CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman/Zagrodzki/Zietz
Assignment 3
Due: Sunday, February 11 before 5pm

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.)

Build your own communications network

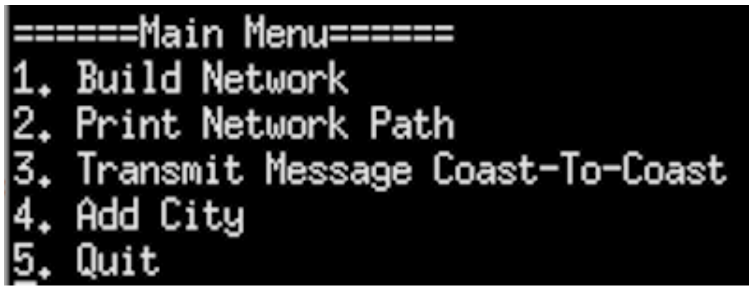
In this assignment, you're going to simulate a communications network using a linked list. Each node in your linked list will represent a city and you need to be able to send a message between nodes from one side of the country to the other. Your program also needs to provide the capability to update the network by adding cities and still be able to transmit the message.

(Note: I'll refer to the linked list as the network throughout this document.)

Include the following cities in your network:
Los Angeles
Phoenix
Denver
Dallas
St. Louis
Chicago
Atlanta
Washington, D.C.
New York
Boston

Implement each city as a struct with a name, a pointer connecting it to the next city in the network, and a place to store the message being sent. (You can assume 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
4. Add City
5. Quit
```

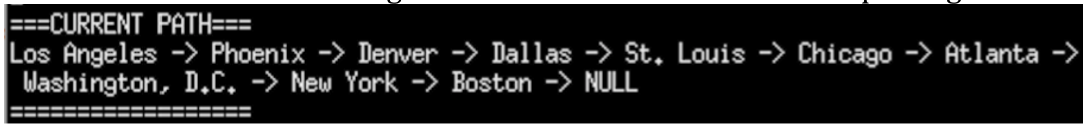
The user will select the number for the menu option and your program should respond accordingly to that number. Your menu options need to have the following functionality.

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:

Los Angeles -> Phoenix -> Denver -> Dallas -> St. Louis -> Chicago -> Atlanta -> Washington, D.C. -> New York -> Boston -> NULL

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

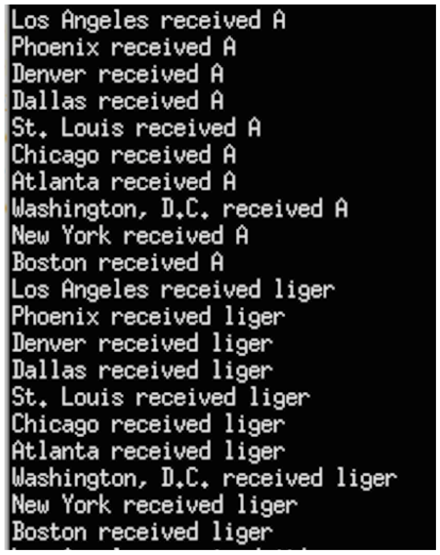


```
===CURRENT PATH===
Los Angeles -> Phoenix -> Denver -> Dallas -> St. Louis -> Chicago -> Atlanta ->
 Washington, D.C. -> New York -> Boston -> NULL
==================
```

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:** 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. Using the cities in this write-up, the message would go from Los Angeles to Boston, 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 I get after transmitting the first two words in the file:



```
Los Angeles received A
Phoenix received A
Denver received A
Dallas received A
St. Louis received A
Chicago received A
Atlanta received A
Washington, D.C. received A
New York received A
Boston received A
Los Angeles received liger
Phoenix received liger
Denver received liger
Dallas received liger
St. Louis received liger
Chicago received liger
Atlanta received liger
Washington, D.C. received liger
New York received liger
Boston received liger
```

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 Tucson after Phoenix in the network, then the first four cities in the network would be:

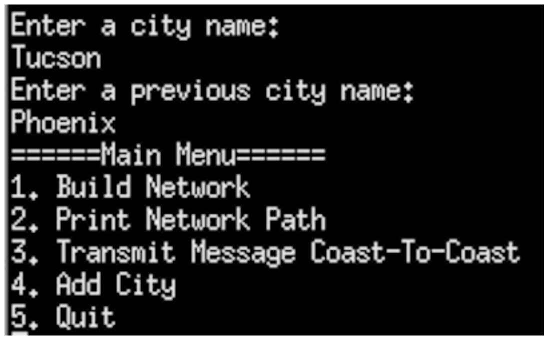
   Los Angeles -> Phoenix -> Tucson -> Denver…

   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 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. *(Note: for this week, we won't be testing that your code can add a new head node. However, you will need to implement this functionality in next week's assignment.)*

   If the user wants to add a new city to the tail of the network, e.g. the previous city name would be the current tail, then they could enter the name of the current tail city or "". Your code needs to handle the case where an empty string is passed as the previous city.

   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:
Tucson
Enter a previous city name:
Phoenix
======Main Menu======
1. Build Network
2. Print Network Path
3. Transmit Message Coast-To-Coast
4. Add City
5. Quit
```

5. **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.

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

Each of the menu options needs to be handled in a separate function. Included below is the definition for the city struct and the required function prototypes.

```
struct city{
        std::string cityName;
        std::string message;
        city *next;
        city(){}; // default constructor
         city(std::string initName, city *initNext, std::string
initMessage)
         {
                cityName = initName;
                next = initNext;
                message = initMessage;
        }
};
```

*city *buildNetwork()*
/*Create a linked list from the list of cities provided above and return the head of the list.
*/

*city *addCity(city *head, city *previous, string cityName)*
/*Add a new city to the linked list between the city *previous and the city that follows it in the network. The name of the new city is in the argument cityName. Return the head of the linked list.
*/

*void transmitMsg(city *head)*

/*Open the file messageLn.txt 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 printPath(city *head)*
/*Go through each city in the network starting at the head and print the name of the city.*/

Suggestions for completing this assignment
There are several components to this assignment that can be treated independently. My 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. If you're planning to implement the functionality to clear the network, to get a head start on next week, do that last.

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.

Also, start early.

Submitting Your Code:
Submit your assignment to CodeRunner on Moodle:
https://moodle.cs.colorado.edu/mod/quiz/view.php?id=21754

Go to the quiz like submission link labeled "Assignment 3 Submit" on Moodle. For each question, paste your code for the function asked in the input window provided. CodeRunner will give you feedback on errors in your code or incompatibilities with the environment and give you the output it is looking for.

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.

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 << "NULL" << endl;
cout << "==================" << endl;
```

Transmit Message

```
cout<<sender->cityName<<" received "<<sender->message<<endl;

//if network not built yet, head = NULL
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);
```

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;
```