CMS 250: Data Structure and Algorithms

Assignment 3

total points: 10

Due date: October 22

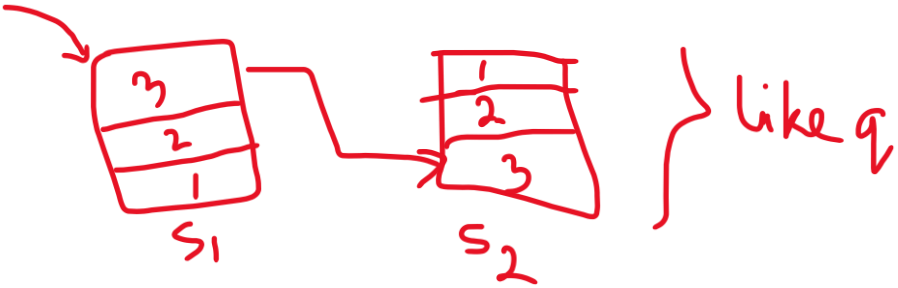
Where to submit: on Canvas. Just upload the zipped project

Objectives: The purpose of this assignment is for student to demonstrate

1. Basic understanding of Stack and Queue structure and how to implement them to solve a problem.
2. Implementation of file in Java. Getting input and printing output in a file.

Question 1: Queue and Stack implementation (3 points)

(a) Implement enqueue/offer() and dequeue/poll() like function. Hints: You just need to use Stack class with two objects of Stack. Use them in a way that like a queue a new item gets added at the end and the first item gets removed first. **(1.5 points)**



(b) Design and implement an algorithm to sort the elements of a **queue** in ascending order using a **stack** as an auxiliary data structure. You are not allowed to use any other data structure (like arrays or lists). **(1.5 points)**

Algorithm Steps (Hint)

- Create an empty stack.
- Dequeue elements one by one from the queue.
- For each element:
  - While the stack is not empty **and** the top of the stack is greater than the current element:
    - Pop elements from the stack and enqueue them back to the queue.
  - Push the current element onto the stack.
- Once all elements are processed, push all elements from the stack back into the queue.
- The queue will now be sorted in ascending order.

Question 2: Palindrome (3 points)

Problem Description: A palindrome is a word, number, phrase, or other sequence of symbols that reads the same backwards as forwards, such as madam or racecar. Write a palindrome checker using Stack. **Take input from "palindrome.txt" file and output to file "out.txt"**. Read the **last two pages of Stack-For Midterm2.pdf** slides on canvas for how to use file.

Sample input: madam

Sample output: madam is a palindrome.

Question 3: Infix to Postfix Conversion in Java (4 points)

Problem Description:

In mathematical expressions, **infix notation** is commonly used, where operators are written between operands (e.g., 3 + 5). However, evaluating such expressions requires handling **operator precedence** and **parentheses**. A more efficient notation for computation is **postfix notation** (Reverse Polish Notation, RPN), where operators follow their operands (e.g., 3 5 +).

Your task is to implement a Java method **toPostfix(String infix)** that converts a given infix expression into its corresponding postfix notation using the **Shunting Yard Algorithm** https://mathcenter.oxford.emory.edu/site/cs171/shuntingYardAlgorithm/ .

Input Format:

- The input is a string representing an **infix mathematical expression**.
- The expression **only contains single-digit numbers**, operators (+, -, *, /), and **parentheses**.
- **Tokens are separated by whitespace.**
- The input is always valid.

Output Format:

- Return a **string** representing the equivalent **postfix expression**, where each token is separated by a **single space**.

Constraints:

- 1 ≤ length of expression ≤ 100
- Supported operators: +, -, *, /
- Multiplication (*) and division (/) have **higher precedence** than addition (+) and subtraction (-).

- Addition (+) and subtraction (-) have the **lowest precedence** and are **left-associative**.
- Parentheses () override precedence rules.

Example 1:

Input:

"3 + 5 * 2"

Output:

"3 5 2 * +"

Example 2:

Input:

"(1 + 2) * (3 / 4)"

Output:

"1 2 + 3 4 / *"

Example 3:

Input:

"8 / 4 + 2 * 3"

Output:

"8 4 / 2 3 * +"

Additional Notes:

- You **must use a stack** to handle operators and ensure proper precedence.
- **Do not evaluate** the postfix expression, only convert it.
- Consider implementing **helper functions** for checking operators and precedence.
- Starter code is at ***InfixToPostFix.java***