Lab 02: Postfix Expression Evaluation (PEE)

Overview

We commonly write arithmetic operations in *infix* notation, wherein operators are written in between its two operands, as in the following expression:

```
6 / 2 * (1 + 2)
```

This has been our conventional writing of arithmetic expressions; however, it has a disadvantage: the need for grouping symbols (*e.g.* parentheses) to indicate the order in which operators are evaluated. As a result, grouping symbols, in a way, complicate the evaluation process of such expressions. Evaluation would be much simpler if we can solve operators from left to right. Obviously, this is not the case in infix notation; fortunately, we can do this using *postfix* notation. In postfix notation of arithmetic expressions, operators appear right after its operands. For example, the expression above in postfix notation is written as follows:

```
6 2 / 1 2 + *
```

It can be observed that the order of operands for both notations are similar (reading from left to right: `6, 2, 1, 2`); however, the order of the operators are now changed - it is in the order of its evaluation. Notice also that parentheses are removed in the postfix notation. It is difficult to comprehend at first; however, it is easier to evaluate. All we need is a stack to perform these operations.

Assume, the input expression is an arithmetic expression in postfix notation which consists of a sequence of single-digit, non-negative integers and the four basic arithmetic operators: `+`, `-`, `*`, and `/` (for simplicity, treat as integer division). Using a stack of integer, the postfix expression can be evaluated using the following algorithm:

- Read the characters from left to right.
- For each character being read:
    - If the character is an integer: push it onto the stack.
    - If the character is an operator:
        - Pop a value from the stack. Call it `A`.
        - Pop another value from the stack. Call it `B`.
        - Solve for `C = B operator A`.
        - Push `C` onto the stack.
- After all characters were read, pop a value (the last value) from the stack. This is the result of the expression.

Using the example above as input for this algorithm, we have:

- Expression: **6** `2 / 1 2 + *` (character being read is in **bold**)
    - Action: push `6`
    - Stack: `[6]`

- Expression: `6 2 / 1 2 + *`
    - Action: Push `2`
    - Stack: `[2, 6]`
- Expression: `6 2 / 1 2 + *`
    - Action: pop `2 -> A`
    - Stack: `[6]`
    - Action: pop `6 -> B`
    - Stack: `[]`
    - Action: compute `C = B / A -> 6 / 2 = 3`
    - Action: push `C = 3`
    - Stack: `[3]`
- Expression: `6 2 / 1 2 + *`
    - Action: push `1`
    - Stack: `[1, 3]`
- Expression: `6 2 / 1 2 + *`
    - Action: push `2`
    - Stack: `[2, 1, 3]`
- Expression: `6 2 / 1 2 + *`
    - Action: pop `2 -> A`
    - Stack: `[1 3]`
    - Action: pop `1 -> B`
    - Stack: `[3]`
    - Action: compute `C = B + A -> 1 + 2 = 3`
    - Action: push `C = 3`
    - Stack: `[3, 3]`
- Expression: `6 2 / 1 2 + *`
    - Action: pop `3 -> A`
    - Stack: `[3]`
    - Action: pop `3 -> B`
    - Stack: `[]`
    - Action: compute `C = B * A -> 3 * 3 = 9`
    - Action: push `C = 9`
    - Stack: `[9]`
- Expression: `6 2 / 1 2 + *` (whole expression is already read)
    - Action: pop `9 -> result`
    - Stack: `[]`

Tasks

You will create a program that evaluates a postfix expression as described in the algorithm above.
Use the `STACK` ADT you have implemented by importing `stack.h`. Save your program as `postfix.c`. Take note that the input is just a sequence of single-digit values and operators (maximum of 100 characters).

The output must be an integer, the result after evaluating the expression.

Evaluate the following postfix expressions first and use them as test cases for your program.

Sample Input	Expected Output
62/	_
62/12+*	_
72*3+5-	_
3564-+*	_
02/	_
9	_

Submission

Create a `.zip` file named `SurnameEx02.zip` containing the following:

1. `list.h` and `list.c`
2. `stack.h` and `stack.c`
3. `postfix.c`
4. `Makefile` (create a `Makefile` to automate compilation and running.)

Deadline is within laboratory hours.

Questions?

If you have any questions, approach your lab instructor.