Arithmetic Expressions and Stack Operations

1. Introduction to Arithmetic Expressions

Arithmetic expressions consist of operators and operands. Operators specify the operations to be performed on operands. Expressions can be represented in different forms:

- Infix: Operators are between operands (e.g., A + B).

- Prefix: Operators are before operands (e.g., +AB).

- Postfix: Operators are after operands (e.g., AB+).

2. **Why Use Stacks for Expression Evaluation?**

Stacks are a Last-In-First-Out (LIFO) data structure. They are used to reverse the order of operators in arithmetic expressions and handle operator precedence. The stack helps in processing expressions from infix to postfix and evaluates postfix expressions efficiently.

3. Operator Precedence Table

The following table shows the precedence of operators and their associativity:

Operator	**name**
( )	parentheses
^	power
* , /	Multiplication, Division
+ , -	Addition, Subtraction

Why using postfix notation?

1. It is the most suitable notation system to calculate any expression duo to its reversing characteristic and the absence of using parenthesis.
2. It is universally accepted notation for designing the arithmetic logic unit (ALU) of the processor.
3. It is the way computers look toward any arithmetic expression.
4. **Conversion Rules :(Infix to Postfix Conversion)**

There are a set of rules need to be performed when using the stack in converting infix notation to postfix notation, these rules are:

1. When reading a number or data item, perform pop immediately.
2. When reading an operator, pop until the top of the stack has an element of lower precedence and push this element.
3. When any ending parenthesis or brackets like) or] is pushed, then perform pop until reaching the matching (or [.
4. The parenthesis or brackets have the lowest precedence inside the stack.
5. When the end of the input is reached, pop until the stack is empty.

step	**Input**	**Stack**	**Output**
1	A		A
2	+	+	A
3	B	+	AB
4	*	+ *	AB
5	C	+ *	ABC
6			ABC*+

The postfix expression is: ABC*+

2. 3 + 4 * 5 / 6

Solution:

step	Input	Stack	Output
1	3		3
2	+	+	3
3	4	+	34
4	*	+*	34
5	5	+*	345
6	/	+/	345*
7	6	+/	345*6
8			345*6/+

The postfix expression is: 345*6/+

3. (30+23)*(43-21)/(84+7)

Solution:

step	Input	Stack	Output
1	(	(	
2	30	(	30
3	+	(+	30
4	23	(+	30 23
5	)		30 23 +
6	*	*	30 23 +
7	(	*(	30 23 +
8	43	*(	30 23 + 43

9	-	*(-	30 23 + 43
10	21	*(-	30 23 + 43 21
11	)	*	30 23 + 43 21 -
12	/	/	30 23 + 43 21 - *
13	(	/(	30 23 +43 21 - *
14	84	/(	30 23 +43 21 - * 84
15	+	/(+	30 23 +43 21 - * 84
16	7	/(+	30 23 +43 21 - * 84 7
17	)	/	30 23 + 43 21 - * 84 7 +
18			30 23 + 43 21 - * 84 7+ /

The postfix expression is: 30 23 + 43 21 - * 84 7+ /

4. (4+8)*(6-5)/((3-2)*(1+2))

Solution:

step	**Input**	**Stack**	**Output**
1	(	(	
2	4	(	4
3	+	(+	4
4	8	(+	4 8
5	)		4 8 +
6	*	*	4 8 +
7	(	*(	4 8 +
8	6	*(	4 8 + 6
9	-	*(-	4 8 + 6

10	5	*(-	4 8 + 6 5
11	)	*	4 8 + 6 5 -
12	/	/	4 8 + 6 5 - *
13	(	/(	4 8 + 6 5 - *
14	(	/((	4 8 + 6 5 - *
15	3	/((	4 8 + 6 5 - * 3
16	-	/((-	4 8 + 6 5 - * 3
17	2	/((-	4 8 + 6 5 - * 3 2
18	)	/(	4 8 + 6 5 - * 3 2 -
19	*	/(*	4 8 + 6 5 - * 3 2 -
20	(	/(*(	4 8 + 6 5 - * 3 2 -
21	1	/(*(	4 8 + 6 5 - * 3 2 - 1
22	+	/(*(+	4 8 + 6 5 - * 3 2 - 1
23	2	/(*(+	4 8 + 6 5 - * 3 2 – 1 2
24	)	/(*	4 8 + 6 5 - * 3 2 – 1 2 +
25	)	/	4 8 + 6 5 - * 3 2 – 1 2 + *
26			4 8 + 6 5 - * 3 2 – 1 2 + * /

The postfix expression is: 4 8 + 6 5 - * 3 2 – 1 2 + * /

Tutorials:

1. What is the obtained number after performing the following sequence of push and pop: PUSH(1), POP, PUSH(2), POP, PUSH(3), PUSH(4), POP, PUSH(5), POP,POP
2. What is the required sequence of push and pop to obtain the string (25431) from the initial input (12345).
3. Convert the following infix expression to the equivalent postfix notation: