­­



Submitted to

:

Sir Mohsin

***BS***

***-***

***Software Engineering***

***3***

***rd***

***-***

***E***

Title: Lab Report 2

DSA

Hamza Mehmood

Roll# SP-21-110

NUML

-

S21

-

2352

9



**National University of Modern Languages**

DATA STRUCTURES & ALGORITHMS

|  |  |
| --- | --- |
| **Subject:** Data Structures and Algorithms | **Instructor:** Mohsin Abbas |
| **Lab Report:** 2  **Class:** BSSE III (Evening) | **Due Date:** 17-05-2022 |
| **Student Roll No: SP21-110** | **Student Name: Hamza Mehmood** |
| **Total Marks:** 10 | **Obtained Marks:** |

**Note: This assignment should be submitted in both soft copy and hard copy. Soft copy should be submitted in word document (.docx file) on Google Classroom. Hard copy should be submitted in coming lab class (no need to hard bind or something, just staple document or can place it in strip file). It is an individual assignment. The report (softcopy & hardcopy) must contain this page as the first page.**

**Assigned Task:**

1. Create a class “Node” with proper data members (as discussed in class), save it as header file. Write complete code for Linked List class that uses the header file of Node class. Linked List class contains following functions:
   1. addToHead(double element)
   2. addToTail(double element)
   3. addAfter(double existing, double element)
   4. addBefore(double existing, double element)
   5. deleteFromHead()
   6. deleteFromTail()
   7. deleteSpecificNode(double existing)
   8. traverselist()
2. Write a menu driven program that implements Linked List data structure using class based implementation. Provide menu for user to select operations mentioned above. Take values from user where necessary.
3. Attach screenshot(s) of proper result and working of program.

“DNODE” Class

class Node

{

  public:       //These data members are editable by the user

    double data;   //Declare the data part of node where value is stored

    Node\*next; //Declare the next part of node where address of next node is stored

    Node(double d=0, Node\*n=0) //Initially data and address are both equal to 0

    {

      data=d;

      next=n;

    }

};

“ Linked List” Class

#include <iostream>

#include "NodeClass.h"//User Defined Library

using namespace std;

class linkedlist

{

private:

    Node \*head; //Head declare the starting point of the list

    Node \*tail; //Tail declare the ending point of the list

public:

    linkedlist() //This is a constructor

    {

        head = 0;

        tail = 0;

    }

    void addToHead(double element);

    void addToTail(double element);

    void addAfter(double existing, double element);

    void addBefore(double existing, double element);

    void deletefromHead();

    void deletefromTail();

    void deleteSpecific(double element); //this function is used to delete specific value from the list

    void traverselist();//Display Function

};

void linkedlist::addToHead(double element) //a function for adding value at the start of list

{

    Node \*newNode = new Node(element); //creates a new node to enter data in the list

    if (head == 0 && tail == 0)          //condition shows list is empty

    {

        head = tail = newNode; //new Node is added at the head

    }

    else

    {

        newNode->next = head; //shift the address of head to new node's next part

        head = newNode;       //make head as a new Node

    }

}

void linkedlist::addToTail(double element) //a function for adding value at the end of the list

{

    Node \*newNode = new Node(element);

    if (head == 0 && tail == 0)

    {

        tail = newNode;

    }

    else

    {

        tail->next = newNode; //add null in the next part of null to make new node as the tail

        tail = newNode;

    }

}

void linkedlist::addAfter(double existing, double element) //this function add value after an existing value and the existing is provided by the user

{

    if (tail->data == existing) //Check if value of tail equals to existing (call the function of addToTail) so that we don't need to search the whole list for existing which saves our time

    {

        addToTail(element);

    }

    else

    {

        Node \*current = head;                            //Make head as the current node and start searching the existing value which is provided by the user

        while (current != 0 && current->data != existing) //this condition is true till the user do not find an existing value or the current is not equal to 0

        {

            current = current->next; //Till the above condition is true so the reference of current is shifting forward in the list

        }

        if (current == 0) //means existing not present in our list

        {

            cout << "Existing not present in the list";

        }

        else

        {

            Node \*newNode = new Node(element);

            newNode->next = current->next; //when the existing value is found store the address  of current or present node in the new node's next part

            current->next = newNode;

        }

    }

}

void linkedlist::addBefore(double existing, double element) //this function add value before an existing value and the existing is provided by the user

