CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman/Zagrodski/Zietz
Assignment 5
Due: Wednesday, February 28th before 5pm.

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 circular queue. You will also implement the functionality to dequeue individual words and print the contents of the circular queue.

Structuring your program

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

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. 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.

Apart from this *Queue.cpp* file, you will also need to create an *Assignment5.cpp* file that will create a Queue object and call functions on that object based on user input. The menu will be displayed in this file.

Each of the menu options presented to the user needs to be handled in a separate function. Be aware that any helper functions you implement must be defined outside of the class, otherwise you may run into issues when trying to get your code to run in CodeRunner. Included below are the required function prototypes.

void Queue::enqueue(std::string word)
/*Insert a new word into the circular queue, print out the added word and the index of the head and tail positions. If the queue is completely full, print "Queue is full."
*/

void Queue::dequeue()
/* Perform the dequeue operation on the circular queue and print out the indices of the head and tail positions. Also print the word that was dequeued. If the queue is empty, print "Queue is empty."
*/

void Queue::printQueue()
/* Print all of the elements in the queue in the specified format from Appendix A. If the queue is empty print "Empty".
*/

bool Queue::queueIsFull()
/* Returns true of false whether or not the queue is full.
*/

bool Queue::queueIsEmpty()
/* Returns true or false whether or not the queue is empty.
*/

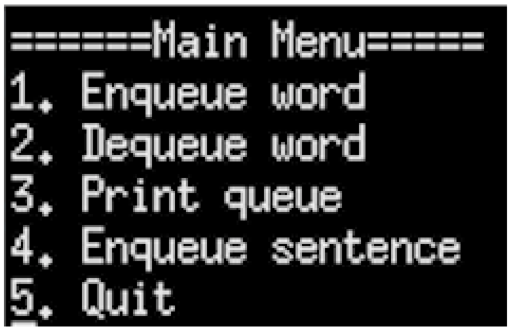
First, do some system setup
Outside of the loop that controls the user's input to Quit the program, create an instance of the Queue class. Use a queue size of 10. Your Queue constructor and destructor should include the following settings:

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

Queue::~Queue() {
   delete [] arrayQueue;
   arrayQueue = nullptr;
}
```

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
```

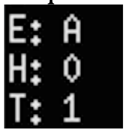
The user will select the number for the menu option and your program should respond accordingly to that number. Your menu options need to have the following functionality.

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. Because you will be implementing a circular queue, you will need to connect the last position of your queue array to the first position.

   Consider the scenario where you fill your queue completely and then dequeue the first three elements. Your code will need to allow the addition of more elements to this queue even though tail will be pointing to the last position in the array.

   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 CodeRunner will expect after the word "A" is enqueued to an empty queue.

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

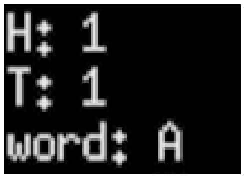
   The head of the queue is still at index 0 and the tail of the queue is now at index 1.

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>

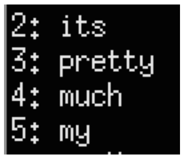
Here is the output that CodeRunner will expect after dequeueing the word "A":



```
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:



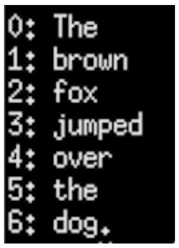
```
2: its
3: pretty
4: much
5: my
```

(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:

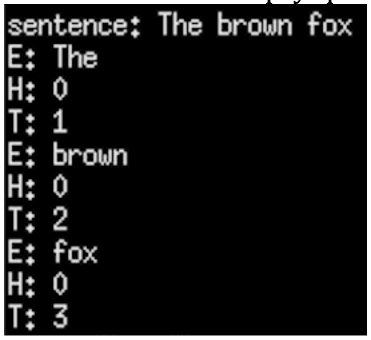


