/\*--- 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.

Chris Dang Class: CSCI 1107

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

#ifndef LINKEDLIST

#define LINKEDLIST

#include <iostream>

using namespace std;

typedef int ElementType ;

class LinkedList

{

public:

//------ LinkedList OPERATIONS

// Precondition: None.

// Postcondition: This list's data members have been initialized

// for an empty list.

LinkedList(){first = NULL; mySize = 0;} ;

// Prototype and document the size() operation here

//Function: size

//Precondition: none

//Postcondition: returns size of linked list as an integer

int size() const ;

// Prototype and document display() here

//Function: display

//Precondition: ostream object must be used

//Postcondition: contents of current node are insirted into ostream object

void display(ostream &out) const ;

// Prototype insert() here

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

Insert a value into a LinkedList at a given index.

Precondition: The first parameter, index, is an unsigned value with

0 <= index <= mySize; the second parameter, dataValue, is an

ElementType value. index = 0 denotes insertion at the beginning

of the list, and index = mySize denotes insertion at the end

(after the current last element).

Postcondition: dataValue has been inserted into this LinkedList

object at the position determined by index (provided index

is a legal position).

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

void insert(int index, int dataValue) ;

// Prototype erase() here

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

erase() removes a value from a LinkedList at a given index.

Precondition: The parameter index, an unsigned value, satisfies

0 <= index < mySize.

Postcondition: The data value at the position determined by index

(provided index is a legal position) has been removed from

this LinkedList object.

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

void erase(int index) ;

// 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 operator=(const LinkedList & origList) ;

private:

class Node

{

public:

ElementType data ;

Node \*next ;

//------ Node OPERATIONS

// Prototype the Node constructor here

/\* --- 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.

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

Node(ElementType dataValue ){data = dataValue, next = NULL ;};

}; //--- end of Node class

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

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

Node \*first ;

int mySize ;

typedef Node \*NodePointer;

}; //--- end of LinkedList class

// Put prototype of operator<<() here

//-------------Non-member function -------------

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

#endif

//-LinkList.cpp Implementation File-----------------------------

// Written by: Chris Dang Class: CSCI 1107

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

#include <iostream>

using namespace std;

#include "LinkedList.h"

int LinkedList::size() const {

return mySize ;

}//end size getter

void LinkedList::display(ostream &out) const {

LinkedList::NodePointer ptr = first ;

while (ptr != NULL) {

out << ptr -> data << " " ;

ptr = ptr -> next ;

}// end while

}// end display ;

void LinkedList::insert(int index, int dataValue) {

Node \*nPtr = new(nothrow) Node(dataValue) ;

//error checking: if index to be passed in is out of bounds of list,

//the index isn't used, the linked list remains unchanged,

//and exectution continues

if (index < 0 || index > mySize)

cerr << "Index " << index << " is not a valid index.\n" ;

else {

//if trying to insert new node to first of the list,

//this if will handle the inseration

if (index == 0) {

nPtr -> next = first;

first = nPtr ;

}//end if

//if trying to insert somewhere else in the list

//other than first position, this else will will handle the insertation

else {

Node \*predPtr = first ;

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

predPtr = predPtr -> next ;

nPtr -> next = predPtr -> next ;

predPtr -> next = nPtr ;

}// end else

++mySize ;

}// end else

}//end insert

void LinkedList::erase(int index) {

//error checking: if index to be passed in is out of bounds of list,

//the index isn't used, the linked list remains unchanged,

//and exectution continues

if (index < 0 || index > mySize)

cerr << "Index " << index << " is not a valid index.\n" ;

else {

//if trying to erase node at beginning of list

//this if will handle the erase

if (index == 0) {

Node \*ptr = first ;

first = first -> next ;

delete ptr ;

}// end if

else {

Node \*predPtr = first ;

Node \*ptr = first -> next ;

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

predPtr = ptr ;

ptr = ptr -> next ;

}//end for

predPtr -> next = ptr -> next ;

delete ptr ;

//if deleting the last node from list,

//this will handle setting a new last node by setting

//the next pointer to null

if (index == mySize - 1)

predPtr -> next = NULL ;

