Evaluation of Postfix Expressions

Do Now

What is a prefix expression?

What is an infix expression?

What is a postfix expression?

Infix expression

An operator is written in between two operands.

Example: 4 * 10

Prefix Expression

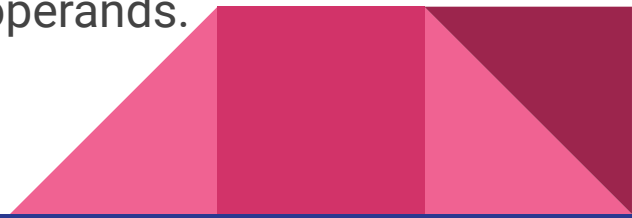
It requires that all operators precede the two operands that they work on.

Example: * 4 10

Postfix Expression

In this type of expression an operator is written after its operands.

Example: 4 10 *



Specifications to design your calculator to evaluate postfix expressions

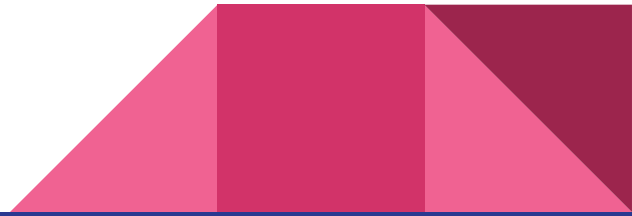
- Operands could be valid numbers (int or double). For our calculator let's use double numbers.

- Valid operators:

  Add (+) Subtract (-)

  Multiply(*) Divide(/)

  Remainder(%)



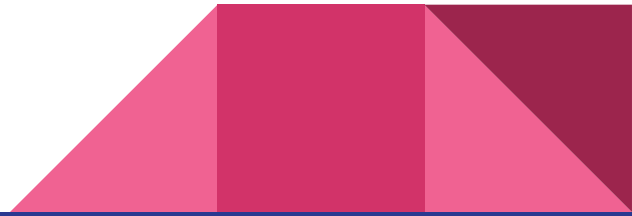
Specifications to design your calculator to evaluate postfix expressions

- The operators work on 2 values (4 10 *)
- The expressions we are going to evaluate are strings and all operands and operators are separated by a single space.

  Examples:

  “10 2.5 /” “8 2 + 99 9 - * 2 + 9 -”

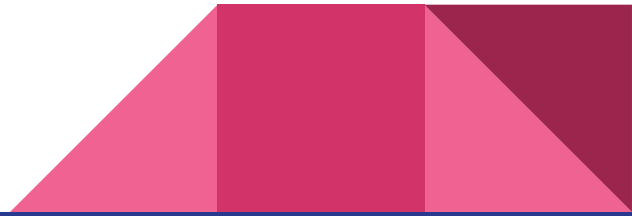
  “4 5 - 2 + 1.5” “2 4 6 8 10 + * - -”



Group discussion

Let’s think on an algorithm that can help us evaluate a postfix expression. Consider the following questions:

- How can you read postfix expression (string)?
- You must use a data structure to evaluate the expression, which one would you use?
- Explain how your algorithm would work.

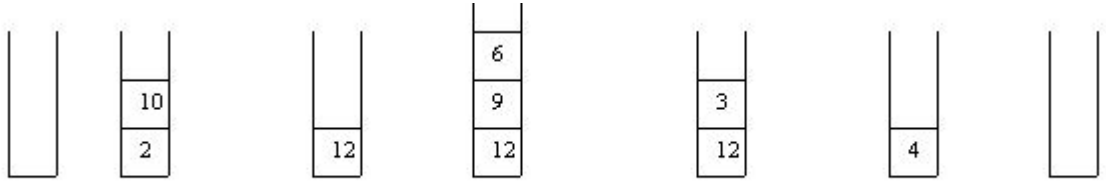


Would your algorithm work for the following postfix expressions?

Postfix Expression	Infix Equivalent	Result
4 5 7 2 + − ×	4 × (5 − (7 + 2))	−16
3 4 + 2 × 7 /	((3 + 4) × 2)/7	2
5 7 + 6 2 − ×	(5 + 7) × (6 − 2)	48
4 2 3 5 1 − + × + ×	? × (4 + (2 × (3 + (5 − 1))))	not enough operands
4 2 + 3 5 1 − × +	(4 + 2) + (3 × (5 − 1))	18
5 3 7 9 + +	(3 + (7 + 9)) ... 5???	too many operands

Could a stack work?

