NATIONAL UNIVERSITY OF MODERN LANGUAGES

NUML

11/7/2021

Department Of Computer Science

Subject: Data Structure and Algorithm	**Instructor:** Ma'am Zainab Malik
Lab No: 4	**Date:** 7-11-2021
Class: BSCS-3B	

Students' Name

1)	Madina Javed Iqbal Khokhar	2426
2)	Esha Mansoor	2413
3)	Sultan Zahid	2411
4)	Abdul Moeed	2419

NUML (National University Of Modern Languages)

Lab Repot 3

Task:

Implement a program that will illustrate the usage of linkedlist and perform the following functions on it.

- Add to head
- Add to tail
- Add After
- Add before
- Remove from Head
- Remove from Tail
- Searching
- Remove All
- Destructor
- Traversing

Description:

Because of some limitation in array for instance it occupies consecutives slots and has fixed size we use another type of data structure which is linked list. A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In the given program we are illustrating the usage of linked list and perform certain functions to make it more feasible and accurate.

Firstly, we have created a project in which our first class is Node. We should keep this thing in mind that we will use template class in order to make our code more generic. Templates are a way of making classes more abstract by letting us define the behavior of the class without actually knowing what datatype will be handled by the operations of the class. Our next step is towards creating a variable named info and next. As next store the address of next slot so its data type must be node. Then we have use a constructor (Named of a class and constructor name will always be same). Then we will use getter and setter and set our values for info and next. Lastly.we will get our our required values.

In the next step,we have created another class named Linkedlist and will include our Node class in it.In the constructor of Linked list we have initialize head and tail equal to zero. Then we have build our required functions. Our first function is add to head in which we have use if-else statement. If our list 0is empty our if statement will execute which means head and tail are at 0.If there exist a element then else statement will run which will storing our store the value of next head in ptr.Then, we have , use add to tail function again using the same logic but this time our ptr will store the tail value.Then we have use add to head and add to tail function,in the given code each step is illustrating their working

We have also build a function of traversing. Then , we move towards removing a info from head and tail using function. Lastly, we have bulid a destuctor which will remove the entire info until our head becomes zero which means now out list is empty.

In the main function , we will include our Linkedlist class and call our each function according to our requirements.

Code:

Node Class:

```
#include<iostream>

using namespace std;

template<class T> // Templates actually increase flexibility, they're easy

// to update, and they provide consistency across the project

class Node

{

        private:

                T info;  //  variable name use to store information

                Node<T> *next;  // variable use to store address  of next node,that's
why its data type

                                                // is node and "T"  is used for Template



        public:

                Node(T i=0,Node<T> *n=0):info(i),next(n)  // constructor

                {    // constructor { having same name as class}
        }

        void setInfo(T i);  // using setter and getter
        T getInfo();
        void setNext(Node<T> *n); // calling setter and getter
        Node<T>* getNext();
};//EOC

template<class T>
void Node<T>::setInfo(T i)
{
    info=i;    // setting our info
}

template<class T>
void Node<T>::setNext(Node<T> *n)
{
    next=n; // setting our next
}

template<class T>
T Node<T>::getInfo()   // getting our info
{
        return info;
}

template<class T>
Node<T>* Node<T>::getNext()  // getting our next
{
        return next;
}
```

LinkedList Class

```
#include <iostream>
#include "Node.h"   // here we are including our node class
using namespace std;
template<class T> // Templates actually increase flexibility, they're easy
// to update, and they provide consistency across the project

class LinkedList   // class name
{
        private:
                Node<T> *head;   // head is variable whose data type is node
                Node<T> *tail;   // tail is variable whose data type is node
        public:
                LinkedList()     // constructor { having same name as class}
                {
                        head=0;    // tail and head  as initially our slot is empty
                        tail=0;
                }
                ~LinkedList();
        void addToHead(T element);    // here, we are calling each functions to perform a particular task
        void addToTail(T element);
        void addAfter(T existing, T element);
        void addBefore(T existing, T element);
        Node<T>* searching(T element);
        void traversing();
        T removeFromHead();//it will delete first Node<T> and will return of deleted info
        //T removeFromTail();
        void remove(T element);//will deleted Node<T> having provided info
        void removeAll();
};
template<class T>
LinkedList<T>::~LinkedList() // ~ is a destructor sign
{
 removeAll();
}//destructor

template<class T>
void LinkedList<T>::addToHead(T element) // add to head function
     //element=9
{
      /*Node<T> *ptr=new Node<T>();//info=0 and next=0
      ptr->setInfo(element);//info=9
```

```
        ptr->setNext(0);//next=0*/

        Node<T> *ptr=new Node<T>(element);//info=5 & next=0
                                                                    // storig address of newly
constructed node
        if(head==0 && tail==0)//list is empty
        {
                head=ptr; // for the first time , it will work, as head and tail are equal to 0
                tail=ptr;
        }
        else //only one element or >1 element
        {
                ptr->setNext(head);//next of 9 is 23
                                                    // ptr(next) = head
                head=ptr;//head will now poT Node<T> with value 9
        }

}//addToHead

template<class T>
void LinkedList<T>::traversing()
{
        Node<T> *ptr=head;
        while(ptr!=0)
        {
                cout<<ptr->getInfo()<<" ";
                ptr=ptr->getNext();
        }
}//traversing

template<class T>    // add to tail function
void LinkedList<T>::addToTail(T element)
{
        Node<T> *ptr=new Node<T>(element); // storig address of newly constructed node

        if(head==0 && tail==0)//list is empty
        {
                head=ptr;
                tail=ptr;
        }
        else //only one element or >1 element
        {
                tail->setNext(ptr);  //ptr(next) = tail
                tail=ptr; // storing tail in ptr
        }
}//addToTail
template<class T>
Node<T>* LinkedList<T>::searching(T element) // Searching fucnction
{
        Node<T> *loc=0; // setting loc variable equal to zero whose data type is node

        Node<T> *ptr=head; // setting ptr equal to head
```

```
    while(ptr!=0)  // means until ptr reaches at the and of our list
    {
        if(ptr->getInfo()==element)
        {
            loc=ptr;
            return loc;//whenever it will be executed it will return true hexadecimal loc
instead 0
        }
        ptr=ptr->getNext();//we need to take ptr to its next if element does not match
    }
    return loc;//whenever it will be executed it will always return 0
}
template<class T>
void LinkedList<T>::addAfter(T existing, T element)  // add after function
{
    if(head==0)// as list is empty and there is no value where we can add our new info
      // so error will occur
    {
        cerr<<"List is empty therefore, existing cannot exist"<<endl;
    }
    else if(existing==tail->getInfo())//only 1 element & existing is at that single element || if
>1 element and existing gets match with last node
    {                          //the existing element is at the tail
        /*Node<T> *ptr=new Node<T>(element);
        tail->setNext(ptr);
        tail=ptr;*/
        addToTail(element);//it will always execute else part of this function because list is non empty
    }
    else//list is non empty and also existing does not exist at tail
    {
        //it this else is execute it means that existing is somewhere before tail or it does not exist at all
        Node<T> *loc=searching(existing);//it may return 0 (if existing not found) or it may return hexdecimal address if existing found
        if(loc==0)
        {
            cerr<<"Existing not found"<<endl;
        }
        else//existing found somewhere before tail
        {
            Node<T> *ptr=new Node<T>(element);
            ptr->setNext(loc->getNext());
            loc->setNext(ptr); // replacing the value of ptr
        }

    }
}

template<class T>
void LinkedList<T>::addBefore(T existing, T element) // add before function
{
    if(head==0) // as list is empty and there is no value where we can add our new info
```

```
          // so error will occur
        {
                cerr<<"Existing cannot be found"<<endl;
        }
        else if(head->getInfo()==existing)//existing is on head node
        {
                addToHead(element);  // calling add to head function
        }
        else//element can exist after head node
        {
                Node<T> *temp=head; // creating a new node
                while(temp!=tail && temp->getNext()->getInfo()!=existing)//temp->getNext()!=0 && temp->getNext()->getInfo!=existing


                 {
                        temp=temp->getNext();
                }
                if(temp==tail)//not found
                {
                        cerr<<"Existing not found"<<endl;
                }
                else//need to adjust addresses
                {
                        Node<T> *ptr=new Node<T>(element); // replacing our values
                        ptr->setNext(temp->getNext());
                        temp->setNext(ptr);
                }
        }

    }//addBefore

    template<class T>
    T LinkedList<T>::removeFromHead() // fuction use to remove from head
    {
        if(head==0) // as list is empty and there is no value where we cannot delete any info
          // so error will occur
        {
                cerr<<"nothing to delete"<<endl;
        }
        else if(head==tail)   // in case there is only one element exist
        {
                T info=head->getInfo();
                delete head; // deleting our head
                head=0; // after deletig , we will set head and tail equal to 0
                tail=0;
                return info;
        }
        else//more than one element
        {
                Node<T> *temp=head; // storing head in temp
                head=head->getNext();
```

```
                T info=temp->getInfo();
                delete temp;
                return info; // here we will return the value which was deleted
        }
}

template<class T>
void LinkedList<T>::remove(T element) // using remove function
{
        if(head==0) //as list is empty and there is no value which we can delete
          // so error will occur
        {
                cerr<<"Nothing to delete"<<endl;
        }
        else if(head==tail && head->getInfo()==element)
        {
                delete head;
                head=tail=0;
        }
        else if(head->getInfo()==element)
        {
                removeFromHead(); // calling remove from head function
        }
        else if(tail->getInfo()==element)
        {
                //removeFromTail(); // calling remove from tail function
```

```
    }
    else
    {
        Node<T> *temp=head;
        while(temp!=tail && temp->getNext()->getInfo()!=element)
        {
            temp=temp->getNext();
        }
        if(temp==tail)
        {
            cerr<<"Element not found"<<endl;
        }
        else
        {
            Node<T> *ptr=temp->getNext();
            temp->setNext(ptr->getNext());
            delete ptr;
        }
    }
}//remove

template<class T>
void LinkedList<T>::removeAll()  // it works as a destructor
{
    Node<T> *ptr=head;
    while(head!=0)
```

```
	{
		removeFromHead();
	}
}
```

