Algorithm for Infix to Postfix

Token	Action
Operand	Append to end of output list
open parenthesis	push token to Stack
close parenthesis	while element on Stack is not an open parenthesis: pop elements from Stack and append to output list Discard/pop open parenthesis from Stack
operator	while operator at top of Stack has a higher or equal precedence compared to precedence of operator token pop operator from Stack and append to output list push operator token on stack
End of tokens	Append all remaining items on Stack to the output list

Example Input : 3 + 4 * 2 / (5 - 1) + (2 + 3)

	Token	Operator/Parenthesis Stack	Output List
1.	3	[]	[3]
2.	+	[+]	[3]
3.	4	[+]	[3, 4]
4.	*	[+, *]	[3, 4]
5.	2	[+, *]	[3, 4, 2]
6.	/	[+, /]	[3, 4, 2, *]
7.	(	[+, /, (]	[3, 4, 2, *]
8.	5	[+, /, (]	[3, 4, 2, *, 5]
9.	-	[+, /, (, -]	[3, 4, 2, *, 5]
10.	1	[+, /, (, -]	[3, 4, 2, *, 5, 1]
11.	)	[+, /]	[3, 4, 2, *, 5, 1, -]
12.	+	[+]	[3, 4, 2, *, 5, 1, -, /, +]
13.	(	[+, (]	[3, 4, 2, *, 5, 1, -, /, +]
14.	2	[+, (]	[3, 4, 2, *, 5, 1, -, /, +, 2]
15.	+	[+, (, +]	[3, 4, 2, *, 5, 1, -, /, +, 2]

16.	3	[+, (, +]	[3, 4, 2, *, 5, 1, -, /, +, 2, 3]
17.	)	[+]	[3, 4, 2, *, 5, 1, -, /, +, 2, 3, +]
			[3, 4, 2, *, 5, 1, -, /, +, 2, 3, +, +]

3 4 2 * 5 1 - / + 2 3 + +

Postfix evaluation using a stack

1. go through expression from left to right
2. push operands on the stack
3. if it is an operator,
   pop the last two operands from the stack
   apply the operator on the last two operands
   push the result back onto the stack

[] -> [3] -> [3, 4] -> [3, 4, 2] -> [3, 8] -> [3, 8, 5] -> [3, 8, 5, 1] -> [3, 8, 4] -> [3, 2] -> [5] -> [5, 2] -> [5, 2, 3] -> [5, 5] -> [10]