CSCI 2270 Data Structures and Algorithms Summer 2017
Instructor: Holler/MonteroQuesada
Assignment 4

Due Friday, June 30th, by 11am Assignment 4 Part A (Pseudocode)
Due Wednesday, July 5th, by 8am Assignment 4 Part B (Implementation)

# Build your own word queue

**Objectives:**

- Create, add, and delete, and manipulate a queue

**Background**

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 will need to work on the implementation of the class to be defined in your .hpp file. The 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 function prototypes.

**First, do some system setup**

Create an instance of the Queue class. The queue size should be taken as a command line argument (CLA) input.

Your Queue constructor should include the following settings:

```
Queue::Queue(int qs) {
     queueSize = qs;
     arrayQueue = new string[queueSize];
     queueHead = 0;
     queueTail = 0;
     queueCount = 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 moodle 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 is 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 moodle 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 1 and the tail is at index 5, then the output would be the following:

   ```
   1: its
   2: pretty
   3: much
   4: 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 (print 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:

```
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:

```
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.
* (max size of queue = 10 for this example)

5. Quit: This option allows the user to exit the program. You should also free all memory allocated at this time. It should print out:

```
Goodbye!
```

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. Our 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.

Although you need to submit a single code for the implementation and driver, while developing your code you should practice working with three separate files - Queue.hpp, Queue.cpp, and Assignment4.cpp. You need to write the Queue.hpp, the Queue.cpp and the Assignment4.cpp files. You can compile your code on the command-line using g++

g++ -std=c++11 Queue.cpp Assignment4.cpp -o Assignment4
and then run your program on the command-line using

./Assignment4 <size of queue>

Also, remember to start early.

### Pseudocode (20%)

For this assignment, the pseudocode portion will correlate to the functionalities 1 and 2, to Enqueue and Dequeue

### Implementation (80%)

Submit your Assignment 4 code by opening the link to the CodeRunner question and pasting your code on the text box. Make sure your code is commented enough to describe what it is doing. Include a comment block at the top with your name, assignment number, and course instructor.

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. Consider the TA office hours if you think that the discussion would be longer than 10min.

### 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

### Queue is empty

```
cout << "Queue is empty" << endl;
```

### Queue is full

```
cout << "Queue is full" << endl;
```

### Enqueue

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

### Dequeue

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

**Print Queue**

```
cout << index << ": " << arrayQueue[index] << endl;
```

**Print menu**

```
cout << "======Main Menu=====" << endl;
cout << "1. Enqueue word" << endl;
cout << "2. Dequeue word" << endl;
cout << "3. Print queue" << endl;
cout << "4. Enqueue sentence" << endl;
cout << "5. Quit" << endl;
```

**Quit**

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