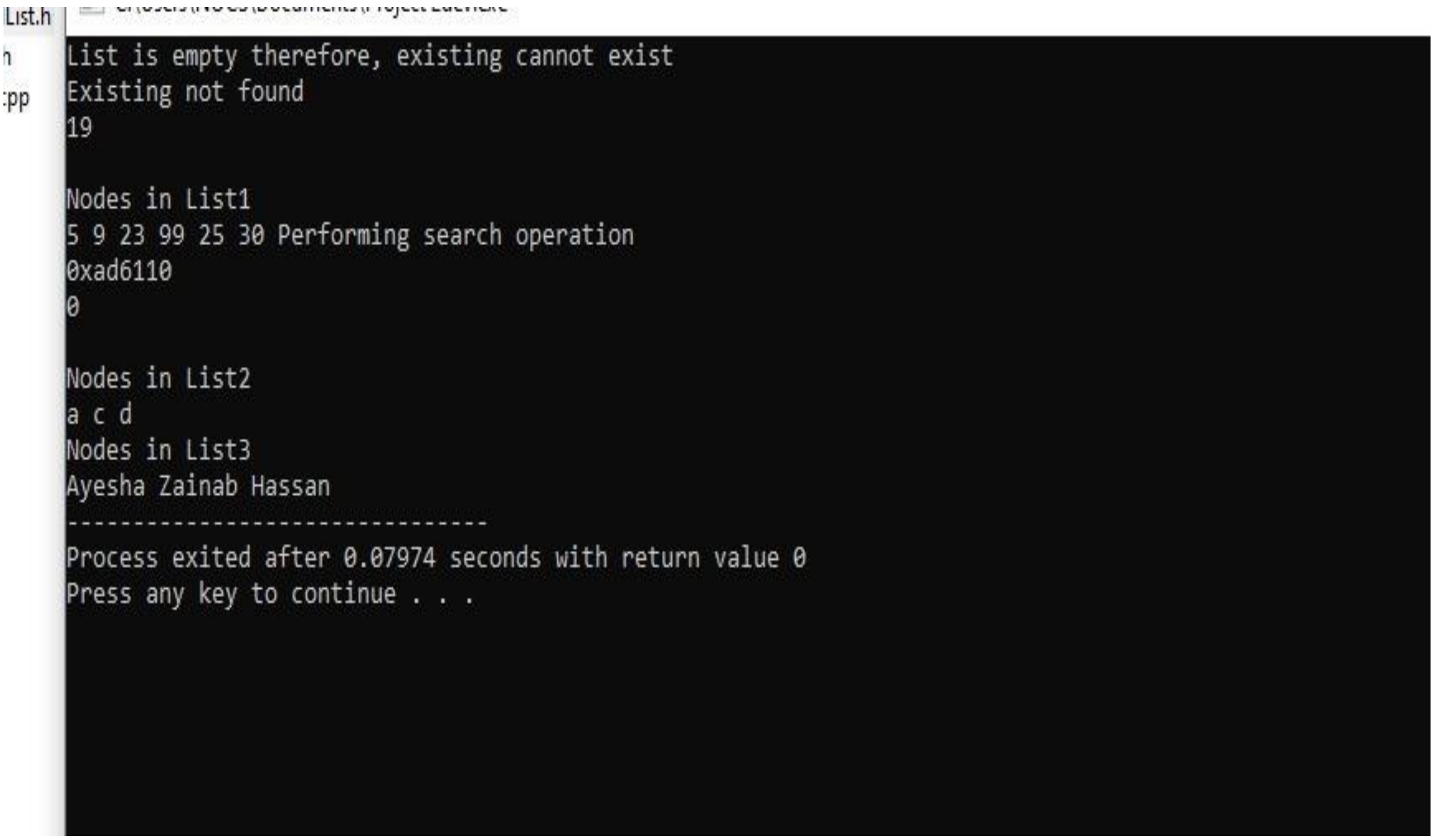
Main class:

```
#include <iostream>
#include <string.h>
#include "LinkedList.h"  // here we  are adding  our LinkedList CLass
using namespace std;
/* run this program using the console pauser or add your own getch, system("pause") or input loop */


int main(int argc, char** argv) {


	LinkedList<int> list1;//it will call constructor and constructor will head=0 and tail=0
	// calling our function one by one
   list1.addAfter(23,99);
	list1.addToHead(23);//before this list is empty, 23 would be the first Node<T> of list1
	list1.addToHead(9);//list is non empty, it already has 23 in it which means that head and tail are not equal to 0
	list1.addToHead(5);
	list1.addToTail(25);
	list1.addAfter(23,99);//it will add 99 after 23 by executing else of else part
	list1.addToHead(19);
```

```
list1.addToTail(30);
list1.addAfter(97,98);//it will giver error by executing if of else part
//19 5 9 23 99 25 30
cout<<list1.removeFromHead()<<endl; // calling remove from  head function
cout<<endl<<"Nodes in List1"<<endl;
list1.traversing();  // calling traversing  function

cout<<"Performing search operation"<<endl;
cout<<list1.searching(9)<<endl; //0x...
cout<<list1.searching(100)<<endl;//0

LinkedList<char> list2;
list2.addToHead('a'); // calling add  to head function
list2.addToTail('c');    // calling add  to tail function
list2.addToTail('d');       // caliing add  to tail function
cout<<endl<<"Nodes in List2"<<endl;
list2.traversing();  // calling traversing  function



LinkedList<string> list3;
list3.addToHead("Zainab"); // calling add  to head function
list3.addToTail("Hassan"); // calling add  to tail function
list3.addBefore("Zainab","Ayesha"); // calling add before function
cout<<endl<<"Nodes in List3"<<endl;
list3.traversing();       // calling traversing  function
```

```
        return 0;
}
```

<u>Output:</u>



```
List is empty therefore, existing cannot exist
Existing not found
19

Nodes in List1
5 9 23 99 25 30 Performing search operation
0xad6110
0

Nodes in List2
a c d
Nodes in List3
Ayesha Zainab Hassan
--------------------------------
Process exited after 0.07974 seconds with return value 0
Press any key to continue . . .
```

THANKS