CSCI 2270 Data Structures and Algorithms Summer 2017
Instructor: Holler/MonteroQuesada
Assignment 3

Due Friday, June 23th, by 11am Assignment 3 Part A (Pseudocode)
Due Wednesday, June 28st, by 8am Assignment 3 Part B (Implementation)

Communication Between Towers

Objectives

- **Create, traverse, add, and delete to/from a linked-list**

Background

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 www.google.com is typed from campus, the request might follow this path:

```
 1  10.233.16.1 (10.233.16.1)  1.757 ms  1.939 ms  1.381 ms
 2  engr-engr2.colorado.edu (128.138.81.36)  1.532 ms  1.560 ms  1.800 ms
 3  hut-engr.colorado.edu (128.138.81.134)  1.123 ms  1.022 ms  0.993 ms
 4  fw-hut.colorado.edu (128.138.81.250)  1.249 ms  1.140 ms  5.517 ms
 5  juniper-fw.colorado.edu (128.138.81.193)  2.001 ms  1.829 ms  2.457 ms
 6  frgp-i1-ucb.colorado.edu (198.59.55.10)  2.709 ms  2.732 ms  2.789 ms
 7  xe-0-0-1.core-910.frgp.net (192.43.217.170)  6.084 ms  3.180 ms  2.868 ms
 8  72.14.194.239 (72.14.194.239)  3.135 ms  3.095 ms  2.961 ms
 9  72.14.234.59 (72.14.234.59)  3.294 ms  3.386 ms  3.062 ms
10  209.85.250.237 (209.85.250.237)  4.240 ms  3.633 ms  3.684 ms
11  den03s09-in-f4.1e100.net (216.58.217.4)  3.414 ms  3.710 ms  4.347 ms
```

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 destination of the message. (Note: this path can be obtained by typing traceroute www.google.com in a terminal window. From campus, you will see a different path.)

Build your own communications network

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 and still be able to transmit the message.
(Note: We'll refer to the linked list as the network throughout this document.)

Include the following cities in your network:

Miami
New York
Chicago
Seattle
Denver
Dallas

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.)***

When you initially build your network, the order of the cities should be the same as the order listed above. After the network is built, you will provide the option of adding additional cities.

First, display a menu
When your program starts, you should display a menu that presents the user with options for how to run your program. The menu needs to look like the one shown here:

```
======Main Menu======
1. Build Network
2. Print Network Path
3. Transmit Message Coast-To-Coast-To-Coast
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 need to have the following functionality.

1. **Build Network: This option builds the linked list using the 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:

nullptr <- Miami <-> New York <-> Chicago <-> Seattle <-> Denver <-> Dallas -> nullptr

Here is a screenshot showing the format that CodeRunner is expecting:

```
===CURRENT PATH===
nullptr <- Miami <-> New York <-> Chicago <-> Seattle <-> Denver <-> Dallas -> nullptr
==================
```

2. **Print Network Path: 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 the format in Build Network.**
3. **Transmit Message Coast-to-Coast-to-Coast: This option reads word by word from the *messageIn.txt* file and transmits the message starting at the beginning of the network and ending at the end of the network, and then sends the message back again to the beginning of the network. Using the cities in this write-up, the message would go from Miami to Dallas, passing through each city along the way, and then back to Miami, passing through each city along the way. When a city receives the message, you should print**

<city name> received <word>

where *<city name>* is the name of the city and *<word>* is the word received. When a city receives a word, the word should be deleted from the sender city, i.e set the message for the sender city to an empty string. Here is a screenshot of the output after transmitting the first word in the file:

```
Enter name of file: messageIn.txt
Miami received A
New York received A
Chicago received A
Seattle received A
Denver received A
Dallas received A
```

Note: The name of the file that contains the message should be enter by the user using `"Enter name of file: "`.

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 Atlanta after Miami in the network, then the first four cities in the network would be:**

Miami <-> Atlanta <-> New York <-> Chicago <-> Seattle…

You don't need to print anything when you add a new city, just call the Print Network function again from the menu if you want to verify that the city has been added.

If the user wants to add a new city to the head of the network, e.g. replace Miami as the starting city, then they should type "First" when prompted for the previous city and your code should handle this special case.

Here is a screenshot showing the expected output for the add city functionality when the user selects Add City from the menu.

```
Enter a city name:
Atlanta
Enter a previous city name:
Miami
```

5. **Delete City: This option allows the user to delete a city from the network. When the user selects this option, they should be prompted for the name of the city to delete. Your code should then update the next and previous pointers for the surrounding cities and free the memory for the deleted city.**

```
Enter a city name:
Dallas
```

While developing, you can print the network to verify that node Dallas is absent:

```
===CURRENT PATH===
nullptr <- Miami <-> Atlanta <-> New York <-> Chicago <-> Seattle <-> Denver -> nullptr
==================
```

6. **Clear Network: This option allows the user to delete all cities in the network starting at the head city. After this functionality executes, the head of the list should be nullptr and all cities should be deleted from the network. When a city is deleted, print the name of the city just before freeing the memory. Your clear network method should be called in the destructor of CommunicationNetwork, in addition to being a menu option.**

