National University of Singapore
School of Computing
IT5003: Data Structure and Algorithm
Semester I, 2019/2020

Tutorial x Lab 4
Stack and Queue

General idea: Tutorial type questions will be listed first in this document. Some tutorial question will then be implemented in the lab portion (later part of this document). Lab exercise cases to be submitted on Coursemology will be marked [SUBMISSION] at the front.

1. [Using Stack ADT] Suppose you are given a bunch of 1's and 0's, and you are asked to check whether there are more 1's than 0's (or vice versa). However, you are not allowed to **count** the numbers of 1's and 0's and can only use **a single stack** to solve the problem. Give either pseudo-code or actual Python code for the answer. You can use the following function prototype as a guide:

```
def whichIsMore( input ):
"""
[input] is a string, where each character is '1' or '0'.
Return:
= 1 if there are more 1's than 0's
= 0 if there are more 0's than 1's
= -1 if there are equal number of 1's and 0's

Restriction: You are not allowed to count the numbers of 1's and
0's.
"""
```

   Extra Challenge: Can you modify the answer so that the difference in number of 1's and 0's is computed as well? e.g. 1011101 gives 3 as there are 3 more 1's than 0's.

2. [Stack ADT and arithmetic expression] An arithmetic expression can be expressed in three notations: *prefix*, *infix*, and *postfix*. The infix notation is the one most of us are familiar with, where **each binary operator appears between its two operands**. For example, "2 * (3 + 4)" is an **infix** expression. On the other hand, in the **postfix** notation, each binary operator appears behind its two operands. For example, "2 3 4 + *" is the postfix expression equivalent to the example infix expression.

   For this question, we are going to learn how to evaluate a postfix expression. Below is the evaluation algorithm, expressed as pseudo code:

```
Create an empty stack
for each item of the expression,
      if it is an operand,       #i.e. numbers
          push it on the stack
      if it is an operator,      #i.e. +, -, * etc
          pop arguments from stack;
          perform the operation
          push the result onto the stack
```

   The last result on the stack is the result of the entire postfix expression.

a. Trace the algorithm for "2 3 4 + *".

b. Implement / Sketch a Python implementation of this algorithm. For simplicity, we assume:
   - The input to the function is a Python list, where each "item" (formally known as a **token**) of the postfix expression is stored as separate strings. For example, ["12", "34", "+"] is the postfix expression for "12 + 34".
   - We only support addition "+" and subtraction "-" only.

   The Python function has the following definition:

```
def evaluatePostfix( eList ):
"""
[eList] is a List of strings. Each strings is either and
operand, e.g. "123"; or an operator, e.g. "+".

This function returns the result (a single integer value) of
evaluating the postfix expression.
"""
```

3. [Using Queue ADT] The FIFO property of Queue ADT is very useful in simulating real world queue. Let us take the consultation queue in a clinic as example. If we assume that time is represented as an integer (i.e. Time = 0, 1, 2… etc), then each patient can be represented by using two integer values, the *arrival time* (when he/she goes intothe waiting queue) and the *consultation time* (how long is the consultation when the doctor sees the patient). Given the sequence of patients information, write a function (Psuedo-Code is acceptable) to calculate:

    a. Average Queue Length (number of persons in the queue)

    b. Average Waiting Time (average time units a person stays in the queue)

For simplicity, each patient is represented as a 3-tupe (name, arrival time, consultation time), e.g. ("Jane Doe", 2, 5) means Ms.Jane Doe arrives at the clinic at time 2 and will need 5 time unit for consultation.

The function prototype:

```
def simulateQueue( patients ):
"""
[patients] is a list of patients stored in order of arrival time.

This function simulate a clinic with 1 doctor and print out
events happening, e.g. patients arriving, queueing up,
consulating, leaving etc.

This function also returns a 2-tuple (avg. queue length, avg.
waiting time) at the end.
"""
```

Sample Input (4 patients):

```
Amanda 1 3 //Format: Name, Arrival Time, Consultation Time
Bernard 2 1
Christa 5 2
Donald 5 5
```

Sample Output:

```
* Time 1 *
Amanda added to queue
Amanda consulting with doctor. Wait time = 0
Queue length = 0

* Time 2 *
Bernard added to queue
Queue length = 1
```

```
* Time 3 *
Queue length = 1

* Time 4 *
Amanda left clinic
Bernard consulting with doctor. Wait time = 2
Queue length = 0

* Time 5 *
Bernard left clinic
Christa added to queue
Donald added to queue
Christa consulting with doctor. Wait time = 0
Queue length = 1

* Time 6 *
Queue length = 1

* Time 7 *
Christa left clinic
Donald consulting with doctor. Wait time = 2
Queue length = 0

* Time 8 to 11 not shown as they are very similar *

* Time 12 *
Donald left clinic
Queue length = 0

** Statistics **
Average Waiting Time = 1.00
Average Queue Length = 0.33
```

Extra: This technique is known as *Discrete Event Simulation*, where time flows in discrete unit (1, 2, 3, …) and events may only happen at each time unit. You can think about how to simulate multiple queues with one doctor, or one queue with multiple doctors. This can help to decide the optimum number of queues and doctors….

4. [Past midterm from similar modules] Given the following algorithm:

   Step I. Read one input **X** from user
   Step II. Either:
      i. Store **X** in a data structure **S**, **OR**
      ii. Output **X** immediately
   Step III. Output zero or more items in **S**
   Step IV. Goto ***Step I***

   If user enters the number 1, 2, 3, …. 10, and **S** is a **stack**, which of the following output sequences is/are possible?

   i. **1, 2, 3, 4, 5, 10, 9, 8, 7, 6**
   ii. **6, 5, 7, 4, 8, 3, 9, 2, 10, 1**
   iii. **8, 6, 7, 5, 4, 3, 9, 10, 2, 1**

5. Given the same algorithm as above. If user enters the number 1, 2, 3, …. 10, and S is a **queue**, which of the following output sequences is/are possible?

   i. **6, 7, 8, 1, 2, 3, 9, 10, 4, 5**
   ii. **2, 1, 4, 3, 6, 5, 8, 7, 10, 9**
   iii. **5, 1, 7, 10, 2, 9, 3, 4, 6, 8**

~~~ Lab Questions ~~~

1. [**The master juggler** - Submission] The problem description is very simple:

   Given a list of numbers in any order, return a **sorted list (in ascending order)**. However, you must respect the following restrictions:

   a. You can go through the list of numbers **once**. To help illustrating this point, the function you need to write, **`makeSorted`,** has the main loop written for you. You can only access the numbers in this loop.
   b. You can **only use at most two stacks to store the number temporarily.** i.e. your code **cannot use additional data structure to store the numbers in any fashion.** Again, to help you, the two stacks are already declared for you. You are **not allowed to declare / use any additional variables in the function.**
   c. Similarly, the output list should be used to store the result **only**. Each number should be **appended to the output list** and not accessed again.
   d. If there are duplicate numbers in the list, you need to respect the relatively order (i.e. as if it is a **stable sorting**).

   For simplicity, you can assume that the **`StackList`** implementation is given.

   Note: Since the correctness of your solution lies mainly on whether you follow the restrictions, please do not take the test case output as the sole indicator.

   We will evaluate both correctness and programming style of your submitted code:

   a. [Correctness 70%]: The code works according to the functional requirement, e.g. output value, output format, side effect, efficiency etc.

   b. [Programming Style 30%]: Reuse own code whenever appropriate, modularity, good variable / method naming convention, comments (only if appropriate)