}// end else

--mySize ;

}//end else

}// end erase

LinkedList::~LinkedList() {

// creates ptr and points toward first node in list

NodePointer ptr = first -> next ;

// traverses list, reassigns first of the list, and

// deletes nodes in traversal

for (int i = 1; i < mySize; i++) {

first = ptr -> next ;

delete ptr ;

ptr = first ;

} // end for

if (first == 0)

cout << "List destroyed\n" ;

else

cout << "List not destroyed\n" ;

} // end destructor

LinkedList::LinkedList(const LinkedList & origList) {

// copies size of original list to new list

mySize = origList.mySize ;

// if original list is emtpy, first is set to NULL pointer

if (origList.first -> next == NULL)

first = NULL ;

else {

NodePointer origPtr, lastPtr ;

origPtr = origList.first -> next ;

Node \*nPtr = new(nothrow) Node(origList.first -> data) ;

lastPtr = nPtr ;

first = lastPtr ;

// traverses the original list, constructing nodes based of the

// originial list and adding them to the new list

for (int i = 1; i < mySize ; i++) {

Node \*nPtr = new(nothrow) Node(origPtr -> data) ;

lastPtr -> next = nPtr ;

lastPtr = lastPtr -> next ;

origPtr = origPtr -> next ;

} // end for ;

} // end else

} // end copy constructor

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

// checks to see if this is a self assignemnet

// if it is a self assignment, returns itself

if (this != &origList) {

// destroys value of left hand side of assignment

// to avoid memory leak

this -> ~LinkedList() ;

// copies size of original list to new list

mySize = origList.mySize ;

// if original list is emtpy, first is set to NULL pointer

if (origList.first -> next == NULL)

first = NULL ;

// creates a copy of the rhs list to the left hand side

else {

NodePointer origPtr, lastPtr ;

origPtr = origList.first -> next ;

Node \*nPtr = new(nothrow) Node(origList.first -> data) ;

lastPtr = nPtr ;

first = lastPtr ;

// traverses the original list, constructing nodes based of the

// originial list and adding them to the new list

for (int i = 1; i < mySize ; i++) {

Node \*nPtr = new(nothrow) Node(origPtr -> data) ;

lastPtr -> next = nPtr ;

lastPtr = lastPtr -> next ;

origPtr = origPtr -> next ;

} // end for ;

} // end else

}// end if

return \*this ;

} // end overloaded assignment operator

// Non-member functions

ostream &operator<<(ostream &out, const LinkedList &List) {

List.display(out) ;

return out ;

}//end overloaded insertation operator

/\*--- 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

Chris Dang Class: CSCI 1107

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

#include <iostream>

using namespace std;

#include "LinkedList.h"

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

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

{

intList.insert(i/2, 10\*i); // -- insert 10\*i at position i/2

cout << intList << endl ; // test output

}

//-- Test insert at end of list:

intList.insert(intList.size(), 999);

cout << "\ninsert 999 at end of list:\n"

<< intList << endl;

//-- Test for illegal inserts

cout << "Try inserting at positions -1 and 20:\n";

intList.insert(-1, -99);

intList.insert(20, 200);

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

// ---- PART 4 ---- TEST ERASE OPERATION

cout << "\nRemove last node:\n";

intList.erase(intList.size() - 1);

cout << intList << endl;

cout << "\nRemove first node:\n";

intList.erase(0);

cout << intList << endl;

cout << "\nRemove node at position 4:\n";

intList.erase(4);

cout << intList << endl;

//-- Test for illegal deletes

cout << "Try deleting at positions -1 and 20\n";

intList.erase(-1);

intList.erase(20);

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

// ---- PART 5 ---- TEST DESTRUCTOR

{

LinkedList anotherList ;

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

anotherList.insert(i, 20 \* i);

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

cout << "Now destroying this list\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;

cout << LinkedList(intList);

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

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

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

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

//creates linked list with values 11, 22, 33, 44, and 55

LinkedList var , test ;

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

var.insert(i, 11 \* (i + 1)) ;

