­­



Submitted to

:

Sir Mohsin

***BS***

***-***

***Software Engineering***

***3***

***rd***

***-***

***E***

Title: Lab Report 3

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:** 3  **Class:** BSSE III (Evening) | **Due Date:** 19-05-2022 |
| **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 “DNode” with proper data members (as discussed in class), save it as header file. Write complete code for Doubly 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. forwardtraverse()
   9. reversetraverse()
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 DNode

{

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

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

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

        DNode\*prev; //Declare the prev part of node where address of Previous node is stored

        DNode(int d=0, DNode\*n=0,DNode\*p=0) //Initially data and address are both equal to 0

        {

            value=d;

            next=n;

            prev=p;

        }

};

“Doubly Linked List” Class

#include <iostream>

#include "DNodeClass.h"

using namespace std;

class Dlinkedlist

{

private:

    DNode \*head;

    DNode \*tail;

public:

    Dlinkedlist()

    {

        head = 0;

        tail = 0;

    }

    void addToHead(int element)

{

     DNode\* newItem =new DNode();

    if(head==NULL)

    {

        head=newItem;

        newItem->prev=NULL;

        newItem->value=element;

        newItem->next=NULL;

        tail=newItem;

    }

    else

    {

        newItem->next=head;

        newItem->value=element;

        newItem->prev=NULL;

        head->prev=newItem;

        head=newItem;

    }

}

void addToTail(int element)

{

     DNode\* newItem=new DNode(element);

    newItem->value=element;

    if(head==NULL)

    {

        head=newItem;

        newItem->prev=NULL;

        newItem->next=NULL;

        tail=newItem;

    }

    else

    {

        newItem->prev=tail;

        tail->next=newItem;

        newItem->next=NULL;

        tail=newItem;

    }

}

void addAfter(int existing, int element)

{

    DNode\* newItem =new DNode();

     DNode\* temp=head;

    if(head==NULL)

    {

        cout<<"could not insert"<<endl;

        return;

    }

    if(head==tail)

    {

        if(head->value!=existing)

        {

            cout<<"could not insert"<<endl;

            return;

        }

        newItem->value=element;

        head->next=newItem;

        newItem->next=NULL;

        head->prev=NULL;

        newItem->prev=head;

        tail=newItem;

        return;

    }

    if(tail->value==element)

    {

        newItem->next=NULL;

        newItem->prev=tail;

        tail->next=newItem;

        tail=newItem;

        return;

    }

    while(temp->value!=existing)

    {

        temp=temp->next;

        if(temp==NULL)

        {

            cout<<"Could not insert"<<endl;

            cout<<"element not found"<<endl;

            return;

        }

    }

    newItem->next=temp->next;

    newItem->prev=temp;

    newItem->value=element;

    temp->next->prev=newItem;

    temp->next=newItem;

}

 void addBefore(double existing, double newData)

    {

        DNode \*temp = head;

        while (temp != NULL)

        {

            if (existing == temp->value)

            {

                DNode \*prev = temp->prev;

                DNode \*newNode = new DNode();

                newNode->value=newData;

                prev->next=newNode;

                newNode->prev=prev;

                newNode->next=temp;

                temp->prev=newNode;

            }

            temp = temp->next;

        }

    }

void deleteFromHead()

{

    if(head==NULL)

    {

        return;

    }

    if(head==tail)///one element in the list

    {

         DNode\* currNode;

        currNode=head;

        head=NULL;

        tail=NULL;

        delete currNode;

        return;

    }

    else

    {

         DNode\* currNode=head;

         head=head->next;

         head->prev=NULL;

        delete currNode;

    }

}

void deleteFromTail()

{

    if(head==NULL)

    return;

    if(head==tail)

    {

        struct DNode\* currNode;

        currNode=head;

        head=NULL;

        tail=NULL;

        delete currNode;

        return;

    }

    else

    {

         DNode\* currNode=tail;

        currNode=tail;

        tail=tail->prev;

        tail->next=NULL;

        delete currNode;

