BIM213 – Data Structures and Algorithms

Homework I

Application of Stacks: Expression Evaluator

ZIP file format. **YourNameSurname_HW1.zip** You have to upload your ZIP file until on **Monday, November 11, 2024, until 23:59** to the **estuoys.eskisehir.edu.tr**

Suppose we would like to evaluate the following arithmetic expression: “20+2*3+(2*8+5)*4". To evaluate this expression, we have to realize that * has higher precedence than +, and therefore, compute 2*3 and add it to 20 rather than computing 20+2 and multiplying it with 3. Additionally, we need to put parentheses around (2*8+5) to specify that the result of this part will be multiplied by 4.

A typical evaluation sequence for the given infix expression would be the following:

(1) Store 20 in accumulator A1
(2) Compute 2*3 and store the result 6 in accumulator A2
(3) Compute A1+A2 and store the result 26 in A1
(4) Compute 2*8 and store the result in A2
(5) Compute 5+A2 and store the result 21 in A2
(6) Compute 4*A2 and store the result 84 in A2
(7) Compute A1+A2 and store the result 110 in A1
(8) Return the result, 110, stored in A1

We can write this sequence of operations as follows: "20 2 3 * + 2 8 * 5 + 4 * +". This notation is known as postfix or reverse polish notation.

Postfix notation unambiguously specifies the sequence of operations that will be performed to evaluate an expression without the need for parenthesis. Therefore, it is much easier to evaluate an expression specified in postfix notation.

It turns out, we can easily convert a well-formed infix expression to postfix notation using a stack. We will only consider the operators +, -, *, /, (,), and follow the usual precedence rules: * and / has the highest precedence, + and - come next, and (has the lowest precedence.

Here is a rough sketch of the algorithm:

(1) When an operand is encountered, output it

(2) When (is encountered, push it

(3) When) is encountered, pop all symbols off the stack until (is encountered

(4) When an operator, +, -, *, /, is encountered, pop symbols off the stack until you encounter a symbol that has lower priority

(5) Push the encountered operator to the stack

We start with an initially empty stack. When an operand is encountered it is immediately placed onto the output. When a left parenthesis, (, is encountered, it is placed onto the stack. If we see a right parenthesis, then we pop the stack, writing the symbols out until we encounter a corresponding left parenthesis, which is popped but not output. When an operator is encountered, then we pop entries from the stack until we find an operator of lower priority. That is, when a * or / is encountered, the stack is popped as long as we see * or /. Similarly, when + or - is encountered, the stack is popped as long as we see *, /, + or -, we stop only if the stack is empty or (is encountered.

IMPORTANT NOTES

- Grouping is not allowed in this homework. Please obey the ethical rules.
- **Never change the method names and descriptions.**
- You need to write the codes suitable for the methods left blank in the given java file.

Figure 1 shows the steps in converting the infix expression 20+2*3+(2*8+5)*4 to postfix (reverse polish) notation. Figure 2 shows the steps in evaluation of the postfix expression: 20 2 3 * + 2 8 * 5 + 4 * +

