CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman
Assignment 4
Due: Friday, February 13 before 3pm.

Buffered 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.

In many current communications networks, data buffers are used to temporarily store data as it is read in from an input device before being transferred to an output device. For example, videos downloaded from the Internet can be buffered before they are displayed onscreen to improve the quality of the picture you see. Buffers are implemented by storing the incoming data in a First In First Out queue to maintain the integrity of the order in which the data was received. We discussed queues in lecture on Friday and looked at a few examples of how to implement one using an array or a linked list.

Build your own (buffered) communications network

In this assignment, you're going to build on the linked list you built in Assignment 3 by converting it to a doubly linked list and adding a queue to serve as an input buffer for the message being read in. You will read data in from a text file then transmit it through the network of cities, and then back again once the message has reached the last city. Before transmitting the data, you will write it to a circular array queue for temporary storage, and then send it when the user selects "Dequeue" or "Send Entire Message" from the menu. In this assignment, you don't need to implement the add city and remove city features, but if they already exist in your code, you will not be marked down for having them.

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

Structuring your program

Your communications network, including the linked list and the queue, should all be included in one class. You are provided with a header file for a class-based implementation, called *CommunicationNetwork.h* on Moodle. Your class does not need to look exactly like the one provided, you are welcome to modify it to structure your implementation differently. In the header provided, the cities are implemented with a struct, and the array queue is a dynamically allocated array. The cities and the queue are private within the class. Your class needs to include public methods to enqueue and dequeue the data, print the queue, and build and print the network of cities. Yes, a more-sophisticated implementation would be to have separate classes for the queue and linked list. But, the important learning outcomes here are that you understand how to implement a queue, a doubly linked list, and a class. I'm confident that once you've done that, you could re-structure your code to use multiple classes if you chose to do so.

Each of the menu options presented to the user needs to be handled in a separate function. You are welcome to write additional helper functions to support those functions. Included below are suggested, but not required, function prototypes.

First, do some system setup

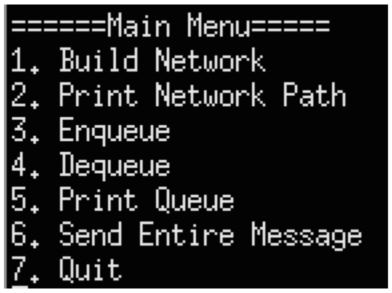
There is a file on Moodle called messageIn.txt that contains the message you will use for this assignment. When your program starts, you should read the text out of that file and store it locally in your code so that don't need to open and close that file multiple times throughout your program. Storing the file data will also make it easier to track which words have been sent to the queue.

Outside of the loop that controls the user's input to Quit the program, create an instance of the CommunicationNetwork class. Use a queue size of 10. Your CommunicationNetwork constructor should include the following settings:

```
CommunicationNetwork::CommunicationNetwork(int qs) {
     queueSize = qs;
     arrayQueue = new string[queueSize];
     queueHead = 0;
     queueTail = 0;
}
```

Next, 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 expected menu is shown here:



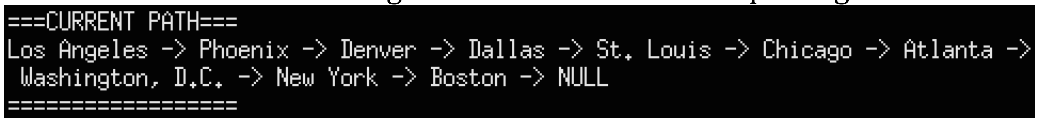
```
======Main Menu=====
1. Build Network
2. Print Network Path
3. Enqueue
4. Dequeue
5. Print Queue
6. Send Entire Message
7. Quit
```

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, a pointer to the previous city, and a message value, which will initially be an empty string. This option should be selected first to build the network. Once the network is built, you should print the name of each city in the network 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 COG is expecting:



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

(Note: We have removed the extra space before NULL that was needed to pass COG in Assignment 3.)

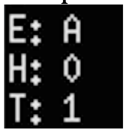
2. **Print Network Path:** This option prints out the linked list in order from beginning to end by following the next pointer for each city. It could be very useful to you when debugging your code. The format should be the same as the format in Build Network.
3. **Enqueue:** This option should enqueue the next word. For example, in your code to setup your setup described above, if you've read the file into an array called fileData, and fileData contains "A liger its pretty much my favorite animal", then

`Obj->enqueue(fileData[0])` would add "A" to the queue
`Obj->enqueue(fileData[1])` would add "liger" to the queue

In your main function, you need to keep track of which words have been added to the queue so that you can enqueue them in order. For example, if you are using a variable called *x* to track the index of the word, then you could increment *x* each time you call: Obj->`enqueue(fileData[x])`.

