Infix to postfix - Application of stacks

Find the result of the following infix expression:

1. $3+5*7-4\wedge 2$

$3+5*7-16$

$3+35-16$

$38-16$

22

2. $500+5\wedge 3-(10+56)*9$

$500+5\wedge 3-66*9$

$500+125-66*9$

$625-594$

31

3. $3+5\wedge 6-7$
4. $254+10-9$
5. $365*52*7+10/10$
6. $3+5*(7-4)\wedge 2$
7. $8*(5\wedge 4+2)-6\wedge 2/(9*3)$.

Infix to postfix Expression Convert

1. A + B / C * D − E / (F + G).

A + B / C * D − E / (F G +)

A + BC / * D − E / (F G +)

A + BC / D * − E / (F G +)

A + BC / D * − E F G + /

A BC / D * + − E F G + /

ABC / D * + E F G + / −

How to Evaluate a postfix Expression

Consider Infix expression

$3 + 5 * (5 / 5) - 2 \wedge 2 = 4$

(1) (2)

Postfix expression

3 + 5 * 55 / − 2 ^ 2

3 + 5 * 55 / − 22 ^

3 + 555 / * − 22 ^

3555 / * + − 22 ^

3 5 5 5 5 / * + 2 2 ^ −

uation of postfix Expression

3 5 5 5 / * + 2 2 ^ -

1. 5 * 5 = 1

3 5 1 * + 2 2 ^ -

2. 5 * 1 = 5

3 5 + 2 2 ^ -

3. 3 + 5 = 8

8 2 2 ^ -

4. 2 ^ 2 = 4

8 4 -

5. 8 - 4 = 4

Postfix Expression conversion using stack

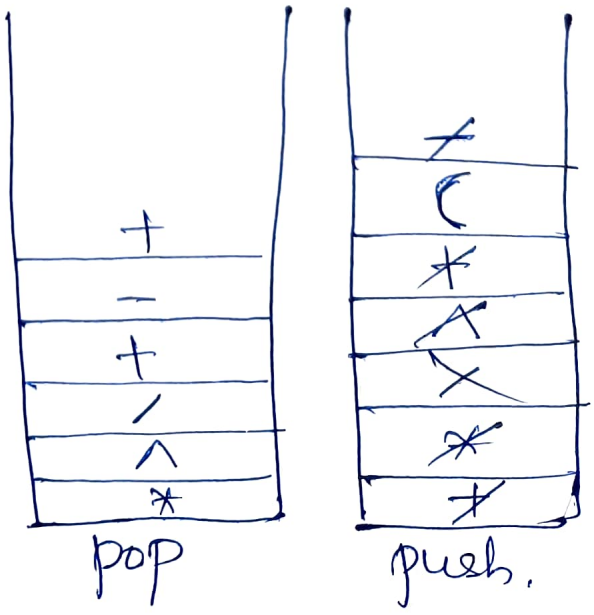
Consider the Infix expression

7 + 5 * 3 / 5 ^ 1 + (3 - 2).

1. Consider 7.
   7 is operand so print 7
2. Consider +
   + is operator, so push +
3. Consider 5
   5 is operand so print 5
4. Consider *
   + > *, so we can push *
5. Consider 3
   3 is operand so print 3.

print

7 5 3 * 5 ^ / + 3 2 - +



sider /

= /

o pop * and print *

o push / chuck

+ ≯ /

push /

7. print 5

8. Consider ^

/ ≯ ^

push ^

9. print 1

10. Consider +

^ > +

So pop ^ and print ^

/ > +

So pop / and print /

+ = +

So pop + and print +

Now push +

11. Consider (

When '(' operator simply push (

12. print 3

13. Consider −

(≠ −

So push −

14. print 2

15. ~~when~~ Consider ')' right operator when ')' operator, pop all operator in the stack

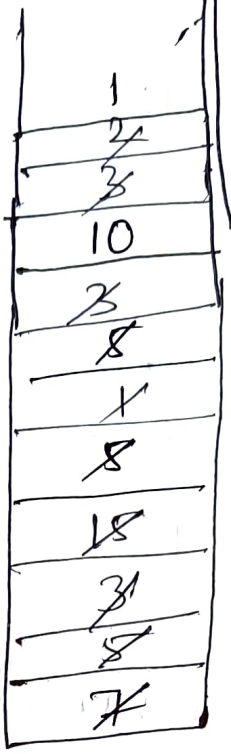
16. pop − & print −

17. '(' operator, pop it simply.

18. pop + & print +

Create the postfix expression

7 5 3 * 5 1 ^ / + 3 2 - +



B * A or
B + A or
B ^ A

1. push 7
2. push 5
3. push 3
4. we got * operator
   so pop last 2 symbols and perform evaluation

   pop 5 & 3 and perform *

   5 * 3 = 15

   And store back 15 on to the stack

   push 15
5. push 5
6. push 1
7. we encountred ^ operator.
   so pop 5 & 1 apply ^

   5 ^ 1 = 5

   push 5
8. we encountred / operator
   so pop 15 & 5 apply /

   15 / 5 = 3

9. we encountered +

pop 7 & 3.

