EECS268:Lab3

From ITTC

Navigation

Home

Information

Syllabus
Schedule

Classwork

Labs
Submitting Work

Contents

Due time

This lab is due 7 days the end of your lab session. If your lab starts, let's say, at 2pm on Thursday, you have until 1:59pm the following Thursday to submit.

Required Files

- main.cpp
- Node.h and Node.cpp
- LinkedList.h and LinkedList.cpp
- Test Code
  - Test.h (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab03/Test.h) Test.cpp (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab03/Test.cpp)
    - Test base class
    - You don't need to declare and instance or use this class.
  - Test_LinkedList.h (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab03/Test_LinkedList.h) Test_LinkedList.cpp (http://people.eecs.ku.edu/~jgibbons/eecs268/2016spring/lab03/Test_LinkedList.cpp)
    - Contains all the methods to test your LinkedList class
    - NOTE: You must name your methods as specified on the wiki, **case sensitive!**
- Makefile (https://wiki.ittc.ku.edu/ittc_wiki/index.php?title=EECS268:Makefiles)

Libraries You May Include

- iostream
- vector

Overview

This lab is all about getting used to pointers, creating and using linked structures. We will create a container class called Node, that will house a single int and a pointer to another node. The pointer will act as our link between two nodes. We will then create a LinkedList class that will create lists or a series of nodes and allow us to interact with the list in various ways.

Makefile

In this lab we will be using a feature from c++11, the nullptr key word. This doesn't really impact your algorithm in any significant way, but you may need to update your makefile. Here is an example makefile with the c++11 flag:

```
Program: main.o MyClass.o
	g++ -g -std=c++11 main.o MyClass.o -o Program

main.o: main.cpp MyClass.h
	g++ -g -std=c++11 -c main.cpp

MyClass.o: MyClass.h MyClass.cpp
	g++ -g -std=c++11 -c MyClass.cpp

clean:
	rm *.o Program
```

Classes

Node Class

A Node is a simple container. It has the follow public members:

- Node()
  - Constructor
  - Sets m_value to zero
  - Sets m_next to nullptr
- void setValue(int val)
  - sets m_value to val
- int getValue() const
  - returns m_value
- void setNext(Node* prev)
  - sets m_next
- Node* getNext() const
  - returns m_next

Node class private members:

- int m_value
  - the value in the node
- Node* m_next
  - a pointer to another node

LinkedList Class

The LinkedList class will be in control of generating a list or sequence of Nodes. The class will have the following methods:

public methods:

- LinkedList();
  - constructor
  - sets m_front to nullptr and m_size to zero
- ~LinkedList();
  - Deletes all nodes in the list.
- bool isEmpty() const;
  - returns true if the list is empty, false otherwise.
- int size() const;
  - returns the number of elements in the list.
- bool search(int value) const;
  - returns true is the value is in the list, false otherwise.
- void printList() const;
  - prints the list to the console.
  - If the list is empty print the empty string, ""
- void addBack(int value);
- One new element added to the end of the list.
- void addFront(int value);
  - One new element added to the front of the list.
- bool removeBack();
  - One element is removed from the back of the list.
  - Returns true if the back element was removed, false if the list is empty.
- bool removeFront();
  - One element is removed from the front of the list.
  - returns True if the front element was removed, false if the list is empty.
- std::vector<int> toVector() const;
  - returns a standard vector with the contents of the list put inside
  - In short, you'll create a new vector, traverse your list, copying the contents into the vector then return that vector.
  - Vector reference (http://www.cplusplus.com/reference/vector/vector/)
  - This method is used by the test class to verify the contents of your list

private members:

- Node* m_front;
- int m_size;

Test Class

You have been provided a Test class (See Test.h/cpp files). **DO NOT** alter the code for this class code for this class. You may call individual tests as you please, but do not alter how the tests are performed. We'll be checking this at grading time.

Purpose

The Test class has public methods for individual test and a method that runs all the tests and scores your code called runTests(). You can call these methods from an instance of the Test class. It will put your LinkedList class through a series of stress tests. Here's what it will test for:

- Test #1: size of empty list is zero (2pts)

- Test #2: size returns correct value after 1 addFront (2pts)
- Test #3: size returns correct value after 1 addBack (3pts)
- Test #4: size returns correct value after multiple addFront (5pts)
- Test #5: size returns correct value after multiple addBack (5pts)
- Test #6: size returns correct value after adds and removeFront (5pts)
- Test #7: size returns correct value after adds and removeBack (5pts)
- Test #8: search returns false on empty list (2pts)
- Test #9: search returns false when value not in list (5pts)
- Test #10: search returns true if value is in large list (10pts)
- Test #11: toVector returns empty vector when called with empty list (2pts)
- Test #12: toVector creates vector with the contents of large list (5pts)
- Test #13: removeFront on empty list returns false (2pts)
- Test #14: removeBack on empty list returns false (2pts)
- Test #15: size preserved by removeFront on populated list (5pts)
- Test #16: size preserved by removeBack on populated list (10pts)
- Test #17: order preserved by removeFront on populated list (10pts)
- Test #18: order preserved by removeBack on populated list (10pts)