//creates linked list with values 22, 44, 66, and 88

for (int i = 0; i < 4; i++)

test.insert(i, 22 \* (i + 1)) ;

cout << "\nTest list var contains:\n" << var << endl ;

cout << "\nTesting self assignement. var before:\n"

<< var << endl ;

var = var ;

cout << "\nvar after self assignment:\n"

<< var << endl

<< "\nTesting assignment of var = test\n" ;

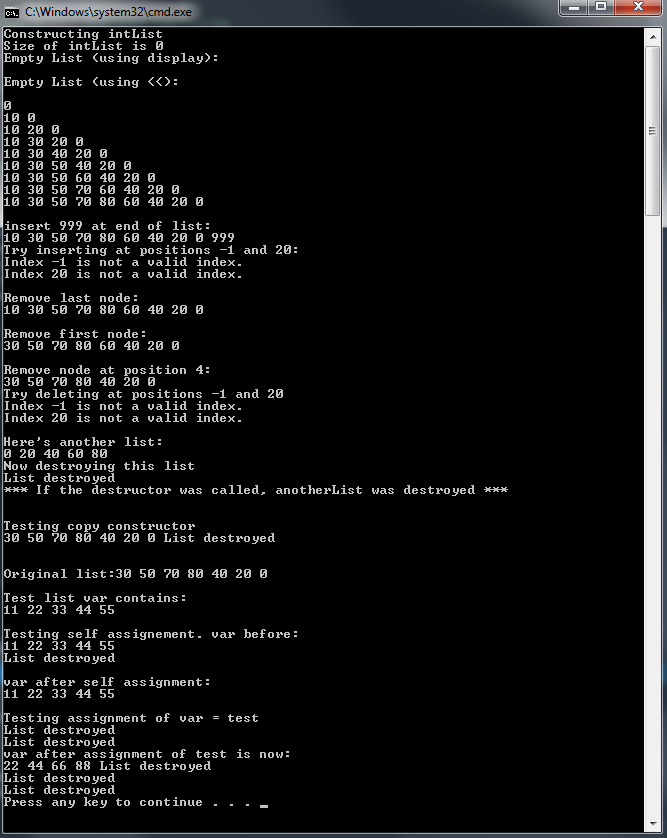
var = test ;

cout << "var after assignment of test is now:\n"

<< var ;

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

}



/\*--- HashLink.h --------------------------------------------------------

Chris Dang Class: CSCI 1107

Header file to be used with HashTable.h, HashTable.cpp, and HashLink.cpp

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

#ifndef LINKEDLIST

#define LINKEDLIST

#include <iostream>

#include <string>

using namespace std;

typedef string ElementType ;

class LinkedList

{

public:

//------ LinkedList OPERATIONS

// Precondition: None.

// Postcondition: This list's data members have been initialized

// for an empty list.

LinkedList(){first = NULL; mySize = 0;} ;

// Prototype and document the size() operation here

//Function: size

//Precondition: none

//Postcondition: returns size of linked list as an integer

int size() const ;

// Prototype and document display() here

//Function: display

//Precondition: ostream object must be used

//Postcondition: contents of current node are insirted into ostream object

void display(ostream &out) const ;

// Prototype insert() here

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

Insert a value into a LinkedList at a given index.

Precondition: The first parameter, index, is an unsigned value with

0 <= index <= mySize; the second parameter, dataValue, is an

ElementType value. index = 0 denotes insertion at the beginning

of the list, and index = mySize denotes insertion at the end

(after the current last element).

Postcondition: dataValue has been inserted into this LinkedList

object at the position determined by index (provided index

is a legal position).

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

void insert(int index, ElementType dataValue) ;

// Prototype erase() here

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

erase() removes a value from a LinkedList at a given index.

Precondition: The parameter index, an unsigned value, satisfies

0 <= index < mySize.

Postcondition: The data value at the position determined by index

(provided index is a legal position) has been removed from

this LinkedList object.

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

void erase(int index) ;

// 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 operator=(const LinkedList & origList) ;

private:

class Node

{

public:

ElementType data ;

Node \*next ;

//------ Node OPERATIONS

// Prototype the Node constructor here

/\* --- 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.

