List In C++ STL | C++ Tutorials for Beginners #72

List In C++ STL | C++ Tutorials for Beginners #72

Before this tutorial, we covered templates, STL, and the last video was an efficient introduction to the vectors. Today, we'll learn about Lists in C++ STL.

A List is a bi-directional linear storage of elements. Few key features as to why a list should be used is,

1. It gives faster insertion and deletion operations.
2. Its access to random elements is slow.

What makes a list different from an array?

An array stores the elements in a contiguous manner in which inserting some element calls for a shift of other elements, which is time taking. But in a list, we can simply change the address the pointer is pointing to. I'll

show you how these work via an illustration.

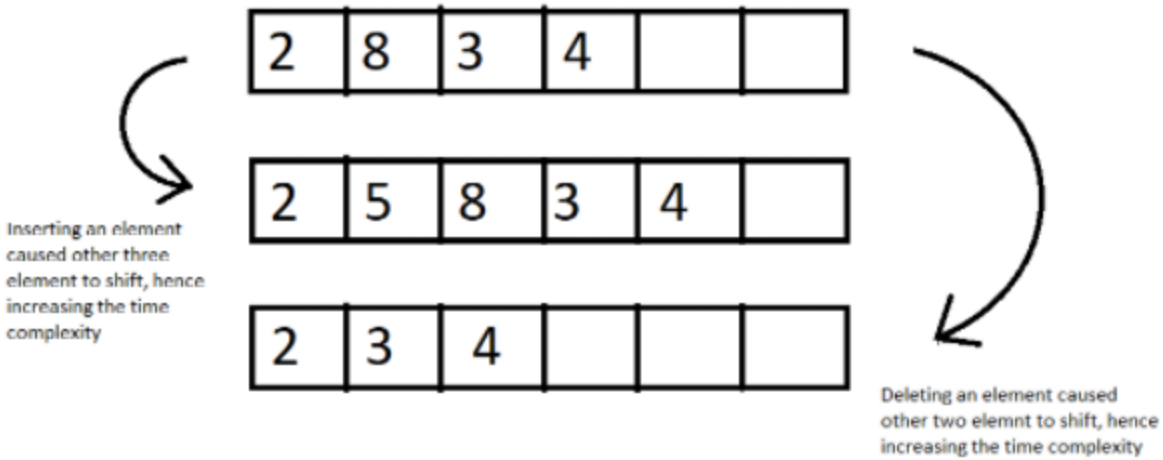

Figure 1: Inserting and deleting in an array

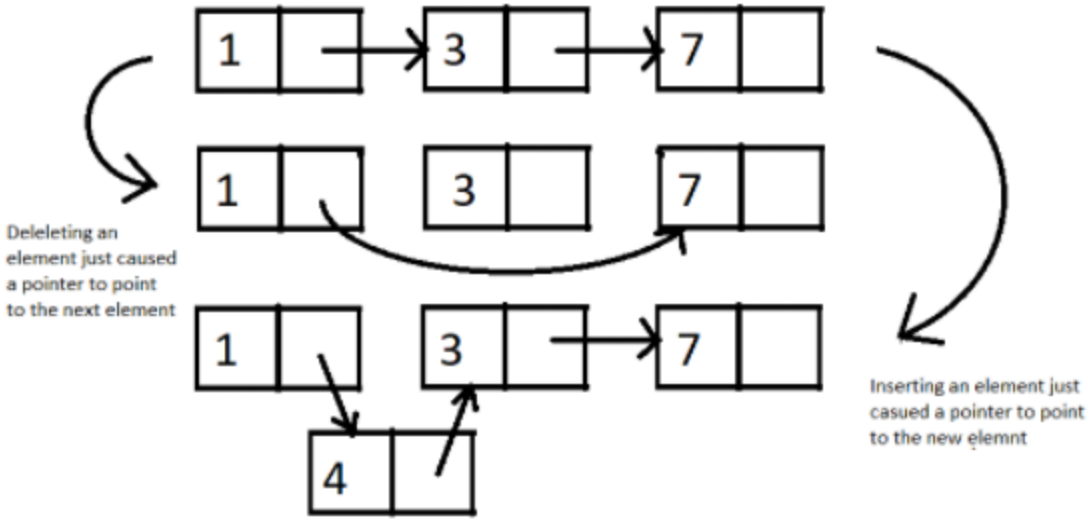

Figure 2: Insertion and deletion in a list

Let's move on to our editors and write some code using lists and its methods.