    }

}

void deleteSpecificItem(int element)

{

     DNode\* temp=head;

    if(head==tail)

    {

        if(head->value!=element)

        {

            cout<<"could not delete"<<endl;

            return;

        }

        head=NULL;

        tail=NULL;

        delete temp;

        return;

    }

    if(head->value==element)

    {

        head=head->next;

        head->prev=NULL;

        delete temp;

        return;

    }

    else if(tail->value==element)

    {

        temp=tail;

        tail=tail->prev;

        tail->next=NULL;

        delete temp;

        return;

    }

    while(temp->value!=element)

    {

        temp=temp->next;

        if(temp==NULL)

        {

            cout<<"element not found"<<endl;

            return;

        }

    }

    temp->next->prev=temp->prev;

    temp->prev->next=temp->next;

    delete temp;

}

void forwardtraverse()

{

     DNode\* temp=head;

    while(temp!=NULL)

    {

        cout<<"\t"<<temp->value;

        temp=temp->next;

    }

}

void reversetraverse()

{

    DNode\* temp=tail;

    while(temp!=NULL)

    {

        cout<<temp->value<<"\t";

        temp=temp->prev;

    }

}

    };

int main()

{

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

    Dlinkedlist 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: Delete Specific" << endl;

        cout << "Press 8: Forward Traverse the List" << endl;

        cout << "Press 9: Reverse Traverse the List" << endl;

        cout << "Press 0: Exit" << endl;

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

        cin >> choice;

        if(choice==1)

        {

            int element;

            cout<<"Enter element\_";

            cin>>element;

            list.addToHead(element);

            list.forwardtraverse();

        }

        else if(choice==2)

        {

            int element;

            cout<<"Enter element\_";

            cin>>element;

           list.addToTail(element);

            list.forwardtraverse();

        }

        else if(choice==3)

        {

            int existing,newitem;

            cout<<"Enter Old Item\_";

            cin>>existing;

            cout<<"Enter new Item\_";

            cin>>newitem;

            list.addAfter(existing,newitem);

            list.forwardtraverse();

        }

        else if(choice==4)

        {

            int existing,newitem;

            cout<<"Enter Old Item\_";

            cin>>existing;

            cout<<"Enter new Item\_";

            cin>>newitem;

            list.addBefore(existing,newitem);

            list.forwardtraverse();

        }

        else if(choice==5)

        {

            list.deleteFromHead();

            list.forwardtraverse();

        }

        else if(choice==6)

        {

            list.deleteFromTail();

            list.forwardtraverse();

        }

        else if(choice==7)

        {

            int element;

            cin>>element;

            list.deleteSpecificItem(element);

            list.forwardtraverse();

        }

        else if(choice==8)

        {

            list.forwardtraverse();

        }

        else if(choice==9)

        {

            list.reversetraverse();

        }

        else if(choice==0)

        {

            break;

        }

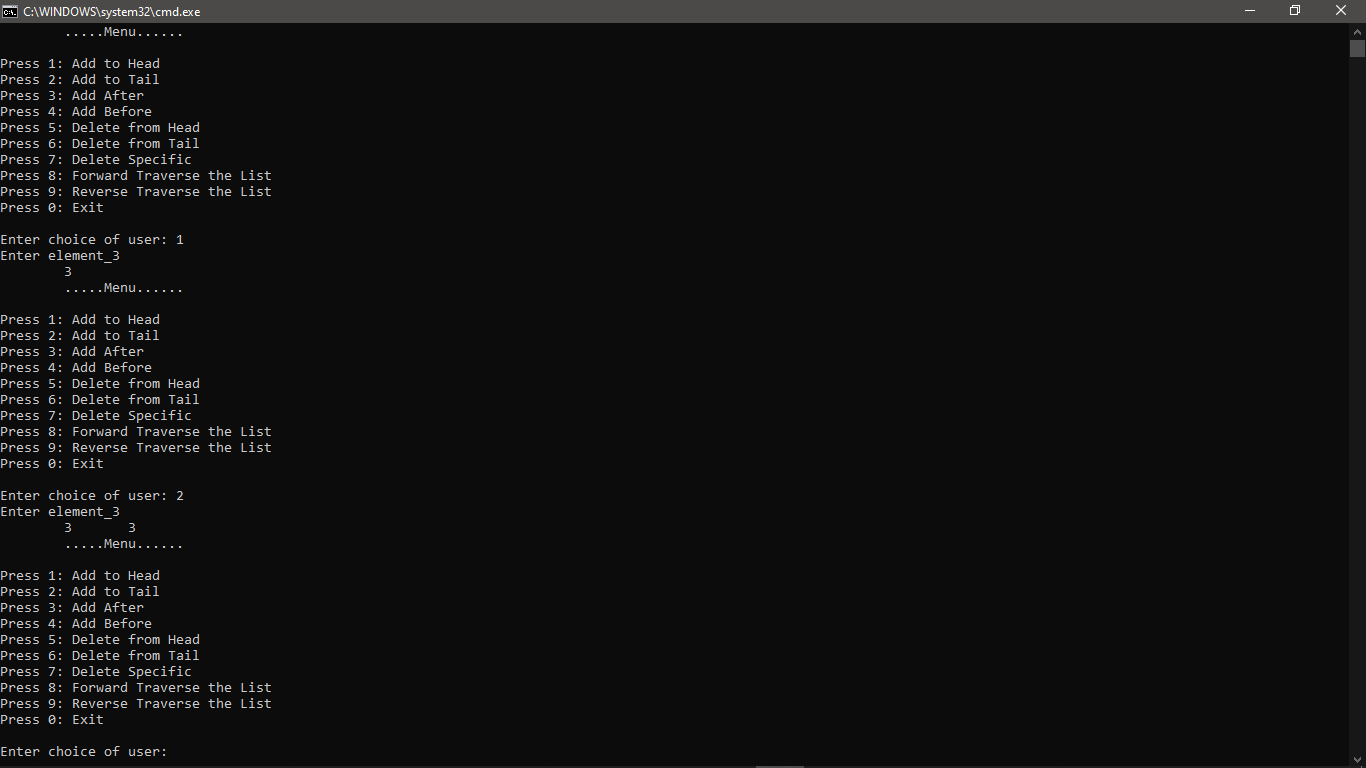
    }while (choice != 0);

return 0;