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

Node(ElementType dataValue ){data = dataValue, next = NULL ;};

}; //--- end of Node class

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

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

Node \*first ;

int mySize ;

typedef Node \*NodePointer;

}; //--- end of LinkedList class

// Put prototype of operator<<() here

//-------------Non-member function -------------

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

#endif

/\*--- HashTable.h -------------------------------------------------------------

Written by: Chris Dang

Written for: CSCI 1107

Header file for Hash Table Class

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

#include <iostream>

#include <string>

using namespace std;

#include "Hashlink.h"

const int tableSize = 13 ;

const int hashSize = 3 ;

class HashTable

{

public:

// default constructor

// pre: LinkedList.h must be included, tableSize must be definced as

// a global constant.

// post: A HashTable with linked list array will be created.

HashTable(){} ;

// destructor

// pre: none

// post: deletes linked lists in HashTable

~HashTable() ;

// function: hash

// pre: must pass in a string object

// post: stores string into hash table

void hash(string &str) ;

void display(ostream &out) const ;

private:

LinkedList hashTable[tableSize] ;

} ; // end HashTable

// Non-member functions

ostream &operator<<(ostream &out, const HashTable &hashTable) ;

//-HashLink.cpp Implementation File-----------------------------

// Written by: Chris Dang Class: CSCI 1107

//Implement file to be used with HashTable.h, HashTable.cpp, and HashLink.h

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

#include <iostream>

#include <string>

using namespace std;

#include "HashLink.h"

int LinkedList::size() const {

return mySize ;

}//end size getter

void LinkedList::display(ostream &out) const {

LinkedList::NodePointer ptr = first ;

while (ptr != NULL) {

out << ptr -> data << " " ;

ptr = ptr -> next ;

}// end while

}// end display ;

void LinkedList::insert(int index, ElementType dataValue) {

Node \*nPtr = new(nothrow) Node(dataValue) ;

//error checking: if index to be passed in is out of bounds of list,

//the index isn't used, the linked list remains unchanged,

//and exectution continues

if (index < 0 || index > mySize)

cerr << "Index " << index << " is not a valid index.\n" ;

else {

//if trying to insert new node to first of the list,

//this if will handle the inseration

if (index == 0) {

nPtr -> next = first;

first = nPtr ;

}//end if

//if trying to insert somewhere else in the list

//other than first position, this else will will handle the insertation

else {

Node \*predPtr = first ;

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

predPtr = predPtr -> next ;

nPtr -> next = predPtr -> next ;

predPtr -> next = nPtr ;

}// end else

++mySize ;

}// end else

}//end insert

void LinkedList::erase(int index) {

//error checking: if index to be passed in is out of bounds of list,

//the index isn't used, the linked list remains unchanged,

//and exectution continues

if (index < 0 || index > mySize)

cerr << "Index " << index << " is not a valid index.\n" ;

else {

//if trying to erase node at beginning of list

//this if will handle the erase

if (index == 0) {

Node \*ptr = first ;

first = first -> next ;

delete ptr ;

}// end if

else {

Node \*predPtr = first ;

Node \*ptr = first -> next ;

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

predPtr = ptr ;

ptr = ptr -> next ;

}//end for

predPtr -> next = ptr -> next ;

delete ptr ;

//if deleting the last node from list,

//this will handle setting a new last node by setting

//the next pointer to null

if (index == mySize - 1)

predPtr -> next = NULL ;

}// end else

--mySize ;

}//end else

}// end erase

LinkedList::~LinkedList() {

// creates ptr and points toward first node in list

if (first != NULL) {

NodePointer ptr = first -> next ;

// traverses list, reassigns first of the list, and

// deletes nodes in traversal

for (int i = 1; i < mySize; i++) {

first = ptr -> next ;

delete ptr ;

ptr = first ;

} // end for

}// end if

} // end destructor

LinkedList::LinkedList(const LinkedList & origList) {

// copies size of original list to new list

mySize = origList.mySize ;