7+3=10

10. push 3.

11. push 2

12. we encountered −

pop 3 & 2

3−2=1

push 1

13. we encountered +

pop 10 & 1

10+1=11

push 11.

14. print 11

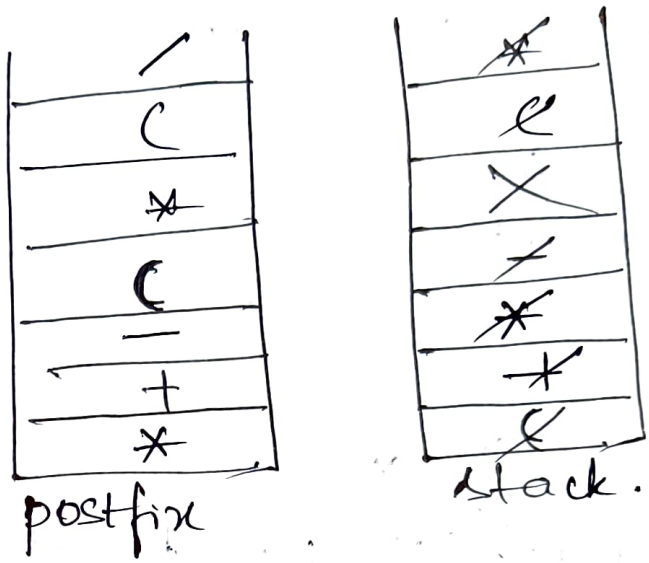
pl 2

x expression to postfix expression using stack.

(a+b*c-d)/(e*f)

~~print~~

1. '('
   push '('
2. 'a'
   print 'a'
3. '+'
   (≯ +
   push '+'
4. 'b'
   print b
5. '*'
   + ≯ *
   push '*'
6. 'c'
   print 'c'
7. '-'
   * > -
   pop * to postfix & print *
8. '-'
   + = -
   pop + to postfix & print +
   (≯ -
   so push '-' on to stack
9. 'd'
   print d
10. ')'
    pop - to postfix & print -
    (= pop



print

abc*+d-ef*/

10) '/'

Stack is empty, no need to compare, ju
push it on to stack.

11) '('

push to stack

12) 'e'

print e

13) '*'

C ∯ *

push '*' on to stack

14) 'f'

print f

15) ')'

pop * and print *

16) '('

pop '('

17) '/'

pop / and print /

Example 3

a+b*(c^d-e)^(f+g*h)-i

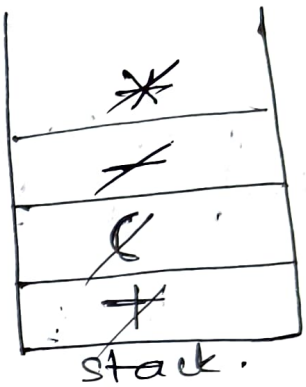
Tracing of Infix to postfix conversion

infix:

7	+	(	9	−	5	)	*	2	\0
0	1	2	3	4	5	6	7	8	

postfix

7	9	5	−	2	*	+	'\0'		



⇒ i = 0, j = 0

symbol = infix[0]

symbol = 7

⇒ i = 1, j = 1

symbol = infix[1].

symbol = + push(symbol) on to stack.

⇒ i = 2, j = 1

symbol = infix[2].

symbol = (

push(symbol) onto stack

⇒ i = 3, j = 1

symbol = infix[3].

symbol = 9

push(symbol) on to postfix array. j++

⇒ i = 4; j = 2

symbol = infix[4].

symbol = −

while (!IsEmpty() = True
&&
precedence(stack[top]) >= precedence(symbol)

precedence('(') >= precedence('-')

0 ≠ 1

push(symbol)
i.e., push('-') on to the stack.

⇒ i = 5, j = 2
symbol = infix[5]
symbol = 5
push(symbol = 5) onto the ~~stack~~ postfix array. j++.

⇒ i = 6, j = 3
symbol = infix[6].
symbol = ')'

while (next = pop() != '(')
next = - != '(' true.
pop '-' on insert it onto postfix
~~push~~

postfix[3++] = - & pop ')' from stack.
j = 4

7, J = 4

symbol = infix[7]

pop '+' that it.

⇒ I = 8, J = 4

symbol = infix[8]

⇒ I = 7, J = 4

symbol = infix[7].

symbol = *

while (!u.Empty() → True.

p(stack[top]) >= p(symbol)

+ >= * False.

push '+' onto the stack.

⇒ I = 8, J = 4

symbol = infix[8].

symbol = 2

push '2' onto the postfix array.

J = 5

⇒ I = 9, J = 5

9 ≠ 9.

⇒ while (!u.Empty() = True.

postfix[5] = pop() = * J = 6

⇒ while (!is Empty ()) → True.

postfix [6] = pop() = +

⇒ postfix [7] = '\0'.