PARSING ARITHMETIC EXPRESSIONS

TABLE OF CONTENTS

INTRO

Infix format: e.g. 6 * (5 + 4) * 10

Postfix format: 6 5 4 + * 10 *

The following show you the 2 steps process –

1. convert the infix expression to postfix expression
2. evaluate the postfix expression to obtain the final value

LET'S TRY SOME SAMPLES:

Sample 1: 7 + 3 * 10 / 5

Infix to postfix

Add to postfix exp.	Read	Push to stack	
7	7		
7	+	+	
7 3	3	+	
7 3	*	* +	If ord(stack top) >= ord(curr) Pop stack top Push curr on stack
7 3 10	10	* +	
7 3 10 *	/	/ +	
7 3 10 * 5	5	/ +	
7 3 10 * 5 / +		Pop remaining token off the stack	

7 3 10 *	/	/ +	
7 3 10 * 5	5	/ +	
7 3 10 * 5 / +		Pop remaining token off the stack	

Evaluate postfix

Add to postfix exp.	Read	Operation	Push to stack	
7 3 10 * 5 / +	7		7	
3 10 * 5 / +	3		3 7	
10 * 5 / +	10		10 3 7	
* 5 / +	*	3 * 10 = 30	30 7	
5 / +	5		5 30 7	
/ +	/	30 / 5 = 6	6 7	
+	+	7 + 6	13	

Sample 2: 6 * (5 + 4) * 10

Add to postfix exp.	Read	Push to stack	
6	6		
6	*	*	
6	(	(*	
6 5	5	(*	
6 5	+	+ (*	
6 5 4	4	+ (*	
6 5 4 +	)	*	Since it is ')'.. pop until it sees '('
6 5 4 +	*	*	If ord(stack top) >= ord(curr) Pop stack top & add to expression Push curr on stack
6 5 4 + *	10	*	
6 5 4 + * 10 *		Pop remaining token off the stack	

Evaluate postfix

Add to postfix exp.	Read	Operation	Push to stack	
6 5 4 + * 10 *	6		6	
	5		5 6	
	4		4 5 6	
	+	4 + 5	9 6	Push 9 on stack
	*	9 * 6	54	Push 54 on stack
	10		10 54	
	*	10 * 54 = 540	540	Push 540 on stack
Empty				Pop 540 off the stack

Sample 3: 6 * (5 + 4) + 10

Add to postfix exp.	Read	Push to stack	
6	6		
6	*	*	
6	(	(*	
6 5	5	(*	
6 5	+	+ (*	
6 5 4	4	+ (*	
6 5 4 +	)	*	Since it is ')'.. pop until it sees '('
6 5 4 + *	+	+	If ord(stack top) >= ord(curr) Pop stack top Push curr on stack
6 5 4 + * 10	10	+	
6 5 4 + * 10 +		Pop remaining token off the stack	

Sample 4: 6 * (5 + 4) / 10

Add to postfix exp.	Read	Push to stack	
6	6		
6	*	*	
6	(	(*	
6 5	5	(*	
6 5	+	+ (*	
6 5 4	4	+ (*	
6 5 4 +	)	*	Since it is ')'.. pop until it sees '('
6 5 4 + *	/	/	If ord(stack top) >= ord(curr) Pop stack top Push curr on stack
6 5 4 + * 10	10	/	
6 5 4 + * 10 /		Pop remaining token off the stack	

Sample 5: 6 + (5 + 4) * 10

Add to postfix exp.	Read	Push to stack	
6	6		
6	*	+	
6	(	(+	
6 5	5	(+	
6 5	+	+ (+	
6 5 4	4	+ (+	
6 5 4 +	)	+	Since it is ')'.. pop until it sees '('
6 5 4 +	*	* +	If ord(stack top) >= ord(curr) Pop stack top Push curr on stack
6 5 4 + 10	10	* +	
6 5 4 + 10 * +		Pop remaining token off the stack	

Sample 6: 6 * (5 + 4) / 10

Add to postfix exp.	Read	Push to stack	
6	6		
6	*	*	
6	(	(*	
6 5	5	(*	
6 5	+	+ (*	
6 5 4	4	+ (*	
6 5 4 +	)	*	Since it is ')'.. pop until it sees '('
6 5 4 + *	/	/	If ord(stack top) >= ord(curr) Pop stack top Push curr on stack
6 5 4 + * 10	10	/	
6 5 4 + * 10 /		Pop remaining token off the stack	

Let's take a look of the precedency order:

	Precedence Value to reflect the priority order. Higher value means higher precedence	
Valid Operators	In the stack	Outside the Stack
(	0	100
- (unary negation)	8	7
^	5	6
* / %	4	3
+ -	2	1
	Note: ^ outside the stack has a higher precedent over the ^ inside the stack	

