The Infix, Prefix, Postfix Notation:

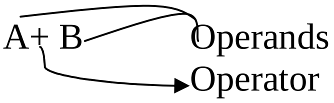
Applications of stack: There are a number of applications of stacks such as;

1) To print characters/string in reverse order.
2) Check the parentheses in the expression.
3) To evaluate the arithmetic expressions such as, infix, prefix and postfix.

Arithmetic expression: An expression is defined as a number of operands or data items combined using several operators. There are basically three types of notations for an expression;

1) Infix notation
2) Prefix notation
3) Postfix notation

Infix notation: It is most common notation in which, the operator is written or placed in-between the two operands. For eg. The expression to add two numbers A and B is written in infix notation as,



In this example, the operator is placed in-between the operands A and B. The reason why this notation is called infix.

Prefix Notation: It is also called Polish notation, named after in the honor of the mathematician Jan Lukasiewicz, refers to the notation in which the operator is placed before the operand as,
+AB
As the operator '+' is placed before the operands A and B, this notation is called prefix (pre means before).

Postfix Notation: In the postfix notation the operators are written after the operands, so it is called the postfix notation (post means after), it is also known as suffix notation or reverse polish notation. The above postfix if written in postfix notation looks like follows;
AB+

Notation Conversions: Let us take an expression A+B*C which is given in infix notation. To evaluate this expression for values 2, 3, 5 for A, B, C respectively we must follow certain rule in order to have right result. For eg.
A+B*C = 2+3*5 = 5*5 = 25!!!!!!!!
Is this the right result? No, this is because the multiplication is to be done before addition, because it has higher precedence over addition. This means that for an expression to be calculated we must have the knowledge of precedence of operators.

Operator Precedence:

Operator	Symbol	Precedence
Exponential	$	Highest
Multiplication/Division	*,/	Next highest
Addition/Substraction	+,-	Lowest

Conversion from infix to postfix expression:

<u>Algorithm</u>

POSTFIX (Q, P)

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

Step1: Push "(" onto STACK and add '')'' to the end of Q.
Step2: Scan Q from left to right and repeat step 3 to 6 for each element of Q, until the STACK is empty.
Step3: If an operand is encountered, push it to STACK.
Step4: If a left parenthesis is encountered, push it to STACK.
Step5: If an operator X○ is encountered, then

a) Repeatedly pop from STACK and add to P each operator (Top of Stack) which has same precedence as or higher precedence than X.○
b) Add X○ to STACK.

Step6: If a right parentheses is encountered, then;

a) Repeatedly pop from STACK and add to P each operator (top of stack) until a left parentheses is encountered.
b) Remove the left parentheses from stack [Do not add it to P]

```
        [End of if]
        [End of step 2 loop].
Step7: Exit.
```

Note: In the Infix to postfix conversion expression algorithm ⊗ means any mathematical operator such as +,-,*,/,$.

Example:

A-B/(C*D$E).

S.N.	Symbol Scan	STACK	P (Postfix Expression)
		(	
1.	A	(	A
2.	-	(-	A
3.	B	(-	AB
4.	/	(-/	AB
5.	(	(-/(	AB
6.	C	(-/(	ABC
7.	*	(-/(*	ABC
8.	D	(-/(*	ABCD
9.	$	(-/(*$	ABCD
10.	E	(-/(*$	ABCDE
11.	)	(-/	ABCDE$*
12.	)		ABCDE$*/-

Required postfix expression (P) = ABCDE$*/-

Evaluating a postfix expression:

Algorithm for evaluating postfix expression:

Let P is an expression written in postfix notation.

1) STACK=empty stack.
2) Scan P from left to right and repeat step 3 and 4 for each symbol in P until end of expression.
3) If an operand is encountered, push it on STACK.
4) If an operator x○ encountered then;
   a) Operand 2= pop (STACK).
   b) Operand 1= pop (STACK).
   c) Value= operand1 x○ operand 2.

```
        d) Push value on STACK.
    5) Return the value at top of the STACK.
    6) Exit.
```

Tracing of algorithm

P= 623+-382/+*2$3+

S.N.	Symbol Scan	Operand 1	Operand 2	Value	STACK
1.	6				6
2.	2				6,2
3.	3				6,2,3
4.	+	2	3	5	6,5
5.	-	6	5	1	1
6.	3				1,3
7.	8				1,3,8
8.	2				1,3,8,2
9.	/	8	2	4	1,3,4
10.	+	3	4	7	1,7
11.	*	1	7	7	7
12.	2				7,2
13.	$	7	2	49	49
14.	3				49,3
15.	+	49	3	52	52

Conversion from Infix to Prefix:

Algorithm for Infix to prefix:

```
Step 1: Scan character at a time from right to left.
Step 2: Repeat until there is data.
        a) If ')' Push into opstack.
        b) If operand push into prestack.
        c) If operator –if stack is empty –push it into opstack.
           Else
              -repeat while (prece (tos char) >= prece (scan char)) and push to prestack.
           -pop and push to prestack.
           -push scanchar to opstack.
```

d) if ‘)’ found pop and push to prestack until the matching ‘)’ is found and ignore (cancel) both.
Step 3: pop and push to prestack until stack is empty.
Step 4: pop and display from prestack until stack is empty.

Example;
(A-(B/C))*((D*E)-F)

S.N.	Scan symbol	Prefix stack	Opstack
1.	)		)
2.	F	F	)
3.	-	F	)-
4.	)	F	)-)
5.	E	FE	)-)
6.	*	FE	)-)*
7.	D	FED	)-)*
8.	(	FED*	)-
9.	(	FED*-	
10.	*	FED*-	*
11.	)	FED*-	*)
12.	)	FED*-	*))
13.	C	FED*-C	*))
14.	/	FED*-C	*))/
15.	B	FED*-CB	*))/
16.	(	FED*-CB/	*)
17.	-	FED*-CB/	*)-
18.	A	FED*-CB/A	*)-
19.	(	FED*-CB/A-	*
		FED*-CB/A-*	

Hence, the required prefix expression is *-A/BC-*DEF

Evaluation of prefix expression:

Algorithm

1) Read prefix string from right to left until there is a data.
2) Repeat;
   If char is operand add to prestack
   If char is operator
   -operand 1= pop prestack.

-operand 2= pop prestack.
-result= value after applying operator between operand 1 and operand 2.
-push the result into prestack.

3) pop prestack get required value.

<u>Tracing</u>

+-*+12/421$42

S.N.	Scan Symbol	Operand 1	Operand 2	Value	Prestack
1.	2				2
2.	4				2,4
3.	$	4	2	16	16
4.	1				16,1
5.	2				16,1,2
6.	4				16,1,2,4
7.	/	4	2	2	16,1,2
8.	2				16,1,2,2
9.	1				16,1,2,2,1
10.	+	1	2	3	16,1,2,3
11.	*	3	2	6	16,1,6
12.	-	6	1	5	16,5
13.	+	5	16	21	21

Homework

Q. Convert the following infix expression into postfix expression.

1) (A+B)*C
2) (A+B)*C$E
3) ((A-(B+C))*D)$(E+F)

Q. Evaluate the following postfix expressions

1) AB+C-BA+C$-
2) ABC+*CBA-+*
   Where, A=1, B=2, C=3.

Q. Convert the following infix expression into prefix expression.

1) A/B$C-D
2) ((A+B)*C/D-E)+F$G
3) (A*B+(C/D))-F

Q. Evaluate the following prefix expression

a) -/A$BCD

b) *-A/BC-*DEF

where, A=2, B=1, C=4, D=3, E=5, F=1