11/11/2021

Department Of Computer Science

Subject: Data Structure and Algorithm	**Instructor:** Ma'am Zainab Malik
Lab No: 5	**Date:** 11-11-2021
Class: BSCS-3B	

Students' Name

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

Lab Repot 5

Task:

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

- Add to head
- Add to tail
- Add After
- Add before
- Searching
- Traversing
- Traversing back

Description:

A doubly linked list (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list. Doubly Linked List is Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List

In the given program we are illustrating the usage of doublelinked list and perform certain functions to make it more feasible and accurate.

Firstly, we have created a project in which our first class is TWNode. 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, prev and next. As next and prev store the addresses of next and prev slot respectively so there 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,prev and next. Lastly.we will get our our required values.

In the next step,we have created another class named TWLinkedlist and will include our TWNode 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 is empty our if statement will execute which means head and tail are at 0.If there exist a element then else statement will run in which we have set the addresses for instance we will store the address of our head in the ptr of next and in the prev of our head the address of ptr will be stored.Then, we have, use add to tail function again using the same logic but this time our ptr will store the tail value.In this function we will again set our addresses for instance we will store the address of tail will be stored.

We have also created a function of traversing in which we have stored our head in the ptr and use a while that until our ptr becomes zero it wil print the info of ptr and then our ptr will move to the next of the node. Similarly,we have build a function for travering back,add to after, add before and searching .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 TWNode

{

	private:

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

		// is node

		T info;  //  variable name use to store information

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

		// is node
```

```
        public:
                TWNode(T i=0,TWNode<T> *p=0,TWNode<T>
*n=0):info(i),next(n),prev(p)
                {
                        // constructor { having same name as class}
                }

                void setInfo(T i);  // making setter and getter for info
                T getInfo();
                void setNext(TWNode<T> *n); // making setter and getter for NEXT
node
                TWNode<T>* getNext();
                void setPrev(TWNode<T> *p);  // making setter and getter for PREV
node
                TWNode<T>* getPrev();
};//EOC

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

```
template<class T>

void TWNode<T>::setNext(TWNode<T> *n) // setting our Next

{

	next=n;

}


template<class T>

T TWNode<T>::getInfo() // getting  our info

{

	return info;

}


template<class T>

TWNode<T>* TWNode<T>::getNext() // getting  our NEXT

{

	return next;

}



template<class T>

TWNode<T>* TWNode<T>::getPrev() // getting  our PREV

{

	return prev;
}

template<class T>
void TWNode<T>::setPrev(TWNode<T> *p)   // setting our PREV
{
        prev=p;
}
```

LinkedList Class

```
#include <iostream>
#include "TWNode.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 TWLinkedList
{
        private:
                TWNode<T> *head; // head is variable whose data type is node
                TWNode<T> *tail;  // tail is variable whose data type is node
        public:
                TWLinkedList() // constructor { having same name as class}
                {
                        head=0;   // tail and head  as initially our slot is empty
                        tail=0;
                }
                // here, we are calling each functions to perform a particular task
                void traverseBackward();
                void traversing();
                void addToHead(T element);
                void addToTail(T element);
                void addAfter(T existing, T element);
                TWNode<T>* searching(T element);
                void addBefore(T existing, T element);
                /*~LinkedList();





            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>
void TWLinkedList<T>::traverseBackward()
{
        TWNode<T> *ptr=tail;  // storing tail in ptr
        while(ptr!=0)
        {
                cout<<ptr->getInfo()<<" "; // here we will print our info starting from tail and
                // we will wen at head
                ptr=ptr->getPrev();
        }
}//traverseBackward()

template<class T>
void TWLinkedList<T>::traversing()
{
        TWNode<T> *ptr=head; // storing head in ptr
        while(ptr!=0) // here , we will move towards each value one by one untill our head
become
        // zero and will print our values
        {
                cout<<ptr->getInfo()<<" ";
                ptr=ptr->getNext(); // moving our ptr to next
        }
}//traversing

template<class T>
void TWLinkedList<T>::addToHead(T element)//element=9
{
        /*Node<T> *ptr=new Node<T>();//info=0 and next=0
        ptr->setInfo(element);//info=9
        ptr->setNext(0);//next=0*/

        TWNode<T> *ptr=new TWNode<T>(element);//info=5 & next=0
        // CREATIG A NEW NODE

        if(head==0 && tail==0)//list is empty
        {
                head=ptr;  // if there is no element in list then our ptr is head and also tail
                tail=ptr;
        }
        else //only one element or >1 element
        {
                ptr->setNext(head);//next of 9 is 23
                head->setPrev(ptr); // setting our addresses
                head=ptr;//head will now poT Node<T> with value 9
        }

}//addToHead

template<class T>
void TWLinkedList<T>::addToTail(T element)
{
        TWNode<T> *ptr=new TWNode<T>(element);

        if(head==0 && tail==0)//list is empty
        {
                head=ptr;  // if there is no element in list then our ptr is head and also tail
                tail=ptr;
        }
```

```
	else //only one element or >1 element
	{
		tail->setNext(ptr); // setting addresses
		ptr->setPrev(tail);
		tail=ptr; // set ptr as tail
	}
}//addToTail

template<class T>
void TWLinkedList<T>::addAfter(T existing, T element)
{
	if(head==0)//list is empty
	{
		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
	{
		/*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
                    TWNode<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; // it means our element has not found
                    }
                    else//existing found somewhere before tail
                    {
                              TWNode<T> *ptr=new TWNode<T>(element); // creating a new node
                              ptr->setPrev(loc);
                              ptr->setNext(loc->getNext()); // here we will set our addresses accurately
                              loc->setNext(ptr);
                              ptr->getNext()->setPrev(ptr);


                    }


          }
}


template<class T>
TWNode<T>* TWLinkedList<T>::searching(T element)//100
{
          TWNode<T> *loc=0;
```

```
	TWNode<T> *ptr=head; // storing head in a ptr
	while(ptr!=0)  // to run at the at the end of the list
	{
		if(ptr->getInfo()==element) // if element is founded
		{
			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 TWLinkedList<T>::addBefore(T existing, T element)
{
	if(head==0)//empty
	{
		cerr<<"Existing cannot be found"<<endl;
		// if there is no element in list hence there is nothing to found so error will occur
	}
	else if(head->getInfo()==existing)//existing is on head node
	{
		addToHead(element); // if element is at the head
	}
```

```
    else//existing can exist after head node
    {
        TWNode<T> *loc=searching(existing);
        if(loc==0)
        {
            cerr<<"Existing not found"<<endl; // element does not exist
        }
        else
        {
            TWNode<T> *ptr=new TWNode<T>(element); // creating a new node
            loc->getPrev()->setNext(ptr);
            ptr->setPrev(loc->getPrev());  // here we will set our addresses
            loc->setPrev(ptr);
            ptr->setNext(loc);
        }
    }

}//addBefore
```

