LAB 10: STL ,Containers, Algorithms, Iterators

Lab Objectives

By the end of this lab, students will be able to:

- Understand the internal structure and behavior of std::list (doubly linked list).
- Use essential std::list functions for insertion, deletion, iteration, sorting, merging, reversing, and splicing.
- Apply std::list in an object-oriented scenario.
- Compare std::list with other STL containers like vector and deque.

Introduction:

The **Standard Template Library (STL)** is one of the most influential additions to the C++ language, officially adopted into the ANSI/ISO C++ standard in 1998. It was designed by Alexander Stepanov and his team with the vision of enabling **generic programming**, where algorithms and data structures are written independently of the data types they operate on. STL achieves this using **templates**, allowing developers to write highly flexible and type-safe code. The library provides a comprehensive collection of **predefined classes and functions** that implement common data structures like arrays, linked lists, trees, queues, stacks, and associative containers, all optimized for performance and portability.

A key strength of STL lies in its **well-organized architecture**, built on four main components: **containers, iterators, algorithms, and function objects (functors)**. Containers manage the storage of objects; iterators serve as a generalized pointer mechanism for traversing container elements; algorithms perform tasks such as sorting, searching, counting, or transforming data; and function objects allow functions to be passed as arguments to algorithms, improving flexibility. All components are designed to work together seamlessly through a unified interface. This separation allows programmers to change a container type without rewriting the algorithm, promoting modular and maintainable code.

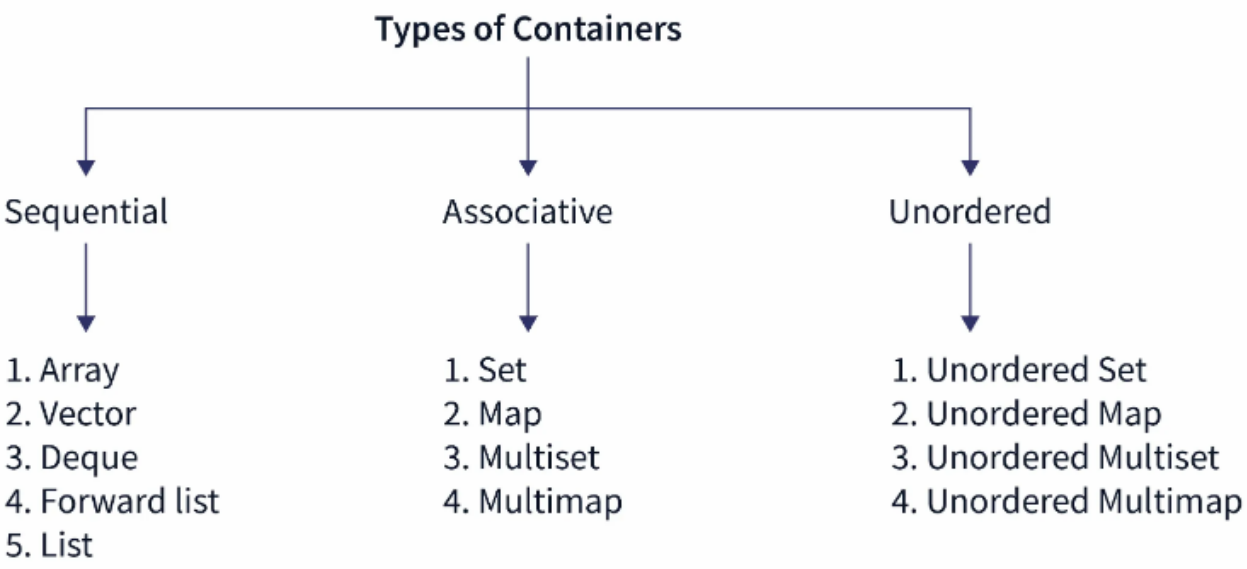
In practice, STL dramatically increases development speed while ensuring high performance, because its implementations are optimized by compiler vendors. Instead of building data structures or algorithms from scratch, developers can rely on these **tested, reusable, and efficient components**, reducing bugs and improving reliability. STL has become a fundamental part of C++ programming, widely used in both academic learning and industrial applications due to its efficiency, scalability, and elegant use of templates.