// if original list is emtpy, first is set to NULL pointer

if (origList.first -> next == NULL)

first = NULL ;

else {

NodePointer origPtr, lastPtr ;

origPtr = origList.first -> next ;

Node \*nPtr = new(nothrow) Node(origList.first -> data) ;

lastPtr = nPtr ;

first = lastPtr ;

// traverses the original list, constructing nodes based of the

// originial list and adding them to the new list

for (int i = 1; i < mySize ; i++) {

Node \*nPtr = new(nothrow) Node(origPtr -> data) ;

lastPtr -> next = nPtr ;

lastPtr = lastPtr -> next ;

origPtr = origPtr -> next ;

} // end for ;

} // end else

} // end copy constructor

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

// checks to see if this is a self assignemnet

// if it is a self assignment, returns itself

if (this != &origList) {

// destroys value of left hand side of assignment

// to avoid memory leak

this -> ~LinkedList() ;

// copies size of original list to new list

mySize = origList.mySize ;

// if original list is emtpy, first is set to NULL pointer

if (origList.first -> next == NULL)

first = NULL ;

// creates a copy of the rhs list to the left hand side

else {

NodePointer origPtr, lastPtr ;

origPtr = origList.first -> next ;

Node \*nPtr = new(nothrow) Node(origList.first -> data) ;

lastPtr = nPtr ;

first = lastPtr ;

// traverses the original list, constructing nodes based of the

// originial list and adding them to the new list

for (int i = 1; i < mySize ; i++) {

Node \*nPtr = new(nothrow) Node(origPtr -> data) ;

lastPtr -> next = nPtr ;

lastPtr = lastPtr -> next ;

origPtr = origPtr -> next ;

} // end for ;

} // end else

}// end if

return \*this ;

} // end overloaded assignment operator

// Non-member functions

ostream &operator<<(ostream &out, const LinkedList &List) {

List.display(out) ;

return out ;

}//end overloaded insertation operator

/\*--- HashTable.cpp ------------------------------------------------------------

Written by: Chris Dang

Written for: CSCI 1107

Implement file for Hash Table Class

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

#include <iostream>

using namespace std;

#include "HashTable.h"

HashTable::~HashTable() {

// inserts empty nodes to force linked list destructor

for (int i = 0; i < tableSize; i++) {

hashTable[i].insert(0, "");

hashTable[i].insert(0, "");

}// end for

} // end destructor

void HashTable::hash(string &str) {

int index = 0;

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

index = index + str[i] ;

index = index % tableSize ;

// any hash will be added to the front of the list as order does not matter

hashTable[index].insert(0, str) ;

//cout << str << " has now been stored in the hash table.\n" ;

} // end hash

void HashTable::display(ostream &out) const {

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

out << "Index " << i << " hash: " << hashTable[i] << endl ;

} // end display

// Non-member functions

ostream &operator<<(ostream &out, const HashTable &hashTable) {

hashTable.display(out) ;

return out ;

}//end overloaded insertation operator

/\*--- HashDriver.cpp --------------------------------------------------------

Chris Dang Class: CSCI 1107

Driver file to test hash table.

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

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "Hashlink.h"

#include "HashTable.h"

int main() {

// file input

ifstream infile;

infile.open("DearMarlin.txt") ;

if (!infile) {

cerr << "Error opening file. --- Terminating program\n\n" ;

exit(1107) ;

} // end if

string str = "Hello";

string str2 = "How are you?" ;

string str3 = "I turn strings into hash" ;

string word ;

HashTable testHash, dearMarlin ;

cout << "Entering Test strings\n" ;

testHash.hash(str) ;

testHash.hash(str2) ;

testHash.hash(str3) ;

cout << "Now displaying hash:\n"

<< "--------------------------\n";

cout << testHash ;

cout << "\nNow adding Dear Marlin letter\n" ;

while (infile >> word)

dearMarlin.hash(word) ;

cout << "Now displaying hash for Dear Marlin letter:\n"

<< "--------------------------\n";

cout << dearMarlin ;

infile.close() ; // close the file

// destructors are called in due to loss of scope of has tables

return 0 ;

} // end main