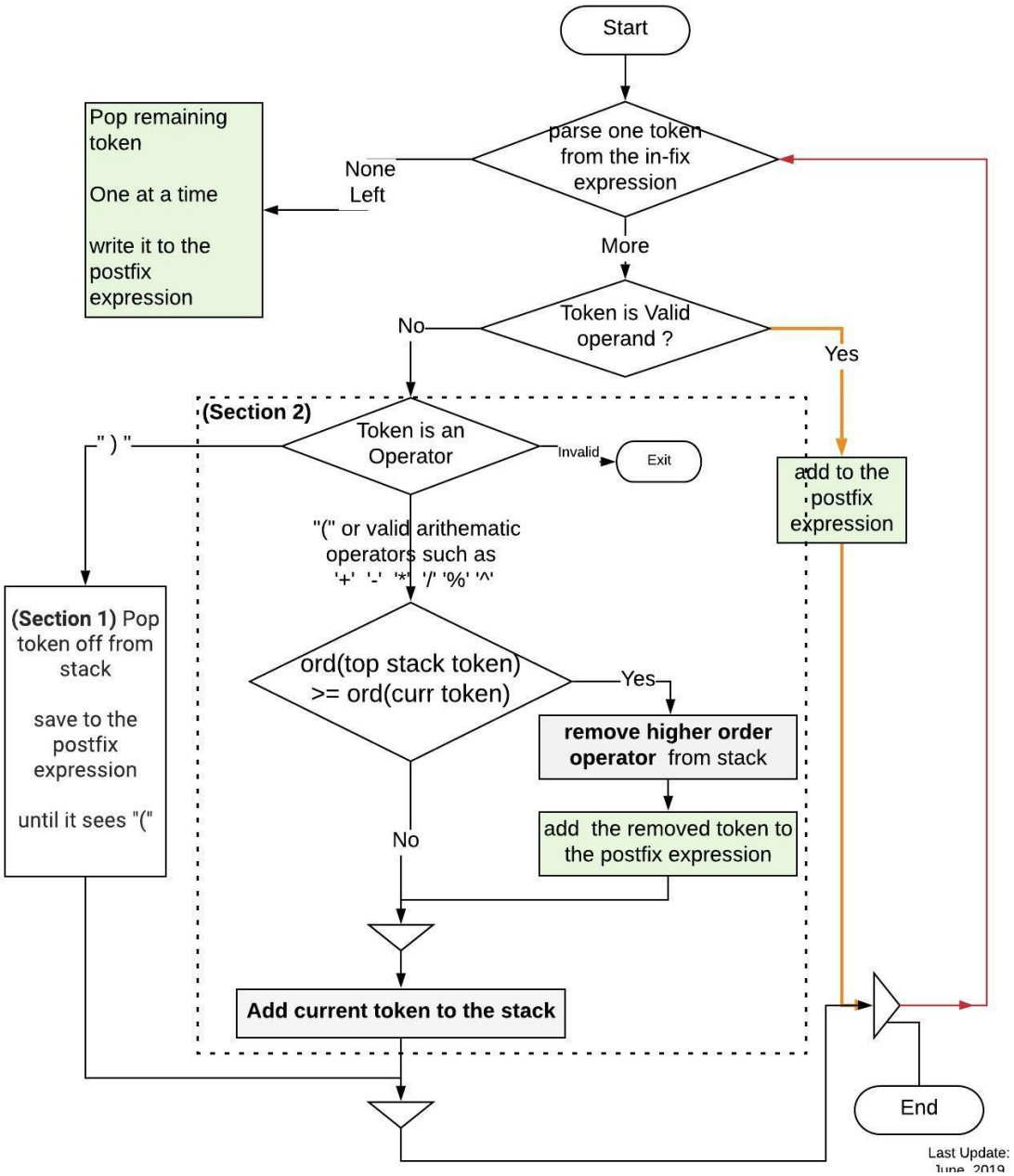
use 100 to allow additional operators in the future.

STEP 1 : CONVERT INFIX TO POSTFIX

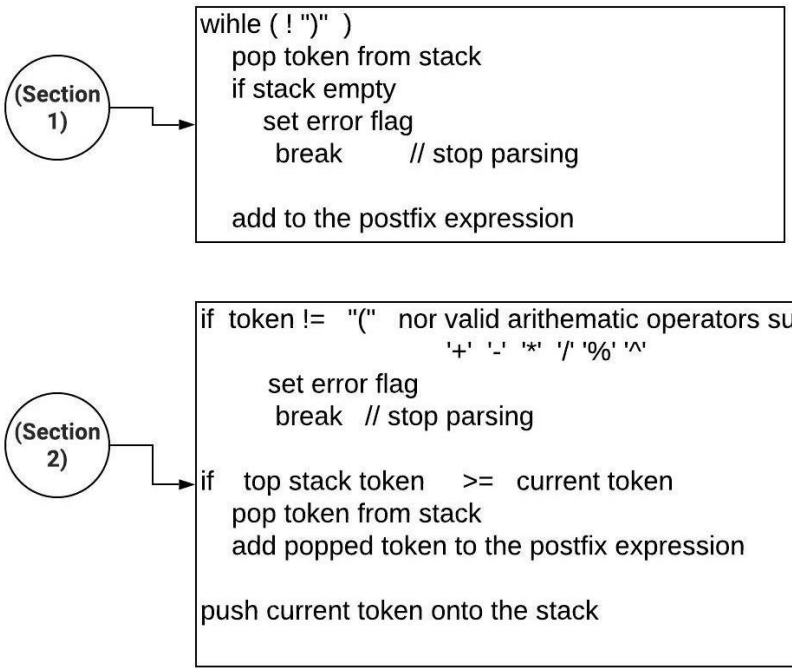
NOTE: Use a stack to store the operators only.

Operators may include + - * / % (^ only.

HIGH LEVEL FLOWCHART



PSUEDOCODE FOR SEGMENTS IN STEP 1



Sample for setting up the presedency order by give each a value:

```
char  plus = '+',  minus='-' , div='/', mult='*', mod='%', ep='^';

char ord[255] = {0};
ord[ plus ] =  ord[ minus ] = 1;
ord[ div ] = ord[ mult ]  = ord[ mod ] = 3;
ord[ ep] = 6;

...

if ( ord[ nextOpFromExpression ] >   ord[ topOpInStack ] )
        ...
```

STEP 2 : CONVERT INFIX TO POSTFIX

NOTE: Use a stack to store the operators only.

HIGH LEVEL FLOWCHART