When you enqueue a word, your code should print the word and the head and tail indices after the word has been added to the queue in the following format:
E: <word>
H: <head index>
T: <tail index>

Here is the output that COG will expect after one Enqueue operation on the sample sentence "A liger is pretty much my favorite animal".

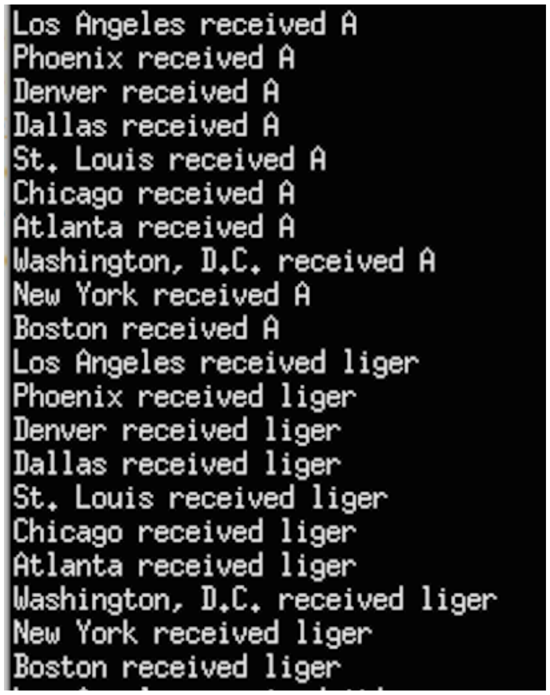


```
E: A
H: 0
T: 1
```

4. **Dequeue:** This option does a dequeue operation on the queue and transmits the word through the network and back again. The word should start in Los Angeles and go 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. Here is a screenshot of the output I get after transmitting the first two words in the file:



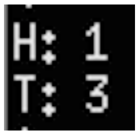
```
Los Angeles received A
Phoenix received A
Denver received A
Dallas received A
St. Louis received A
Chicago received A
Atlanta received A
Washington, D.C. received A
New York received A
Boston received A
Los Angeles received liger
Phoenix received liger
Denver received liger
Dallas received liger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

When the word is received at the other end, it should be sent back through the network to the starting city. Again, print that each city has received the word using the same format, and delete the word from the sender city.

After the message has been sent, output the head and tail indices for the queue in the following format:

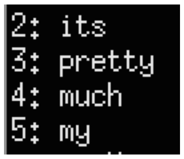
H: <head index>
T: <tail index>

Here is the output that COG will expect after three enqueue operations on the sample sentence and one dequeue operation:



```
H: 1
T: 3
```

5. **Print Queue:** This option will print all words in the queue, starting at the head and stopping at the tail with the index of where the word occurs in the queue. The queue should not be modified in any way. For example, if I do six enqueue operations and two dequeue operations, then printing the queue would display the following:



```
2: its
3: pretty
4: much
5: my
```

6. **Send Entire Message:** This option should send the entire message by repeatedly filling up the queue and sending the words in the queue through the network forward and backward until there are no remaining words to be sent. If there are already words in the queue from previous enqueue operations, then the words enqueued here should be added to the queue until the queue is full, nothing already in the queue should be overwritten. Once all words have been read into the queue, the contents of the queue should be sent through the network even if the queue isn't full.

7. **Quit:** This option allows the user to exit the program. You should also free all memory allocated at this time.

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.

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 updating the menu from the last assignment to include the requirements for this assignment. There is also some setup to do, such as creating an instance of the class and reading the text from the file. Next, work on the enqueue and dequeue functionality, testing that you can enqueue and dequeue words using the pseudocode from lecture. The wrap-around functionality is going to require some thought, tackle that next. Then, move to converting your linked list from assignment 3 to a double linked list that is contained within a class.

Also, start early.

Submitting Your Code:
Submit your assignment to the COG autograder:
https://web-cog.cs.colorado.edu/submit.html.

Login to COG using your identikey and password. Select the CSCI2270 - Hoenigman – HW #04 from the dropdown. Upload your file and click Submit. Zip your CommunicationNetwork.cpp, CommunicationNetwork.h, and Assignment4.cpp files together into one Assignment4.zip archive. **Your file needs to be named Assignment4.zip for the grading script to run.** COG will run its tests and display the results in the window below the Submit button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

In addition to submitting through COG, submit your .cpp file through Moodle using the Assignment 4 Submit link. 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.

If you do not get your assignment to run on COG, 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 COG, 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. There is a Peer Discussion Forum on our Moodle page that is a good place to post technical questions, such as how to iterate through a linked list. When you answer other students' questions on the forum, please do not post entire assignment solutions. The multi-course LAs 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 LAs.