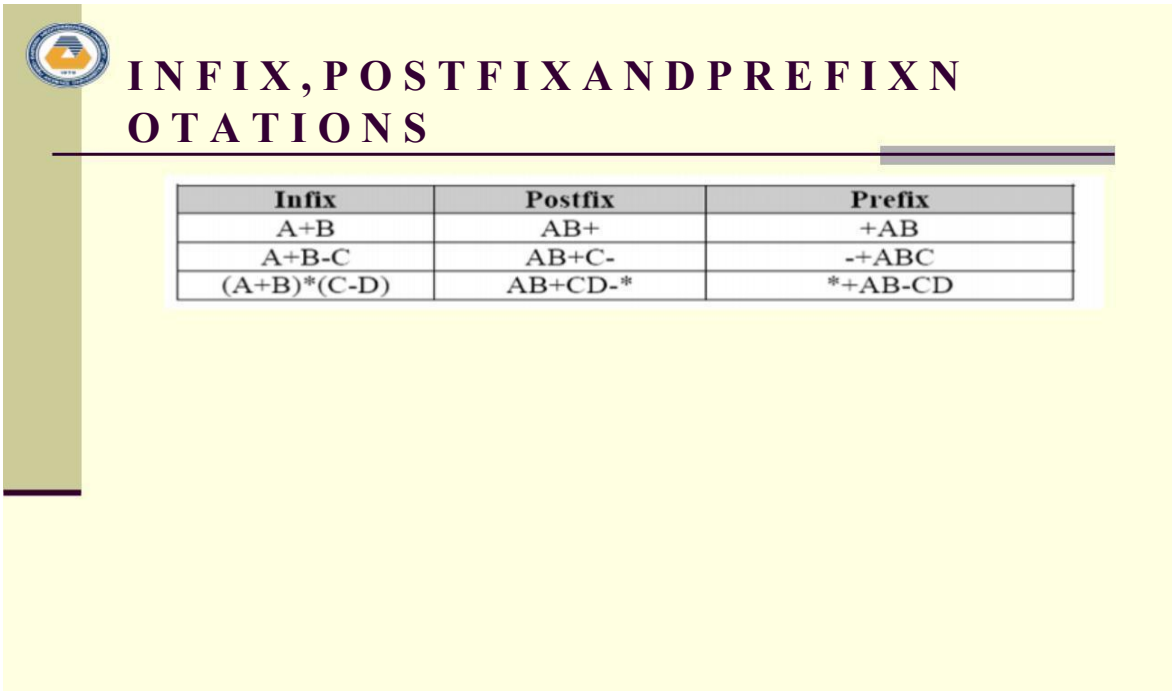
Application of the Stack (Arithmetic Expressions)



Stacks are used by compilers to help in the process of converting infix to postfix arithmetic expressions and also evaluating arithmetic expressions. Arithmetic expressions consisting variables, constants, arithmetic operators and parentheses. Humans generally write expressions in which the operator is written between the operands (**3 + 4**, for example). This is called infix notation. Computers "prefer" postfix notation in which the operator is written to the right of two operands. The preceding infix expression would appear in postfix notation as **3 4 +**.
To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation, and then evaluate the postfix version of the expression. We use the following three levels of precedence for the five binary operations.

Precedence	**Binary Operations**
Highest	Exponentiations (^)
Next Highest	Multiplication (*), Division (/) and Mod (%)
Lowest	Addition (+) and Subtraction (-)

For example:

(66 + 2) * 5 – 567 / 42
to postfix
66 22 + 5 * 567 42 / –

Transforming Infix Expression into Postfix Expression:

The following algorithm transform the infix expression **Q** into its equivalent postfix expression **P**. It uses a stack to temporary hold the operators and left parenthesis.
The postfix expression will be constructed from left to right using operands from **Q** and operators popped from STACK.

D A T A S T R U C T U R E S

Algorithm: Infix_to_PostFix(Q, P)

Suppose **Q** is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression **P**.

1. Push "(" onto STACK, and add ")" to the end of **Q**.
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
3. If an operand is encountered, add it to **P**.
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator © is encountered, then:
   a) Repeatedly pop from STACK and add to **P** each operator (on the top of STACK) which has the same or higher precedence/priority than ©
   b) Add © to STACK.

   [End of If structure.]
6. If a right parenthesis is encountered, then:
   a) Repeatedly pop from STACK and add to **P** each operator (on the top of STACK) until a left parenthesis is encountered.
   b) Remove the left parenthesis. [Do not add the left parenthesis to **P**.]

   [End of If structure.]

[End of Step 2 loop.]

7. Exit.

Convert **Q**: **A+(B * C – (D / E ^ F) * G) * H** into postfix form showing stack status .
Now add **")"** at the end of expression **A+(B * C – (D / E ^ F) * G) * H)**
and also Push a **"("** on Stack.

Symbol Scanned	Stack	Expression Y
	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
^	(+(-(/^	ABC*DE
F	(+(-(/^	ABC*DEF
)	(+(-	ABC*DEF^/
*	(+(-*	ABC*DEF^/
G	(+(-*	ABC*DEF^/G
)	(+	ABC*DEF^/G*-
*	(+*	ABC*DEF^/G*-
H	(+*	ABC*DEF^/G*-H
		ABC*DEF^/G*-H*
)	empty	**+**

D A T A S T R U C T U R E S

Evaluation of Postfix Expression:

If **P** is an arithmetic expression written in postfix notation. This algorithm uses STACK to hold operands, and evaluate **P**.

Algorithm: This algorithm finds the VALUE of **P** written in postfix notation.

1. Add a Dollar Sign "$" at the end of **P**. [This acts as sentinel.]
2. Scan **P** from left to right and repeat Steps 3 and 4 for each element of **P** until the sentinel "$" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator © is encountered, then:
   - **a)** Remove the two top elements of STACK, where **A** is the top element and **B** is the next-to—top-element.
   - **b)** Evaluate **B** © **A**.
   - **c)** Place the result of (b) back on STACK.

   [End of If structure.]

   [End of Step 2 loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.

For example:

Following is an infix arithmetic expression

(5 + 2) * 3 – 9 / 3

And its postfix is:

5 2 + 3 * 9 3 / –

Now add **"$"** at the end of expression as a sentinel.

5 2 + 3 * 8 4 / – **$**

Scanned Elements	Stack	Action to do
5	5	Pushed on stack
2	5, 2	Pushed on Stack
+	7	Remove the two top elements and calculate 5 + 2 and push the result on stack
3	7, 3	Pushed on Stack
*	21	Remove the two top elements and calculate 7 * 3 and push the result on stack
8	21, 8	Pushed on Stack
4	21, 8, 4	Pushed on Stack
/	21, 2	Remove the two top elements and calculate 8 / 2 and push the result on stack

-	19	Remove the two top elements and calculate 21 - 2 and push the result on stack
$	**19**	**Sentinel** **$ encouter ,** Result is on top of the STACK