Rajvaibhav Rahane

17u283 223045

SE-C Comp,Viit,Pune

+917083496708

rajrahane10399@gmail.com

Note-

This is an implementation of DLL in C++ based on the Java Library File LinkedList.java. Feel free to contact me for any queries.

**Code-**

/\*

\*Doubly Linked List Implementation.

\*Implements add,remove operations at all positions

\*Operations that index into the list will traverse the list from beginning or end ,whichever is close to specified index.(n/2 search)

\*Permits classes with an overloaded << extraction operator.

\*Permits dynamically allocated basic datatypes.

\*

\*@author Rajvaibhav D Rahane

\*/

#include<bits/stdc++.h>

#include"my\_exception.cpp"

using namespace std;

//--------------------class LinkedList--------------------------------------------------

template<class E>class LinkedList;

template<class E>ostream & operator<<(ostream &,const LinkedList<E>);

template<class E>

class LinkedList{

public:

LinkedList(){ //default constructor

head=NULL; //to set head to NULL

tail=NULL; //to set tail to NULL

elements=0;

}

friend ostream& operator<< <E>(ostream&,const LinkedList<E> ll); //funtion to print LinkedList

int size(); //return number of elements

void addFirst(E\*); //add element at 1st location

E\* removeFirst(); //remove element at 1st location

void addLast(E\*); //add element at last location

E\* removeLast(); //remove element at last location

void add(int,E\*); //add element at given index

E\* remove(int); //remove element at given index

private:

class Node{ //Single unit of LL-Node //note:change class to template

Node \*prev;

E\* element;

Node\* next;

public:

Node(Node \*prev,E\* element,Node\* next){

this->prev=prev;

this->element=element;

this->next=next;

}

void setElement(E\* element){

this->element=element;

}

E\* getElement(){

return element;

}

Node\* getNextNode(){

return next;

}

void setNextNode(Node \* next){

this->next=next;

}

Node\* getPrevNode(){

return prev;

}

void setPrevNode(Node \* prev){

this->prev=prev;

}

};

Node\* head; //pointer to head of LinkedList

Node\* tail; //pointer to tail of LinkedList

int elements;

bool isPositionIndex(int index);

void checkPositionIndex(int index);

bool isElementIndex(int index);

void checkElementIndex(int index);

Node\* node(int index);

void linkBefore(E\*,Node\*);

E\* unLink(Node\* removedNode);

};

template<class E>

void LinkedList<E>::addFirst(E \*n){ //function to insert data at 1st location

Node \*newNode=new Node(NULL,n,head);

if(head==NULL)

tail=newNode;

else

head->setPrevNode(newNode);

head=newNode; //this statement only if an SLL

/\*E \*newNode=n;

newNode->setNextNode(head);

head=newNode;\*/

elements++;

}

template<class E>

E\* LinkedList<E>::removeFirst(){ //function to remove data at 1st location

if(head!=NULL){

Node \* deletedNode=head;

head=head->getNextNode();

deletedNode->setNextNode(NULL);

E\* removedNode=deletedNode->getElement();

deletedNode->setElement(NULL);

delete deletedNode;

elements--;

if(head==NULL) //for DLL

tail=NULL;

else

head->setPrevNode(NULL);

return removedNode; //check whether to return ptr or variable

}

throw NoSuchElementException(); //else throw NoSuchElementException()

}

template<class E>

void LinkedList<E>::addLast(E \*element){

Node \*newNode=new Node(tail,element,NULL);

if(tail==NULL)

head=newNode;

else

tail->setNextNode(newNode);

tail=newNode;

elements++;

}

template<class E>

E\* LinkedList<E>::removeLast(){

if(tail!=NULL){

Node \* deletedNode=tail;

tail=tail->getPrevNode();

deletedNode->setPrevNode(NULL);

E\* removedNode=deletedNode->getElement();

deletedNode->setElement(NULL);

delete deletedNode;

elements--;

if(tail==NULL)

head=NULL;

else

tail->setNextNode(NULL);

return removedNode;

}

throw NoSuchElementException(); //else throw NoSuchElementException()

}

template<class E>

int LinkedList<E>::size(){ //returns size of linkedList

return elements;

}

template<class E>

ostream& operator<<(ostream & out,const LinkedList<E> ll){ //funtion to print LinkedList

if(ll.head==NULL){

out<<"Empty Linked-List";

}

else{

typename LinkedList<E>::Node\*p=ll.head;

out<<"{";

while(p!=NULL){

out<<"["<<\*(p->getElement())<<"]";

if(p->getNextNode()!=NULL)

out<<",\n";

p=p->getNextNode();

}

out<<"}";

}

}

template<class E>

bool LinkedList<E>::isElementIndex(int index){ //index=element is invalid, index=[0,elements-1]

return index>=0 && index < elements; //fx used when deleting element

}

template<class E>

bool LinkedList<E>::isPositionIndex(int index){ //index=element is valid for position

return index >= 0 && index <= elements; //fx used when inserting element

}

template<class E>

void LinkedList<E>::checkElementIndex(int index){

if(!isElementIndex(index))

throw std::out\_of\_range("Index out of Bounds:"); //note:throw with msg of index and size

}

template<class E>

void LinkedList<E>::checkPositionIndex(int index){

if(!isPositionIndex(index))

throw std::out\_of\_range("Index out of Bounds:"); //note:throw with msg of index and size

}

template<class E>

void LinkedList<E>::add(int index,E\* element){ //Not fully Implemented-for index!=size

checkPositionIndex(index);

if(index==elements)

addLast(element);

else linkBefore(element,node(index)); //add a Node at index

}

template<class E>

typename LinkedList<E>::Node \* LinkedList<E>::node(int index){ //find node at index in elements/2 time

Node \*returnedNode;

if(index<(elements>>1)){

returnedNode=head;

for(int i=0;i<index;i++){ //first half

returnedNode=returnedNode->getNextNode();

}

}else{ //next half

returnedNode=tail;

for(int i=elements-1;i>index;i--){

returnedNode=returnedNode->getPrevNode();

}

}

return returnedNode;

}

template<class E>

void LinkedList<E>::linkBefore(E\* element,Node\* nextNode){ //add a Node before nextNode

Node\* prevNode=nextNode->getPrevNode();

Node \*newNode=new Node(prevNode,element,nextNode);

nextNode->setPrevNode(newNode);

if(prevNode==NULL){

head=newNode;

}else{

prevNode->setNextNode(newNode);

}

elements++;

}

template<class E>

E\* LinkedList<E>::unLink(Node\* removedNode){

E\* returnedElement=removedNode->getElement();

removedNode->setElement(NULL);

Node \*next=removedNode->getNextNode();

Node \*prev=removedNode->getPrevNode();

if(prev==NULL){

head=next;

}else{

prev->setNextNode(next);

removedNode->setPrevNode(NULL);

}

if(next==NULL){

tail=prev;

}else{

next->setPrevNode(prev);

removedNode->setNextNode(NULL);

}

delete removedNode;

elements--;

return returnedElement;

}

template<class E>

E\* LinkedList<E>::remove(int index){

checkElementIndex(index);

return unLink(node(index));

}

//-------------------------------End Of Class----------------------------------------------