CS151:Spring 2023

Lab 5

Lab exercises due by Assignment 4 due date - yes lab4 and lab5 are both due with A4

1. Download `PostfixEvaluator.java` from `~dxu/handouts/labs/05`. Study the code. This is an implementation of the postfix evaluator we discussed in class with the built-in Java `Stack`. Test it using `TestPostfixEvaluator.java`. Make sure you understand how it works.

   Modify `PostfixEvaluator.java` to convert a postfix expression to a parenthesized infix expression and display it to the user. Modifications to `PostfixEvaluator.java` should not be extensive. The only difference is that when encountering an operator, instead of pushing the result of the arithmetic operation involving this operator and the top two operands onto the stack, i.e. a number, you will consruct the string representing that arithmetic operation and push the string onto the stack instead.

   ```
   Sample Input 1
   5 6 + 9 *

   Output
   ( ( 5 + 6 ) * 9 )

   Sample Input 2
   8 9 10 + *

   Output
   ( 8 * ( 9 + 10 ) )
   ```

2. Implement a `DoubleStack` class such that

   - a single underlying array stores two different stacks (stack 1 and stack 2), one grows from index `0` upward, one grows from the end of the array down. So these two stacks grow towards each other. The top indexes are denoted by `top1` and `top2` for stack 1 and stack 2, respectively. Thus, there are three instance variables: `E[] theArray`, `int top1`, `int top2`
   - `theArray` locations `0` to `top1` contain elements in stack 1 and `theArray` locations `theArray.length-1` downto `top2` stores the elements in stack 2.

   (a) Write methods

   i. `push(int stackId, E e)`: push `e` onto stack `stackId` (1 or 2). In other words, it will push onto stack 1 if `stackId==1` and onto stack 2 if `stackId==2`. Throw an `IllegalStateException` if stack is full - for now.

   ii. `E pop(int stackId)`: pop from `stackId`, return `null` if empty.

   iii. `E top(int stackId)`: top element from `stackId`, return `null` if empty.

iv. `int size(int stackId)`: return size of stack `stackId`

v. `boolean isEmpty(int stackId)`

vi. `printStack(int stackId)`

(b) What is the big-O of each of your methods above? If any besides `printStack` is not $O(1)$, you are writing unnecessary loops.

(c) Change `push` so that if the array gets full, instead of throwing an exception, resize the array to double size. What is the worst-time complexity of your `push` now?