Assignment 2: Solving Expressions in Postfix Notation using Stacks

In this assignment, you will write a program which reads in a numerical infix expression from the user and convert it to a postfix expression. Once converted, you will then solve the postfix expression and print the result for the user.

Infix expression: The expression of the form <operand> <operator> <operand>. When an operator is in-between every pair of operands. For example: 3 + (5 – 2) * 8

Postfix expression: The expression of the form <operand> <operand> <operator>. When an operator is followed for every pair of operands. For example: 3 (4 2 *) +

General Notes:

- For this assignment, we will only be considering numerical expressions
    - The only valid characters are single digits 0-9 and +, -, *, /, ^, (,)
- The user should input an expression with minimum 3 characters and maximum 20 characters.
- The user should be prompted to re-enter an expression if an invalid character is encountered or if the number of characters falls outside of the range (3-20 inclusive). This should be checked before proceeding with the algorithm.
- Your program **must** use the *ArrayStack* implementation from Blackboard.
- The precedence of the mathematical operators is as follows:
    - **^** (Math.pow) has the highest precedence followed by ***** or **/**, followed by **+** or **–**
- You will need to create a utility method which returns a value for an operator, based on its precedence, so that two of them can be compared.
- In order to carry out the mathematical operations, you will need to ensure that the operands are casted to an appropriate number type. ArrayStack works with *Objects*.
- Even though we are dealing with single digit integer numbers for the operands, keep in mind that the result of an expression might be a decimal number.
    - For example: 3 * 4 / 5 = 2.4
    - If we only cast using integers, then the result will be incorrectly given as 2.

Infix to Postfix Algorithm:

1. Scan the infix expression from left to right. The expression will be input as a String from the user and can then be converted to a character array. You can use a loop to read each character at a time.
2. **If** the scanned character is an operand (number): append it to an output String.
3. **Else**:
    3.1 **If** the precedence of the scanned operator is greater than the precedence of the operator in the stack (or the stack is empty or the stack contains an opening parenthesis '('): push it.
    3.2 **Else**, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator and append them to the output string. After doing that, Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. **If** the scanned character is an '(', push it to the stack.

5. **If** the scanned character is an ')', pop the stack and append to the output until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until all of the infix expression is scanned.
7. Pop (and append to the output) any remaining content from the stack.
8. You now have the postfix expression stored as the output string.

Evaluation of postfix expressions:

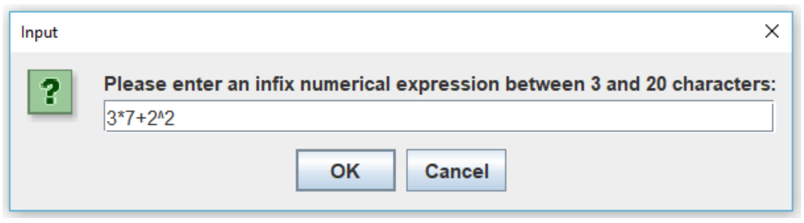
1. Create a stack (*ArrayStack*) to store operands
2. Scan the postfix expression (created using the algorithm above) and do the following for every scanned element.
   1. If the element is a number (operand), push it onto the stack
   2. If the element is an operator, pop two operands for the operator from the stack. Evaluate the operator and push the result back to the stack
3. When the expression has been fully read, the number on the stack is the final answer

Submission Notes:

- You should submit a *single PDF document* with the following sections:
  - **Problem Statement (3 Marks)**
    - Describe the problem. You can carry out your own research to describe the overall problem in sufficient detail to demonstrate that you fully understand it.
  - **Analysis and Design Notes (5 Marks)**
    - Before you begin coding, you should analyse the overall problem and create design notes for yourself. This can include identifying methods that will be required, writing basic pseudocode and outlining the flow of control for the program. You can then use this as a guide when you begin programming.
  - **Code (16 Marks)**
    - **IMPORTANT**: You must copy and paste your code as text into this section
    - **If you submit a screenshot of the code you will receive 0 marks.**
    - Your code must also contain plenty of *meaningful* comments to fully describe the functionality of each part.
  - **Testing (6 Marks)**
    - You should extensively test each aspect of the code and provide screenshots of the testing output. A very basic example is shown at the end of this document. You will need to run comprehensive tests for each part of the program to receive full marks in this section. You could also include a table listing tests with expected and actual results.

- You have two lab sessions to complete this assignment.
  - Due date: **Wednesday the 8th of February**

Sample Program Usage:

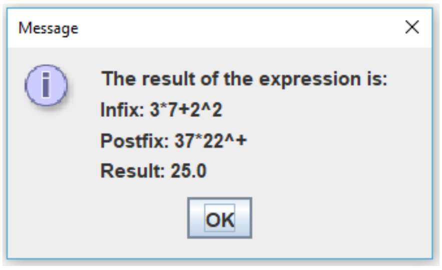

Input

Please enter an infix numerical expression between 3 and 20 characters:

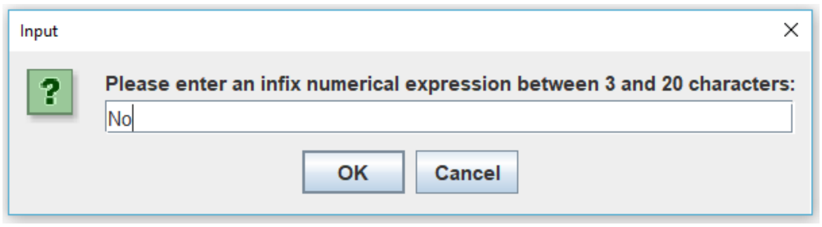
3*7+2^2

Console:

```
3.0 * 7.0 = 21.0
2.0 ^ 2.0 = 4.0
21.0 + 4.0 = 25.0
```

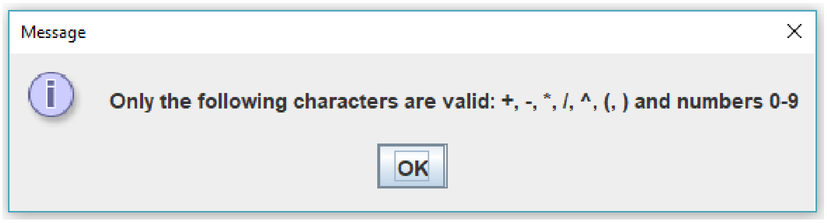

Message

The result of the expression is:
Infix: 3*7+2^2
Postfix: 37*22^+
Result: 25.0

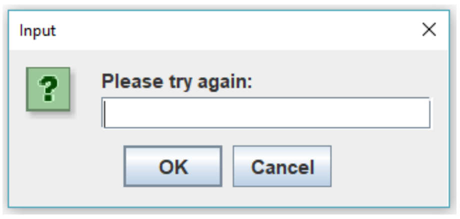

Input

Please enter an infix numerical expression between 3 and 20 characters:

No


Message

Only the following characters are valid: +, -, *, /, ^, (,) and numbers 0-9


Input

Please try again: