Andrew Tran

Lab Experience 7

LINKEDLIST.H - HEADER

/\*--- LinkedList.h --------------------------------------------------------

This header file contains the declarations of LinkedList, a class for

singly-linked lists.

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #5.1 and Projects 5.1 & 5.2

Add a list of the basic operations including brief descriptions.

Add your name here and other info requested by your instructor.

--------------------------------------------------------------------------\*/

#ifndef LINKEDLIST

#define LINKEDLIST

#include <iostream>

#include <new>

using namespace std;

//----- Add typdef statement here

typedef int ElementType;

class LinkedList

{

public:

//------ LinkedList OPERATIONS

// Prototype the class constructor here

/\* --- LinkedList constructor --------------------------------------

Constructs an empty LinkedList object.

Precondition: None.

Postcondition: This list's data members have been initialized

for an empty list.

---------------------------------------------------------------------\*/

LinkedList();

// Prototype and document the size() operation here

int size() const;

// Prototype and document display() here

void display(ostream &) const;

// Prototype insert() here

void insert(ElementType);

/\*----------------------------------------------------------------------

Insert a value into the LinkedList in a sorted order.

Precondition: The parameter contains the data item to insert into an

ordered linked list. The field variable mySize needs

to be modified.

Postcondition: dataValue has been inserted into this LinkedList

object at the position determined by its value.

-----------------------------------------------------------------------\*/

// Prototype erase() here

void erase(ElementType value);

/\*----------------------------------------------------------------------

erase() removes a node containing the value from the LinkedList.

Precondition: A data element from the list

Postcondition: The data value at the position determined by its value

has been removed(depending upon if it is in the list)

from this LinkedList object.

-----------------------------------------------------------------------\*/

// Prototype and document the destructor here

~LinkedList();

// Prototype and document the copy constructor here

LinkedList( const LinkedList &origList);

// Prototype and document the assignment operator here

LinkedList &LinkedList::operator=(const LinkedList &origList);

private:

class Node

{

public:

//------ DATA MEMBERS OF Node

// Define data and next members here

ElementType data;

Node \*next;

//------ Node OPERATIONS

// Prototype the Node constructor here

Node(ElementType dataValue = ElementType(0)) :

data(dataValue), next(0) {}

/\* --- The Node class constructor initializes a Node's data members.

Precondition: None

Receive: dataValue, an ElementType value;

Postcondition: The data and next members have been set to

dataValue and 0, respectively.

-------------------------------------------------------------------\*/

}; //--- end of Node class

typedef Node \*NodePointer;

//------ DATA MEMBERS OF LinkedList

// declare first as a pointer to a Node and declare mySize

Node \*first;

int mySize;

int index = 0;

}; //--- end of LinkedList class

// Put prototype of operator<<() here

ostream &operator<<(ostream &out, const LinkedList &s);

#endif

LINKEDLIST.CPP – IMPLEMENTATION

#include "LinkedList.h"

using namespace std;

//Linked List constructor

//Precons: None

//Postcons: None

LinkedList::LinkedList()

{

first = 0;

mySize = 0;

}

//Size method

//Precons: None

//Postcons: Size of Linked List

int LinkedList::size() const

{

return mySize;

}

//Display function, prints linked lists

//Precons: Ostream

//Postcons: None

void LinkedList::display(ostream &) const

{

LinkedList::NodePointer ptr;

ptr = first;

while (ptr != NULL)

{

cout << ptr->data << endl;

ptr = ptr->next;

}

}

//Overloaded cout operator

//Precons: out, Linked List

//Postcons: out

ostream &operator<<(ostream &out, const LinkedList &s)

{

s.display(out);

return out;

}

//Insert function, inserts new nodes into linked lists

//Precons: dataValue to be inserted

//Postcons: None

void LinkedList::insert(ElementType dataValue)

{

NodePointer nPtr = new(nothrow) Node(dataValue);

if (index == 0)

{

nPtr->next = first;

first = nPtr;

}

else

{

NodePointer predPtr = NULL;

NodePointer currPtr = first;

for (int i = 1; i <= index - 1; ++i)

{

predPtr = currPtr;

currPtr = currPtr->next;

}

predPtr->next = nPtr;

nPtr->next = currPtr;

}

mySize++;

}

//Erase function, deletes nodes from linked lists

//Precons: value to be deleted

//Postcons: None

void LinkedList::erase(ElementType value)

{

NodePointer temp = NULL, prev = NULL;

temp = first;

while (temp != NULL)

{

if (temp->data == value)

{

if (temp == first)

{

first = temp->next;

delete temp;

mySize--;

break;

}

else

{

prev->next = temp->next;

delete temp;

mySize--;

break;

}

}

else

{

prev = temp;

temp = temp->next;

}

}

}

//Destructor method, destroys lists

//Precons: None

//Postcons: None

LinkedList::~LinkedList()

{

NodePointer ptr = first;

while (ptr != 0)

{

first = ptr->next;

delete ptr;

ptr = first;

}

if (first == 0)

{

cout << "List destroyed" << endl;

}

else

{

cout << "List not destroyed" << endl;

}

}

//Copy function, copies entire linked lists

//Precons: A linked list to be copied

//Postcons: None

LinkedList::LinkedList(const LinkedList &origList)

{

mySize = origList.mySize;

if (origList.mySize == 0)

{

first = NULL;

