CMP 210 – Data Structures and Algorithms
Fall 2019
LAB-4

Issue Date: Oct 17, 2019

ALERT!

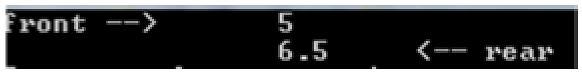
1. The objective of this lab is understanding and finding FIFO behavior in different problems.
2. This is an individual lab, you are strictly **NOT** allowed to discuss your solution with fellow colleagues, even not allowed asking how is he/she is doing, it may result in negative marking. You can **ONLY** discuss with your TAs or with me.
3. Beware of memory leaks and dangling pointers.
4. Pay attention to **GOOD coding conventions** e.g.
   - Proper indentation.
   - Meaning variable and function names.
   - Use camelCase naming convention
   - Use meaningful prompt lines/labels for all input/output
5. **Anyone caught in act of plagiarism would be awarded an "F" grade in this Lab.**

Task 01: [10 Marks]

Implement queue ADT. In addition to standard queue functionality, implement a *showStructure* function that should print the queue with its front and rear pointing to its correct locations on the console.

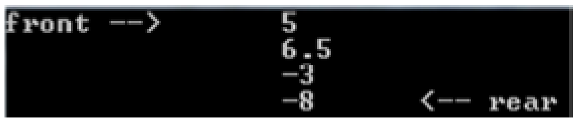
Sample Run:

```
queue.Enqueue(5.0);
queue.Enqueue(6.5);
queue.showStructure();
```



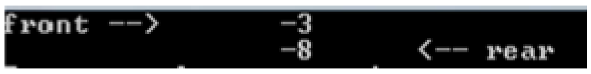
```
front -->          5
                   6.5        <-- rear
```

```
queue.Enqueue(-3.0);
queue.Enqueue(-8.0);
queue.showStructure();
```



```
front -->          5
                   6.5
                   -3
                   -8         <-- rear
```

```
queue.Dequeue( );
queue.Dequeue( );
queue.showStructure();
```



```
front -->          -3
                   -8         <-- rear
```

Task 02: [10 Marks]

A double-ended queue *(deque)* is like a queue, except that access is allowed at both ends. Much of its functionality can be derived from the Queue class. Rather than the terms *enqueue* and *dequeuer*, the terms used are *addFront*, *addRear*, *removeFront*, and *removeRear*. You should privately derive it from the simple Queue that you implemented in Task 01. Deque implementation is relatively simple using private inheritance.