University of Colorado Boulder

CSCI-2270
Assignment 3
Instructor: Boese

Linked List
Communication Between Towers

Objectives

- Create, traverse, add, and delete nodes in a linked-list

Background

Communication between towers
In the Lord of the Rings trilogy, there is a scene where the first beacon is lit in the towers of Minas Tirith. The second beacon then sees the fire, and knows to light its fire to send a signal to the third beacon, and so forth. This was a means of communicating in the days before telegraphs were invented as it was much faster than sending a human rider to deliver a message. Communication towers were equipped with signaling mechanisms, such as mirrors, that could spell out messages using the positions of the mirrors.

Today, there are several examples of communication networks that are conceptually similar, but much more technically advanced, that route messages through multiple hubs between the sender and the receiver. For example, when you type a URL into a web browser, a request is sent through a network of service providers to the destination, and then packets of information are sent back to your machine. If I type www.google.com from my home in Boulder, my request follows this path:

```
1 192.168.2.1 (192.168.2.1)
2 c-24-9-60-1.hsd1.co.comcast.net (24.9.60.1)
3 te-9-7-ur02.boulder.co.denver.comcast.net
4 xe-13-3-1-0-ar01.aurora.co.denver.comcast.net
5 he-3-10-0-0-cr01.denver.co.ibone.comcast.net (68.86.92.25)
te-1-1-0-4-cr01.chicago.il.ibone.comcast.net (68.86.95.205)
6 xe-2-0-0-0-pe01.910fifteenth.co.ibone.comcast.net (68.86.82.2)
7 as15169-1-c.910fifteenth.co.ibone.comcast.net (23.30.206.106)
8 72.14.234.57 (72.14.234.57)
9 209.85.251.111 (209.85.251.111)
10 den03s06-in-f16.1e100.net (74.125.225.208)
```

Each IP address is a hop in the network for my request, which is received at each service provider and then forwarded to the next service provider in the network, depending on the final destination of the message.

(Note: I got this path by typing traceroute www.google.com in a terminal window. From campus, you will see a different path.)

<u>Build your own communications network</u>

<u>Overview</u>: In this assignment, you're going to simulate a communications network using a linked list. Each node in your linked list will represent a city and you need to be able to send a message between nodes from one side of the country to the other. Your program also needs to provide the capability to update the network by adding cities. After adding, it should still be able to transmit the message. *(Note: We'll refer to the linked list as the network throughout this document.)*

You will include the following cities in your network by default:

Los Angeles, Phoenix, Denver, Dallas, Atlanta, & New York

<u>Details</u>: We have defined the class `CityNetwork` to hold the linked list, and we've declared all necessary methods inside it. You will need to complete these methods to implement the linked list functionality (*see the starter code*).

Implement each city as a `struct` with *(use the starter code)*

- a name,
- a pointer connecting it to the next city in the network, and
- a place to store the message being sent. *(The message is a string.)*
- an integer for the number of messages that have been sent through this city

<u>Program operation:</u> When your program starts, it will display a menu allowing the user to interactively modify your network and send messages. The menu looks like the one shown here:

```
Select a numerical option:
+=====Main Menu=========+
1. Build Network
2. Print Network Path
3. Transmit Message
4. Add City
5. Delete City
6. Clear Network
7. Quit
+-----------------------------------+
```

The user will select the number for the menu option and your program should respond accordingly to that number. Your menu options will support the following functionality:

1. <u>Build Network</u>: This option builds the linked list using the default cities listed above, in the order they are listed. Each city needs to have a name, a pointer to the next city, and a message value, which will initially be an empty string. This option should be selected first to build the network, and can be selected anytime the user wants to rebuild the starting network after adding cities. As part of the Build Network functionality, you should print the name of each city in the network once the network is built in the following format by calling the function for Option #2:

```
Los Angeles -> Phoenix -> Denver -> Dallas -> Atlanta -> New York -> NULL
```

Here is a screenshot showing the format that is expected*:*

```
== CURRENT PATH ==
Los Angeles -> Phoenix -> Denver -> Dallas -> Atlanta -> New York ->  NULL
===
```

2. <u>Print Network Path</u>: This option prints out the linked list in order from the head to the tail by following the `next` pointer for each city. You should print the name of each city. The function could be very useful to you when debugging your code. The format should be the same as shown above.
There is one space before and after each arrow which is a dash and a >

3. <u>Transmit Message</u>: This option prompts the user to supply a message and a destination, then transmits the message. The message should start at the beginning of the network and end at the city specified. Each city which the message passes through should update its `numberMessages` variable to count the total number of messages passed through that city.