}

else

{

NodePointer origPtr,lastPtr;

origPtr = origList.first;

lastPtr = new Node(origPtr->data);

first = lastPtr;

if(origPtr->next != NULL)

{

origPtr = origPtr->next;

lastPtr->next = new Node(origPtr->data);

lastPtr = lastPtr->next;

}

}

}

//Overloaded assignment operator

//Precons: a linked list

//Postcons: None

LinkedList &LinkedList::operator=(const LinkedList &origList)

{

if (this != &origList)

{

mySize = origList.mySize;

if (origList.mySize == 0)

{

first = NULL;

}

else

{

mySize = origList.mySize;

this->~LinkedList();

NodePointer origPtr, lastPtr;

origPtr = origList.first;

lastPtr = new Node(origPtr->data);

first = lastPtr;

NodePointer temp;

while (origPtr != NULL) //cycle through origList

{

temp = new Node(origPtr->data); //copy data in current node of origList

lastPtr->next = temp; //add temp to end of this list

lastPtr = lastPtr->next; //advance lastPtr to end of this list

origPtr = origPtr->next; //go to next node in origList

}

}

}

return \*this;

}

LINKTEST.CPP – TESTER

/\*--- linktester.cpp --------------------------------------------------

A program for testing class LinkedList.

Written by: Larry R. Nyhoff

Written for: Lab Manual for ADTs, Data Structures, and Problem

Solving with C++, 2E

Lab #5.1 and Projects 5.1

Add your name here and other info requested by your instructor.

---------------------------------------------------------------------\*/

#include <iostream>

using namespace std;

#include "LinkedList.h"

//\*---- PART 6 ---- TEST COPY CONSTRUCTOR

void f(LinkedList aList) // LinkedList value parameter

{ // to test the copy constructor

for (int i = 1; i < 5; i++)

aList.insert(100\*i); // insert into the copy

cout << aList << endl; // output the copy

}

//---- END PART 6 ----\*/

int main()

{

LinkedList intList; // TEST THE CONSTRUCTOR

cout << "Constructing intList\n";

// ---- PART 1 ---- TEST SIZE OPERATION

cout << "Size of intList is " << intList.size() << endl;

/\*---- END PART 1 ----\*/

//\* ---- PART 2A ---- TEST OUTPUT OF EMPTY LIST USING display()

cout << "Empty List (using display): \n";

intList.display(cout);

cout << endl;

//---- END PART 2A ----\*/

//\* ---- PART 2B ---- TEST OUTPUT OF EMPTY LIST USING <<

cout << "Empty List (using <<): \n"

<< intList << endl;

//---- END PART 2B ----\*/

//\* ---- PART 3 ---- TEST INSERT OPERATION

cout << "PART 3" << endl;

cout << "Inserting the values of 0, 10, ..., 80 into the list. \n\n";

for (int i = 0; i < 9; i++)

intList.insert(10\*i); // -- insert 10\*i into the list in order

cout << intList << endl << endl; // Were the items inserted correctly?

//-------------------------------------------------------------------

// Test insert at end of list:

//-------------------------------------------------------------------

cout << "\nInserting a data element, 999, at end of list:\n";

intList.insert(999);

cout << "\nAfter inserting 999 at end of list:\n"

<< intList << endl;

//-------------------------------------------------------------------

// Test insert into the middle of the list:

//-------------------------------------------------------------------

cout << "\nInserting a data element, 25, into the middle of the list:\n";

intList.insert(25);

cout << "\nAfter inserting 25 at end of list:\n"

<< intList << endl;

//---- END PART 3 ----\*/

//\* ---- PART 4 ---- TEST ERASE OPERATION

cout << "PART 4" << endl;

cout << "\nTesting the removal of the last node from the list. The original list:\n"

<< intList << endl;

intList.erase(999);

cout << "\nThe list with the last node removed:\n"

<< intList << endl;

cout << "\nTesting the removal of the first node from the list. The original list:\n"

<< intList << endl;

intList.erase(0);

cout << "\nThe list with the first node removed:\n"

<< intList << endl;

cout << "\nTesting the removal of a node, 25, from the middle of the list. The original list:\n"

<< intList << endl;

intList.erase(25);

cout << "\nThe list with the node containing 25 removed:\n"

<< intList << endl;

cout << "\nTesting the removal of a node that is not contained in the list. The original list:\n"

<< intList << endl;

intList.erase(1107);

cout << "\nThe list after trying to remove a node not in the list:\n"

<< intList << endl;

//---- END PART 4 ----\*/

//\* ---- PART 5 ---- TEST DESTRUCTOR

{

LinkedList anotherList; // this is now a local object to the block

for (int i = 0; i < 5; i++)

anotherList.insert(20 \* i);

cout << "\nHere's another list:\n" << anotherList << endl;

cout << "Now destroying this list\n";

}

cout << "\*\*\* anotherList has lost scope, therefore the destructor should have been called.\n";

cout << "\*\*\* If the destructor was called, anotherList was destroyed \*\*\*\n";

//---- END PART 5 ----\*/

//\* ---- PART 6 ---- TEST COPY CONSTRUCTOR

cout << "\n\nTesting copy constructor" << endl;

f(intList);

cout << "\n\nOriginal list:"; // output the original to make sure

cout << intList << endl; // it hasn't been changed.

//---- END PART 6 ----\*/

// ---- PART 7 ----

LinkedList test;

test = intList;

cout << "assignment test:" << test << endl;

// ---- END PART 7 ----

}







