Infix and postfix expressions

In a postfix expression,
- an operator is written **after** its operands.
- the infix expression *2+3* is *23+* in postfix notation.
- For postfix expressions, operations are performed in the order in which they are written (left to right).
- No parentheses are necessary. '
- the infix expression *2+3*4* is *234*+* in postfix notation
- the infix expression *3*4+2*5* translates to *34*25*+* in postfix notation.
- the infix expression 3*(4+2)*5 translates to 342+*5*

Evaluation of postfix expressions.

*2+3*4 (infix) / 234*+ (postfix)* expression. Notice:

- the operands (*2,3,and 4)* appear in the same order in both expressions.
- in the postfix version the operators (* *and +*) appear in the order in which they are performed -- the multiplication before the addition
- writing the operators in the order in which they are performed makes postfix expressions easy to evaluate using the following algorithm:

> 1. scan the expression, left to right, until you encounter an operator, @ (@ means + - * or /)
> 2. Perform the operation @. The operands **precede** the operator
>   3 4 + = 3+4= 7
> 3. In the expression, replace @ and its operands with the computed value
> 4. repeat 1-3 the process until no more operators exist.

Look at *234*+*.
Here is the sequence of operations:
- 2 **3 4 *** +   * is the first operator. Perform the operation 34*
- **2 12 +**   3 4*is replaced by 12 , the value of 3*4
                          + is next operator, perform 2 12+
- replace 2 12+ with 14.   Done

The value of the expression is 14. Another example, *3 4 * 2 5 * +* which in infix notation is 3*4 + 2*5.

3 4 * 2 5 * +   * is the first operator 3 4 * is replaced by 12
12 **2 5 *** +   2 5 * is replaced by 10
12 10 +   12 10 + is replaced by 22
22

Postfix notation does not require parentheses.

Evaluation of postfix with a stack"
- Scan the string left to right.
- When you encounter an operand push it on the stack;
- when you encounter an operator, pop the corresponding operands off the stack,
- perform the operation, and push the result back on the stack.

- When you are finished scanning the expression, the final value remains on the stack.

For example, consider the postfix expression *234*+*

Input	Stack *(top is on the left)*	
2 3 4 * +	*empty*	Push 2
3 4 * +	2	Push 3
4 * +	3 2	Push 4
* +	4 3 2	Pop 4, pop 3, do 3 *4 , push 12
+	12 2	Pop 12, Pop 2, do 2 + 12, push 14
	14	

Input	Stack	
3 4 * 2 5 * +	*empty*	Push 3
4 * 2 5 * +	3	Push 4
* 2 5 * +	4 3	Pop 4, pop 3, do 3*4, Push 12
2 5 * +	12	Push 2
5 * +	2 12	Push 5
* +	5 2 12	Pop 5, Pop 2, do 2*5, Push 10
+	10 12	Pop 10, Pop 12 do 12 + 10, push 22
	22	

Here is an algorithm to evaluate postfix expressions.

To eliminate some unnecessary and non-instructive details make a few simplifying assumptions:

1. all input numbers are in the form of single digits 0..9
   There is no whitespace in the input string. Thus 345+* is valid
   but 3 4 5 +* is not.

2. the only operators allowed are the binary
   operators +,-,*, and /, where / signifies **integer** division.

3. all input data is correct.

Thus a typical input string is 23*73/+, which in infix notation is 2*3 + 7/3 (value is 8).

Making these assumptions, the algorithm for postfix evaluation is

while characters remain in the postfix string
    1. read a character
    2. if the character is a digit, convert to int and push
    3. if the character is an operator
        pop the stack twice obtaining the two operands
        perform the operation
        push the result
Pop the final value from the stack.

How to convert Infix to postfix.

hHow do we convert it to postfix notation.
 For example, the infix expression *(2+3)*(4+5)* in postfix notation is *23+45+**
and the infix expression *2+3*4+5* in postfix notation is *234*+5+*.

Also, since our four operators are left associative, *2 + 3 + 4* translates to *23+4+* and not *234++.* While both of these postfix expressions evaluate to 7, the first is interpreted as *(2+3)+4* (correct) and the second as *2 + (3+4)* (incorrect associativity). By ignoring the associativity of operators, you could run into trouble with subtraction and division. The infix expression *2-3+4* is evaluated as (2-3)+4 = (-1)+4 = 3. The correct postfix is 23-4+ and not 234+- (which is equivalent to 2- (3+4) and evaluates to -5).

Once again, we can use a stack to facilitate the conversion of infix to postfix. This time, however, we will use a stack of characters to store the operators in the expression. To convert correctly formed infix expressions to postfix we will use the following algorithm.

While characters remain in the infix string
 1. read the next character in the infix string
 2. if the character is an operand, append the character to the postfix
  expression
 3. if the character is an operator @
   while the stack is not empty and an operator of greater or equal
        priority is on the stack
    pop the stack and append the operator to the postfix
  push @
 4. if the character is a left parenthesis (
  push the parenthesis onto the stack
 5. if the character is a right parenthesis)
   while the top of the stack is not a matching left parenthesis (
    pop the stack and append the operator to postfix
   pop the stack and discard the returned left parenthesis
 Pop any remaining items on the stack and append to postfix.

Examples.

Input	Stack	Postfix
2*3 + 4*5	empty	
*3+4*5	empty	2
3+4*5	*	2
+4*5	*	23
4*5	+	23*
*5	+	23*4
5	*+	23*4
	*+	23*45
	+	23*45*
	empty	23*45*+

Input	Stack	Postfix
2-3+4-5*6	empty	
-3+4-5*6	empty	2
3+4-5*6	-	2
+4-5*6	-	23
4-5*6	+	23-
-5*6	+	23-4
5*6	-	23-4+
*6	-	23-4+5
6	*-	23-4+5
	*-	23-4+56
	-	23-4+56*
	empty	23-4+56*-

Input	Stack	Postfix
(2-3+4)*(5+6*7)	empty	
2-3+4)*(5+6*7)	(	
-3+4)*(5+6*7)	(	2
3+4)*(5+6*7)	(-	2
+4)*(5+6*7)	(-	23
4)*(5+6*7)	(+	23-
)*(5+6*7)	(+	23-4
*(5+6*7)	empty	23-4+
(5+6*7)	*	23-4+
5+6*7)	(*	23-4+
+6*7)	(*	23-4+5
6*7)	+(*	23-4+5
*7)	+(*	23-4+56
7)	*+(*	23-4+56
)	*+(*	23-4+567
	*	23-4+567*+
	empty	23-4+567*+*