**C++ Lists**

**A. What are lists?**

Lists are sequence containers that fall under the category of class templates. They are also used to store information in a linear fashion. Only elements of a single data type can be stored in a list. In a list, we have faster insertion and deletion of elements as compared to other containers such as arrays and vectors.  Although, accessing elements at some random position, is comparatively slower.

**B. What makes inserting in and deleting from a list faster?**

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.

**C. Using a list in our programs**

To be able to use lists in our code, the header file <list> must be included. And the syntax to define a list is

list<data\_type> list\_name;

The data\_type could be replaced by any data type. One benefit of using lists is that lists support a bidirectional and provide an efficient way for insertion and deletion operations.

Lists provide certain methods to be used to access and utilize the elements of a list, the first one being, the push\_back method. To access all the methods and member functions in detail, one can visit this site, [std::list](https://cplusplus.com/reference/list/list/).

**D. Initialising a list**

A list could be initialized in a very similar way we used to initialize a vector.: A list could be initialized with all the elements inserted in the list at the time it is defined.

list<int> l = {1, 2, 3, 4};

OR it could be even be initialised with elements to be inserted as a parameter.

list<int> l{1, 2, 3, 4};

**E. Inserting elements in a list**

We can insert elements in a list in different ways. The insertion of any element in a list takes constant time, making it faster than insertion in vectors or arrays. Insertion in a list could be done using the push\_back() and the push\_front() methods. The push\_back() function gets the elements to be inserted as a parameter and the element gets pushed at the back. In contrast, the push\_front() function gets the elements to be inserted as a parameter and the element gets pushed at the front.

#include <iostream>

#include <list>

using namespace std;

void display(list<int> l)

{

cout << "The elements are: ";

for (auto it : l)

{

cout << it << " ";

}

cout << endl;

}

int main()

{

list<int> l = {1, 2, 3, 4};

display(l);

l.push\_back(5);

display(l);

l.push\_front(0);

display(l);

}

Output:

The elements are: 1 2 3 4

The elements are: 1 2 3 4 5

The elements are: 0 1 2 3 4 5

Another method to insert an element allows us to insert at any random position. Here, we use the insert() method. The syntax for using the insert method is

list\_name.insert(iterator, element);

Here, the iterator is the pointer to that position where the element gets inserted. An example that demonstrates the use of the insert() method is

list<int> l = {1, 2, 3, 4};

display(l);

auto it = l.begin();

it++;

it++;

l.insert(it, 5); //5 is inserted at the third position

display(l);

Output:

The elements are: 1 2 3 4

The elements are: 1 2 5 3 4

**F. Accessing/Changing elements in a list**

Changing or accessing any element in a list is a costlier process in terms of time. Any element at any specified position could not be simply accessed using its index number or anything that we could be done for arrays or vectors.

We have to use the iterators to manually traverse through the list to reach a specific element or position and then dereference the iterator to retrieve the value at that position.

list<int> l = {1, 2, 3, 4};

list<int>::iterator it = l.begin();

it++;

it++;

cout << "Element at index 2 is " << \*it << endl;

Output:

Element at index 2 is 3

**G. Removing elements from a list**

We can remove elements from a list in different ways. The removal of any element in a list takes constant time, making it faster than removal in vectors or arrays. Removal in a list could be done using the pop\_back() and the pop\_front() methods. The pop\_back() function needs nothing as a parameter and the element gets popped from the back while the pop\_front() function pops the element from the front.

#include <iostream>

#include <list>

using namespace std;

void display(list<int> l)

{

cout << "The elements are: ";

for (auto it : l)

{

cout << it << " ";

}

cout << endl;

}

int main()

{

list<int> l = {1, 2, 3, 4};

display(l);

l.pop\_back();

display(l);

l.pop\_front();

display(l);

}