Programming Language

Assignment 2

Due date: 2025/10/19 11:59pm

Explanation

- Title : Building a Parse Tree Based Arithmetic Expression Calculator

- Objective
  - Implement a program that takes a mathematical expression as a string, parses it into a **parse tree**, and calculates the result.
  - You must implement the entire pipeline **without using external libraries** such as ast, eval, or operator.
  - The process should follow these steps: **Tokenization → Parsing & Validation → Calculation**

Tokenizer

- How?
  - Split the input string into numbers, operators, and parentheses.
  - Supported operators: ^, +, -, *, /
  - Parentheses () are supported.
- Example
  - Expression: 1 + 2 * 3
  - Tokens: ['1', '+', '2', '*', '3']

Parser

- How?
  - Use a **recursive descent parser** implementation.

  - Operator precedence:
    - Exponentiation(^) > Multiplication (*), Division (/) > Addition (+), Subtraction (-)
    - Parentheses have the highest precedence.

- Parse tree structure
  - Number: integer (e.g., 3)
  - Binary operation: tuple (op, left, right)

- Example
  - Expression: 1 + 2 * 3
  - Tokens: ['1', '+', '2', '*', '3']
  - Parse Tree: ('+', 1, ('*', 2, 3))

Validation

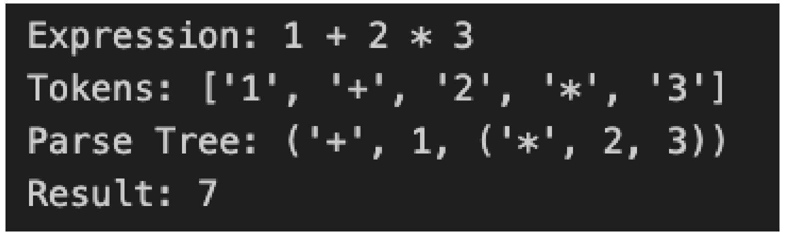
- During parsing, detect invalid expressions and raise errors.
- Example invalid cases:
  - "1 + * 2”
    - "(3 + 4" (missing parenthesis)
    - "1 2 + 3" (unexpected tokens)

Calculator

- Evaluate the parse tree recursively.

- Hint:
  - (+, left, right) → eval(left) + eval(right)
  - (-, left, right) → eval(left) - eval(right)
  - (*, left, right) → eval(left) * eval(right)
  - (/, left, right) → eval(left) / eval(right)

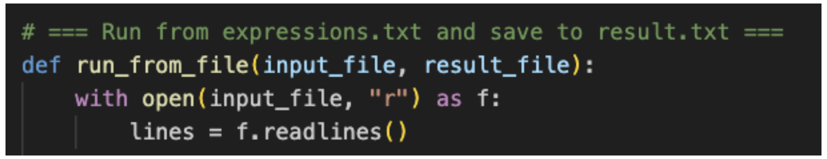
- Example
  - Expression: 1 + 2 * 3
  - Tokens: ['1', '+', '2', '*', '3']
  - Parse Tree: ('+', 1, ('*', 2, 3))
  - Result: 7



```
Expression: 1 + 2 * 3
Tokens: ['1', '+', '2', '*', '3']
Parse Tree: ('+', 1, ('*', 2, 3))
Result: 7
```

Submission Format

- Important!!

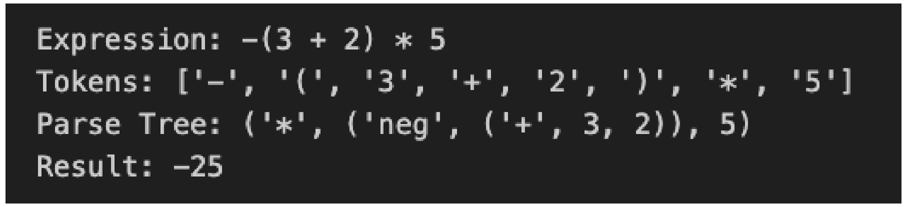


```
# === Run from expressions.txt and save to result.txt ===
def run_from_file(input_file, result_file):
    with open(input_file, "r") as f:
        lines = f.readlines()
```

- About code structure
  - In the code, you must read the “expressions.txt” file I have provided and save the output as “result.txt”. The format must be identical to the one in the given file.
  - When I run the main.py file you submitted, it must generate the result.txt file. If you want to experiment with more test cases, you can add additional expressions to expressions.txt.
  - You must implement (i) a tokenizer function that takes expressions, (ii) a parser that takes tokens, and (iii) a calculator that takes a parse tree, each as a separate function. Both the parser and the calculator must be implemented using recursive functions.
- Submission
  - Please submit both your main.py file and the report in a folder named {student_id}_{name}.”
  - Your report should contain a detailed explanation of the code design, and it may be written in either Korean or English.

Evaluation Criteria

- The final score will be determined based on our own test cases (70%) and the report (30%).
- An additional 5% bonus will be awarded if you implement unary minus. However, the total score cannot exceed 100%.



```
Expression: -(3 + 2) * 5
Tokens: ['-', '(', '3', '+', '2', ')', '*', '5']
Parse Tree: ('*', ('neg', ('+', 3, 2)), 5)
Result: -25
```

- Please verify that result.txt is produced correctly from the expressions.txt file I have provided.
- In cases of plagiarism, code sharing with peers, or the use of LLMs, the final grade will be recorded as F.
- If you have any question, please feel free to contact the TA.