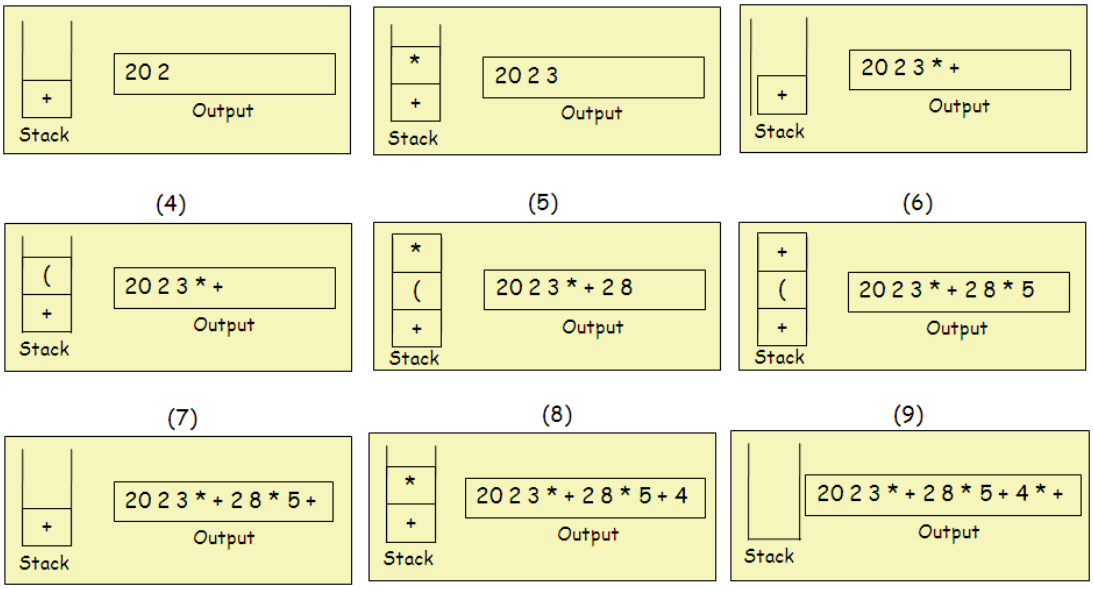

Figure 1: Steps in converting the infix expression 20+2*3+(2*8+5)*4 to postfix (reverse polish) notation.

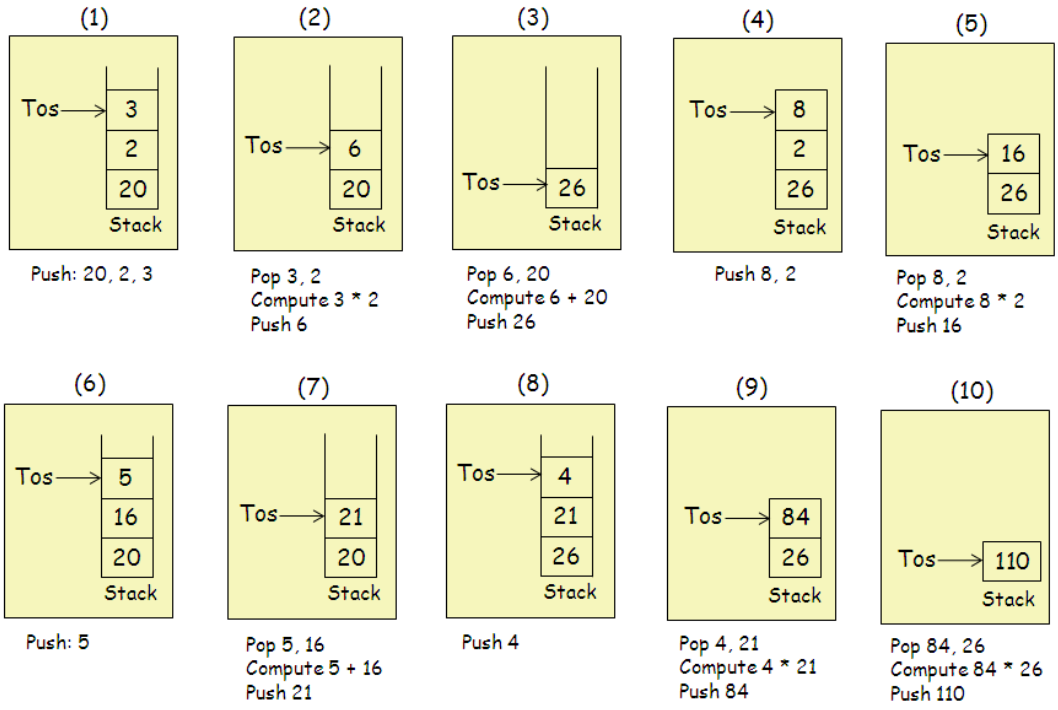

Figure 2: Steps in evaluating the postfix expression 20 2 3 * + 2 8 * 5 + 4 * +