Lab 6

Circular array implementation of the Queue ADT

1. Create a new directory for this lab and call it Lab6.

2. Define a generic class $AQueue<T>$ to represent a queue implemented as a circular array.

$AQueue$ has 4 private instance variables: an array $arr$ for $T$ type elements and 3 int type variables: front and end for indexes of the first and last elements, and count for the actual number of elements in the queue. **Important note:** in this implementation the checking of the fullness and emptiness of the queue is done by checking the count and NOT the front and end indexes.

$AQueue$ has a static public nested class $MyException$ with two constructors.

$AQueue$ has one constructor with one parameter for the size of the array. This constructor creates an empty queue – it allocates memory for the array $arr$, and initializes the three instance variables. Note that with the above agreement, we can initialize front to be 0 (this will shorten the work of the enqueue which will no longer need to separately deal with the ‘empty queue’ case); as for the end and count variables, they will be initialized by -1 and 0 respectively.

$AQueue$ has the following public methods:
- **enqueue**: no return value; $T$ type parameter is added to the end of the queue and count is incremented (end is incremented, wrapping around edge of the array if needed, and the element is added at the end index). **Important**: if the queue is full, you must increase the size of the array holding the queue. Thus, BEFORE you add a new item, you need to check if the queue is full or not (via count variable) and if it is, you need to create a new array twice the size of the original array and copy the queue content from the old array to the new one (elements at indexes front..end in the old array are copied into the indexes 0..count-1 in the new array). Adjustments need to be done to the front and end indexes (they become 0 and count-1 respectively)
- **dequeue**: returns a $T$ type value; the front element is removed from the queue (the element at front index is returned and the front is incremented wrapping around the edge of the array if needed), and the count is decremented. **Note**: NO need to reset the front and end if the queue gets empty (it is an extra check). A $MyException$ type exception is thrown if the queue is empty (checking done via count).
- **isEmpty**: no parameters; returns a boolean value – true if the queue is empty and false otherwise (should check the count – if the queue is empty, the value of count is 0).

Agreement: For learning/educational purposes we will include a printArray method in our Queue ADT to be able to see the content of the array and make sure the circular array implementation is done correctly. In the printout it will be less confusing if the unoccupied cells will not show garbage (previous elements of the queue that have been deleted). Therefore, let’s agree that when an element is dequeued, it is replaced with the null value and only then the front index is incremented.

- **printArray**: (this is for our testing purposes only): no return value. Prints the content of the whole array on the screen, i.e. the content of all cells from index 0 till $arr.length-1$ (values need to be printed on the same line separated by spaces).

3. Define an application class $AQueueDriver$ to test the functionality of your $AQueue$ class:

This driver is identical to the $LQueueDriver$ from Lab3 with the following differences:
- you need to define an object of $AQueue$ class instead of $LQueue$.
- when creating the $AQueue$ object, you need give a parameter value for the size of the queue (empty queue of the requested size will be created). For easy testing give 5 for the size – when we enter more values, the queue will be resized.
- You need to add a printArray option in the menu and another case segment in switch for its execution.

Attention: be sure to use the a,d,e,p,q letters for menu’s add, delete, isEmpty, print, quit options respectively.

4. Write another application class $AQueueClient$ to make use of the $AQueue$ class.

Problem description: values are entered from the keyboard until Ctrl+D is pressed and then ONLY numeric values are output on the screen (on one line, separated by spaces) in the order they were inputted. **Hint**: define a queue object to hold numeric values (Float type objects) initializing the size of the queue to be some integer value (for easy testing give 5 for size – the array will be resized later if needed). Then start inputting – if the value is numeric (this means it can be parsed into float) add it to the queue, otherwise ignore it (read it out of the way). When the input is finished, empty the queue and output all dequeued elements on the screen.

5. Edit, compile and thoroughly test BOTH $AQueueDriver$ and $AQueueClient$ programs.