Tutorial 72 - Lists in C++ STL

Introduction to Lists

- A **List** is a **bi-directional** (doubly linked) linear data structure in C++ STL.
- **Advantages of Lists**:
  a. **Faster Insertion & Deletion** – No shifting of elements is required, unlike arrays.
  b. **Slower Random Access** – Elements are not stored contiguously.

Differences Between Arrays & Lists

Feature	Arrays	Lists
Storage	Contiguous memory	Non-contiguous memory
Insertion & Deletion	Slow (requires shifting)	Fast (pointer manipulation)
Random Access	Fast (direct indexing)	Slow (must traverse)

Working with Lists in C++

1. Basic List Operations

Declaring & Displaying a List

- **Before using lists, include the `<list>` header file.**
- **Defining a List**:

```
list<int> list1;  // Empty list of integers

```

- **Iterating & Displaying a List (Using Iterator)**:

```
#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 << " ";
    }
    cout << endl;
}

int main() {
    list<int> list1;
    list1.push_back(5);
    list1.push_back(7);
    list1.push_back(1);
    list1.push_back(9);
    list1.push_back(12);

```

```
    display(list1);
    return 0;
}

```

Output:

```
5 7 1 9 12

```

2. Inserting Elements Using Iterator

- **Using Iterators for Direct Insertion**:

```
list<int> list2(3);  // List of size 3
list<int>::iterator it = list2.begin();

*it = 45; it++;
*it = 6; it++;
*it = 9; it++;

display(list2);

```

Output:

```
45 6 9

```

---

3. List Methods in C++ STL

a) `push_back()` & `push_front()`

- `push_back()`: Inserts an element at the end.
- `push_front()`: Inserts an element at the beginning.

b) `pop_back()` & `pop_front()`

- `pop_back()`: Removes the last element.
- `pop_front()`: Removes the first element.

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

```

Output:

```
5 7 1 9
7 1 9

```

c) `remove(value)`

- Removes **all occurrences** of a specified element.

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

```

Output:

```
5 7 1 12

```

d) `sort()`

- **Sorts the list in ascending order.**

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

```

Output:

```
1 5 7 9 12

```

Short Notes on List in C++ STL

- **List is a doubly linked list (bi-directional).**
- **Key Methods:**
  - `push_back()` → Add element at the end
  - `push_front()` → Add element at the front
  - `pop_back()` → Remove last element
  - `pop_front()` → Remove first element
  - `remove(value)` → Removes all occurrences of `value`
  - `sort()` → Sorts the list in ascending order
- **Iterators** are used to traverse and modify elements in a list.
- **Lists are preferred over arrays when frequent insertions & deletions are needed.**

Conclusion

- Lists in C++ STL provide **efficient insertion and deletion** but **slow random access**.
- More details can be explored at **std::list - C++ Reference**.
- Next, we will learn about **Maps in C++ STL**. 🚀