German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Data Structures and Algorithms, Winter term 2020
Practice Assignment 5

Exercise 5-1 Reverse Queue

Write a method `reverse` that takes as a parameter a queue containing a sequence of objects and reverse it. You should implement the method using only queues and stacks.

```
public class Stack {

public Stack(int s);
public void push(object k);
public Object pop();
public int size();
public int top();
public boolean isEmpty();
public boolean isFull();
}

public class Queue {

public Queue(int s);
public void enqueue(Object k);
public Object dequeue();
public int size();
public int peek();
public boolean isEmpty();
public boolean isFull();
}
```

Exercise 5-2 Mirror using Queues

Write a method `mirror` that takes as a parameter a queue containing a sequence of integers and returns a queue having the mirror image of this sequence. The mirror image of a sequence contains the original sequence in reverse order followed by the original sequence. For example, if the original sequence is:

```
1, 8, 15, 7, 2
```

the mirror image is:

```
2, 7, 15, 8, 1, 1, 8, 15, 7, 2
```

Notice that the mirror-image has twice as many elements as the original sequence because it contains both the original sequence and the reversed sequence. Also notice that the reversed sequence comes first. You are allowed to destroy the original queue.

Only use queues!

Exercise 5-3 Prefix Evaluation

A **pefix expression** is one where the operators are written before their operands. For example, 1 + 2 is written + 1 2. You can take a look here: http://scanftree.com/Data_Structure/infix-to-prefix to get an idea about infix to prefix conversion.
Write a program that allows for the evaluation of prefix expressions using a static method int evaluate (String exp). Your evaluate method will pass a String representing a prefix expression containing **space-separated-elements**, and it is required to evaluate the expression and return the resulting value. Use a queue of String objects to solve this problem, given the algorithm mentioned below:

WHILE the queue has more than one element DO

- Inspect the front element
- If it is an **operator followed by its two operands**, insert the value to the rear of the queue and remove operator and operands from the front of the queue.
- If it is an **operator not followed by two operands**, remove it from the front and copy it to the rear.
- If it is an **operand**, remove it from the front and copy it to the rear.

For example, you have the following Expression:

```
Prefix Expression: + 10 * 2 3
1:                   10 * 2 3 +
2:                      * 2 3 + 10
3:                            + 10 6
4:                                 16
```

Hint: You may find it useful to use the String method split.

Exercise 5-4 Queue using Stacks

Implement the Queue ADT using only stacks. Implement the constructor and all the basic methods for queues.

Exercise 5-5 Shift the Zeros

Write a static method `shiftZeroes` which takes an instance of the `Queue` class. The method should take all the zeroes in a queue and place them at the back. For instance, if we have the following instance of a `Queue q` with the sequence

```
[5, 0, 1, 4, 3, 0, 0, 6, 1, 0]
```

we could call `shiftZeroes(q)`, and the sequence would be

```
[5, 1, 4, 3, 6, 1, 0, 0, 0, 0]
```

after the call. **Note** that the order of the non-zero elements stays the same.

The only objects you may use to solve this problem are instances of the `Stack` and `Queue` classes. Assume the classes have the usual methods (Queue has `enqueue`, `dequeue`, `peek`, `size` and `isEmpty`; Stack has `pop`, `peek`, `push`, `size` and `isEmpty`).

Exercise 5-6 Every Third Element

Write a method that takes an integer queue `q` as an argument, and returns a queue containing (in reverse) every third element of `q` starting with the last element.

For example,

- if `q` contains `{a, b, c, d, e}` then a queue containing `{e, b}` should be returned. **Note:** for simplicity reasons, the queue in this example is a queue of characters.
- if `q` contains `{a, b, c, d, e, f, g, h, i, j}` then a queue containing `{j, g, d, a}` should be returned.

Only use queues and stacks!

Exercise 5-7 Anagrams

An anagram is a word formed by reordering the letters of another word, using all letters of the original word exactly once. For example, the word `post` is an anagram of `stop`. Your task is to implement a method that takes as input two strings and returns either `true` meaning that they are anagrams or `false` meaning that they are not. You should use the following technique to obtain your results:

- for each of the two input strings, enqueue all the characters forming the string to an initially empty queue,
- now that you have obtained two queues, each representing a string, iterate through one of the queues by dequeuing a character from its front and enqueuing it again at the rear, and repeating this process as many times as needed,
- if the front character of both queues is the same, dequeue the front character of each of the queues,
- in case you have iterated through the whole queue and the front characters were never equal then you have determined that the strings are not anagrams,
- finally if you end up with two empty queues, then you have determined that the strings are anagrams.

Use the following header for your method: `static boolean anagrams(String a, String b)`

Exercise 5-8 Palindrome using Stacks and Queues

You are required to create a class that can detect palindromes. A palindrome is a word or sentence that spells the same forwards as backwards.
There are many ways to detect if a phrase is a palindrome. The methods that you will use in this assignment is by using a stack and a queue. This works in the following way: Push half of the characters onto the stack and enqueue the second half in the queue, then compare the characters as you pop and dequeue from the stacks and queue.
Create a class called `Palindrome` that has a single method called `isPalindrome(String...)`. Inside this method create a `Stack` and a `Queue`, and determine if the given `String` is a palindrome.

Exercise 5-9 Dequeue

Chapter 4 of the textbook (page 143) describes a doubly-ended queue, which is often called a **deque**.

In class you have seen and implemented both stack (LIFO) and queue (FIFO) abstract data types. A deque (pronounced "deck") is an abstract data type that combines what a stack can do, and what a queue can do. Thus, you are allowed to add items to either end of a deque, and inspect/remove them from either end.

Implement the `Deque` ADT using an array used in a circular fashion (as was done for the queue ADT). The Deque class will contain the following methods:

- `public void insertFirst(Object o)`: inserts an object at the beginning of the Deque.
- `public void insertLast(Object o)`: inserts an object at the end of the Deque.
- `public Object removeFirst()`: removes the object at the begining of the Deque.
- `public Object removeLast()`: removes the object at the end of the Deque.

- `public Object peekFront()`: returns the front object of the Deque, without removing it.
- `public Object peekRear()`: returns the last object of the Deque, without removing it.
- `public boolean isEmpty()`: returns true if the Deque is empty, false otherwise.
- `public boolean isFull()`: returns true if the Deque is full, false otherwise.
- `public int size()`: returns the number of elements currently present in the Deque.

Be sure to test your class implementation thoroughly. Show output demonstrating that each method of your implementation is correct.