```
0: The
1: brown
2: fox
3: jumped
4: over
5: the
6: dog.
```

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:

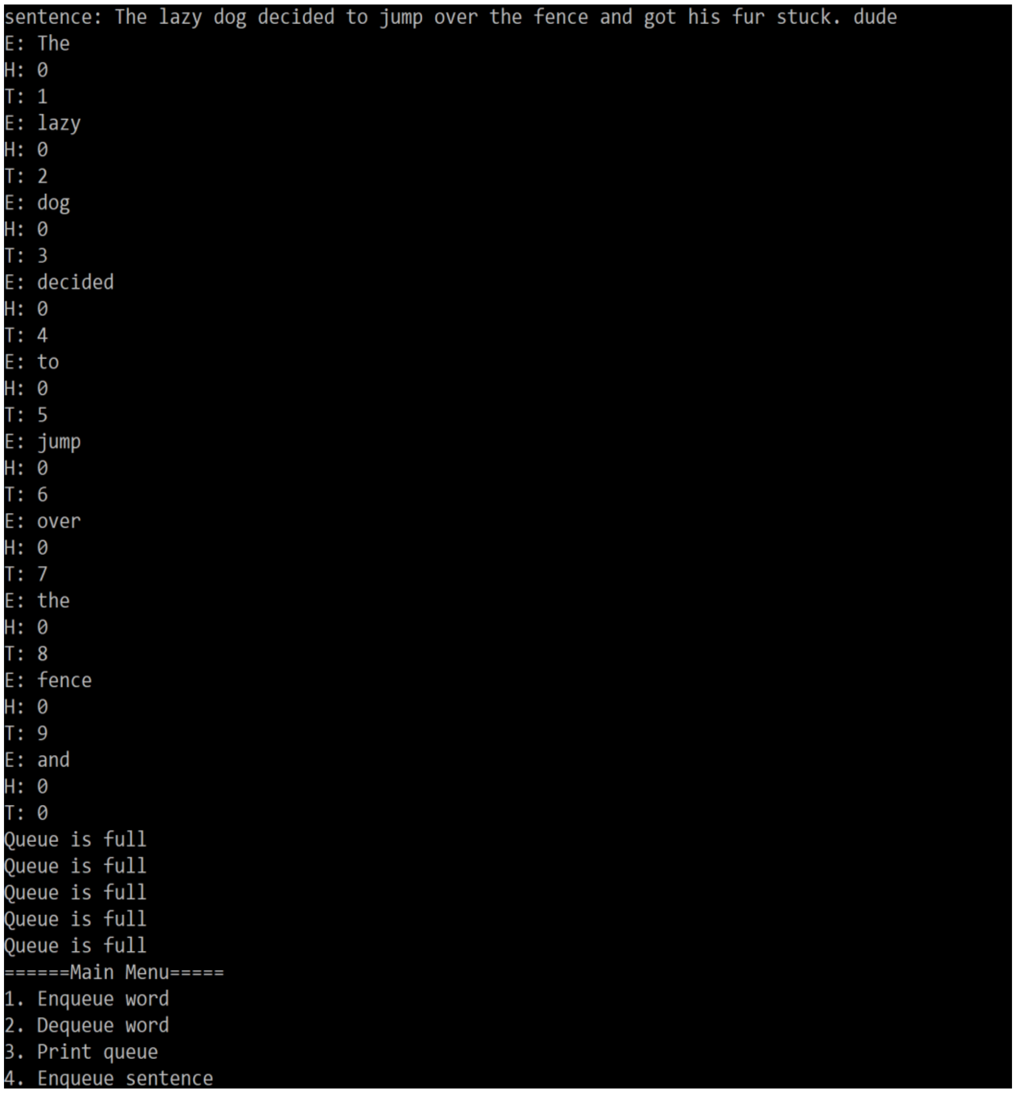


```
sentence: The brown fox
E: The
H: 0
T: 1
E: brown
H: 0
T: 2
E: fox
H: 0
T: 3
```

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: The lazy dog decided to jump over the fence and got his fur stuck. dude
E: The
H: 0
T: 1
E: lazy
H: 0
T: 2
E: dog
H: 0
T: 3
E: decided
H: 0
T: 4
E: to
H: 0
T: 5
E: jump
H: 0
T: 6
E: over
H: 0
T: 7
E: the
H: 0
T: 8
E: fence
H: 0
T: 9
E: and
H: 0
T: 0
Queue is full
Queue is full
Queue is full
Queue is full
Queue is full
======Main Menu=====
1. Enqueue word
2. Dequeue word
3. Print queue
4. Enqueue sentence
```

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. The destructor will be called for the queue object implicitly as the program is shutting down, freeing the allocated memory.

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 Assignment 5 Submission link on Moodle:
https://moodle.cs.colorado.edu/mod/quiz/view.php?id=22080

The CodeRunner quiz for this assignment will have a question for each one of your implemented class functions and a final main function test. If your code doesn't run correctly on CodeRunner, read the error messages carefully, correct the mistakes in your code, and try again. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

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. There is a Piazza discussion forum 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 CAs 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 Piazza or the CAs.

Appendix A – cout statements that CodeRunner 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
```

Quit

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