Example:

```
#> 3
Enter name of the city to receive the message:
Atlanta
Enter the message to send:
The Broncos game is a blast!
Los Angeles [# messages passed: 1] received: The Broncos game is a blast!
Phoenix [# messages passed: 1] received: The Broncos game is a blast!
Denver [# messages passed: 1] received: The Broncos game is a blast!
Dallas [# messages passed: 1] received: The Broncos game is a blast!
Atlanta [# messages passed: 1] received: The Broncos game is a blast!

Select a numerical option:
+=====Main Menu=========+
1. Build Network
2. Print Network Path
3. Transmit Message
4. Add City
5. Delete City
6. Clear Network
7. Quit
+-----------------------+
#> 3
Enter name of the city to receive the message:
New York
Enter the message to send:
Moving to east coast
Los Angeles [# messages passed: 2] received: Moving to east coast
Phoenix [# messages passed: 2] received: Moving to east coast
Denver [# messages passed: 2] received: Moving to east coast
Dallas [# messages passed: 2] received: Moving to east coast
Atlanta [# messages passed: 2] received: Moving to east coast
New York [# messages passed: 1] received: Moving to east coast
```

4. Add City: This option allows the user to add a new city to the network. If the user selects this option, then they should be prompted for the name of the city and the city that the new city should follow in the network. For example, if the user wants to add Washington D.C. after Dallas in the network, then the first few cities in the network would be:

Los Angeles -> Phoenix -> Denver -> Dallas -> Washington D.C. -> ……

If the user wants to add a new city to the head of the network, e.g. replace Los Angeles as the starting city, then they should type First when prompted for the previous city and your code should handle this special case. You do not have to handle the case where the user enters an invalid previous city (we won't test this).

Here are sample screenshots showing the expected output for the add city functionality when the user selects Add City from the menu.

Example 1:

```
#> 4
Enter a new city name:
Washington D.C.
Enter the previous city name (or First):
Dallas
adding: Washington D.C. (prev: Dallas)
== CURRENT PATH ==
Los Angeles -> Phoenix -> Denver -> Dallas -> Washington D.C. -> Atlanta -> New York ->  NULL
===
```

Example 2: Tucson added to the head of the network

```
#> 4
Enter a new city name:
Tuscon
Enter the previous city name (or First):
First
adding: Tuscon (HEAD)
== CURRENT PATH ==
Tuscon -> Los Angeles -> Phoenix -> Denver -> Dallas -> Washington D.C. -> Atlanta -> New York
->  NULL
===
```

5. Delete City: Delete one of the cities in the network. If the user selects this option, then they should be prompted for the name of the city to delete. You do not have to handle the case where the user enters an invalid city.

```
#> 5
Enter a city name:
Dallas
== CURRENT PATH ==
Tuscon -> Los Angeles -> Phoenix -> Denver -> Washington D.C. -> Atlanta -> New York ->  NULL
===
```

6. Clear Network: Deletes all cities in the network. Inside the `deleteEntireNetwork` function, as each node is deleted, print the name of the city being deleted.

```
#> 6
deleting: Tuscon
deleting: Los Angeles
deleting: Phoenix
deleting: Denver
deleting: Washington D.C.
deleting: Atlanta
deleting: New York
Deleted network
```

7. Quit: This option allows the user to exit the program.
For each of the options presented (except Quit), after the user makes their choice and your code runs for that option, you should re-display the menu to allow the user to select another option.

Program Specifications

- Use the starter code on moodle (**HW3-LinkedListNetwork.cpp**). Do not modify completed functions, only add your code where indicated.
- Your code needs to be readable, efficient, and accomplish the task provided.
- Make sure your code is commented enough to describe what it is doing. Include a comment block at the top of the .cpp file with your name, assignment number, and course instructor, and anyone you worked with.
- You must fill in the functions as specified by TODO comments. You do not need any additional functions.
- Consider border cases: what happens when you add or delete cities in the beginning, middle, and end of the list?

Submitting Your Code:

Log into Moodle and go to the Homework link. It is set up in the quiz format. Follow the instructions on each question to submit all or parts of each assignment question. You can check your solution to each question by clicking on the "Check" button. Note that you can only submit your homework once.

Note: There was confusion on the last assignment with moodle build errors. You can use the same options moodle does on your machine to get the same errors:

```
g++ -std=c++11 -Wall -Werror HW3-LinkedListNetwork.cpp
```

Note*: there is no late period on assignments! If you miss the deadline or do not do well, you can sign up for an optional grading interview to get up to half the points missed back. There is also an optional extra credit assignment at the end of the semester you can use to replace one of your homework scores.*