Data Structure and Algorithm Lab Manual (Week-04)

Lab Instructor: Ms. Rabeeya Saleem

Session: 2024 (Fall 2025)

Implement LinkedList class in C++ which must have following functions.

Node Structure

```
// Node structure for singly linked list
struct Node {
   int data;
   Node* next;
   Node(int value) {
      data = value;
      next = nullptr;
   }
};
```

LinkList Class

```
#include <iostream>
using namespace std;

// Linked List class
class LinkList {
private:
   Node* head;   // Pointer to the first node

public:
   // Constructor
   LinkList() {
      head = nullptr;
   }

   // Destructor - deletes all nodes
   ~LinkList() {
      Node* current = head;
      while (current != nullptr) {
         Node* nextNode = current->next;
         delete current;
         current = nextNode;
      }
      head = nullptr;
   }
   // Check if list is empty
   bool isEmpty() {
      return head == nullptr;
   }
};
```

```
// Insert at head
Node* insertAtHead(int x) {
    Node* newNode = new Node(x);
    newNode->next = head;
    head = newNode;
    return head;
}

// Insert at end
Node* insertAtEnd(int x) {
    Node* newNode = new Node(x);
    if (head == nullptr) {
        head = newNode;
        return head;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    temp->next = newNode;
    return head;
}

// Insert at a specific index (Position-based)
Node* insertNode(int index, int x) {
    Node* newNode = new Node(x);

    if (index == 0) {  // Insert at head
        newNode->next = head;
        head = newNode;
        return head;
    }

    Node* temp = head;
    int count = 0;

    while (temp != nullptr && count < index - 1) {
        temp = temp->next;
        count++;
    }

    if (temp == nullptr) {
        cout << "Index out of range!\n";
        delete newNode;
        return nullptr;
    }
```

```
    newNode->next = temp->next;
    temp->next = newNode;
    return head;
  }
```

```
// Delete all occurrences of x
  bool deleteNode(int x) {
    if (head == nullptr)
      return false;

    bool deleted = false;

    // Delete from the start if needed
    while (head != nullptr && head->data == x) {
      Node* toDelete = head;
      head = head->next;
      delete toDelete;
      deleted = true;
    }

    // Delete from middle or end
    Node* temp = head;
    while (temp != nullptr && temp->next != nullptr) {
      if (temp->next->data == x) {
        Node* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
        deleted = true;
      } else {
        temp = temp->next;
      }
    }

    return deleted;
  }

  // Delete from start
  bool deleteFromStart() {
    if (head == nullptr)
      return false;
    Node* temp = head;
    head = head->next;
    delete temp;
    return true;
  }

  // Delete from end
  bool deleteFromEnd() {
    if (head == nullptr)
      return false;
```

```
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return true;
    }

    Node* temp = head;
    while (temp->next->next != nullptr) {
        temp = temp->next;
    }

    delete temp->next;
    temp->next = nullptr;
    return true;
}
```

```
// Find a node with value x (Searching)
bool findNode(int x) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == x)
            return true;
        temp = temp->next;
    }
    return false;
}
```

```
// Display list
void displayList() {
    Node* temp = head;
    cout << "List: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

```
// Main function to test
int main() {
    LinkList list;

    list.insertAtEnd(10);
    list.insertAtEnd(30);
    list.insertAtHead(5);
```

```
    list.insertNode(2, 20);  // Insert at index 2
    list.displayList();     // 5 10 20 30

    list.deleteNode(10);
    list.displayList();     // 5 20 30
    list.displayList();     // 30 20 5

    list.displayList();     // 5 20 30

    return 0;
}
```

Class Activity:

Update Node Class in the above LinkList, Create a new class with Data, Next and Prev pointer, Create DoublyLinkList and rewrite all the operations where required. Compare the time complexity of operations in LinkList and DoublyLinkList.

Problems1-7:

Write a function to find the middle node in a Singly Linked List using the two-pointer technique (slow and fast pointers). If the list has an even number of nodes, return the second middle node	**Input:** 10 → 20 → 30 → 40 → 50 **Output:** 30 **Input:** 5 → 10 → 15 → 20 → 25 → 30 **Output: 20**
Implement a function to **detect a loop** in a **Singly Linked List.**	10 → 20 → 30 → 40 → 50 ↑ ↓ ← Output: yes

Given an array of integers, the task is to Implement a function to **reverse a Singly Linked List**. Describe the process involved in reversing the list	**Input:**
	`10 → 20 → 30 → 40 → 50`
	Output:
	`50 → 40 → 30 → 20 → 10`

Implement a function to **rotate a singly linked list** by `k` nodes.	**Input**: `1 -> 2 -> 3 -> 4 -> 5` and `k = 2`, **Output:** `3 -> 4 -> 5 -> 1 ->`

Write a function to **segregate even and odd nodes** in a **Singly Linked List**, such that all even nodes appear first, followed by all odd nodes.	**Input:**
	`17 → 15 → 8 → 9 → 2 → 4 → 6`
	Output:
	`8 → 2 → 4 → 6 → 17 → 15 → 9`

Write a function to **remove duplicates** from a un**sorted singly linked list**.	**Input:** `10 → 20 → 10 → 30 → 20 → 40` **Output:** `10  20 → 30 → 40`

Write functions to find the **union and Intersection** of two **unsorted singly linked lists**.	`List1: 10 15 4 20`
	`List2: 8 4 2 10`
	`Union: 10 15 4 20 8 2`
	`Intersection: 10 4`