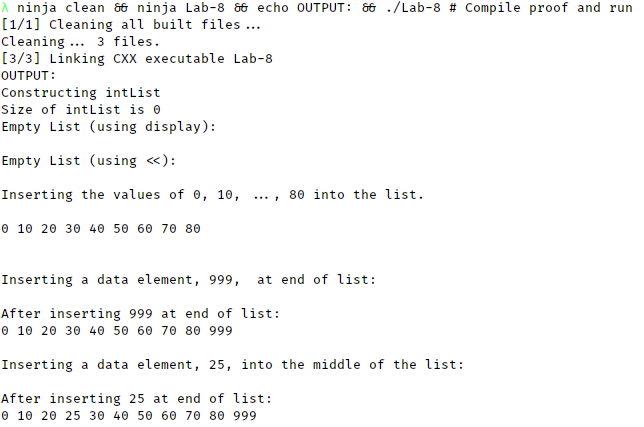
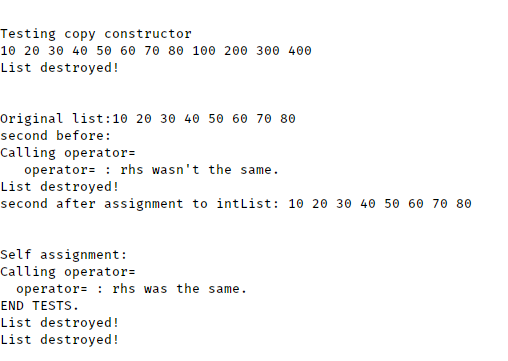
Lab 5.1 Screenshots:







**Code:**

/\*--- 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.

Johnathan Lee CSCI 1107

Due 03/20/18

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

#ifndef LINKEDLIST

#define LINKEDLIST

#include <iostream>

using namespace std;

//----- Add typdef statement here

typedef int ElementType;

class LinkedList {

public:

//------ LinkedList OPERATIONS

// Prototype the class constructor here

**LinkedList**();

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

Constructs an empty LinkedList object.

Precondition: None.

Postcondition: This list's data members have been initialized

for an empty list.

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

// Prototype and document the size() operation here

int **size**() const;

/\* --- size() --------------------------------------------------------------

\* Gives the number of elements in the LinkedList.

\* PostCondition: this->mySize is returned.

\*/

// Prototype and document display() here

/\* --- display() -----------------------------------------------------------

\* Displays all elements of this LinkedList

\* Postconditions: All elements of this, if there are any, are printed to

\*/

void **display**(ostream& out) const;

// Prototype insert() here

void **insert**(ElementType newElem);

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

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 target);

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

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**();

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

\* Deletes the contents of the entire list.

\* Pre: None

\* Post: All elements are destroyed and their memory freed.

\*/

// Prototype and document the copy constructor here

**LinkedList**(const LinkedList& rhs);

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

\* Deep copies the list.

\* Pre: None.

\* Post: \*this has been constructed with the contents of rhs, but NOT the

\* same memory locations.

\*/

// Prototype and document the assignment operator here

const LinkedList& *operator*=(const LinkedList& rhs);

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

\* Deep copies the list IF the rhs is not the same object.

\* Pre: None

\* Post: \*this is the same as rhs, and a reference to \*this is returned.

\*/

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 ele);

/\* --- 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

NodePointer first;

}; //--- end of LinkedList class

// Put prototype of operator<<() here

ostream& *operator*<<(ostream& out, const LinkedList& list);

#endif

/\* LinkedList.cpp

\* Definitions for LinkedList.h

\* Johnathan Lee CSCI 1107

\* Due 03/20/18

\*/

#include "LinkedList.h"

#include <cassert>

#include <new>

LinkedList::LinkedList() {

first = NULL;

}

int LinkedList::size() const {

int count = 0;

Node\* cur = first;

while (cur) {

cur = cur->next;

count++;

}

return count;

}

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

NodePointer ptr = *this*->first;

while (ptr) {

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

ptr = ptr->next;

}

}

void LinkedList::insert(ElementType newElem) {

Node \*cur = first, \*pred = NULL;

Node\* newNode = new Node(newElem);

while (cur && cur->data < newElem) {

pred = cur;

cur = cur->next;

}

if (pred)

pred->next = newNode;

else

first = newNode;

newNode->next = cur;

}

void LinkedList::erase(ElementType target) {

Node \*cur = first, \*pred = NULL;

while (cur && cur->data != target) {

pred = cur;

cur = cur->next;

}

if (cur != NULL) { // We found it and/or list wasn't empty.

// Deletion in the middle or end

// I.e cur isn't == first.

if (pred != NULL)

pred->next = cur->next;

// Deletion at beginning

// Make sure we either know it's empty OR know the next link.

if (cur == first)

first = cur->next;

delete cur;

} // Otherwise don't bother doing anything.

}

LinkedList::~**LinkedList**() {

cout << "List destroyed!\n";

Node\* cur = first;

while (cur != NULL) {

first = cur->next;

delete cur;

cur = first;

}

}

LinkedList::LinkedList(const LinkedList& rhs) {

// Note for code maintenance: The below code is the same as that in operator=

if (rhs.first == NULL) {

first = NULL;

} else {

Node \*origPtr = rhs.first, \*lastPtr = NULL;

lastPtr = new Node(origPtr->data);

first = lastPtr;

origPtr = origPtr->next;

while (origPtr != NULL) {

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

lastPtr = lastPtr->next;

origPtr = origPtr->next;

}

}

}

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

cout << "Calling operator=\n";

if (this != &rhs) {

cout << " operator= : rhs wasn't the same.\n";

*this*->~LinkedList(); // Destroy value of var

if (rhs.first == NULL) {

first = NULL;

} else {

Node \*origPtr = rhs.first, \*lastPtr = NULL;

lastPtr = new Node(origPtr->data);

first = lastPtr;

origPtr = origPtr->next;

while (origPtr != NULL) {

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

lastPtr = lastPtr->next;

origPtr = origPtr->next;

}

}

} else // FOR DEBUG AND PART 7's testing.

cout << " operator= : rhs was the same.\n";

return \*this;

}

ostream& *operator*<<(ostream& out, const LinkedList& list) {

list.display(*out*);

return out;

}

LinkedList::Node::Node(ElementType ele) {

data = ele;

}

/\*--- 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

Johnathan Lee CSCI 1107

Due 03/20/18

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

#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 << "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 << "\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 ----

// ADD STATEMENTS HERE TO TEST ASSIGNMENT OPERATOR

LinkedList second;

cout << "second before: " << second << endl;

second = intList;

cout << "second after assignment to intList: " << second << endl;

cout << "\n\nSelf assignment: \n";

second = second;

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

cout << "END TESTS.\n";

}