lab07 : Linked lists

num	ready?	description	assigned	due
lab07	true	Linked lists	Tue 11/19 09:00AM	Mon 11/26 11:59PM

Goals of this lab

The goal of this lab is get more practice with iterating through linked lists and solving problems. Continue to practice code tracing to reason about your code. We request that you DO NOT ask the staff to debug your code. They have been specifically instructed not to debug for you, rather to guide in the process.

You are also required to apply the style guidelines as described in this presentation: link to style guide

If you are using a style different from what is recommended and feel strongly about continuing in that style, you may include a note as a comment in the README of your project explaining the rationale behind your choices. Cite any sources as relevant.

If you are using any instructor written code or hints from a previous offering of this course, cite your source. Remember that code that is not written by you is in violation of the academic honesty for this class.

Step by Step Instructions: PLEASE READ CAREFULLY!

Step 1: Getting Started

1. Decide if you are working alone, or working in a pair. Pair programming is OPTIONAL for this lab.
2. Choose who will be the first driver and who will start as navigator, and then remember to switch (at least once) during the lab.
3. Go to github and create a git repo for this lab following the naming convention specified in previous labs (this step carries style points, see our feedback on previous labs to understand what we are looking for). If you are working with a partner only one of you needs to create the repo.
4. If you are working with a partner and you are the one who created the github repo, add your partner as a collborator on the repo
5. Decide on initial navigator and driver.
6. Driver, log on to your CSIL account.
7. Open a terminal window and log into the correct machine.
8. Change into your CS 16 directory

Note: Remember to push your work to github at the end of EVERY work session. That way, both partners always have access to the latest version of the code even if the code is being developed on one partner's CoE account.

Step 2: Obtaining the starter code

- Navigate to your cs16 directory and clone the git repository you created

```
git clone git@github.com:ucsb-cs16-s18-mirza/lab07_alily_jgaucho.git
```

- cd into this new directory

```
cd lab07_alily_jgaucho
```

- Copy the starter code by typing the following command:

```
cp /cs/faculty/dimirza/cs16/labs/lab07/* ./
```

Typing the list (ls) command should show you the following files in your current directory

```
[dimirza@csil-03 lab07-startercode]$ ls
linkedListFuncs.cpp  Makefile                tddFuncs.cpp
linkedListFuncs.h    moreLinkedListFuncs.cpp  tddFuncs.h
linkedList.h         moreLinkedListFuncs.h
llTests.cpp          README.md
[dimirza@csil-03 lab07-startercode]$
```

Step 3: Reviewing the files and what your tasks are

Here is a list of your tasks for this lab:

Step 3a: Familiarize yourself with the big picture

Type "make tests" and you will see some tests pass, but some fail.

You are finished when all the tests pass. We have implemented a few function that involve linked lists in `linkedListFuncs.cpp`. There is only one file you need to edit this week:

`moreLinkedListFuncs.cpp` contains more functions that deal with linked lists.

Step 3b: Work on the linked list functions

Working on the linked list functions below is one of the most important things you can do to prepare for the final exam.

There are 7 functions you will need to write for this lab:

- `addIntToEndOfList`
- `addIntToStartOfList`
- `pointerToMax`
- `pointerToMin`
- `largestValue`
- `smallestValue`
- `sum`

Each one has a set of tests which can be found under its corresponding heading when you type `make tests`. For example, the addIntToEndOfList tests look like this to start:

```
./llTests 1
-------------ADD_INT_TO_END_OF_LIST-------------
PASSED: linkedListToString(list)
   FAILED: linkedListToString(list)
     Expected: [42]->[57]->[61]->[12]->null Actual: [42]->[57]->[61]->null
   FAILED: linkedListToString(list)
     Expected: [42]->[57]->[61]->[12]->[-17]->null Actual: [42]->[57]->[61]->null
PASSED: linkedListToString(empty)
   FAILED: linkedListToString(empty)
     Expected: [0]->null Actual: null
   FAILED: linkedListToString(empty)
     Expected: [0]->[19]->null Actual: null
```

You should replace each function stub with the correct code for the function until all of the tests for each one pass. It is recommended that you work on the functions one at a time in the order that they are presented above. That is, get all the tests to pass for addIntToStartOfList then addIntToEndOfList and so on. When all the tests pass, move on to the next step.

