers-the-c-standard-template-library-stl-set-1), with an addition of push and pop operations for both front and back.

| #include <iostream>  #include <deque>    using namespace std;    void showdq(deque <int> g)  {  deque <int> :: iterator it;  for (it = g.begin(); it != g.end(); ++it)  cout << '\t' << \*it;  cout << '\n';  }    int main()  {  deque <int> gquiz;  gquiz.push\_back(10);  gquiz.push\_front(20);  gquiz.push\_back(30);  gquiz.push\_front(15);  cout << "The deque gquiz is : ";  showdq(gquiz);    cout << "\ngquiz.size() : " << gquiz.size();  cout << "\ngquiz.max\_size() : " << gquiz.max\_size();    cout << "\ngquiz.at(2) : " << gquiz.at(2);  cout << "\ngquiz.front() : " << gquiz.front();  cout << "\ngquiz.back() : " << gquiz.back();    cout << "\ngquiz.pop\_front() : ";  gquiz.pop\_front();  showdq(gquiz);    cout << "\ngquiz.pop\_back() : ";  gquiz.pop\_back();  showdq(gquiz);    return 0;  } |
| --- |

The output of the above program is :

The deque gquiz is : 15 20 10 30

gquiz.size() : 4

gquiz.max\_size() : 4611686018427387903

gquiz.at(2) : 10

gquiz.front() : 15

gquiz.back() : 30

gquiz.pop\_front() : 20 10 30

gquiz.pop\_back() : 20 10

**Methods of Deque:**

* [deque insert() function in C++ STL](https://www.geeksforgeeks.org/deque-insert-function-in-c-stl/): Inserts an element. And returns an iterator that points to the first of the newly inserted elements.
* [deque rbegin() function in C++ STL](https://www.geeksforgeeks.org/deque-rbegin-function-in-c-stl/): Returns a reverse iterator which points to the last element of the deque (i.e., its reverse beginning).
* [deque rend() function in C++ STL](https://www.geeksforgeeks.org/deque-rend-function-in-c-stl/): Returns a reverse iterator which points to the position before the beginning of the deque (which is considered its reverse end).
* [deque cbegin() in C++ STL](https://www.geeksforgeeks.org/deque-cbegin-in-c-stl/): Returns a constant iterator pointing to the first element of the container, that is, the iterator cannot be used to modify, only traverse the deque.
* [deque max\_size() function in C++ STL](https://www.geeksforgeeks.org/deque-max_size-function-in-c-stl/): Returns the maximum number of elements that a deque container can hold.
* [deque assign() function in C++ STL](https://www.geeksforgeeks.org/deque-assign-function-in-c-stl/): Assign values to the same or different deque container.
* [deque resize() function in C++ STL](https://www.geeksforgeeks.org/deque-resize-function-in-c-stl/): Function which changes the size of the deque.
* [deque::push\_front() in C++ STL](https://www.geeksforgeeks.org/dequepush_front-c-stl/): This function is used to push elements into a deque from the front.
* [deque::push\_back() in C++ STL](https://www.geeksforgeeks.org/dequepush_back-c-stl/): This function is used to push elements into a deque from the back.
* [deque::pop\_front() and deque::pop\_back() in C++ STL](https://www.geeksforgeeks.org/dequepop_front-dequepop_back-c-stl/): **pop\_front()** function is used to pop or remove elements from a deque from the front. **pop\_back()** function is used to pop or remove elements from a deque from the back.
* [deque::front() and deque::back() in C++ STL](https://www.geeksforgeeks.org/dequefront-dequeback-c-stl/): **front()** function is used to reference the first element of the deque container. **back()** function is used to reference the last element of the deque container.
* [deque::clear() and deque::erase() in C++ STL](https://www.geeksforgeeks.org/dequeclear-dequeerase-c-stl/): **clear()** function is used to remove all the elements of the deque container, thus making its size 0. **erase()** function is used to remove elements from a container from the specified position or range.
* [deque::empty() and deque::size() in C++ STL](https://www.geeksforgeeks.org/dequeempty-dequesize-c-stl/): **empty()** function is used to check if the deque container is empty or not. **size()** function is used to return the size of the deque container or the number of elements in the deque container.
* [deque::operator= and deque::operator[] in C++ STL](https://www.geeksforgeeks.org/dequeoperator-dequeoperator-c-stl/):  
  **operator=** operator is used to assign new contents to the container by replacing the existing contents. **operator[]** operator is used to reference the element present at position given inside the operator.
* [deque::at() and deque::swap() in C++ STL](https://www.geeksforgeeks.org/dequeat-dequeswap-c-stl/): **at()** function is used reference the element present at the position given as the parameter to the function. **swap()** function is used to swap the contents of one deque with another deque of same type and size.
* [deque::begin() and deque::end in C++ STL](https://www.geeksforgeeks.org/dequebegin-dequeend-c-stl/): **begin()** function is used to return an iterator pointing to the first element of the deque container. **end()** function is used to return an iterator pointing to the last element of the deque container.
* [deque::emplace\_front() and deque::emplace\_back() in C++ STL](https://www.geeksforgeeks.org/deque-emplace_front-deque-emplace_back-cpp-stl/): **emplace\_front()** function is used to insert a new element into the deque container. The new element is added to the beginning of the deque. **emplace\_back()** function is used to insert a new element into the deque container. The new element is added to the end of the deque.

