CSC 228 Data Structures and Algorithms, Fall 2019
Instructor: Dr. Natarajan Meghanathan

Quiz 3: Insertion and Deletion of Data at an Arbitrary Index in a Queue

Due by: Oct. 22nd, 11.59 PM

In this quiz, you will develop and implement algorithms to insert at an arbitrary index of a queue as well as delete the data at an arbitrary index of a queue. The index of an element (data node) in the queue is the position of the element (data node) starting from the front (first data node) of the queue.

For example, consider a queue as shown below:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Data	34	21	90	45	87	22	43	55	81	98

where 34 is the data at the front of the queue (the data node for 34 is next to the head node) and 98 is the data at the end of the queue (the data node for 98 is previous to the tail node)

A call to the function to insert a new data (say, 40) at index 4 would result in the contents of the queue ordered as shown below:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Data	34	21	90	45	**40**	87	22	43	55	81	98

A call to the function to delete data at index 7 on the above modified queue would result in the contents of the queue ordered as shown below (node that the data 43 at index 7 above is no longer available):

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Data	34	21	90	45	40	87	22	55	81	98

You are provided the code for the implementation of a Queue ADT using a doubly linked list. The code also has two auxiliary functions (*getQueueLength* and *Print*) to get the length of a queue and print the contents of the queue (starting from the data node next node the head node).

You are supposed to only use the enqueue and dequeue functions of the Queue class and the auxiliary function getQueueLength to implement the other two auxiliary functions *QueueInsertAtIndex*(Queue queue, int insertIndex, int insertData) and *QueueDeleteAtIndex*(Queue queue, int deleteIndex) that are currently blank in the code provided. Note that you could use some temporary variables inside these functions, but the space complexity should be $\Theta(1)$; i.e., the amount of additional space used should not grow with the queue size.

You need not modify the main function; it already has the code to test run your implementations. To demonstrate that your functions work as required, test run your code for a queue size of 10 with the range of values for the integer being (1... 100).
Run for the following test cases:

(i) Insert at index 4 and delete at index 7
(ii) Insert at index 0 and delete at index 9
(iii) Insert at index 9 and delete at index 0
(iv) Insert at index 0 and delete at index 10
(v) Insert at index 9 and delete at index 1

Submission:
(1 - 80 pts) Submit a separate .cpp file containing the completed version of the startup code provided, including the implementations of the *QueueInsertAtIndex* and *QueueDeleteAtIndex* functions.
(2 - 20 pts) Test run your code for a queue size of 10 with the range of values for the integer being (1... 100). Submit a PDF file that contains the screenshots of the output of your code for the test cases (i)-(v) listed above.