VECTORS IN C++

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() – Returns an iterator pointing to the first element in the vector
2. end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
3. rbegin() – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
4. rend() – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)
5. cbegin() – Returns a constant iterator pointing to the first element in the vector.
6. cend() – Returns a constant iterator pointing to the theoretical element that follows the last element in the vector.
7. crbegin() – Returns a constant reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
8. crend() – 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() – Returns the number of elements in the vector.
2. max_size() – Returns the maximum number of elements that the vector can hold.
3. capacity() – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
4. resize(n) – Resizes the container so that it contains ‘n’ elements.
5. empty() – Returns whether the container is empty.
6. shrink_to_fit() – Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
7. reserve() – 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] – Returns a reference to the element at position ‘g’ in the vector
2. at(g) – Returns a reference to the element at position ‘g’ in the vector
3. front() – Returns a reference to the first element in the vector
4. back() – Returns a reference to the last element in the vector
5. data() – 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() – It assigns new value to the vector elements by replacing old ones
2. push_back() – It push the elements into a vector from the back
3. pop_back() – It is used to pop or remove elements from a vector from the back.
4. insert() – It inserts new elements before the element at the specified position
5. erase() – It is used to remove elements from a container from the specified position or range.
6. swap() – It is used to swap the contents of one vector with another vector of same type. Sizes may differ.
7. clear() – It is used to remove all the elements of the vector container
8. emplace() – It extends the container by inserting new element at position
9. emplace_back() – 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
```

EXAMPLES OF VECTORS

Sorting a vector:

```
// C++ program to sort a vector in non-decreasing
// order.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> v{ 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };

    sort(v.begin(), v.end());

    cout << "Sorted \n";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
      }
}
```

REVERSING A VECTOR

Approach: Reversing can be done with the help of reverse() function provided in STL.

Syntax:

```
reverse(start_index, last_index);
```

```
// C++ program to reverse Vector
// using reverse() in STL

#include <bits/stdc++.h>
using namespace std;

int main()
{

    // Get the vector
    vector<int> a = { 1, 45, 54, 71, 76, 12 };

    // Print the vector
    cout << "Vector: ";
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
    cout << endl;

    // Reverse the vector
    reverse(a.begin(), a.end());

    // Print the reversed vector
    cout << "Reversed Vector: ";
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
    cout << endl;


}
```

2D VECTOR:

What is a 2D Vector?

A 2D vector is vector of vectors. It is an matrix implemented with the help of vectors.

```
// C++ code to demonstrate 2D vector
#include<iostream>
#include<vector>
using namespace std;

int main()
```

```
{
    // Initializing 2D vector "vect" with
    // values
    vector< vector<int> > vect{{1, 2, 3},
                               {4, 5, 6},
                               {7, 8, 9}};

    // Displaying the 2D vector
    for (int i=0; i<vect.size(); i++)
    {
        for (int j=0; j<vect[i].size() ;j++)
            cout << vect[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```