Features

- Fast insertion/deletion anywhere (O(1)).
- No random access (list[i] ✖).
- Bidirectional iterators.

There are three major components of STL in C++:

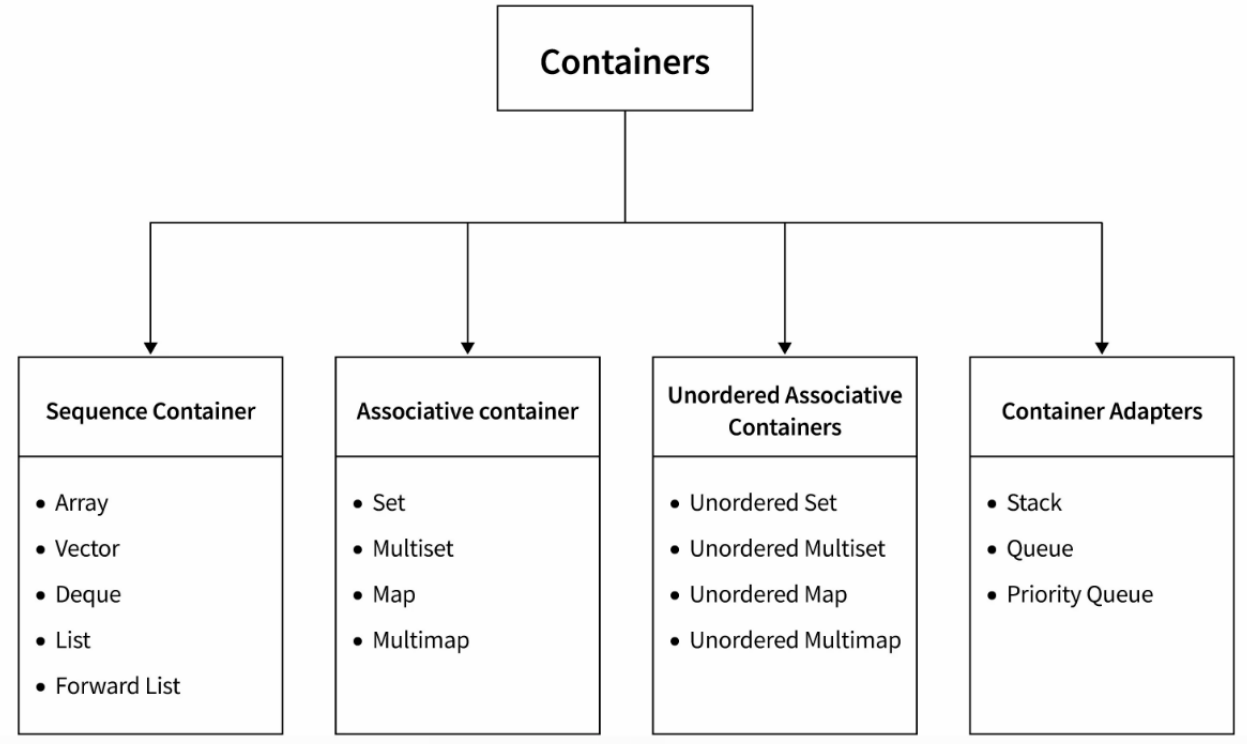
- Containers
- Iterators
- Algorithms



Common Functions

Function	Description
push_back(x) / push_front(x)	Add at end / start
pop_back() / pop_front()	Remove from end / start
insert(pos, x)	Insert at specific iterator position
erase(pos)	Remove element at iterator
remove(x)	Remove all occurrences of x
remove_if(cond)	Remove elements matching condition
sort()	Sort the list

reverse()	Reverse the order
merge(other_list)	Merge two sorted lists
splice(pos, other_list)	Move nodes from another list without copying



1. Array

Arrays are sequential homogeneous containers of fixed size. The elements are stored in contiguous memory locations.

