COMP-2540 Data Structures and Algorithms - Winter 2023
Lab Assignment 1

Deadline: **January 23/25, 2023** (15 minutes prior to the end of the lab)

Rules: This assignment must be done individually. Any similarity between your code/answers and another student's, or a carbon-copy of an answer found on the Web will be considered plagiarism. None of the built-in classes/methods/functions of Java/C/C++/Python or any other language can be used here. You must implement **your own** ADT and its operations.

Objective: The aim of this assignment is to obtain hands on experience in implementing the stack ADT, and understanding the concepts of complexity studied in class. At the end of this assignment, you should know how to implement a Stack ADT, how to analyze the complexity of its operations and how to apply it to real problems.

Tasks:

1. Using your favorite programming language (Java is suggested), design and implement a stack on an array. Implement the following operations: push, pop, top, size, isEmpty. Make sure that your program checks whether the stack is full in the push operation, and whether the stack is empty in the pop operation.

2. Practical application 1: Arithmetic operations.
   (a) Design an algorithm that takes a string, which represents an arithmetic operation, as input and checks whether or not the brackets are correct (balanced) or incorrect (unbalanced). The input string may contain a combination of the following characters: {,},[,],(,),0,1,2,3,4,5,6,7,8,9,+,-,*,/. An obvious hint for this algorithm is to use the stack-based solution discussed in class as a template, and hence you can use the stack implemented in #1. Your algorithm must **not** check the correctness of the arithmetic operators/operands, but only check for balanced brackets. Your algorithm should also show an error message when the string contains a character that is not one of those listed above. Provide the pseudocode of the algorithm.

   (b) Implement the algorithm of (a) in your favorite programming language. Run the program in several cases, including the following (more examples to be asked when the assignment is submitted):

   i. (9*[3*{[(3+3)/5]*7}])
   ii. {3*(2+[3-[4/[6/9]]]})
   iii. ((3*(9-(4*(6-5))))
   iv. {2-{3*{6/[[[(((9-0)))]]]}}/7}

   (c) Consider an input string of size *n*: (i) what is the worst-case time that your algorithm takes to decide whether or not the string is correct (balanced) or incorrect (unbalanced)? (ii) Why? Give your answers in terms of the O-notation.

3. Practical application 2: Context-free language recognizer.

(a) Design an algorithm that recognizes (accepts) strings of the following context-free language: L = $\{0^n1^n : \text{where } n \geq 1\}$. That is, strings that contain a number of zeros followed by the same number of ones. Examples of strings in the language are: 01, 0011, 000111, etc. Strings that are *not* in the language are, for example, 10, 00, 111, 1100, 0101, and many more. The algorithm should take a string of 0's and 1's as input and output "Yes" if the string belongs to the language, and "No" if the string doesn't belong to the language. The algorithm should use a stack. When it reads a 0 from the string, it pushes 0 onto the stack. When it reads a 1, it checks if the stack has a 0, in which case that 0 is popped from the stack. When the stack is empty *and* the end of the string is reached, the string is recognized as correct (Yes). Otherwise, the string is not in the language (No).

(b) Implement the algorithm of (a) in your favorite programming language and run it on different strings as follows: 01, 000111, 00000001111111, and 10, 00, 00111, 0101, and others.

(c) Consider an input string of size *n*: (i) what is the worst-case time that your algorithm takes to decide whether or not the string belongs to the language? (ii) Why? Give your answers in terms of the O-notation.

(d) [**Optional] Run your algorithm on strings of lengths n = 2, 4, 8, 16, 32, …, 2^{20}. Compute the running time in nanoseconds and draw a table with the results. Draw a table or a plot of the results for different values of *n* and compare the results obtained with the complexity you found in (c). Note: if all (or most of) the running times become 0, try running the program 1,000 times and compute the average.

Submission:

1. Your assignment must be submitted during the lab session in the section you are registered in. Any submission *within the last 15 minutes* before the end of the lab session will **not** be accepted and a **zero** mark will be given. Late assignments will be given a **zero** mark. Submissions by email will **not** be accepted.
2. Provide the source code for all your programs.
3. Explain how your stack works, how it helps check parenthesis matching and strings in the language. These will be asked when the lab assignment is being submitted and marks will be deducted if not clearly explained.