Spring 2026 MSML606 HW2 (deadline: 2/25, Wednesday)

Overview

This assignment focuses on expression trees, tree traversal algorithms, and stack-based expression evaluation. You will implement data structures and algorithms that demonstrate the relationship between tree representations of mathematical expressions and their evaluation.

You can find the boilerplate Python script as well as three test cases in a zip file that you can download directly from ELMS.

Instructions:

- Do not directly copy content from textbooks, other individuals' work, AI-generated outputs, online sources, or any material that isn't your own. While collaboration and consulting various resources are encouraged, it's essential that you express your understanding in your own words, reflecting your personal learning and insights.
- Add justifications/explanations to all answers, as there will be points awarded for that. Please write in your own words.

Important!

Before submitting your work, review the external-source policy in the syllabus. You must either (1) cite any external sources you used, including AI tools, or (2) explicitly state that you did not use any. **Submissions without this AI usage statement will receive no points!!!**

Part I (programming): Stacks and Binary Trees

NB: Please include your Github repository link for this part. We will check the commit history and will consider the entire assignment pushed all at once a red flag. Please write your comments in the source code in your own words.

Problem 1. Generate an expression tree (Binary tree) from a postfix expression

Requirements:

- ➢ The input is a list of strings (e.g., [“3”, “4”, “+”, “2”, “*”]), which is parsed from a comma-separated CSV
- ➢ Support basic operators: +, -, *, and /
- ➢ Return the root node of the constructed expression tree

Example: Let out input be a list “3, 4, +, 2, *”. Then the expected tree structure is this:

```
      *
     / \
    +   2
   / \
  3   4
```

Problem 2. Implement functions to print prefix, infix, postfix expressions using traversal methods. Use pre-order traversal (root, left, right) to generate prefix notation. Use in-order traversal (left, root, right) for infix notation with appropriate parentheses. Use post-order traversal (left, right, root) to generate postfix notation.

Requirements:

- ➢ Return the expression as a list of elements
- ➢ For infix: Add parentheses to maintain correct operator precedence (even for the outermost expressions. Also, treat parentheses as individual elements in the returned list)
- ➢ Handle empty trees gracefully

Example (using the tree from Problem 1):

Prefix: *, +, 3, 4, 2 Infix: (, (, 3, +, 4,), *, 2,) Postfix: 3, 4, +, 2, *

Problem 3. Evaluate a postfix expression using a stack. That is, implement a function that evaluates a postfix expression without constructing an expression tree.

Requirements:

- ➢ Accept postfix expressions as space-separated strings (unlike comma-separated in Problem 1)
- ➢ Implement your stack using either an array or a list (i.e., implement the functions based on the Stack ADT we covered in class)
  - ○ Constraints: You may use Python’s list structure as the underlying storage. While you can use .append() to add elements, please ensure the implementation strictly follows the logic we discussed in class (e.g., manually managing the "top" of the stack.
  - ○ Use <u>your own</u> stack implementation to solve problem 3
- ➢ Support operators: +, -, *, and /
- ➢ Return the numeric result of the evaluation
- ➢ Handle division by zero appropriately (e.g., raise a ZeroDivisionError or handle it so that the test case catches the error)

Example: Assume our input is “5 1 2 + 4 * + 3 -”. From this we get the infix expression:

5 + ((1 + 2) * 4) - 3 = 5 + 12 - 3 = 14. Therefore, the expected output is 14.

Problem 4. Explain how edge cases are handled.

Edge cases to address:

- ➢ Empty postfix expressions
- ➢ Malformed postfix expressions (insufficient operands, too many operands)
- ➢ Division by zero
- ➢ Invalid tokens (non-numeric operands, unsupported operators)
- ➢ Very large numbers or results
- ➢ Negative numbers in the expression

Requirements:

- ➢ Include code comments explaining edge case handling
- ➢ Implement appropriate error handling (exceptions or return values)
- ➢ Provide test cases demonstrating edge case handling
- ➢ Document your approach in the report

Rubric for Part I: (Maximum Score: 21 Marks)

Problem	**Criteria**	**Marks**
Problem 1: Construct expression tree	- Correct Tree Construction (3 Marks) - Handling Invalid Sequences (3 Marks)	6
Problem 2: Print functions	- Correct print of prefix expression (2 marks) - Correct print of infix expression (2 marks) - Correct print of postfix expression (2 marks)	6
Problem 3: Evaluate a postfix expression via stack	- Correct expression evaluation (3 Marks) - Handling Division by Zero(3 Marks)	6
Problem 4: Edge Cases Handling	- Authenticity (1.5 Marks) - Clear explanation of all edge cases (1.5 Marks)	3

Part II (written): Recurrence relations

NB: Please write your explanations in their own words.

This part requires no submission and, therefore, will not be graded. It is assumed that you will complete this assignment on your own. The sole purpose of this part is to provide you more practice and creative interaction with recurrence relations and the Master Theorem.

Problem 1. Solve the following recurrence relations:

(a) $T(n) = 2T(n/2) + n^2$; $T(1) = 1$
(b) $T(n) = T(n/2) + T(n/3) + n$; $T(0) = 1$

Problem 2. Give and solve a recurrence relation for the description of the following algorithm: Given an input array of size n, create 5 separate arrays, each one third of the size of the original array. This takes linear time in n to accomplish. Then call the same algorithm on each of the 5 arrays.

Problem 3. Find Big-Theta for this recurrence relation: $7T(n/2) + 3n^2 + 2$
(Hint: apply Master's theorem case 1)

Problem 4. Which of the following recurrence relations allow the use of the Master's theorem and why?

(a) $T(n) = 2T(n/2) + 2^n$
(b) $T(n) = 2T(n/3) + \sin(n)$
(c) $T(n) = T(n - 2) + 2n^2 + 1$
(d) None of these
(e) All of the above

Problem 5. Solve the following recurrence relations by drawing recursion trees. Computer total work per level and derive T(n)

(a) $T(n) = 2T(n/2) + n$; $T(1) = 1$
(b) $T(n) = 3T(n/2) + n$; $T(1)=1$
(c) $T(n) = T(n/2) + T(n/4) + n$; $T(1) = 1$
(d) $T(n) = 3T(n/3) + n*\log(n)$; $T(1)=1$