

**Data Structures and Algorithms**

**Lab Assignment 3**

**SUBMITTED BY:**

Hasaan Ahmad SP22-BSE-017

**SUBMITTED TO: Sir Syed Ahmad Qasim**

**Code:**

Code contains all the solved lab activities as well as graded lab tasks. Graded lab tasks are mentioned in comments of code.

#include <iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*next;

    Node \*prev;

    Node(int val)

    {

        this->data = val;

        this->next = NULL;

        this->prev = NULL;

    }

};

// Graded Lab 3

class SinglyLinkedList

{

public:

    Node \*head;

    SinglyLinkedList()

    {

        this->head = nullptr;

    }

    void insert(int val)

    {

        Node \*newNode = new Node(val);

        if (head == nullptr)

        {

            head = newNode;

            return;

        }

        Node \*temp = head;

        while (temp->next != nullptr)

        {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    void display()

    {

        Node \*temp = head;

        while (temp != nullptr)

        {

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

            temp = temp->next;

        }

        cout << endl;

    }

};

class DoublyLinkedList

{

public:

    Node \*head;

    Node \*tail;

    DoublyLinkedList()

    {

        this->head = NULL;

        this->tail = NULL;

    }

    void insertAtEnd(int val)

    {

        Node \*newNode = new Node(val);

        if (head == NULL)

        {

            head = newNode;

            tail = newNode;

            return;

        }

        tail->next = newNode;

        newNode->prev = tail;

        tail = newNode;

    }

    void insertAtHead(int val)

    {

        Node \*newNode = new Node(val);

        if (head == NULL)

        {

            head = newNode;

            tail = newNode;

            return;

        }

        head->prev = newNode;

        newNode->next = head;

        head = newNode;

    }

    void insertAtSpecific(int val, int key)

    {

        Node \*newNode = new Node(val);

        Node \*temp = head;

        while (temp->data != key)

        {

            temp = temp->next;

        }

        newNode->next = temp->next;

        temp->next->prev = newNode;

        temp->next = newNode;

        newNode->prev = temp;

    }

    void deleteFirst()

    {

        Node \*temp = head;

        head = head->next;

        head->prev = NULL;

        delete temp;

    }

    void deleteLast()

    {

        Node \*temp = tail;

        tail = tail->prev;

        tail->next = NULL;

        delete temp;

    }

    void deleteSpecific(int key)

    {

        Node \*temp = head;

        while (temp->data != key)

        {

            temp = temp->next;

        }

        temp->prev->next = temp->next;

        temp->next->prev = temp->prev;

        delete temp;

    }

    void traverseReverse()

    {

        Node \*temp = tail;

        while (temp != NULL)

        {

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

            temp = temp->prev;

        }

        cout << endl;

    }

    void display()

    {

        Node \*temp = head;

        while (temp != NULL)

        {

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

            temp = temp->next;

        }

        cout << endl;

    }

    void completeDelete()

    {

        Node \*temp = head;

        while (temp != NULL)

        {

            Node \*toDelete = temp;

            temp = temp->next;

            delete toDelete;

        }

        head = NULL;

        tail = NULL;

    }

    // Graded Lab 2

    void swapTwoNodes(int key1, int key2){

        Node \*temp1 = head;

        Node \*temp2 = head;

        while(temp1->data != key1){

            temp1 = temp1->next;

        }

        while(temp2->data != key2){

            temp2 = temp2->next;

        }

        int temp = temp1->data;

        temp1->data = temp2->data;

        temp2->data = temp;

    }

    // Graded Lab task

    void reverseAllNodes()

    {

        Node \*temp = head;

        while (temp != NULL)

        {

            Node \*temp2 = temp->next;

            temp->next = temp->prev;

            temp->prev = temp2;

            temp = temp2;

        }

        temp = head;

        head = tail;

        tail = temp;

    }

};

DoublyLinkedList makeDoubly(SinglyLinkedList list)

{

    DoublyLinkedList newList;

    Node \*temp = list.head;

    while (temp != NULL)

    {

        newList.insertAtEnd(temp->data);

        temp = temp->next;

    }

    return newList;

}

int main()

{

    DoublyLinkedList \*list = new DoublyLinkedList();

    list->insertAtEnd(1);

    list->insertAtHead(2);

    list->insertAtHead(3);

    list->insertAtSpecific(4, 2);

    list->display();

    list->reverseAllNodes();

    list->display();

    list->traverseReverse();

    cout<<"TESTING DOUBLY METHODS";

    cout<<endl;

    // Making Singly Linked List to check if it can be converted in doubly

    SinglyLinkedList \*l1 = new SinglyLinkedList();

    l1->insert(1);

    l1->insert(2);

    l1->insert(3);

    l1->insert(4);

    l1->display();

    DoublyLinkedList l2;

    l2 = makeDoubly(\*l1);

    l2.traverseReverse();

    // Testing swapping Nodes

    cout<<"TESTING SWAPPING NODES"<<endl;

    l2.display();

    l2.swapTwoNodes(1,4);

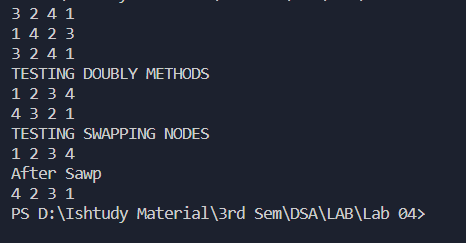
    cout<<"After Sawp" <<endl;

    l2.display();

    return 0;

}

**Output:**

****