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 9, 2025
Semester	Fall 2025

Department of Computer Science

FAST-NU, Lahore, Pakistan

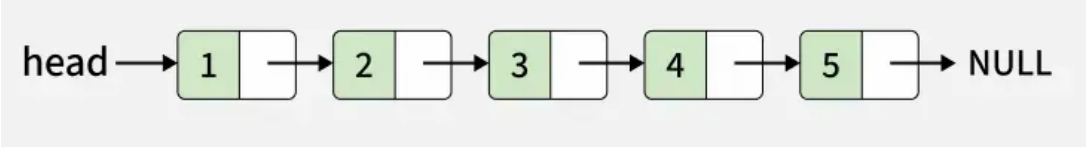
Objectives

In this lab, students will implement both circular and non-circular singly linked lists, as well as some basic operations involving pointer management. In doing so, they will learn when circularity is useful.

Overview

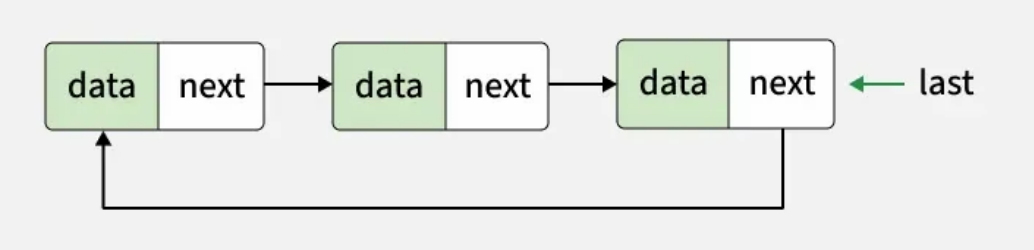
1. Linked Lists

A linked list is a way to store a collection of elements, similar to arrays. Unlike arrays, however, the elements are not stored in **contiguous memory**. Instead, each element is stored in a **node**, and nodes are connected together using **pointers**, forming a chain.



2. Circular Linked List

A circular linked list is a variation of the singly linked list in which the last node points back to the head instead of a *nullptr*. Essentially, there is no end of the list; traversal can continue indefinitely and you can start at any node and eventually return to it by following the links.



3. Why Use Circular Linked Lists?

- Circular linked lists make Rotation operations extremely trivial to implement
- No *nullptr*'s causing NullReferenceExceptions

For these reasons, this data structure is commonly used in Operating System process queues, memory buffers, etc.

In-Lab Exercises

Question 1: Singly Linked Lists

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

Part 1: Basic Implementation

- 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);
```

- Accessing Interface

```
int getSize() const;
T& operator[](int);
```

Part 2: Separate Equal Values into Separate Lists

```
List<List<T>> separateEquals() const;
```

Output

```
[1, 5, 3, 3, 5, 3]   →   [[1], [5, 5], [3, 3, 3]]
["some string"]      →   [["some string"]]
[a, b, b, c, b]      →   [[a], [b, b, b], [c]]
[]                   →   []
```

Part 3: Efficient Pointer-Manipulation Operations

- For the methods in this section, you are NOT allowed to remove or insert any node.

```
void rotateLeft(int k);
void rotateRight(int k);
void reverse();
```

Question 2: Circular Singly Linked Lists

Part 1: Circular List Implementation

Implement a circular linked list with all of the same methods as in Question 1.

Part 2: Implementing a CircularQueue class using the CircularList template from Part 1

```
template <typename T>
class CircularQueue {
private:
     CircularList<T> queue;
public:
     void push_back(const T&);
     const T& pop_front();

     T& get_front() const;
     void next();              // rotate left
};
```