{

    if (head->data == existing)

    {

        addToHead(element);

    }

    else

    {

        Node \*current = head;

        while (current != 0 && current->data != existing)

        {

            current = current->next;

        }

        if (current == 0)

        {

            cout << "Existing not present in the list";

        }

        else

        {

            Node \*newNode = new Node(element);

            newNode->next = current->next;

            current->next = newNode;

        }

    }

}

void linkedlist::deletefromHead() //Delete from head delete data from the head ( from the start of the list)

{

    if (head == 0 && tail == 0)

    {

        cout << "List is empty";

    }

    else

    {

        Node \*current = head;

        head = current->next; //this shift's the address of head to the next node from the head

        delete current;       //delete the current node which is the head of the list

    }

}

void linkedlist::deletefromTail() //Delete from tail delete data from the tail ( from the end of the list)

{

    if (head == 0 && tail == 0)

    {

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

    }

    else

    {

        Node \*current = tail;

        current->next = tail; //store null in next part of current this makes the current as the tail of list

        delete current;

    }

}

void linkedlist::traverselist() //this function is used to display values from the list

{

    if (head == 0 && tail == 0)

    {

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

    }

    else

    {int num=0;

        Node \*currentNode = head; //start from the head

        while (currentNode != 0)

        {

            cout << currentNode->data << "\t";

            currentNode = currentNode->next; //shift the reference of current node to next node of current and display values from head to tail

            num++; //increments number of nodes until nodes does not points to null

        }

    }

}

int main()

{

    cout << "\*\*Linked List\*\*" << endl;

    linkedlist list;

    int choice;

    do

    {

        cout << "\n\t.....Menu......" << endl;

        cout << "\nPress 1: Add to Head" << endl;

        cout << "Press 2: Add to Tail" << endl;

        cout << "Press 3: Add After" << endl;

        cout << "Press 4: Add Before" << endl;

        cout << "Press 5: Delete from Head" << endl;

        cout << "Press 6: Delete from Tail" << endl;

        cout << "Press 7: Traverse the List" << endl;

        cout << "Press 8: Exit" << endl;

        cout << "\nEnter choice of user: ";

        cin >> choice;

        if (choice == 1)

        {

            int Element;

            cout << "Enter element = ";

            cin >> Element;

            list.addToHead(Element);

        }

        else if (choice == 2)

        {

            int element;

            cout << "Enter element = ";

            cin >> element;

            list.addToTail(element);

        }

        else if (choice == 3)

        {

            int Element;

            cout << "Enter element = ";

            cin >> Element;

            int Existing;

            cout << "Enter existing to add value after existing. = " << endl;

            cin >> Existing;

            list.addAfter(Existing, Element);

        }

        else if (choice == 4)

        {

            int Element;

            cout << "Enter element = ";

            cin >> Element;

            int Existing;

            cout << "Enter existing to add value after existing. = " << endl;

            cin >> Existing;

            list.addBefore(Existing, Element);

        }

        else if (choice == 7)

        {

            list.traverselist();

        }

        else if (choice == 5)

        {

            list.deletefromHead();

        }

        else if (choice == 6)

        {

            list.deletefromTail();

        }

        else if (choice == 8)

        {

            cout<<"Thank You for Using The Program"<<endl;

        }

        else if (choice == 9)

        {

            cout << "Invalid choice. Enter again" << endl;

        }

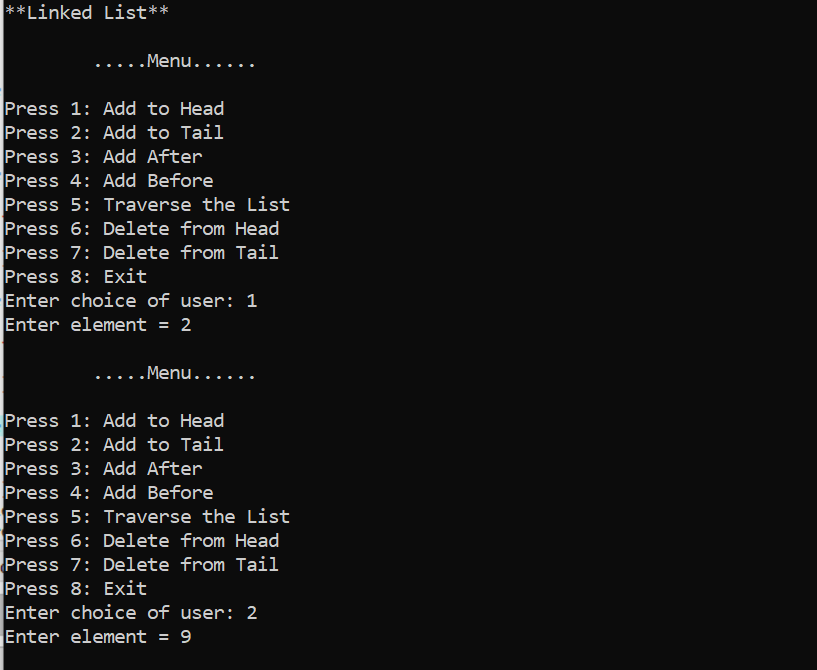
    } while (choice != 8);