Running the tests in main

NOTE: You need to update your Makefile to create a Test.o and Test_LinkedList.o

Main's job is just to call the runTests method, meaning a place for us to test our LinkedList class. You will pass the size of the test as a command line arguement.

```
$> ./lab03 5000
```

```
#include <string>
#include "Test_LinkedList.h"

int main(int argc, char** argv)
{
        int testSize = std::stoi(argv[1]); //convert from char* to int
        Test_LinkedList tester(testSize); //create a tester
        tester.runTests(); //run tests
        return (0);
}
```

Checking for Memory Leaks

An easy way to check for memory leaks is to use a program called valgrind. It's a program that you can hand your lab off to and it will run your lab and tell if you have any memory leaks.

```
valgrind --leak-check=full ./YourProgram
```

Example of running your program with valgrind:

```
valgrind --leak-check=full ./Lab03 500
```

It will tell you if any leaks occurred. Make sure you have the -g flag in your Makefile and you'll get the added bonus of see what lines of code created an object that was never deleted. Science!

Issues with Valgrind

Recently, valgrind began reporting on a buffer that automatically allocated then deallocated after our main ends. This would look like you're doing something wrong. But be aware if you run a program with an empty main, you'll see the following:

```
==23799== HEAP SUMMARY:
==23799==     in use at exit: 72,704 bytes in 1 blocks
==23799==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==23799==
==23799== LEAK SUMMARY:
==23799==    definitely lost: 0 bytes in 0 blocks
==23799==    indirectly lost: 0 bytes in 0 blocks
==23799==      possibly lost: 0 bytes in 0 blocks
==23799==    still reachable: 72,704 bytes in 1 blocks
==23799==         suppressed: 0 bytes in 0 blocks
==23799== Reachable blocks (those to which a pointer was found) are not shown.
==23799== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==23799==
==23799== For counts of detected and suppressed errors, rerun with: -v
==23799== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Note the amount of allocations. My empty main didn't allocate anything, but valigrind is reporting everything it can see. There is a buffer being allocated that we don't control. It is freed, but after our main ends, so Valgrind reports on it.

Just make sure...

- your allocs are only off by 1
- there are no memory errors
- nothing is lost

Rubric

- 90pts Tests
- 5pts Memory leaks: There should be no memory leaks. Use valgrind ./YourProgramName to verify
  - Any memory leaks will result in a loss of all 5 points!
  - Your destructor will play a key role in preventing memory leaks
- 5pts Comments and documentation:
  - @author, @file, @since, @brief (the author, file name, date, and a brief description of a file) at the **top of every file!**
  - @pre, @post, @return, (Pre conditions, Post conditions, Return descriptions) in header files. Not required for cpp files
  - NOTE you don't need to comment the Makefile or the Test files

Remember, if we type "make" into the console and your program fails to compile, we do not grade it.

Submission instructions

Send a tarball with all the necessary files (*.h, *.cpp, and makefile) in a tar file to your GTA email. Do not include any object (*.o) or executable file.

Creating a File Archive Using Tar

The standard Unix utility for created archived files is **`tar`**. Tar files, often called **tarballs**, are like zip files.

1. Create a folder to hold the files you will be sending in. The folder should be named like *LastName-KUID-Assignment-Number*:

   ```
   mkdir Smith-123456-Lab-0#
   ```

2. Now, copy the files you want to submit into the folder:
3. Tar everything in that directory into a single file:

   ```
   tar -cvzf Smith-123456-Lab-0#.tar.gz Smith-123456-Lab-0#
   ```

That single command line is doing a number of things:

- `tar` is the program you're using.
- `-cvzf` are the options you're giving to `tar` to tell it what to do.
  - `c`: **create** a new tar file
  - `v`: operate in **verbose** mode (show the name of all the files)
  - `z`: **zip** up the files to make them smaller
  - `f`: create a **file**
- `Smith-123456-Lab-0#.tar.gz`: the name of the file to create. It is customary to add the `.tar.gz` extension to tarballs created with the `z` option.
- `Smith-123456-Lab-0#`: the directory to add to the tarball

Please note that it is **your** responsibility to make sure that your tarball has the correct files. You can view the contents of a tarball by using:

```
tar -tvzf filename.tar.gz
```

Emailing Your Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line **must** look like "[EECS 268] *SubmissionName*":

> [EECS 268] Lab 0#

Note that the subject should be *exactly* like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.
