Name: Saransh Kalra

R Number: R11528524

LAB section: 502

Linkedlist.hpp

#ifndef LinkedList\_hpp

#define LinkedList\_hpp

#include <stdio.h>

//creating the node class for the node of the linked list

class Node

{

public:

//variable info for storing node info

int info;

//pointer of class node, pointing to the next node of the linked list

Node\* next;

};

//class for linked list

class LinkedList

{

public:

//pointers to point of the head and tail of the list

Node \*head, \*tail;

LinkedList(); //constructor to initialize head and tail to 0 or NULL

~LinkedList(); //deconstructor to free the memory borrowed from heap

//member functions to insert a node at the head and the tail of the linked list

void insertAtHead(int value);

void insertAtTail(int value);

//Print the info value before deletion of the node and then deleting the node from the head and tail

//of the linked list

void deleteAtHead();

void deleteAtTail();

//search the given info found at which node of a linked list

void search(int value);

//member function to display the list

void displayList();

};

#endif /\* LinkedList\_hpp \*/

Linkedlist.cpp

#include "LinkedList.hpp"

#include <stdio.h>

#include <iostream>

using namespace std;

//Initialising the head and tail pointers to NULL

LinkedList::LinkedList(){

this->head=this->tail=NULL;

}

//deconstructor to free the memory occupied by linked list

LinkedList::~LinkedList(){

//while the head doesn't reach till node, traverse through the list and free the memory occupied by each node

while(this->head!=NULL)

{

Node \*ptr;

ptr = this->head;

this->head = this->head->next;

delete ptr;

}

}

//member function to insert a node at the head of the linked list

void LinkedList::insertAtHead(int value){

//if head is NULL, means the list is empty, put the info in the first node and make the tail as head too as single node is both the head and tail

if(this->head==NULL){

//initialise a pointer to store the new info to be added

Node \*ptr = new Node();

ptr->info=value;

//the new pointer's next node is the old head

ptr->next=this->head;

//the updated head points to the new pointer now

this->head=ptr;

//making the tail to be the head

this->tail=this->head;

}

//

else{

//same as above just dont make the head the tail this time as this is not the first node

Node \*ptr = new Node();

ptr->info=value;

ptr->next=this->head;

this->head=ptr;

}

}

//member function to insert a node at the tail of the linked list

void LinkedList::insertAtTail(int value){

//if tail is NULL, means the list is empty, put the info in the first node and make the head as tail too as single node is both the head and tail

if(this->tail==NULL){

//initialise a pointer to store the new info to be added

Node \*ptr = new Node();

ptr->info=value;

//the old tail's pointer to next node is the new pointer

this->tail->next=ptr;

//the updated tail points to the new pointer now

this->tail=ptr;

//making the head to be the tail

this->head=this->tail;

}

else{

//same as above just dont make the tail the head this time as this is not the first node

Node \*ptr = new Node();

ptr->info=value;

ptr->next=NULL;

this->tail->next=ptr;

this->tail=ptr;

}

}

//Print the info value before deletion of the node and then deleting the node at the head of the linked list

void LinkedList::deleteAtHead(){

//if the head points to NULL, means the list is empty, therefore error message printed

if(this->head==NULL){

cout<<"there is nothing to be deleted, the list is empty"<<endl;

}

//if the head's next points to NULL, means there is just one record so print deleted info and make both the head and tail to be NULL, making the list to be empty

else if(this->head->next==NULL){

cout<<"The value being deleted is: "<<this->head->info<<endl;

this->head=this->tail=NULL;

}

//else just make the head to be its next and print the old head info before that

else{

cout<<"The value being deleted is: "<<this->head->info<<endl;

this->head=this->head->next;

}

}

//Print the info value before deletion of the node and then deleting the node at the tail of the linked list

void LinkedList::deleteAtTail(){

//if the head points to NULL, means the list is empty, therefore error message printed

if(this->head==NULL){

cout<<"there is nothing to be deleted, the list is empty"<<endl;

}

//if the head's next points to NULL, means there is just one record so print deleted info and make both the head and tail to be NULL, making the list to be empty

else if(this->head->next==NULL){

cout<<"The value being deleted is: "<<this->tail->info<<endl;

this->head=this->tail=NULL;

}

//else just make the tail's next to be NULL, and the old tail to be the previous record of tail

else{

cout<<"The value being deleted is: "<<this->tail->info<<endl;

Node \*ptr = this->head;

//do that by traversing the list till the pointer whose next is NULL, to find the record who's next is tail and then make it the tail of the linked list

while(ptr->next!=this->tail){

ptr=ptr->next;

}

this->tail=ptr;

this->tail->next=NULL;

}

}

//member function to search the given info found at which node of a linked list

void LinkedList::search(int value){

//counter variable to count at which position given info was found

int count=0;

//flag variable to indicate whether value was found or not

int flag=0;

//if head points to NULL, means the list is empty

if(this->head==NULL)

cout<<"The whole list is empty, so the record was not found"<<endl;

//else traverse the whole linked list to find if some node's info matches with the given info

else{

Node \*ptr = this->head;

while(ptr->info!=value && ptr!=this->tail){

ptr=ptr->next;

count++;

if(ptr->info==value){

flag=1;

}

}

//error message if info was not found

if(flag==0){

cout<<"The given value "<<value<<" was not found in the list!"<<endl;

}

else{

cout<<"The value "<<value<<" was found and its the no. "<<count+1<<" record."<<endl;

}

}

}

//member function to display the linked list

void LinkedList::displayList(){

//if head points to NULL, the list is empty

if(this->head==NULL){

cout<<"The whole list is empty, nothing to display."<<endl;

}

//else traverse through the list displaying each node's info followed by an arrow

else{

Node \*ptr;

ptr = this->head;

while(ptr!=NULL){

cout<<ptr->info<<"->";

ptr = ptr->next;

}

cout<<endl;

}

}

LinkedListTest.cpp

#include <iostream>

#include "LinkedList.hpp"

using namespace std;

int main() {

//initialising instance variable of linked list

LinkedList LinkedList1;

//call to member function to delete at tail

LinkedList1.deleteAtTail();

//call to member function to delete at head

LinkedList1.deleteAtHead();

//call to insert 1 at head

LinkedList1.insertAtHead(1);

//display the list function called

LinkedList1.displayList();

//call to insert 2 at tail

LinkedList1.insertAtTail(2);

//display the list function called

LinkedList1.displayList();

//call to insert 3 at tail

LinkedList1.insertAtTail(3);

//display the list function called

LinkedList1.displayList();

//call to insert 4 at tail

LinkedList1.insertAtTail(4);

//display the list function called

LinkedList1.displayList();

//search 2 in the linked list

LinkedList1.search(2);

//search 4 in the linked list

LinkedList1.search(4);

//delete the value at tail

LinkedList1.deleteAtTail();

//search 4 in the linked list

LinkedList1.search(4);

//delete the value at head

LinkedList1.deleteAtHead();

//display the list

LinkedList1.displayList();

return 0;

}

Output

