Expression Parser Documentation

Course: Algorithms

Team members: Meirambek, Ernar, Yestay

1. Goal

Implement a simple arithmetic expression parser that supports addition, subtraction, multiplication, division and parentheses. The parser builds an abstract syntax tree (AST) that correctly respects standard operator precedence and associativity, prints the tree and evaluates the expression.

2. Supported features

- Operators: `+` `-` `*` `/` `( )`
- Integer literals
- Standard precedence: `*` and `/` bind stronger than `+` and `-`
- Left-associative operators
- Arbitrary nesting of parentheses

3. Parsing algorithm

Classic recursive-descent parser with three precedence levels:

```
text

expr    → term   { ("+" | "-") term }*      // lowest precedence
term    → factor { ("*" | "/") factor }*    // higher precedence
factor  → NUMBER | "(" expr ")"
```

Each level consumes the operators of its own precedence and recursively calls the next higher level, which naturally enforces the required order of operations.

4. Required example

Expression (with explicit grouping for the intended semantics):

```
text

(12 + 3 - (4 + 6)) / (12 * 2)
```

Manual calculation:

```
text

4 + 6 = 10
12 + 3 = 15
15 - 10 = 5
12 * 2 = 24
5 / 24 ≈ 0.208333
```

The parser correctly builds the tree:

```
text

/
       /   \
      -     *
    / \   / \
   +   + 12  2
  / \ / \
 12  3 4  6
```

and returns the exact result 5/24.

5. Functionality provided

- `tokenize()` – converts the input string into a list of tokens
- `Parser` class – builds the AST using recursive descent
- `print_tree()` – pretty-prints the expression tree (visual verification of precedence)
- `evaluate()` – recursively evaluates the tree and returns the numerical result

6. Usage example

```Python
expr = "(12 + 3 - (4 + 6)) / (12 * 2)"
tree = Parser(tokenize(expr)).parse()
print_tree(tree)
print(evaluate(tree))        # → 0.20833333333333334
```

The implementation is concise, fully functional, follows mathematical rules without exceptions and is ready for inclusion in the project report or further extensions (unary minus, exponentiation, etc.).