# Data Structures and Algorithms I: Queues & Deques

Answer questions A3,4-6,7; B1,3, and questions C4.

## Part A: Theory

The first set of questions (1-3) focus on dry runs for array and link based implementations of the Queue ADT. The latter set of questions deals with the pseudo code and dry runs for the Link-based implementation of the Deque ADT. Remember to check your answers to 4-6 with a demonstrator before going any further.

1) Show how the state of both an array-based and a link-based Queue changes after each of the following operations:

   *enqueue(15), enqueue(20), enqueue(5), dequeue(),enqueue(17), enqueue(12), dequeue(),enqueue(7), enqueue(13), enqueue(2)*

   After the last operation, work out the sum of the values stored in the Queue and the sum of the values output from the Queue.

2) Show how the state of both an array-based and a link-based Queue changes after each of the following operations:

   *enqueue('c'), enqueue('a'), enqueue('t'), dequeue(),enqueue('h'), enqueue('o'), dequeue(),enqueue('u'), enqueue('s'), enqueue(),enqueue('e')*

   After the last operation, work out the word held in the Queue and the word that was output via the dequeue operations.

3) Show how the state of both an array-based and a link-based Queue changes after each of the following operations:

   *enqueue('Ireland'), dequeue(),enqueue('England'), dequeue(), enqueue('Wales'), dequeue(), enqueue('Scotland'), dequeue(), enqueue('France'), enqueue('Germany')*

   After the last operation, list the countries that were dequeued from the Queue and the countries held in the Queue.

4) Write out the pseudo code for the insertLast(o) operation of the Deque ADT. The insertFirst(o) pseudo code is given below as an example of a similar operation.

*Algorithm insertFirst(o):*
*Input: an object o*
*Output: N/A*

```
node ← new Node(o)
node.next ← front
if front = null then
        rear ← node
else
        front.prev ← node
front ← node
size ← size + 1
```

*NOTE: once you have written a first version of your pseudo code, test it using two dry runs: (1) inserting into an empty Deque, and (2) inserting into a non-empty Deque.*

5) Write out the pseudo code for the removeLast() and removeFirst() operations of the Deque ADT. As with question 4, for each removal operation consider 2 cases: (1) a Deque containing 1 item and (2) a Deque containing more than 1 item.

6) Write out the pseudo code for the isEmpty(), size(), front() and rear() operations of the Deque ADT.

7) Show how the state of a link-based Deque changes after each of the following operations:

*insertFirst('Ireland'), removeLast(),insertLast('England'), removeFirst(), insertLast('Wales'), insertFirst('Scotland'), insertLast('France'), removeFirst(), removeLast(), insertLast('Germany')*

After the last operation, list the countries that were removed from the Deque and the countries held in the Deque.

## Part B: Queue & Deque Implementations

In this second part of the worksheet, you will be expected to write Java code that corresponds to the two Queue and one Deque implementation strategies discussed in the course. Much of this work will require that you transform pseudo code given in the course into Java code.

1) **ArrayQueue**: Create a new Java class called `ArrayQueue` that realises the array-based Queue implementation strategy. As with the Stack implementations (Assignment 4), remember to implement a toString() method to help you debug / visualise the operation of the class.

2) **LinkedQueue**: Create a new Java class called `LinkedQueue` that realises the link-based Queue implementation strategy. Again, remember to implement a toString() method to help you debug / visualise the operation of the class.

3) **LinkedDeque**: Create a new Java class called `LinkedDeque` that realises the link-based Deque implementation strategy. Your implementation should be based on the pseudo code you completed in questions A4-6. Again, remember to implement a toString() method to help you debug / visualise the operation of the class.

## Part C: Evaluation

The final part of this worksheet requires that you test your implementations against the dry runs that you performed in part A. For each Queue question, you should write 2 program start classes: one for each implementation strategy; and for each Deque question, you should write 1 program start class. As a naming convention you should use the following: PCQxA.java for the ArrayQueue implementation strategy and PCQxL.java for the LinkedQueue / LinkedDeque implementation strategys. x should be the question number.

NOTES:

- Because your Queue/Deque implementations work with objects, you cannot insert primitive data types directly onto the Queue/Deque. Instead, you have to convert them to their object equivalents. For example, if you had an integer value , `15`, then the object equivalent of this is `new Integer(15)`. Similarly, the char value `'a'` has an object equivalent `new Character('a')`.
- When you remove objects that correspond to primitive data types from a Queue/Deque, you need to convert them back to their primitive data types before you can use them in calculations. To do this, you use a XXXValue() method that is associated with the object equivalent. For example, to convert an Integer object referenced by the variable, `intObj`, into an integer value, you do the following: `intObj.intValue()`,. Similarly, to convert a Character object, `ch`, into a char, you do the following `ch.charValue()`.

1) Write two main methods that correspond to the series of enqueue and dequeue operations outlined in question A1. You do not need to work out the totals in the program, only print out the state after each operation (also print out the state before the first operation).

2) Write two main methods that correspond to the series of enqueue and dequeue operations outlined in question A2. You do not need to work out the words in the program, only print out the state after each operation (also print out the state before the first operation).

3) Write two main methods that correspond to the series of enqueue and dequeue operations outlined in question A3. You do not need to print out the countries in the program, only print out the state after each operation (also print out the state before the first operation).

4) Write two main methods that correspond to the series of insertion and removal operations outlined in question A7.