Main class:

```
#include <iostream>
#include "TWLinkedList.h" // including TWLinkedList.h
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) {
	TWLinkedList<int> list1;
	// calling our function and passing values
	list1.addToHead(25);
	list1.addToHead(20);
	list1.addToHead(10);
	list1.addToTail(30);
	list1.addAfter(10,15);
	list1.addAfter(15,13);
	//10 13 15 20 25 30
	list1.traversing();
	cout<<endl; // it will traverse our enetered elements
	list1.traverseBackward(); // using traverse back function
	return 0;
}
```

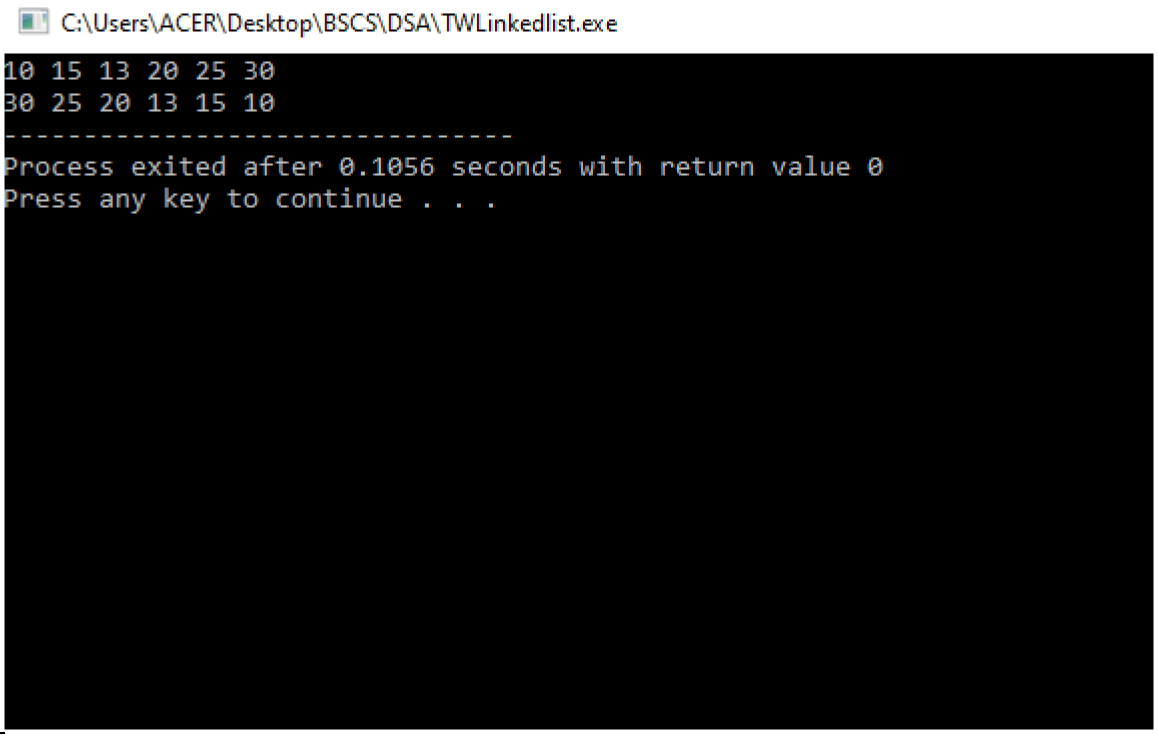
Output:

```
#include <iostream>
#include "TWLinkedList.h" // including TWLinkedList.h
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) {
	TWLinkedList<int> list1;
	// calling our function and passing values
	list1.addToHead(25);
```

```
        list1.addToHead(20);
        list1.addToHead(10);
        list1.addToTail(30);
        list1.addAfter(10,15);
        list1.addAfter(15,13);
        //10 13 15 20 25 30
        list1.traversing();
        cout<<endl; // it will traverse our enetered elements
        list1.traverseBackward(); // using traverse back function
        return 0;
}
```

Output:



C:\Users\ACER\Desktop\BSCS\DSA\TWLinkedlist.exe

```
10 15 13 20 25 30
30 25 20 13 15 10
--------------------------------
Process exited after 0.1056 seconds with return value 0
Press any key to continue . . .
```

THANKS