**National University of Modern Languages**

|  |  |
| --- | --- |
| **Subject:** Data Structures and Algorithms | **Instructor:** Mohsin Abbas |
| **Lab Report:** 2  **Class:** BSSE III-B | **Due Date:** 08-04-2022 |
| **Student Roll No:** SP21057 | **Student Name:** MEHRAB SABIR |
| **Total Marks:** 10 | **Obtained Marks:** |

Department of Software Engineering

**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.

Note: The functions which are “**bold**” are the task given. Rest of the functions are already discussed and implemented in class. All you have to do is to merge all the code in single Linked List class.

**Node Class:**

#include<iostream>

using namespace std;

class LLNode

{

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

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

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

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

{

data=d;

next=n;

}

};

**Linked List Class:**

#include<iostream>

#include "Node.h"

using namespace std;

class LinkedList

{

private:

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

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

public:

LinkedList() //This is a constructor

{

head=0;

tail=0;

}

//~LinkedList();

void addToHead(double element);

void addToTail(double element);

void addAfter (double existing, double element);

void addBefore(double existing, double element);

void addSorted(double element);

void deletefromHead();

void deletefromTail();

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

void FindNode(double element); //the function is used to locate or to find a node in the list

void Traverse();

bool IsEmpty(); //this function is used to check whether the list is empty or not

};

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

{

LLNode\*newNode= new LLNode(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

{

LLNode\*newNode= new LLNode(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

{

LLNode\*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

{

LLNode\*newNode = new LLNode(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

{

LLNode\*current=head;

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

{

current=current->next;

}

if(current==0)

{

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

}

else

{

LLNode\*newNode = new LLNode(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

{

LLNode\*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

{

LLNode\*current=tail;

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

delete current;

}

}

void LinkedList::Traverse() //this function is used to display values from the list

{

if(head==0 && tail==0)

{

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

}

else

{

LLNode\*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

} }

}

**Main Class:**

#include<iostream>

#include "lList.h"

using namespace std;

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: Traverse the List"<<endl;

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

cout<<"Press 7: Delete from Tail"<<endl;

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

cout<<"Enter 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==5)

{

list.Traverse();

}

else if(choice==6)

{

list.deletefromHead();

}

else if(choice==7)

{

list.deletefromTail();

}

else if(choice==8)

{

exit(0);

}

else if (choice==9)

{

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

}}

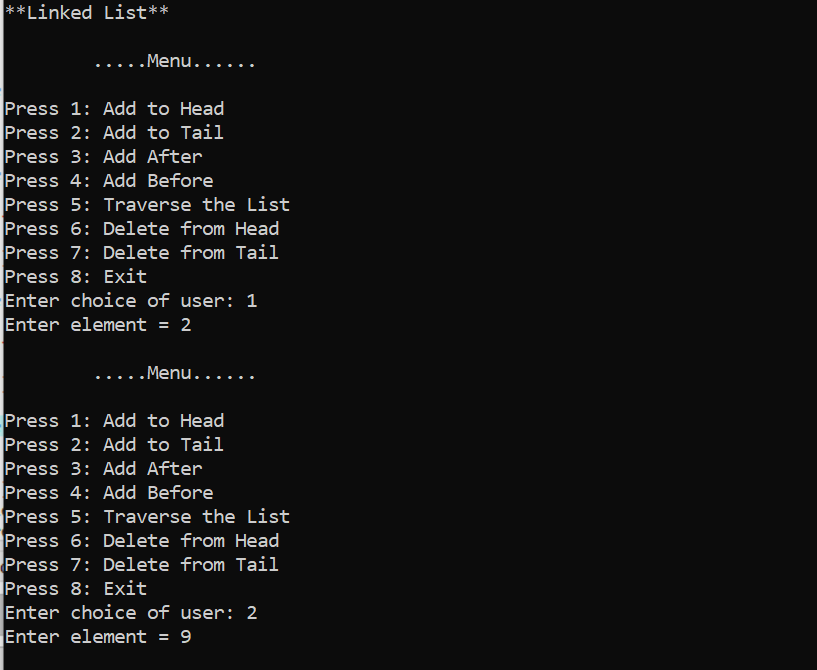
while(choice=8);

return 0;

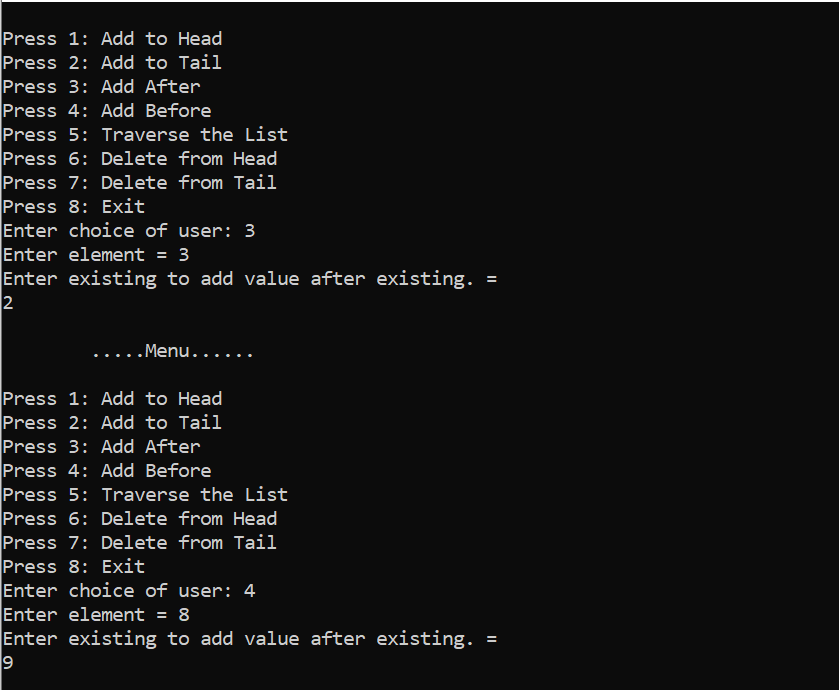
}

**OUTPUT:**

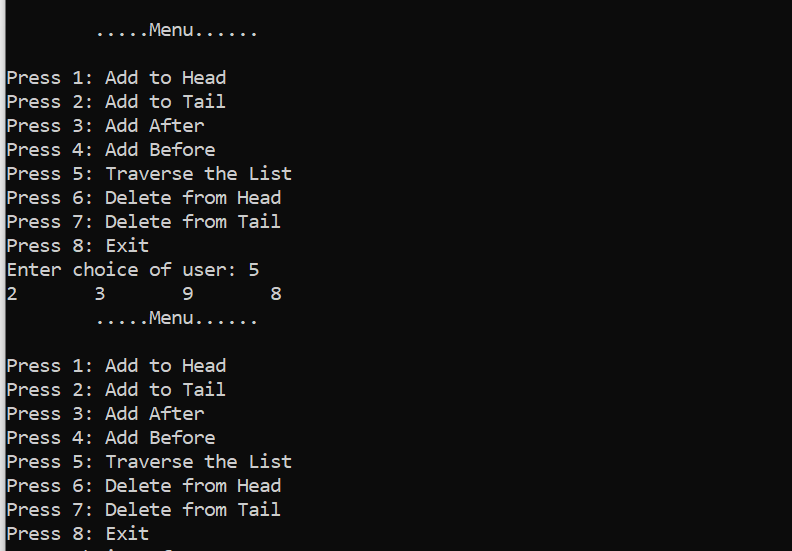
1. **Add To Head & Add To Tail:**



1. **Add After & Before:**



1. **Traverse the List:**



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

