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

3A. 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')

--- Array Based // need to know the size before hand // fixed size !!! => {1, 2, _, _, _, _} of 4 null (garbage unused)

state of stack (input)	command	num of elements	output
[null]	enqueue('Ireland'),	1	[‘Ireland']
['Ireland']	dequeue()	0	[null]
[null]	enqueue('England')	1	['England']
['England']	dequeue()	0	[null]
[null]	enqueue('Wales')	1	['Wales']
['Wales']	dequeue()	0	[null]
[null]	enqueue('Scotland')	1	['Scotland']
['Scotland']	dequeue()	0	[null]
[null]	enqueue('France')	1	['France']
['France']	enqueue('Germany')	2	['France', 'Germany']

--- Linked List Based // NO UNUSED MEMORY => extra memory for pointer

state of stack (input)	command	Rear	Front	size	output
[null]	enqueue('Ireland')	Ireland	Ireland	1	
---	---	---	---	---	---
['Ireland']	dequeue()	null	null	0	['Ireland']
---	---	---	---	---	---
[null]	enqueue('England')	England	England	1	
---	---	---	---	---	---
['England']	dequeue()	null	null	0	['England']
---	---	---	---	---	---
[null]	enqueue('Wales')	Wales	Wales	1	
---	---	---	---	---	---
['Wales']	dequeue()	null	null	0	['Wales']
---	---	---	---	---	---
[null]	enqueue('Scotland')	Scotland	Scotland	1	
---	---	---	---	---	---
['Scotland']	dequeue()	null	null	0	['Scotland']
---	---	---	---	---	---
[null]	enqueue('France')	France	France	1	
---	---	---	---	---	---
['France']	enqueue('Germany')	Germany	France	2	

4A. 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 NOTE: once you have written a first version of your pseudo code, test it using two dry runs:

(1) inserting into an empty Deque,

```
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
```

(2) inserting into a non-empty Deque.

```
Algorithm insertLast(o):
  Input: an object o
  Output: N/A
  node <- new Node(o)
  node.prev <- rear
  if  rear = null then
    front <- node
  else
    rear.next <- node
  rear <- node
  size <- size + 1
```

5A. 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

```
Algorithm removeFirst():
  Input: N/A
  Output: N/A
  if size = 1 then
    front <- null
    rear <- null
    size <- 0
  else if size > 1 then
    front <- front.next
    front.prev <- null
    size <- size – 1
```

(2) a Deque containing more than 1 item.

```
Algorithm removeLast():
  Input: N/A
  Output: N/A
  if size = 1 then
    front <- null
    rear <- null
    size <- 0
  else if size > 1 then
    rear <- rear.prev
    rear.next <- null
    size <- size - 1
```

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

```
  // isEmpty
Algorithm isEmpty():
  Input: N/A
  Output: N/A
  if size = 0 then
    return true
  else
    return false

  // size
Algorithm size():
  Input: N/A
  Output: an integer
  return size

  // front
Algorithm front():
  Input: N/A
  Output: an object
  return front

  // rear
Algorithm rear():
  Input: N/A
  Output: an object
  return re
```

7A. 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

state of stack (input)	command	num of elements	Rear	Front	output
[null]	insertFirst('Ireland'),	1	Ireland	Ireland	['Ireland']
['Ireland']	removeLast()	0	null	null	[null]
[null]	insertLast('England')	1	England	England	['England']
['England']	removeFirst()	0	null	null	[null]
[null]	insertLast('Wales')	1	Wales	Wales	['Wales']
['Wales']	insertFirst('Scotland')	2	Wales	Scotland	['Scotland','Wales']
['Scotland','Wales']	insertLast('France')	3	France	Scotland	['Scotland','Wales','France']
['Scotland','Wales','France']	removeFirst()	2	France	Wales	['Wales','France']
['Wales','France']	removeLast()	1	Wales	Wales	['Wales']
['Wales']	insertLast('Germany')	2	Germany	Wales	['Wales','Germany']

Countries removed from Deque:
Ireland
England
Scotland
France

Countries held in Deque:
Wales
Germany

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

// in file folder

3B. LinkedDeque: Create a new Java class called LinkedDeque that realizes 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 / visualize the operation of the class.

// in file folder

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

// in file folder