CENG 202: DATA STRUCTURES

Programming Assignment 3

Deadline: WEDNESDAY, 19 APRIL 2023 by **20:00**

Rules

1. You will use Java to implement the tasks
2. You will lose 20 points for each day your assignment is submitted late.
3. No deadline extensions will be granted.
4. Your code must be developed individually. You cannot collaborate with your peers. You cannot post your code on any public repository, nor can you share it with anybody. Any violations to this code of conduct will be treated as academic misconduct.
5. Submitted code will be automatically checked using tools that detect plagiarism.
6. In case cheating is detected, the grade of this assignment will be zero and a further penalty of 50 points will be applied to the next homework assignment.
7. You may be asked to explain your code and rewrite some parts of it in front of the instructor.
8. Submit your codes to AYBUZEM.
9. Write your name as a comment line in each file you submit.
10. In order to be accepted, your submission
    - must be of text format with a '.java' extension (no Word or other formats and no archives will be accepted)
    - You will receive a zero grade if you do not submit the assignment or you do not adhere to this rule!

Note: You can only use code/pseudo code from the slides and the textbook. You need to develop the rest of the code yourself.

What to turn in electronically:

- Your source code (Single *.java file).
- Your file needs to have only Node and MyStack classes.

Task:
In this task, you are asked to implement Stack ADT to evaluate arithmetic expressions consisting of operands and operators in postfix notation. You have Node and MyStack classes to construct your Stack ADT. Given parts of the code should not be changed. After you construct your Stack structure, you need to complete the **evaluate** function in Test class. **evaluate** function, works as follows:

- Function takes a single parameter; a **string** which can consist of a single command or multiple commands separated with **space** characters.
- The commands consist of operands and operators in postfix notation.
- When the user sends an integer(s) as a command, push that integer(s) onto the stack.
- When the user enters a valid operator (**+**, **-**, *, /, **%**) as a command, pop two integers off the stack, perform the requested operation, and then push the result back onto the stack. Note: If the user enters **/** to perform division, then make sure the division is the integer division (not floating point). Be careful about the order of numbers and analyze the sample outputs.
- When the user enters **`print`**, add the current state of the stack with the exact form of the output to the result of your function. The content of the stack should be in brackets. If there are more than 1 item there should be a comma and space after each item until the end. Analyze the sample outputs.
- When the user enters **`pop`** as a command, pop the top element off the stack and add only that element (not the entire stack) to the result of your function.
- When the user enters **`exit`** as a command, add $ sign to the result of your function.

There are a number of error conditions your program must consider for full credit. For example, if the user enters an operator other than the 5 operators given above, then your program should ignore it but should not stop and an `“Invalid operator”` message should be added to the result of your function. Or, if the user enters a valid operator, however, there are no integers in the stack or there is only one integer in the stack, then your program adds a warning message `“Not enough integers in the stack”` and should ignore the operation. Be careful about ignoring the operation and analyze the outputs carefully.

Sample input and outputs are given in the skeleton code. **Remove** those lines before you submit your file.

Results of **evaluate** function must match exactly with the outputs. There is a new line character after each line of the result string including the last line.