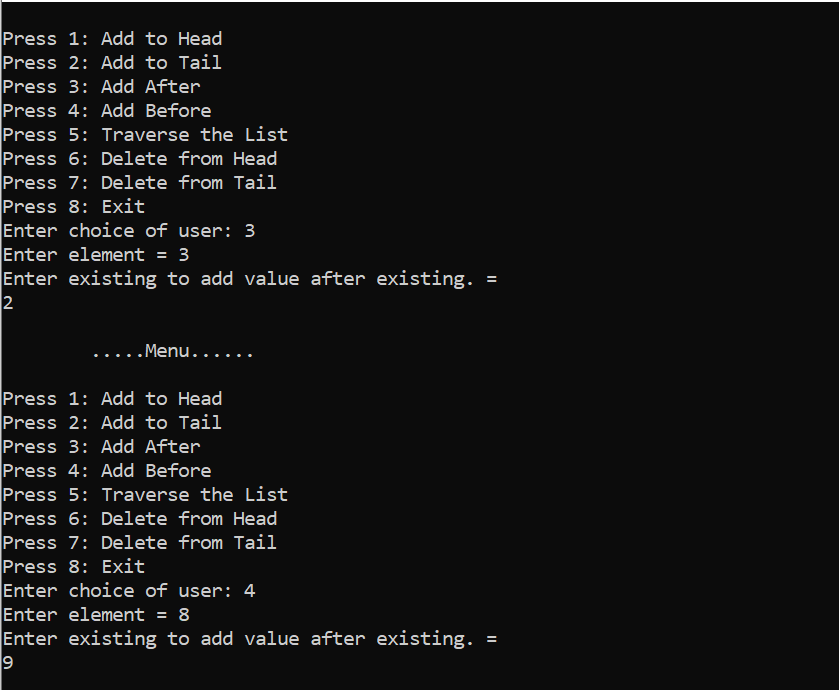
}

Screenshot of Task

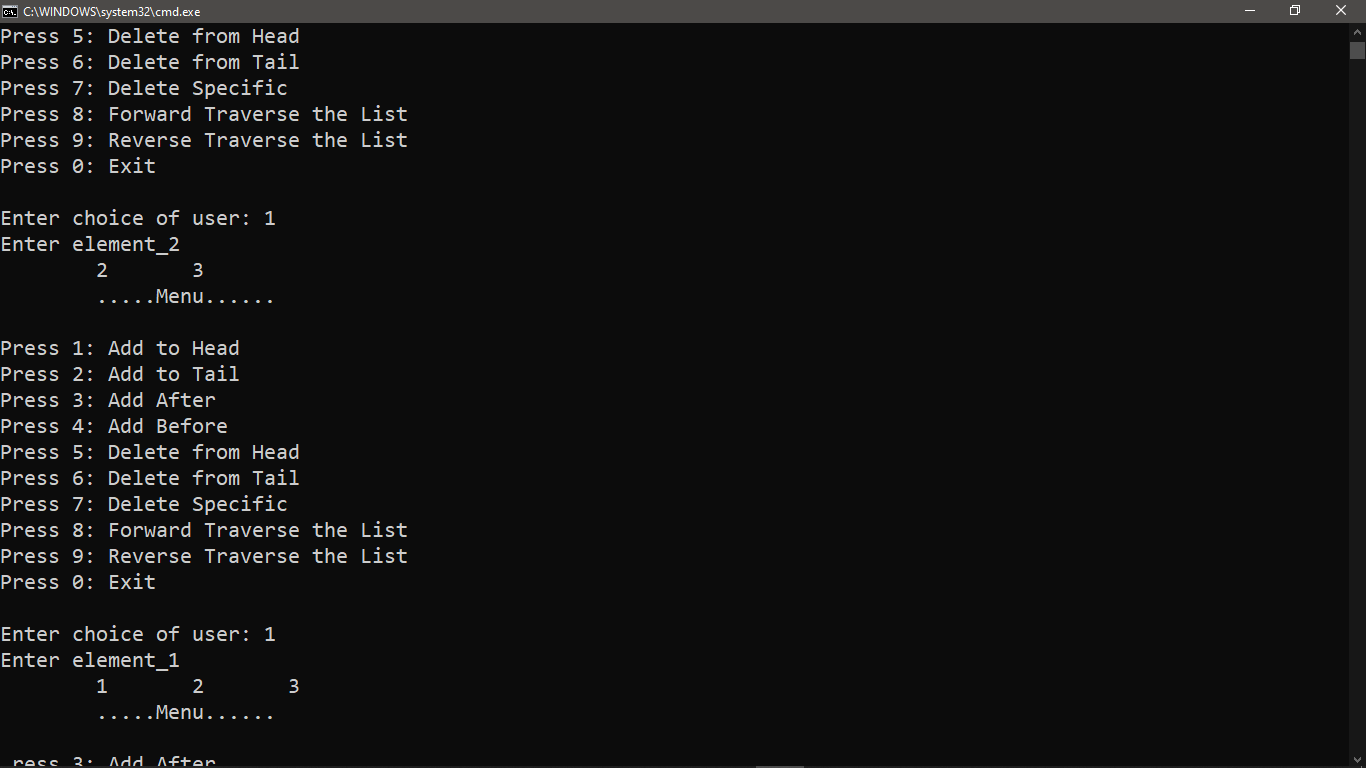
1. **Add to Head & Add to Tail:**

****

**2.Add After & Before:**



**3.Traverse the List:**



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