```
deleting Miami
deleting Atlanta
deleting New York
deleting Chicago
deleting Seattle
deleting Denver
```

7. **Quit: This option allows the user to exit the program.**

For each of the options presented, 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. For simplicity, assume that the user would enter only valid data (e.g. city names).

Structuring your program
The specific cout statements that CodeRunner expects are shown in Appendix A.

The functionality for your network will be implemented in a class called CommunicationNetwork. There is a header file called CommunicationNetwork.hpp that is provided for you on Moodle - to use optionally. If you choose to use the .hpp provided though, you will be restricted to the methods and variables implemented for you already (i.e. don't change anything).

Whether you use the header file or not, each of the menu options needs to be handled by calling methods in your CommunicationNetwork instance. Your code needs to be readable, efficient, and accomplish the task provided.

```
void CommunicationNetwork::addCity(string previousCity, string newCity)
/*Insert a new city into the linked list after the previousCity. The name of the new city is in the argument newCity.
*/

void CommunicationNetwork::transmitMsg(string filename)
/*Open the file and transmit the message between all cities in the network word by word. A word needs to be received at the end of the network before sending the next word. Only one city can hold the message at a time; as soon as it is passed to the next city, it needs to be deleted from the sender city.
*/

void CommunicationNetwork::printNetwork()
/*Start at the head of the linked list and print the name of each city in order to the end of the list. */

void CommunicationNetwork::buildNetwork()
/*Build the initial network from the cities given in this writeup. The cities can be fixed in the function, you do not need to write the function to work with any list of cities. */

void CommunicationNetwork::deleteCity(string cityNameDelete)
/*Find the city in the network where city name matches cityNameDelete. Change the next and previous pointers for the surrounding cities and free the memory.
*/
```

void CommunicationNetwork::clearNetwork()
/*Delete all cities in the network, starting at the head city.
***/**

Suggestions for completing this assignment
There are several components to this assignment that can be treated independently. Our advice is to tackle these components one by one, starting with printing the menu and getting user input. Next, build the network and print it. Then, add the functionality to add additional cities. At last, work on the functionalities to delete a city and clear the network.

Once you get one feature completed, test, test, test, to make sure it works before moving on to the next feature.

There are several examples of how to work with linked lists in Chapter 5 in your book, and we will also be covering these concepts in lectures this week.

Although you need to submit a single code for the implementation and driver, while developing your code you should practice working with three separate files - CommunicationNetwork.hpp, CommunicationNetwork.cpp, and Assignment3.cpp - whether you use the provided header or not. You need to write both the CommunicationNetwork.cpp and the Assignment3.cpp files. You can compile your code on the command-line using g++

g++ -std=c++11 CommunicationNetwork.cpp Assignment3.cpp -o Assignment3

and then run your program on the command-line using

./Assignment3 messageIn.txt

Also, start early.

Pseudocode (20%)
For this assignment, the pseudocode portion will correlate to the functionalities 3 and 4, transmit a message and add a city.

Implementation (80%)
Submit your Assignment3 code by opening the link to the CodeRunner question and pasting your code on the text box. Make sure your code is commented enough to describe what it is doing. Include a comment block at the top with your name, assignment number, and course instructor.

If you do not get your assignment to run on CodeRunner, you will have the option of scheduling an interview grade with your TA to get a grade for the assignment. Even if you do get the assignment to run on CodeRunner, you can schedule the interview if you

just want to talk about the assignment and get feedback on your implementation. Consider the TA office hours if you think that the discussion would be longer than 10min.

What to do if you have questions

There are several ways to get help on assignments in 2270, and depending on your question, some sources are better than others. The Piazza discussion forum is a good place to post technical questions, such as how to add a node to a linked list. When you answer other students' questions on the forum, please do not post entire assignment solutions. The CAs and TAs are also a good source of technical information, especially questions about C++. If, after reading the assignment write-up, you need clarification on what you're being asked to do in the assignment, the TAs and the Instructor are better sources of information than the discussion forum or the CAs.

Appendix A – cout statements that CodeRunner expects

Print path

```
cout << "===CURRENT PATH===" << endl;
cout << tmp->name << " -> ";   //for all nodes in network
cout << "nullptr" << endl;
cout << "==================" << endl;
```

Transmit Message

```
cout<<"Enter name of file: "<<endl;
cout<<sender->cityName<<" received "<<sender-message<<endl;

//if network not built yet, head = nullptr
cout << "Empty list" << endl;
```

Adding a new city

```
cout << "Enter a city name: " << endl;
getline(cin,cityNew);
cout << "Enter a previous city name: " << endl;
getline(cin,cityPrevious);
```

Delete city

```
//if city name not found
cout<<cityNameIn<<"not found"<<endl;
```

Clear network

```
cout<<"deleting "<<tmp->cityName<<endl;  //for all nodes in
network
```

Print menu

```
cout << "======Main Menu======" << endl;
cout << "1. Build Network" << endl;
cout << "2. Print Network Path" << endl;
cout << "3. Transmit Message Coast-To-Coast" << endl;
cout << "4. Add City" << endl;
cout << "5. Quit" << endl;
```

Quit

```
cout << "Goodbye!" << endl;
```