

**DSA (Data Structure and Algorithms) Lab**

**LAB REPORT # 9**

**Semester**: 3rdSemester

**Section**: C

**Submitted To:**

**Abdullah Shahrose**

**Submitted By:**

**Name**: Abdul Ahad

**Roll No**: 22-CS-071

**Task 1:**

#include <iostream>

using namespace std;

struct Node

{

    int data;

    Node \*prev;

    Node \*next;

};

Node \*head = nullptr;

Node \*tail = nullptr;

void insertAtStart(int data)

{

    Node \*newNode = new Node;

    newNode->data = data;

    newNode->prev = nullptr;

    newNode->next = head;

    if (head != nullptr)

    {

        head->prev = newNode;

    }

    if (tail == nullptr)

        tail = head;

    head = newNode;

}

void printList()

{

    Node \*temp = head;

    if (temp == nullptr)

    {

        cout << "List is empty" << endl;

        return;

    }

    while (temp != nullptr)

    {

        cout << temp->data << endl;

        temp = temp->next;

    }

}

void reversePrintList()

{

    Node \*temp = tail;

    if (temp == nullptr)

    {

        cout << "List is empty" << endl;

        return;

    }

    while (temp != nullptr)

    {

        cout << temp->data << endl;

        temp = temp->prev;

    }

}

int main()

{

    int choice;

    while (true)

    {

        cout << "--------- Main Menu ---------" << endl;

        cout << "1. Insert at the start" << endl;

        cout << "2. Print the List" << endl;

        cout << "3. Reverse Print the List" << endl;

        cout << "4. Quit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter data to insert: ";

            int data;

            cin >> data;

            insertAtStart(data);

            break;

        case 2:

            printList();

            break;

        case 3:

            reversePrintList();

            break;

        case 4:

            exit(0);

        default:

            cout << "Invalid choice" << endl;

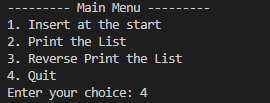
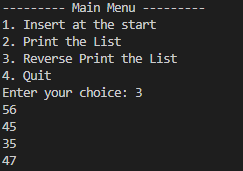
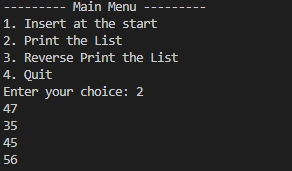
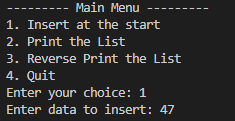
        }

    }

    return 0;

}

**Output:**

****

**Task 2:**

#include <iostream>

struct Node

{

    int data;

    Node \*prev;

    Node \*next;

};

class DoublyLinkedList

{

private:

    Node \*head;

public:

    DoublyLinkedList()

    {

        head = nullptr;

    }

    void insert(int value)

    {

        Node \*newNode = new Node;

        newNode->data = value;

        newNode->prev = nullptr;

        newNode->next = nullptr;

        if (head == nullptr)

        {

            head = newNode;

        }

        else

        {

            Node \*temp = head;

            while (temp->next != nullptr)

            {

                temp = temp->next;

            }

            temp->next = newNode;

            newNode->prev = temp;

        }

    }

    int search(int value)

    {

        Node \*temp = head;

        int index = 0;

        while (temp != nullptr)

        {

            if (temp->data == value)

            {

                return index;

            }

            temp = temp->next;

            index++;

        }

        return -1;

    }

};

int main()

{

    DoublyLinkedList dll;

    dll.insert(10);

    dll.insert(20);

    dll.insert(30);

    dll.insert(40);

    dll.insert(50);

    int element = 30;

    int index = dll.search(element);

    if (index != -1)

    {

        std::cout << "Element " << element << " found at index " << index << std::endl;

    }

    else

    {

        std::cout << "Element " << element << " not found" << std::endl;

    }

    return 0;

}

**Output:**

****

**Task 3:**

#include <iostream>

// Node structure for doubly linked list

struct Node {

    int data;

    Node\* prev;

    Node\* next;

};

// Function to create a new node

Node\* createNode(int data) {

    Node\* newNode = new Node();

    newNode->data = data;

    newNode->prev = nullptr;

    newNode->next = nullptr;

    return newNode;

}

// Function to insert a node at the beginning of the doubly linked list

void insertNode(Node\*\* head, int data) {

    Node\* newNode = createNode(data);

    if (\*head == nullptr) {

        \*head = newNode;

    } else {

        newNode->next = \*head;

        (\*head)->prev = newNode;

        \*head = newNode;

    }

}

// Function to display the doubly linked list

void displayList(Node\* head) {

    Node\* temp = head;

    while (temp != nullptr) {

        std::cout << temp->data << " ";

        temp = temp->next;

    }

    std::cout << std::endl;

}

// Function to reverse the doubly linked list using the swap algorithm

void reverseList(Node\*\* head) {

    Node\* temp = \*head;

    while (temp != nullptr) {

        std::swap(temp->prev, temp->next);

        \*head = temp;

        temp = temp->prev;

    }

}

int main() {

    Node\* head = nullptr;

    // Inserting nodes into the doubly linked list

    insertNode(&head, 1);

    insertNode(&head, 2);

    insertNode(&head, 3);

    insertNode(&head, 4);

    insertNode(&head, 5);

    std::cout << "Original Doubly Linked List: ";

    displayList(head);

    // Reversing the doubly linked list

    reverseList(&head);

    std::cout << "Reversed Doubly Linked List: ";

    displayList(head);

    return 0;

}

**Output:**