Understanding code snippet 2:

- Before using lists, we must include the header file <list>.
- Using a simple program, we'll iterate through the list and display its contents.
- As we did for vectors, first define a list list1.
- And push_back a few elements, and pass the list to a display function via reference.
- Due to the fact that a list element cannot be directly accessed by its index, we must traverse through each element and print them.
- We define a list iterator using this syntax:

```
list<int> :: iterator it;
```

Code Snippet 1: Syntax for defining a list iterator

- We use two methods, **begin()** and **end()** to define the starting and the end of the loop. **end()** returns the pointer next to the last element.
- We dereference the list iterator, using * to print the element at that index.

```
#include<iostream>
#include<list>

using namespace std;

void display(list<int> &lst){
    list<int> :: iterator it;
    for (it = lst.begin(); it != lst.end(); it++)
```

```
    {
        cout<<*it<<" ";
    }

}

int main(){

    list<int> list1;  //empty list of 0 length

    list1.push_back(5);
    list1.push_back(7);
    list1.push_back(1);
    list1.push_back(9);
    list1.push_back(12);

    display(list1);

    return 0;
}
```

Code Snippet 2: A program using list

```
5 7 1 9 12
PS D:\MyData\Business\code playground\C++ course>
```

Figure 3: Output of the above program

We can also enter elements in a list using the iterator and its dereferencer. See the snippet below.

```
int main(){

    list<int> list2(3);  //empty list of length 3
    list<int> :: iterator it = list2.begin();
    *it = 45;
    it++;
    *it = 6;
    it++;
    *it = 9;
    it++;

    display(list2);

    return 0;
}
```

Code Snippet 3: Inserting in list using its iterator

```
45 6 9
PS D:\MyData\Business\code playground\C++ course>
```

Figure 4: Output of the above program

- **Using pop_back() and pop_front():**

We can use pop_back() to delete one element from the back of the list everytime we call this method and pop_front() to delete elements from the front. These commands decrease the size of the list by 1. Let me show you how these work by using them for list1 we made.

```
list1.pop_back();
display(list1);
list1.pop_front();
display(list1);
```

Code Snippet 4: Using pop_back and pop_front in list

The output of the above program is:

```
5 7 1 9
7 1 9
PS D:\MyData\Business\code playground\C++ course>
```

Figure 5: Output of the above program

- **Using remove():**

We can remove an element from a list by passing it in the list remove method. It will delete all the occurrences of that element. The remove method receives one value as a parameter and removes all the elements which match this parameter. Refer to the use of remove in the below snippet.

```
int main(){

    list<int> list1;  //empty list of 0 length
```

```
    list1.push_back(5);
    list1.push_back(7);
    list1.push_back(1);
    list1.push_back(9);
    list1.push_back(9);
    list1.push_back(12);

    list1.remove(9);
    display(list1);

    return 0;
}
```

Code Snippet 5: Deleting elements in list using remove()

The output of the above program is:

```
5 7 1 12
PS D:\MyData\Business\code playground\C++ course>
```

Figure 6: Output of the above program

- **Using sort():**

We can sort a list in ascending order using its sort method. Look for the demo below.

```
display(list1);
list1.sort();
display(list1);
```

Code Snippet 6: Sorting elements in list using sort()

```
5 7 1  9 12
1 5 7  9 12
PS D:\MyData\Business\code playground\C++ course>
```

Figure 7: Output of the above program

I consider this much to be enough for lists. There are still a lot of them, but you will require no more, and even if you feel like exploring more, move onto std::list - C++ Reference and read about all the lists methods. This was all from my side. For more information on linked lists, visit my DSA playlist, Data Structures and Algorithms Course in Hindi.

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 Maps in C++ STL. Till then keep coding.

Previous Next