National University of Computer and Emerging Sciences

Laboratory Manual

for

Data Structures Lab

Course Instructor	Mr. Muhammad Naveed
Lab Instructor	Mr. Durraiz Waseem
Lab Demonstrator	Ms. Adeela Nasir
Section	BDS-3A
Date	Sept 16, 2025
Semester	Fall 2025

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives

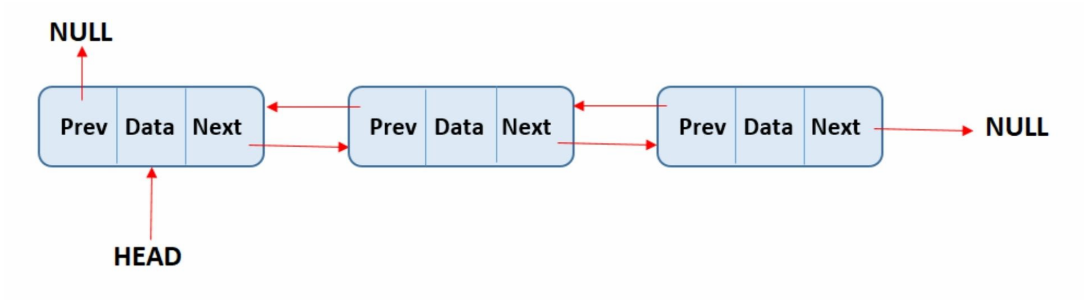
In this lab, students will implement a doubly linked list from scratch, including both basic operations and an iterator class for traversal.

Overview

1. **Doubly Linked Lists**

A doubly linked list is a data structure in which each node stores a value and two pointers: one pointing to the next node and one pointing to the previous node. This bidirectional linking **makes it easier to traverse the list from either direction** and simplifies certain insertion and deletion operations compared to singly linked lists.

2.



Iterators

An iterator is an abstraction that allows step-by-step navigation over a container *without exposing its underlying structure,* providing a single universal method to navigate many data structures in the C++ standard library.

In-Lab Exercises

Question 1: Doubly Linked Lists

Implement a doubly linked list using a templated class with the following methods.

- Constructor, Copy Constructor, and Destructor
- Insertion Methods
```
void insertHead(const T&);
void insertTail(const T&);
void insert(const T&, int index);
```
- Removal Methods
```
void removeHead();
void removeTail();
void remove(int index);
```
- Public Interface
```
int size() const;
```

Question 2: Iterators

Part 1: Basic Implementation

Implement a **private** 'Iterator' class nested within the Doubly Linked List from Question 1 with the following methods:

```
T& operator*();

List<T>::Iterator operator++();
List<T>::Iterator operator++(int);


List<T>::Iterator operator--();
List<T>::Iterator operator--(int);
```

Part 2: Public Interface in List

Implement the following public member functions in the List class

```
List<T>::Iterator begin() const;    // points to head
List<T>::Iterator end() const;      // points to nullptr


List<T>::Iterator rbegin() const;   // points to tail
List<T>::Iterator rend() const;     // points to nullptr
```

Part 3: Type Aliasing

Currently, the begin and end functions in our List class return a datatype unknown to the public user. To solve this, you must alias the `private List<T>::Iterator` datatype to a `public List<T>::iterator` datatype.

Note that case-sensitivity makes these two different identifiers.

Question 3: Using Iterators

The tasks in this question will be ordinary function templates (meaning they are not member functions to any class). However, you must use iterators to navigate through the list.

Part 1: Basic Traversal

```
void print(List<T>&);

T max(List<T>& list);

void bubbleSort(List<T>& list);
```

Part 2: Advanced Traversal

```
List<List<T>> calculateTranspose (List<List<T>>& matrix);
```