**Infix Postfix Prefix Conversion using stack**

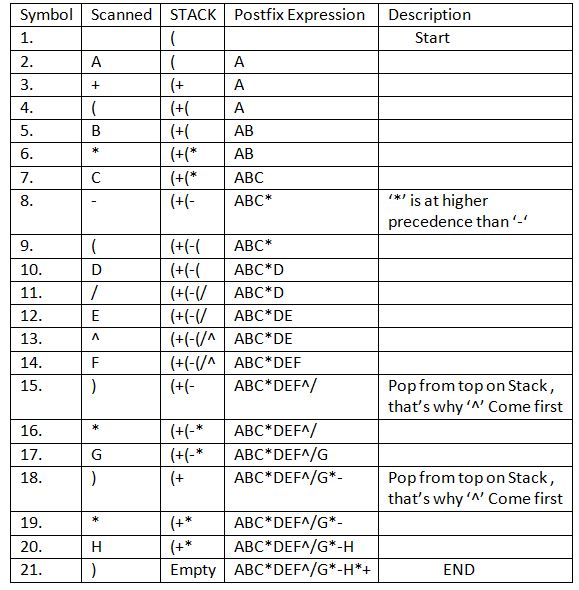
| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| --- | --- | --- |
| A + B | + A B | A B + |
| A + B \* C | + A \* B C | A B C \* + |
| (A + B) \* C | \* + A B C | A B + C \* |
|  | | |
| A + B \* C + D | + + A \* B C D | A B C \* + D + |
| (A + B) \* (C + D) | \* + A B + C D | A B + C D + \* |
| A \* B + C \* D | + \* A B \* C D | A B \* C D \* + |
| A + B + C + D | + + + A B C D | A B + C + D + |

Algorithm to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered, then:
   1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
   2. Add operator to Stack.  
      [End of If]
6. If a right parenthesis is encountered, then:
   1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
   2. Remove the left Parenthesis.  
      [End of If]  
      [End of If]
7. END.

Infix Expression: **A+ (B\*C-(D/E^F)\*G)\*H**, where **^** is an exponential operator.



**Resultant Postfix Expression: ABC\*DEF^/G\*-H\*+**

Infix Expression: **A+ (B\*C-(D/E^F)\*G)\*H**, where **^** is an exponential operator.

**Resultant Postfix Expression: ABC\*DEF^/G\*-H\*+**

(A + B \* C + D)

A + BC\* + D

ABC\*+ + D

ABC\*+D+

A + \*BC + D

+A\*BC + D

++A\*BCD

A\*B + (C/D) / E-F

Prefix: -+\*AB//CDEF

Postfix: AB\*CD/E/+F-

X^Y/(5\*Z) + 2

Prefix: +/^XY\*5Z2

Postfix: XY^5Z\*/2+