Step 4: Checking your work before submitting

When you are finished, you should be able to type `make clean` and then `make tests` and see the following output:

```
-bash-4.2$ make clean
/bin/rm -f llTests *.o
-bash-4.2$ make tests
g++ -Wall -Wno-uninitialized   -c -o llTests.o llTests.cpp
g++ -Wall -Wno-uninitialized   -c -o linkedListFuncs.o linkedListFuncs.cpp
g++ -Wall -Wno-uninitialized   -c -o moreLinkedListFuncs.o moreLinkedListFuncs.cpp
g++ -Wall -Wno-uninitialized   -c -o tddFuncs.o tddFuncs.cpp
g++ -Wall -Wno-uninitialized  llTests.o linkedListFuncs.o moreLinkedListFuncs.o tddFuncs.o -o llTests
./llTests 1
--------------ADD_INT_TO_END_OF_LIST--------------
PASSED: linkedListToString(list)
PASSED: linkedListToString(list)
PASSED: linkedListToString(list)
PASSED: linkedListToString(empty)
PASSED: linkedListToString(empty)
PASSED: linkedListToString(empty)
./llTests 2
--------------ADD_INT_TO_START_OF_LIST--------------
PASSED: linkedListToString(list)
PASSED: linkedListToString(list)
PASSED: linkedListToString(list)
PASSED: linkedListToString(empty)
PASSED: linkedListToString(empty)
PASSED: linkedListToString(empty)
./llTests 3
--------------POINTER_TO_MAX--------------
PASSED: pointerToMax(list1)
PASSED: pointerToMax(list1)
PASSED: pointerToMax(list1)->data
PASSED: pointerToMax(list1)->next->data
PASSED: pointerToMax(list2)
PASSED: pointerToMax(list2)
PASSED: pointerToMax(list2)->data
PASSED: pointerToMax(list3)
PASSED: pointerToMax(list3)
PASSED: pointerToMax(list3)->data
PASSED: pointerToMax(list4)
PASSED: pointerToMax(list4)
PASSED: pointerToMax(list4)->data
PASSED: pointerToMax(list4)->next->data
./llTests 4
--------------POINTER_TO_MIN--------------
PASSED: pointerToMin(list1)
PASSED: pointerToMin(list1)
PASSED: pointerToMin(list1)->data
PASSED: pointerToMin(list1)->next->data
PASSED: pointerToMin(list2)
PASSED: pointerToMin(list2)
PASSED: pointerToMin(list2)->data
PASSED: pointerToMin(list3)
PASSED: pointerToMin(list3)
PASSED: pointerToMin(list3)->data
PASSED: pointerToMin(list4)
PASSED: pointerToMin(list4)
PASSED: pointerToMin(list4)->data
PASSED: pointerToMin(list4)->next->data
./llTests 5
--------------LARGEST_VALUE--------------
PASSED: largestValue(list1)
PASSED: largestValue(list2)
PASSED: largestValue(list3)
PASSED: largestValue(list4)
./llTests 6
--------------SMALLEST_VALUE--------------
PASSED: smallestValue(list1)
PASSED: smallestValue(list2)
PASSED: smallestValue(list3)
PASSED: smallestValue(list4)
./llTests 7
--------------SUM--------------
PASSED: sum(list1)
PASSED: sum(list2)
PASSED: sum(list3)
PASSED: sum(list4)

-bash-4.2$
```

At that point, you are ready to try submitting on gradescope.

Step 5: Submitting via gradescope

Submit the moreLinkedListFuncs.cpp file on gradescope. Make sure to add your partner as a collaborator if you had one.

Grading Rubric

Most of the points will be awarded based on gradescope automatic grading. Other points will be assigned after visual code inspection by TAs.

Gradescope automatic points

Test Name	Value
addIntToEndOfList	(10 pts)
addIntToStartOfList	(10 pts)
pointerToMax	(10 pts)
pointerToMin	(10 pts)
largestValue	(10 pts)
smallestValue	(10 pts)
sum	(10 pts)

Code inspection human-assigned points

- (10 pts) Code style, including but not limited to:
    1. Code can be easily understood by humans familiar with C++ (including both the author(s) of the code, and non-authors of the code.)
    2. Code is neatly indented and formatted, following standard code indentation practices for C++ as illustrated in either the textbook, or example code given in lectures and labs
    3. Variable names choices are reasonable
    4. Code is reasonably "DRY" (as in "don't repeat yourself")—where appropriate, common code is factored out into functions
    5. Code is not unnecessarily or unreasonably complex when a simpler solution is available
    6. Other guidelines as discussed in the style guide for this class: link to style guide
    7. Fix any style issues (inconsistencies or other problems) in the starter code.

An important word about academic honesty and the gradescope system

We will test your code against other data files too—not just these. So while you might be able to pass the tests on gradescope now by just doing a hard-coded "cout" of the expected output, that will NOT receive credit.

To be very clear, code like this will pass on gradescope, BUT REPRESENTS A FORM OF ACADEMIC DISHONESTY since it is an attempt to just "game the system", i.e. to get the tests to pass without really solving the problem.

I would hope this would be obvious, but I have to say it so that there is no ambiguity: hard coding your output is a form of cheating, i.e. a form of "academic dishonesty". Submitting a program of this kind would be subject not only to a reduced grade, but to possible disciplinary penalties. If there is *any* doubt about this fact, please ask your TA and/or your instructor for clarification.