    return 0;

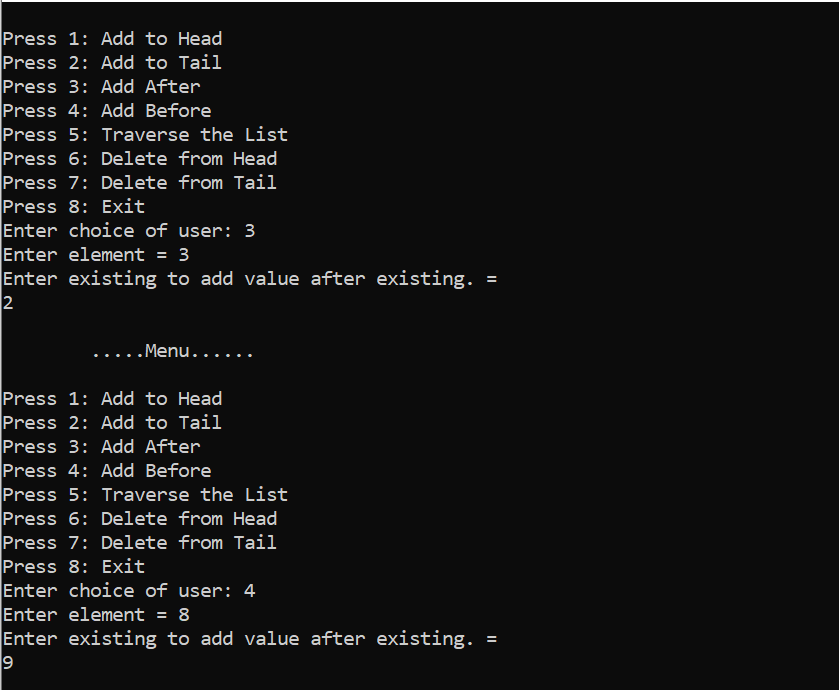
}

Screenshot of Task

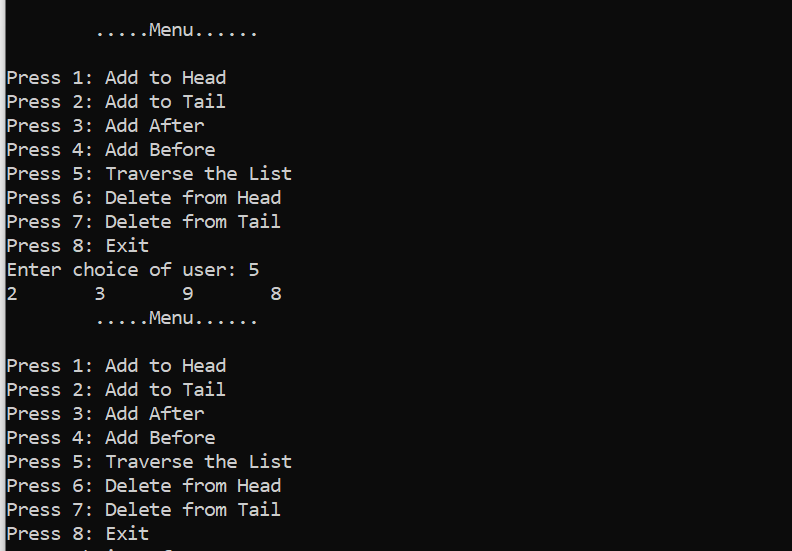
1. **Add to Head & Add to Tail:**



**2.Add After & Before:**



**3.Traverse the List:**



**4.Delete from Head & from Tail:**

