CSCI 2275 – Programming and Data Structures

Instructor: Archana Anand

TA: Himanshu Gupta

Assignment 4

Due: Friday, October 18, before 11:59 pm.

Buffered communication between towers

In this assignment, you're going to build the city network you implemented in Assignment 4 by adding a queue to serve as an input buffer for the message being read in. You will read data in from a text file and store it in a queue, and then transmit it through the network of cities and back again. You will use a normal array queue for temporary storage, and then send the queue when the user selects "Dequeue" or "Empty Queue" 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

Implement each city as a struct with a name, a pointer connecting it to the next city in the network and the previous city in the network, and a place to store the message being sent like in Assignment 4. The array queue is a dynamically allocated array. The cities and the queue are private within the class you implement. Your class needs to include public methods to enqueue and dequeue the data, peek

the queue, and build and print the network of cities.

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.

Some System setup:

There is a file on Canvas called messageQueue.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. You can use any data structure you like to store the data.

Create an instance of the class.
The constructor of your class would look something like below

```
<class-name> {
int queue_size=10;
arrayQueue = new string[queue_size];
queueFront = -1;
queueRear = -1;
}
```

You may also need to initialize your linked list network in the constructor(you would have done this in last assignment)

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 first two menu items were implemented in the previous assignment. The expected menu is shown here:

```
=====Main Menu=====
1. Build Network
2. Print Network Path
3. Enqueue
4. Dequeue
5. Peek Queue
6. Empty Queue
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:

NULL <-> 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. Printing the path could be very useful to you when debugging your code. The format should be the same as the format in Build Network.

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

3. **Enqueue**

This option should enqueue the next word. For example, in your code to set up 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 queueFront and queueRear indices after the word has been added to the queue. The format is something like below
Enqueued: <word>
queueFront: <index>
queueRear: <index>

There is **no Queue Overflow** in this case. You need to keep track of the index queueRear. If queueRear reaches present queue_size-1 of the arrayQueue, you need to do an **array doubling as you did in Assignment 3**. Here you will copy everything from index queueFront to queueRear to the temp array. Delete the present arrayQueue using the "delete" keyword and assign temp to arrayQueue. Please see the implementation of the array doubling.

4. **Dequeue**:

This option does a peek and then dequeue operation on the queue and transmits the word through the network and back again. If the queue is empty, print the "Queue Underflow" error. For peek, you need to implement a method given in Point 5 below. 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. 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 queueFront and queueTail indices for the queue in the following format:
queueFront: <indices>
queueRear: <indices>

5. **Peek Queue**:

This prints and returns the word at queueFront. If the queue is empty, Print "Queue Underflow".

6. **Empty Queue**:

This should send all the messages present in the queue through the network forward and backward. If the queue is empty, Print "Queue Underflow". Else, this method should call the Dequeue method until the queueFront and the queueRear indices are the same. When they are same, set it back to -1.

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.

What to do if you have a question?
Post on Piazza.
Attend Instructor's/TA's office hours.

What to turn in?
Please Turn in a **well-commented Code** with name as
<first_name>_<last_name>.cpp

Also, start early.

Appendix A – cout statements

Enqueue:

```
cout<<"Enqueued: "<<word<<endl;
cout<<"queueFront: "<<queueFront<<endl;
cout<<"queueRear: "<<queueRear<<endl;

//if no more words to enqueue from the file read in
cout << "No more words to queue." << endl;
```

Dequeue

```
cout<< "Peeked Word: "<<word<<endl;
cout<<"queueFront: "<<queueFront<<endl;
cout<<"queueRear: "<<queueRear<<endl;
//dequeue before network built
cout<<"Build a network before attempting to transmit a message." << endl;
//Transmit Message see below
//if the queue is empty
cout << "Queue Underflow" << endl;
```

Peek:

```
cout<< "Peeked Word: "<<word<<endl;
//if the queue is empty
cout << "Queue Underflow" << endl;
```

Print path

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

Transmit Message:

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

Quit

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