# **Evaluating Postfix Expressions**

|  |  |
| --- | --- |
| 3 + 4 | 3 4 + |
| (3 + 4) \* 5 | 3 4 + 5 \* |
| 3 + 4 \* 5 | 3 4 5 \* + |
| 3 \* 4 + 5 \* 6 | 3 4 \* 5 6 \* + |
| 3 \* ( 4 + 5 ) \* 6 | 3 4 5 + \* 6 \* |
| ( 3 \* 4 + 5 ) \* 6 | 3 4 \* 5 + 6 \* |

## ***Infix and Postfix Notation***

A standard expression like 3 + 4 uses *infix* notation. The equivalent *postfix* form is 3 4 +. ("Post" means "after," meaning that the operator appears after its operands.) Because of the order of operations, infix notation sometimes requires parentheses, but postfix expressions never need parentheses.

### ***Algorithm for Evaluating Postfix Expressions***

Assuming the input string contains a valid postfix expression (e.g., 4 5 6 \* + 3 -), loop over the string and process each token. Hint: a stack is a helpful data structure.

Intermediate results are kept on the stack as the loop repeats. When the loop ends the final result will be the only number left on the stack.

## ***Assignment***

PostfixEval.java. At first, limit the operations to add, subtract, multiply, and divide. If we require that every token in the input string is separated by spaces, e.g., "33 -43 + -55 65 + \*", we can evaluate decimals and negatives. Assume that the postfix string is well-formed. Use as test data the postfix expressions below.

**public** **static** **final** String operators = "+ - \* / % ^ !";

**public static double** eval(String str)

{ }

**public static double** eval(**double** a, **double** b, String op)  
 { }

**public static boolean** isOperator(String op)  
 { }

## ***Test Run***

Postfix --> Evaluate  
3 4 5 \* + 23.0  
3 4 \* 5 + 17.0  
1.3 2.7 + -6 6 \* + -32.0  
33 -43 + -55 65 + \* -100.0  
3 4 \* 5 2 / + 5 - 9.5  
8 1 2 \* + 9 3 / - 7.0  
3 4 5 \* 6 + \* 78.0  
3 4 5 - 6 2 \* - + -10.0

## ***Improvements (required)***

Implement modulus, powers, and factorial. Use the input data as shown.

2 7 3 % + 3.0  
2 7 + 3 % 0.0  
2 3 ^ 8.0  
2 -2 ^ 0.25  
5 ! 120.0  
1 1 1 1 1 + + + + ! 120.0