***Lab 03***

***DSA***

**Program:**

#include <iostream>

using namespace std;

class Node // Creating a node

{

public:

    int data;

    Node \*next;

};

class linkedList // creating a linked list

{

public:

    Node \*Head, \*Tail; //The Head and Tail of LL

    linkedList()

    {

        Head = NULL;

        Tail = NULL;

    }

    //Task 1

    void insertNodeAtHead(int value)

    {

        Node \*newNode = new Node; //The node we want to insert

        newNode->data = value;

        newNode->next = NULL;

        if (Head == NULL) //Checks if the LL contains a node or not

        {

            Tail = newNode;

            Head = newNode;

        }

        else

        {

            newNode->next = Head;

            Head = newNode;

        }

    }

    //Task 3

    void displayLinkedList()

    {

        Node \*current = Head;

        while (current != NULL) //Traverses till end of LL

        {

            cout << current->data << " ";

            current = current->next;

        }

        cout << endl;

    }

    //Task 5

    void countList()

    {

        Node \*cur = Head;

        int count = 0;

        while (cur != NULL) //Traverses till end of LL

        {

            count++;

            cur = cur->next;

        }

        cout << "The number of nodes in the LL are: " << count;

    }

    //Task 2

    void insertAtAnyPostion(int position, int value)

    {

        Node \*prev = new Node; // Declared for traversal purposes

        Node \*current = new Node;

        current = Head;

        Node \*newNode = new Node; // The new node

        newNode->data = value;

        newNode->next = NULL;

        if (position < 1) //Checks for invalid position

        {

            cout << "Cant place here";

        }

        else if (position == 1)

        {

            newNode->next = Head;

            Head = newNode;

        }

        else

        {

            for (int i = 1; i < position; i++) //Traverses through the LL till posiiton

            {

                prev = current;

                current = current->next;

                if (current == NULL)

                {

                    cout << "Invalid Position";

                    return;

                }

            }

        }

        prev->next = newNode;//inserts the new node between prev and current node

        newNode->next = current;

    }

    //Task 4

    void deleteNode(int value)

    {

        bool flag = false;

        Node \*current = new Node;

        Node \*previous = new Node;

        previous = NULL, current = Head;

        while (current != NULL) //Traverses through the List

        {

            if (current->data == value && current == Head) //Checks if the node is at head

            {

                Head = current->next;

                free(current);

                flag = true;

                break;

            }

            else if (current->data == value) //if the node is in between the LL

            {

                previous->next = current->next;

                if (current == Tail)

                {

                    Tail = previous;

                }

                free(current);

                flag = true;

                break;

            }

            else //Moves to the next respective nodes

            {

                previous = current;

                current = current->next;

            }

        }

        if (flag == true)

            cout << "Element Deleted\n";

        else

            cout << "Element not found\n";

    }

}; // linked list ends here

int main()

{

    linkedList lst;

    lst.insertNodeAtHead(50);

    lst.insertNodeAtHead(90);

    lst.insertAtAnyPostion(2,100);

    cout<<"The LL obtained when we insert elements at head and at position 3:";

    lst.displayLinkedList();

    cout << endl;

    lst.deleteNode(50);

    lst.displayLinkedList();

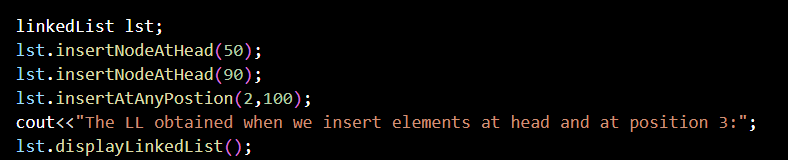
    cout << endl;

    lst.countList();

    cout<<"\nThe resulting Linked List is:\n";

    lst.displayLinkedList();

}

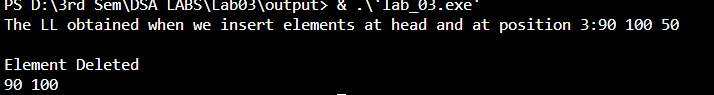
**Output:***Task 1,2,3:  
*

**

*Task 4:*

*A black screen with white text

Description automatically generated*

**

*Task 5:*

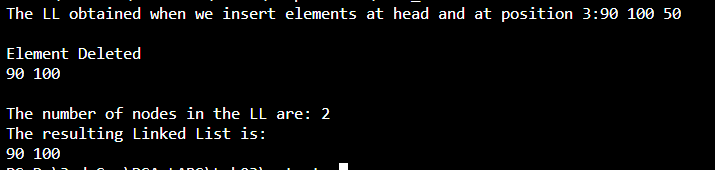
*A screen shot of a computer code

Description automatically generated*

*A black screen with white text

Description automatically generated*

*Task 6:*

**