**Andrew Tran**

**Computer Science 1107**

**Lab Experience Eight**

**Exercise 1:**

Modify the two linked list methods below from the linked list class you created in Lab Seven.

***insert* method modification:**

When inserting a data value into a linked list the insertion point into the list will be determined by the value of the data item. Items inserted into the linked list will be placed dependent upon their value in relation to the other node values currently in the list. When completed the list should be in sorted order, increasing in value from left to right.

The algorithm for insertion into an ordered linked list is listed below:

insert(ListType item)

Construct a new Node pointed to by nPtr. The node should contain the value that is to be inserted into the linked list.

if first is NULL then

Set first to nPtr. ---- This is assuming the Node created in the previous step has been initialized as follows:

The class Node constructor should do this.

|  |  |
| --- | --- |
| item | NULL |

else

Declare another pointer called curPtr and initialize it to first.

Declare another pointer called predPtr and initialize it to NULL

while(curPtr != NULL && curPtr 🡪data < nPtr 🡪data) // order of the test does matter. Why?

Set predPtr to curPtr

Set curPtr to curPtr 🡪next

end while

Set nPtr🡪next to curPtr

if(predPtr == NULL)

Set first to nPtr

else

Set predPtr🡪next to nPtr

// end of insertion

***erase* method modification:**

The erase method will receive as an argument an actual data value to erase from the linked list instead of the location of the item to be erased.

The erase algorithm to implement is listed below:

erase (ListType item)

Declare a pointer called curPtr and initialize it to first

Declare another pointer called predPtr and initialize it to NULL

while(curPtr != NULL && curPtr 🡪data < item) // order of the test does matter. Why?

Set predPtr to curPtr

Set curPtr to curPtr 🡪next

end while

if(curPtr != NULL && curPtr 🡪data == item)

if(predPtr == NULL)

Set first to curPtr 🡪next

else

Set predPtr 🡪next to curPtr 🡪next

delete curPtr

else

Display a message stating the item is not in the list.

**Exercise 2**

Given the following declarations:

class Node{

public:

int data;

Node \*next;

};

Node \*p1, \*p2, \*p3;

Assume the following statements also have been executed:

p1 = new(nothrow) Node;

p2 = new(nothrow) Node;

p3 = new(nothrow) Node;

Tell what will be displayed by each of the code segments or explain why an error occurs.

1. p1->data = 123;

p2->data = 456;

p1->next = p2;

p2->next = NULL;

cout << p1->data << “ “ << p1->next->data << endl;

**123 456**

1. p1->data = 12;

p2->data = 56;

p1 = p2;

p2->next = NULL;

cout << p1->data << “ “ << p2->data << endl;

**56 56**

1. p1->data = 12;

p2->data = 56;

\*p1 = \*p2;

p2->next = NULL;

cout << p1->data << “ “ << p2->data << endl;

**56 56**

1. p1->data = 123;

p2->data = 456;

p1->next = p2;

p2->next = p1;

cout << p2->data << “ “ << p2->next->data << endl;

**456 123**

1. p1->data = 12;

p2->data = 34;

p3->data = 56;

p1->next = p2;

p2->next = p3;

p3->next = NULL;

cout << p1->data << “ “ << p1->next->data << endl;

cout << p2->data << “ “ << p2->next->data << endl;

cout << p1->next->next->data << endl;

cout << p3->data << endl;

**12 34**

**34 56**

**56**

**56**

1. p1->data = 12;

p2->data = 34;

p3->data = 56;

p1->next = p2;

p2->next = p3;

p3->next = p1;

cout << p3->data << “ “ << p3->next->data << endl;

cout << p3->next->next->data << endl;

cout << p3->next->next->next->data << endl;

cout << p3-> next->next->next->next->data << endl;

**56 12**

**34**

**56**

**12**

1. p1->data = 12;

p2->data = 34;

p3->data = 56;

p1->next = p2;

p2->next = p3;

p3->next = NULL;

cout << p1->data << “ “ << p1->next->data << endl;

cout << p2->data << “ “ << p2->next->data << endl;

cout << p2->next->next->data << endl;

cout << p3->data << endl;

**you can’t print** p2->next->next->data **because there’s no node after p3!**

**Exercise 3**

Write an algorithm to reverse a linked list given that the first node in the list is pointed to by the pointer first. Do not copy the list elements, reset the link and the pointers so that first points to the last node and links between the nodes are reversed. NOTE: C++ CODE IS NOT AN ALGORITHM. PSEUDOCODE WOULD BE THE ALGORITHM. SEE THE ERASE ALGORITHM ABOVE FOR AN EXAMPLE.

**Declare pointer next and initialize it to first**

**Declare pointer prev and initialize it to NULL**

**Declare pointer temp and initialize it to NULL**

**While (next != 0)**

**Set temp to next->next**

**Set next->next to prev**

**Set first to next**

**Set prev to next**

**Set next to temp**

**End while**

**Due Date:** As shown on the assignment folder Lab Experience Eight.

**What to hand in:**

1. Print-out of your linked list class containing the two methods described above.
2. A word document containing screen shots of your linked list class.
3. A print out of a word document containing the answers to exercise 2 and 3 above.
4. Compress the word document and all files for your linked list class into a single file called yournamelab8.zip.

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 LINKEDLISTV2

#define LINKEDLISTV2

#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

#include "LinkedListV2.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 (first == NULL)

{

first = nPtr;

}

else

{

NodePointer curPtr = first;

NodePointer predPtr = NULL;

while (curPtr != NULL && curPtr->data < nPtr->data)

{

predPtr = curPtr;

curPtr = curPtr->next;

}

nPtr->next = curPtr;

if (predPtr == NULL)

first = nPtr;

else

predPtr->next = nPtr;

}

mySize++;

}

//Erase function, deletes nodes from linked lists

//Precons: value to be deleted

//Postcons: None

void LinkedList::erase(ElementType value)

{

NodePointer curPtr = first;

NodePointer predPtr = NULL;

while (curPtr != NULL && curPtr->data < value)

{

predPtr = curPtr;

curPtr = curPtr->next;

}

if (curPtr != NULL && curPtr->data == value)

{

if (predPtr == NULL)

first = curPtr->next;

else

predPtr->next = curPtr->next;

delete curPtr;

}

else

{

cout << "Item not found!" << endl;

}

}

//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)

{

temp = new Node(origPtr->data);

lastPtr->next = temp;

lastPtr = lastPtr->next;

origPtr = origPtr->next;

}

}

}

return \*this;

}







