**Department of Computing**

**School of Electrical Engineering and Computer Science**

**CS250 – Data Structures and Algorithms**



**Lab 4: Implementation of Linked List with its Operations**

**Submission Details**

|  |  |
| --- | --- |
| Name | CMS ID |
| Muhammad Umer | 345834 |
| Group | GP – 1 |
| Lab Engineer | Anum Asif |
| Faculty Member | Bostan Khan |
| Class | BEE12 |
| Date | 26/02/2024 |
| Time | 10:00 am – 12:50 pm |

**Table of Contents**

[2 Implementation of Linked List with its Operations 3](#_Toc159833669)

[2.1 Introduction 3](#_Toc159833670)

[2.2 Objectives 3](#_Toc159833671)

[2.3 Tools/Software Requirement 3](#_Toc159833672)

[2.4 Description 3](#_Toc159833673)

[2.5 Deliverables 3](#_Toc159833674)

[3 Lab Task 4](#_Toc159833675)

[4 Conclusion 9](#_Toc159833676)

# Implementation of Linked List with its Operations

## Introduction

Students have learned the fundamental concepts of linked lists in the lectures. This lab will introduce students with the practical implementation of a linked list and different operations that can be performed on a linked list.

## Objectives

The objective of this lab is to get familiar with singly linked list and implement them in C++.

## Tools/Software Requirement

* Visual Studio C++

## Description

A Linked List is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of two parts i.e. data part and a reference part (also known as, a link) to the next node in the sequence. This structure allows efficient insertion or removal of elements from any position in the sequence.

Singly-linked-list.svg

The basic operation consists of

* *Creating* the list
* *Initialize* pointers to NULL
* *Inserting* nodes at beginning, last and from a specific location
* *Deletion* of nodes from beginning, last and from a specific location
* *Traversing* the list
* *Destroying* the list

## Deliverables

Compile a single word document by filling in the solution parts and submit this file on LMS. The name of word document should follow this format. i.e., YourFullName(reg)\_Lab#. You must show the implementation of the tasks in a complete manner to get your work graded.

***Note: Students are required to upload the lab on LMS before deadline.***

# Lab Task

Write a C++ program that can:

1. Create a simple linked list using functions to insert nodes at the head.
2. Make a function that can insert another node at the 3rd location.
3. Make a function that can display the lists made in 1 and 2.
4. Write a function that can delete node from the linked list selected by the user. Display it as well.
5. Write a function that can count the number of nodes present in the list.
6. Create menu in main function to give call to all of the above functions depending upon user’s input.

**Hint:** First you will create the relevant classes, and the functions will belong to the List class.

//class of node

class Node {

public:

int value;

node \*next;

};

Required functions for list class are:

void insertAtHead(int new\_value);

void insertAtLocation(int location,int new\_value)

void delete(int del\_value)

void displayList()

void countList()

Code

#include <iostream>

using *namespace* std;

*void* display\_menu() {

    cout << "\*\*== Task Menu ==\*\*" << endl;

    cout << "1) Insert at head" << endl;

    cout << "2) Insert at location" << endl;

    cout << "3) Display" << endl;

    cout << "4) Delete" << endl;

    cout << "5) Count" << endl;

    cout << "6) Exit" << endl;

    cout << "Enter your choice: ";

}

*class* Node {

*public:*

*int* value;

    Node \*next;

};

*class* List {

*public:*

    Node \*head;

    List() { head = nullptr; }

*void* insertAtHead(*int* *new\_value*);

*void* insertAtLocation(*int* *location*, *int* *new\_value*);

*void* deleteNode(*int* *del\_value*);

*void* displayList();

*void* countList();

};

*void* List::insertAtHead(*int* *new\_value*) {

    Node \*new\_node = new Node;

    new\_node->value = *new\_value*;

    new\_node->next = head;

    head = new\_node;

}

*void* List::insertAtLocation(*int* *location*, *int* *new\_value*) {

    Node \*new\_node = new Node;

    new\_node->value = *new\_value*;

    Node \*temp = head;

    for (*int* i = 1; i < *location* - 1; i++) {

        temp = temp->next;

    }

    new\_node->next = temp->next;

    temp->next = new\_node;

}

*void* List::deleteNode(*int* *del\_value*) {

    Node \*target = head;

    Node \*prev = nullptr;

    if (target != nullptr && target->value == *del\_value*) {

        head = target->next;

        delete target;

        return;

    }

    while (target != nullptr && target->value != *del\_value*) {

        prev = target;

        target = target->next;

    }

    if (target == nullptr) {

        cout << "Value not found" << endl;

        return;

    }

    prev->next = target->next;

    delete target;

}

*void* List::displayList() {

    Node \*temp = head;

    while (temp != nullptr) {

        cout << temp->value << " -> ";

        temp = temp->next;

    }

    cout << "NULL" << endl;

}

*void* List::countList() {

*int* count = 0;

    Node \*temp = head;

    while (temp != nullptr) {

        count++;

        temp = temp->next;

    }

    cout << "Number of nodes: " << count << endl;

}

*int* main() {

    List list;

*int* choice; *// user's choice*

*int* value;

*int* location;

    while (1) { *// infinite loop until user chooses to exit*

        display\_menu();

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value: ";

                cin >> value;

                list.insertAtHead(value);

                break;

            case 2:

                cout << "Enter location: ";

                cin >> location;

                cout << "Enter value: ";

                cin >> value;

                list.insertAtLocation(location, value);

                break;

            case 3:

                list.displayList();

                break;

            case 4:

                cout << "Enter value to delete: ";

                cin >> value;

                list.deleteNode(value);

                break;

            case 5:

                list.countList();

                break;

            case 6:

                exit(0);

            default:

                cout << "Invalid choice" << endl;

        }

        cout << endl;

    }

    return 0;

}

Output

root@Zonularity:/home/zonularity/dsa# cd "/home/zonularity/dsa/lab\_4/" && g++ task.cpp -o task && "/home/zonularity/dsa/lab\_4/"task

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 1

Enter value: 1

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 1

Enter value: 2

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 2

Enter location: 3

Enter value: 3

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 3

2 -> 1 -> 3 -> NULL

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 4

Enter value to delete: 1

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 3

2 -> 3 -> NULL

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 5

Number of nodes: 2

\*\*== Task Menu ==\*\*

1) Insert at head

2) Insert at location

3) Display

4) Delete

5) Count

6) Exit

Enter your choice: 6

# Conclusion

In this lab, we successfully implemented a singly linked list in C++. We investigated various operations on the list, encompassing insertion, deletion, and traversal. This hands-on experience reinforced our theoretical understanding of linked list structures and behaviors, while also providing valuable insights into dynamic memory allocation and pointer manipulation within the context of C++ data structures.