Vector In C++ STL | C++ Tutorials for Beginners #71

Vector In C++ STL | C++ Tutorials for Beginners #71

In this video, we'll cover the Vectors in C++ STL. This is the tutorial we all were waiting for. Enough of the theory part. We'll go into our editors and code. So, to start, we'll have to include the header file <vector>. And the syntax we use to define a vector is:

```
vector<data_type> vector_name;
```

Code Snippet 1: Syntax of declaring a vector

And suppose we want to have a vector of integers; the following program would do the needful:

```
#include<iostream>
#include<vector>

int main(){

    vector<int> vec1;
    return 0;
}
```

Code Snippet 2: Declaring a vector of integers

One benefit of using vectors, is that we can insert as many elements we want in a vector, without having to put some size parameter as in an array. In an array of 10 elements, for adding the 11th one, we'll have to make an array again.

Vectors provide certain methods to be used to access and utilise the elements of a vector, first one being, the push_back method. To access all the methods and member functions in detail, you can visit this site , std::vector - C++ Reference. This will be very handy and useful to you. I'll show you how some of them work in a program. Refer to the code snippet 3.

- **push_back() and size():**

1. First of all, don't forget to include the header file, <vector>.
2. Vectors have a method, push_back(), to insert elements in it from the rear end.
3. We'll define a variable, size, to store the size of the vector.
4. We'll then run a loop of size length, to receive the user input and push them back in the vector vec1.
5. We'll then call the display function.
6. We want to have a display function to display the contents of the vector. And pass reference of vec1 to the function.

7. We have another method size() which returns the size of the vector. We'll use this to traverse through all the elements and print them.
8. So, this is how a vector gets used.

```
#include<iostream>
#include<vector>
using namespace std;
void display(vector<int> &v){
    for (int i = 0; i < v.size(); i++)
    {
        cout<<v[i]<<" ";
    }
    cout<<endl;
}
int main(){
    vector<int> vec1;
    int element, size;
    cout<<"Enter the size of your vector"<<endl;
    cin>>size;
    for (int i = 0; i < size; i++)
    {
        cout<<"Enter an element to add to this vector: ";
        cin>>element;
        vec1.push_back(element);
    }
    display(vec1);
```

```
    return 0;
}
```

Code Snippet 3: A program to demonstrate the use of push_back and size methods

The output of the above program:

```
Enter the size of your vector
3
Enter an element to add to this vector: 5
Enter an element to add to this vector: 3
Enter an element to add to this vector: 7
5 3 7
PS D:\MyData\Business\code playground\C++ course>
```

Figure 1: Output of the above program

Similarly, we can even build float vectors, and we can even templatise the display function.

- **pop_back():**

This method of vectors, deletes the last element of the vector. Refer to the code snippet and the following output below.

```
    display(vec1);
    vec1.pop_back();
    display(vec1);
```

Code Snippet 4: Using pop_back in a vector

So, now you can see how this method deleted the last element 7 from the vector.

```
Enter the size of your vector
3
Enter an element to add to this vector: 5
Enter an element to add to this vector: 3
Enter an element to add to this vector: 7
5 3 7
5 3
PS D:\MyData\Business\code playground\C++ course>
```

Figure 2: Output of the above program

- **Insert (**iterator, element to insert**):**

This method of vectors inserts an element to the position the iterator is pointing to. Now how to evoke that iterator? Refer to the snippet and the output below:

We can generate an iterator using the scope resolution iterator by the following syntax:

```
vector<int> :: iterator iter = vec1.begin();
```

Code Snippet 5: Declaring a vector iterator

Using **begin ()** points the iterator to the starting of the vector. We can now increment the pointer according to our choice and insert any element at that position.

```
display(vec1);
    vector<int> :: iterator iter = vec1.begin();
    vec1.insert(iter,566);
    display(vec1);
```

Code Snippet 6: Demonstrating an insert method

The output of the above program is:

```
Enter the size of your vector
3
Enter an element to add to this vector: 5
Enter an element to add to this vector: 3
Enter an element to add to this vector: 7
5 3 7
566 5 3 7
PS D:\MyData\Business\code playground\C++ course>
```

Figure 3: Output of the above program

Similarly, **v.at(i)** can be used instead of **v[i]**. They will work the same.

We have different ways to declare a vector. I'll list some of them through the snippet below.

1. First one is a vector with no length and elements specified.
2. Second one is a vector of length 4 and no elements.
3. Third one is a vector made from the second one.
4. And last one, is a vector with length 6 and all the elements being 3.

```
vector<int> vec1;      //zero length integer vector
vector<char> vec2(4);  //4-element character vector
vector<char> vec3(vec2);//4-element character vector from vec2
vector<int> vec4(6,3); //6-element vector of 3s
```

Code Snippet 7: Demonstrating different ways to declare a vector

So this was all the basics of a vector, and enough for you to get started using it. With this I'll finish today's tutorial. I'll recommend you all to visit my DSA playlist, Data Structures and Algorithms Course in Hindi to get acquainted with all the data structures and more.

Thank you, for being with me throughout, hope you liked the tutorial. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. I hope you enjoy them all. See you all in the next tutorial where we'll learn about Lists in C++ STL. Till then keep coding.

Previous Next