| **Infix** | **Postfix** | **Prefix** |
| --- | --- | --- |
| A+B | AB+ | +AB |
| A+B-C | AB+C- | -+ABC |
| A\*(B-C) | ABC-\* | \*A-BC |
| A + B \* (C – D) / E + F | ABCD-\*E/+F+ | ++A/\*B-CDEF |
| X = A + B \* C – D^E / (F + G) | X ABC\*+ DE^ FG+/ - = | =X-+A\*BC /^DE+FG |
| 5+6\*2 | 562\*+ | +5\*62 |
| (5+6)\*2 | 56+2\* | \*+562 |
| (565+65)\*22 | 565,65,+,22,\* | \*,+,565,65,22 |

Home Work

X = (A + B \* (C – D)) ^ E / (F + G)

(P - Q ^ (C – D)) + E - F \* G

Converting infix To …

* Infix to postfix (inToPo)
* Infix to prefix (inToPre)

Evaluation of Prefix / Postfix Expresssion

* Prefix to infix (preToIn)
* Postfix to infix (PoToIn)

Evaluate the postfix expression: 562\*+

## Algorithm : Postfix to Infix

1. Scan the given postfix expression from left to right
2. If token is Operand push in on to the stack
3. If token is Operator
   1. Opnd1 = pop()
   2. Opnd2 = pop()
   3. Push back ( Opnd2 operator Opnd1 )
4. When the Postfix expression is over pop and print (there should be only one value on top of the stack)

## Algorithm : Prefix to Infix

1. Scan the given postfix expression from right to left
2. If token is Operand push in on to the stack
3. If token is Operator
   1. Opnd1 = pop()
   2. Opnd2 = pop()
   3. Push back ( Opnd1 operator Opnd2 )
4. When the Postfix expression is over pop and print (there should be only one value on top of the stack)

## Algorithm: Infix to postfix (inToPo)

1. Attach ‘)’ at the end of Infix expression and push ‘(‘ in the stack
2. Scan the given Infix expression from left to right, If token is
   1. Operand : copy it to the output string
   2. Operator :
      1. While stack\_top\_operator\_priority >= incoming\_operator\_priority
         * Pop from stack and print all the operators
         * i. e. Repeatedly pop from STACK each operator (on the top of STACK) which has same or higher precedence than the incoming operator.
      2. Push the incoming operator
   3. ‘)’ : pop and print till an ‘(‘ is encountered and remove ‘(‘
   4. ‘(‘ : push on the stack
3. (at the end of the infix expression the operator stack should be empty)

## Algorithm: Infix to Prefix (inToPre)

1. Attach ‘(’ at the start of Infix expression and push ‘)’ in the stack
2. Scan the given Infix expression from Right to Left, If token is
   1. Operand : copy it to the output string
   2. Operator :
      1. While stack\_top\_operator\_priority > incoming\_operator\_priority
         * Pop from stack and print all the operators
         * i. e. Repeatedly pop from STACK each operator (on the top of STACK) which has higher precedence than the incoming operator.
      2. Push the incoming operator
   3. ‘(’ : pop and print till an ‘)‘ is encountered and remove ‘)‘
   4. ‘)‘ : push on the stack
3. (at the end of the infix expression the operator stack should be empty)
4. Reverse the output string