CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman
Assignment 5
Due: Wednesday, February 24 before 3pm.

Build your own word queue

In this assignment, you're going ask the user to enter individual words and complete sentences and store those words in a queue. You will also implement the functionality to dequeue individual words and print the contents of the queue.

Structuring your program
Your queue functionality should all be included in one class, called Queue. You are provided with a header file for a class-based implementation, called *Queue.h* on Moodle. You will need to create a file called *Queue.cpp* that implements the class defined in the .h file. 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, queue data is stored in a dynamically allocated array. The size of the array is a parameter in the *Queue* constructor. Your class needs to include public methods to enqueue and dequeue the data and print the queue

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

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

Setting the *queueHead* and *queueTail* to 0 initializes the index in the *arrayQueue* for the head and tail positions.

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. Enqueue word
2. Dequeue word
3. Print queue
4. Enqueue sentence
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. **Enqueue:** This option should prompt the user for a word. If the queue is not full, the word should be added to the queue at the tail position. Otherwise, your program should print "Queue full" and not add the word to the queue.

   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>

   You can include these print statements in the enqueue method in your Queue class. Here is the output that COG will expect after the word "A" is enqueued to an empty queue.

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

   The head of the queue is still at index 0 and the tail of the queue is now at index 1.

2. **Dequeue:** This option does a dequeue operation on the queue and prints the head and tail indices and the word. If the queue is empty, your program should print "Queue empty".

   The head and tail indices for the queue and the word should be printed in the following format:

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

   Here is the output that COG will expect after dequeueing the word "A":

```
H: 1
T: 1
word: A
```

Notice that the head has moved to 1, the same index as the tail.

3. **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 there are four words in the queue and the head is at index 2 and the tail is at index 6, then the output would be the following:

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

(Note: the tail position is where the next word will be added and should not be included in the printing.)

4. **Enqueue sentence:** This option should prompt the user for a sequence of words and add all words to the queue in order until the queue is full. The words should all be separated by a space. For example, starting from an empty queue, if the user typed:

The brown fox jumped over the dog.

The contents of the queue would be:

```
0: The
1: brown
2: fox
3: jumped
4: over
5: the
6: dog.
```

For each word, use the enqueue method in your Queue class and print "Queue is full" for each word where the queue is full. For example, if the words

The brown fox

are added to an empty queue, then the output would look like:

```
sentence: The brown fox
E: The
H: 0
T: 1
E: brown
H: 0
T: 2
E: fox
H: 0
T: 3
```

There is room in the queue for all three words. However, if the sentence includes more words than the queue can store, only the words that fit in the queue should be added and the remaining words should be discarded. For example, if the words

jumped over the lazy dog sleeping in the hammock.

are entered as the sentence, then the output would be:

```
sentence: jumped over the lazy dog sleeping in the hammock.
E: jumped
H: 0
T: 4
E: over
H: 0
T: 5
E: the
H: 0
T: 6
E: lazy
H: 0
T: 7
E: dog
H: 0
T: 8
E: sleeping
H: 0
T: 9
E: in
H: 0
T: 0
Queue is full
Queue is full
```

Notice that “Queue is full” prints when the program tries to add “the” to the queue and “hammock.” to the queue because the queue is full.

5. **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 meet the requirements for this assignment. Next, work on the enqueue and dequeue functionality, testing that you can enqueue and dequeue individual words using the pseudocode from lecture and Chapter 7 in your book. The wrap-around functionality is going to require some thought, tackle that next. Finally, implement the functionality to read and enqueue a word sequence.

Also, start early.

Submitting Your Code:
Submit your assignment to the COG autograder:
https://web-cog-csci2270.cs.colorado.edu/submit.html.

Login to COG using your identikey and password. Select the CSCI2270 - Hoenigman – HW #05 from the dropdown. Upload your file and click Submit. Zip your Queue.cpp, Queue.h, and Assignment5.cpp files together into one Assignment5.zip archive. **Your file needs to be named Assignment5.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 5 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.

Appendix A – cout statements that COG expects

Enqueue

```
//prompt the user
cout<<"word: ";
//get input from user

//output after user enters a word and queue is not full
cout<<"E: "<<word<<endl;
cout<<"H: "<<queueHead<<endl;
cout<<"T: "<<queueTail<<endl;

//if the queue is full
cout << "Queue is full." << endl;
```

Dequeue

```
cout<<"H: "<<queueHead<<endl;
cout<<"T: "<<queueTail<<endl;
cout<<"word: "<<word<<endl;

//if the queue is empty
cout << "Queue is empty." << endl;
```

Print Queue

```
cout<<current<<": "<<arrayQueue[current]<<endl;
//where current  is the index
//if queue is empty
cout << "Empty" << endl;
```

Enqueue sentence

```
//prompt user
cout<<"sentence: ";
//get input from user
//use enqueue cout statements for each word added to the queue
```

Quit

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