Syntax:

- array<object_type, size> array_name;

2. List

The list is a sequence container that allows insertions and deletions from anywhere. It is a doubly linked list. They allow non-contiguous memory allocation for the elements.

Syntax:

- list<object_type> list_name;

5. Forward List

Forward Lists are introduced from C++ 11. They are implemented as singly linked list in STL in C++. It uses less memory than lists and allows iteration in only a single direction.

Syntax:

- **forward_list<object_type> forward_list_name;**

Example 1 Basic Operations on std::list

lab10.cpp

```
using namespace std;

int main() {
    list<int> L;

    L.push_back(10);
    L.push_back(40);
    L.push_front(5);
    L.push_front(1);

    cout << "List elements: ";
    for (list<int>::iterator it = L.begin(); it != L.end(); ++it)
        cout << *it << " ";
    cout << endl;

    L.reverse();
    cout << "After reverse: ";
    for (list<int>::iterator it = L.begin(); it != L.end(); ++it)
        cout << *it << " ";
    cout << endl;

    L.sort();
    cout << "After sort: ";
    for (list<int>::iterator it = L.begin(); it != L.end(); ++it)
        cout << *it << " ";
    cout << endl;

    return 0;
}
```

List elements: 1 5 10 40

After reverse: 40 10 5 1

After sort: 1 5 10 40

Task 1: Create a program that stores marks of 5 subjects for each student using `array<int,5>`.
Then store multiple students using a `list` of these arrays.

Requirements:

1. Input marks for *N* students.
2. Store each student's marks in `array<int,5>`.
3. Insert each array into a `list<array<int,5>>`.
4. Write functions to:
    - Calculate each student's total and average
    - Remove students with average < 50
    - Display remaining students

Task 2: Use `forward_list` to store normal patients and `list` to store emergency patients.

Requirements:

1. `forward_list<string>` → normal patients
2. `list<string>` → emergency patients
3. Insert patients based on type.
4. When doctor calls next patient:
    - Serve emergency patient first
    - If no emergency, serve from normal queue
5. Remove the served patient.
6. Display updated queues.

Task 3: You are managing an inventory of products. Each product has:

- ID
- Price
- Quantity

Store each product in an `array<int,3>` and all products in a `forward_list<array<int,3>>`.

Requirements:

1. Insert products at the front of the forward_list.
2. Delete a product using its ID.
3. Update the quantity of a product.
4. Find and display the most expensive product by traversing the `forward_list`.
5. Display the full inventory.

Task 4: Create a music playlist program using a mix of STL containers.

Requirements:

1. Add songs to the playlist (`list`).
2. When a song is played:

    - Move it to `recently played` (`forward_list`).
    - Keep only last 5 recently played songs (delete older ones).
3. Remove a song from the playlist by title.
4. Display full playlist.
5. Display recently played songs.
6. Sort the playlist by song title.

Lab #10 Marks distribution

	ER1	**ER6**	**ER8**
Task	3 points	3 points	4 points

Lab # 10 Rubric Evaluation Guideline:

#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1 < Excellent <=3**
ER 1	**Task Completion**	Minimal or no program functionality was achieved.	Some tasks were completed, but the program has errors or incomplete functionalities.	All tasks were completed, and the program runs without errors.
#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1 < Excellent <=3**
ER 6	Program Output	Output is inaccurate or poorly presented.	Output is mostly accurate but may lack labels, captions, or formatting.	Output is clear, accurate, and well presented with labels, captions, and proper formatting.
#	**Qualities & Criteria**	**0 < Poor <= 0.5**	**0.5< Satisfactory <= 1.5**	**1 < Excellent <= 4**
ER 8	Question & Answer	Answers some questions but not confidently or based on lab task knowledge.	Answers most questions confidently and based on lab task knowledge.	Answers all questions confidently and demonstrates a deep understanding of the given lab task.