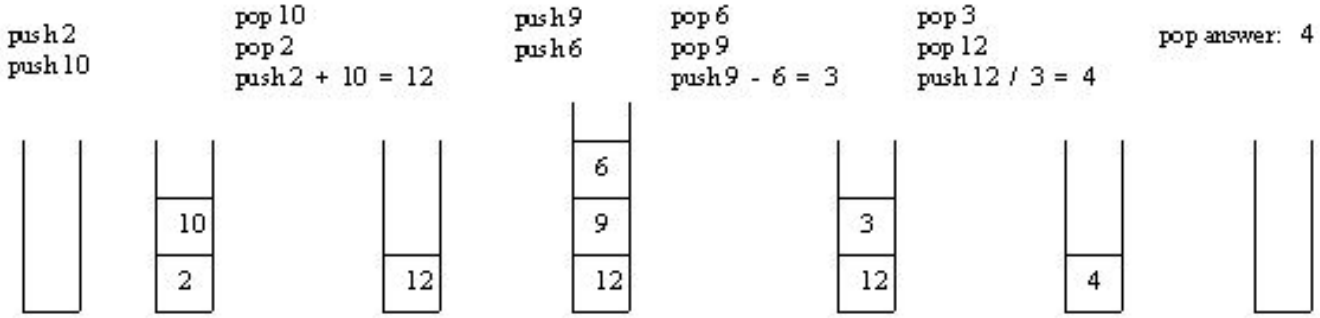
Postfix Expression: **2 10 + 9 6 - /**



- What does it happen when there is an operand in the expression?
- What does it happen when there is an operator in the expression?4
- Is there a way to know if there are too many or too few operands/operators?

Postfix expression using a stack

Expression: **2 10 + 9 6 - /**



Stack Calculator - Implementation

- Create a folder StackCalculator inside the classwork folder in your assignments repo.
- Create a file Calculator.java, which contains a method eval

```
public class Calculator{

   // Evaluate a postfix expression stored in expression
   public static double eval(String expression){
     return 0.0;
   }

}
```

- String expression: Contains ints, doubles, and operators (+ - / * and %) separated by one space
- Return a double value
- Throw an IllegalArgumentException when there are too many or too few operands/operators.

Convert an infix expression to a postfix expression

Convert **A * B + C** to **A B * C +**

Infix token	Operator stack	Postfix string
A		A
*	*	A
B	*	A B
+	+	A B * {pop and print '*' before pushing +'}
C	+	A B * C
		A B * C +

- Print an operand when it is read.

- Push an operator onto the stack if it is empty. If the operator on the top of the stack has higher precedence than the one being read, pop and print the one on top and then push the new operator on. Otherwise push the new sign onto the stack.

- When the end of the expression has been reached, pop the operators on the stack one at a time and print them.

Convert an infix expression to a postfix expression

Convert **A + B * C to A B C * +**

Infix token	Operator stack	Postfix string
A		A
+	+	A
B	+	A B
*	+ *	A B
C	+ *	A B C
		A B C * +

- Print an operand when it is read.

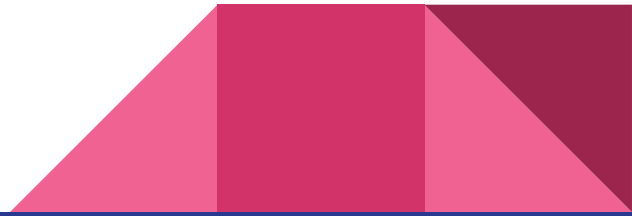
- Push an operator onto the stack if it is empty. If the operator on the top of the stack has higher precedence than the one being read, pop and print the one on top and then push the new operator on. Otherwise push the new sign onto the stack.

- When the end of the expression has been reached, pop the operators on the stack one at a time and print them.

Conversion test cases

Convert 5 * 4 + 2 to 5 4 * 2 + = 22

Convert 10 + 8 * 4 to 10 8 4 * + = 42



Convert an infix expression to a postfix expression

Convert **A * (B + C)** to **A B C + ***

Infix token	Operator stack	Postfix string
A		A
*	*	A
(	* (	A B
B	* (	A B
+	* (+	A B
C	* (+	A B C
)	*	A B C +
		A B C + *

Since expressions in parentheses must be done first, everything on the stack is saved and the left parenthesis is pushed to provide a marker. When the next operator is read, the stack is treated as though it were empty and the new operator (here the '+' sign) is pushed on. Then when the right parenthesis is read, the stack is popped until the corresponding left parenthesis is found. Since postfix expressions have no parentheses, the parentheses are not printed.