# Array class in C++

The introduction of array class from C++11 has offered a better alternative for C-style arrays. The advantages of array class over C-style array are :-

* Array classes knows its own size, whereas C-style arrays lack this property. So when passing to functions, we don’t need to pass size of Array as a separate parameter.
* With C-style array there is more risk of [array being decayed into a pointer](https://www.geeksforgeeks.org/what-is-array-decay-in-c-how-can-it-be-prevented/). Array classes don’t decay into pointers
* Array classes are generally more efficient, light-weight and reliable than C-style arrays.

**Operations on array** :-

**1. at()** :- This function is used to access the elements of array.

**2. get()** :- This function is also used to access the elements of array. This function is not the member of array class but overloaded function from class tuple.

**3. operator[]** :- This is similar to C-style arrays. This method is also used to access array elements.

| // C++ code to demonstrate working of array,  // to() and get()  #include<iostream>  #include<array> // for array, at()  #include<tuple> // for get()  using namespace std;  int main()  {  // Initializing the array elements  array<int,6> ar = {1, 2, 3, 4, 5, 6};    // Printing array elements using at()  cout << "The array elements are (using at()) : ";  for ( int i=0; i<6; i++)  cout << ar.at(i) << " ";  cout << endl;    // Printing array elements using get()  cout << "The array elements are (using get()) : ";  cout << get<0>(ar) << " " << get<1>(ar) << " ";  cout << get<2>(ar) << " " << get<3>(ar) << " ";  cout << get<4>(ar) << " " << get<5>(ar) << " ";  cout << endl;    // Printing array elements using operator[]  cout << "The array elements are (using operator[]) : ";  for ( int i=0; i<6; i++)  cout << ar[i] << " ";  cout << endl;    return 0;    } |
| --- |

Output:

The array elemets are (using at()) : 1 2 3 4 5 6

The array elemets are (using get()) : 1 2 3 4 5 6

The array elements are (using operator[]) : 1 2 3 4 5 6

**4. front()** :- This returns the first element of array.

**5. back()** :- This returns the last element of array.

| // C++ code to demonstrate working of  // front() and back()  #include<iostream>  #include<array> // for front() and back()  using namespace std;  int main()  {  // Initializing the array elements  array<int,6> ar = {1, 2, 3, 4, 5, 6};    // Printing first element of array  cout << "First element of array is : ";  cout << ar.front() << endl;    // Printing last element of array  cout << "Last element of array is : ";  cout << ar.back() << endl;    return 0;    } |
| --- |

Output:

First element of array is : 1

Last element of array is : 6

**6. size()** :- It returns the number of elements in array. This is a property that C-style arrays lack.

**7. max\_size()** :- It returns the maximum number of elements array can hold i.e, the size with which array is declared. The size() and max\_size() return the same value.

| // C++ code to demonstrate working of  // size() and max\_size()  #include<iostream>  #include<array> // for size() and max\_size()  using namespace std;  int main()  {  // Initializing the array elements  array<int,6> ar = {1, 2, 3, 4, 5, 6};    // Printing number of array elements  cout << "The number of array elements is : ";  cout << ar.size() << endl;    // Printing maximum elements array can hold  cout << "Maximum elements array can hold is : ";  cout << ar.max\_size() << endl;    return 0;    } |
| --- |

Output:

The number of array elements is : 6

Maximum elements array can hold is : 6

**8. swap()** :- The swap() swaps all elements of one array with other.

| // C++ code to demonstrate working of swap()  #include<iostream>  #include<array> // for swap() and array  using namespace std;  int main()  {    // Initializing 1st array  array<int,6> ar = {1, 2, 3, 4, 5, 6};    // Initializing 2nd array  array<int,6> ar1 = {7, 8, 9, 10, 11, 12};    // Printing 1st and 2nd array before swapping  cout << "The first array elements before swapping are : ";  for (int i=0; i<6; i++)  cout << ar[i] << " ";  cout << endl;  cout << "The second array elements before swapping are : ";  for (int i=0; i<6; i++)  cout << ar1[i] << " ";  cout << endl;    // Swapping ar1 values with ar  ar.swap(ar1);    // Printing 1st and 2nd array after swapping  cout << "The first array elements after swapping are : ";  for (int i=0; i<6; i++)  cout << ar[i] << " ";  cout << endl;  cout << "The second array elements after swapping are : ";  for (int i=0; i<6; i++)  cout << ar1[i] << " ";  cout << endl;    return 0;    } |
| --- |

Output:

The first array elements before swapping are : 1 2 3 4 5 6

The second array elements before swapping are : 7 8 9 10 11 12

The first array elements after swapping are : 7 8 9 10 11 12

The second array elements after swapping are : 1 2 3 4 5 6

**9. empty()** :- This function returns true when the array size is zero else returns false.

**10. fill()** :- This function is used to fill the entire array with a particular value.

| // C++ code to demonstrate working of empty()  // and fill()  #include<iostream>  #include<array> // for fill() and empty()  using namespace std;  int main()  {    // Declaring 1st array  array<int,6> ar;    // Declaring 2nd array  array<int,0> ar1;    // Checking size of array if it is empty  ar1.empty()? cout << "Array empty":  cout << "Array not empty";  cout << endl;    // Filling array with 0  ar.fill(0);    // Displaying array after filling  cout << "Array after filling operation is : ";  for ( int i=0; i<6; i++)  cout << ar[i] << " ";    return 0;    } |
| --- |

Output:

Array empty

Array after filling operation is : 0 0 0 0 0 0

STL :: Functors

<https://www.geeksforgeeks.org/functors-in-cpp/>

<https://www.geeksforgeeks.org/transform-c-stl-perform-operation-elements/>

#include <bits/stdc++.h>

using namespace std;

int increment(int x) { return (x+1); }

int main()

{

int arr[] = {1, 2, 3, 4, 5};

int arr2[] = {1, 2, 3, 4, 5};

int n = sizeof(arr)/sizeof(arr[0]);

// Apply increment to all elements of

// arr[] and store the modified elements

// back in arr[]

transform(arr, arr+(n-1), arr, increment);

for (int i=0; i<n; i++)

cout << arr[i] <<" ";

cout << "" <<endl;

for (int i=0; i<n; i++)

cout << arr2[i] <<" ";

return 0;

}

// C++ program to demonstrate working of

// functors.

#include <bits/stdc++.h>

using namespace std;

// A Functor

class increment

{

private:

int num;

public:

increment(int n) : num(n) { }

// This operator overloading enables calling

// operator function () on objects of increment

int operator () (int arr\_num) const {

return num + arr\_num;

}

};

// Driver code

int main()

{

int arr[] = {1, 2, 3, 4, 5};

int n = sizeof(arr)/sizeof(arr[0]);

int to\_add = 5;

transform(arr, arr+n, arr, increment(to\_add));

for (int i=0; i<n; i++)

cout << arr[i] << " ";

}

# std::set\_union in C++

OutputIterator set\_union (InputIterator1 first1, InputIterator1 last1,

InputIterator2 first2, InputIterator2 last2,

OutputIterator result);

// CPP program to illustrate

// std :: set\_union

#include <iostream> // std::cout

#include <algorithm> // std::set\_union, std::sort

#include <vector> // std::vector

// Driver code

int main()

{

int first[] = { 5, 10, 15, 20, 25 };

int second[] = { 50, 40, 30, 20, 10 };

int n = sizeof(first) / sizeof(first[0]);

// Print first array

std::cout << "First array contains :";

for (int i = 0; i < n; i++)

std::cout << " " << first[i];

std::cout << "\n";

// Print second array

std::cout << "Second array contains :";

for (int i = 0; i < n; i++)

std::cout << " " << second[i];

std::cout << "\n\n";

std::vector<int> v(10);

std::vector<int>::iterator it, st;

std::sort(first, first + n);

std::sort(second, second + n);

// Using default function

it = std::set\_union(first, first + n, second, second + n, v.begin());

std::cout << "The union has " << (it - v.begin()) << " elements:\n";

for (st = v.begin(); st != it; ++st)

std::cout << ' ' << \*st;

std::cout << '\n';

return 0;

}

// CPP program to demonstrate use of

// std :: set\_symmetric\_difference

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

using namespace std;

// Driver code

int main()

{

string first[] = { "Sachin", "Rakesh", "Sandeep", "Serena" };

string second[] = { "Vaibhav", "Sandeep", "Rakesh", "Neha" };

int n = sizeof(first) / sizeof(first[0]);

// Print students of first list

cout << "Students in first subject :";

for (int i = 0; i < n; i++)

cout << " " << first[i];

cout << "\n";

// Print students of second list

cout << "Students in second subject :";

for (int i = 0; i < n; i++)

cout << " " << second[i];

cout << "\n\n";

vector<string> v(10);

vector<string>::iterator it, st;

// Sorting both the list

sort(first, first + n);

sort(second, second + n);

// Using default operator<

it = set\_union(first, first + n, second, second + n, v.begin());

cout << "Students attending both subjects are :\n";

for (st = v.begin(); st != it; ++st)

cout << ' ' << \*st;

cout << '\n';

}