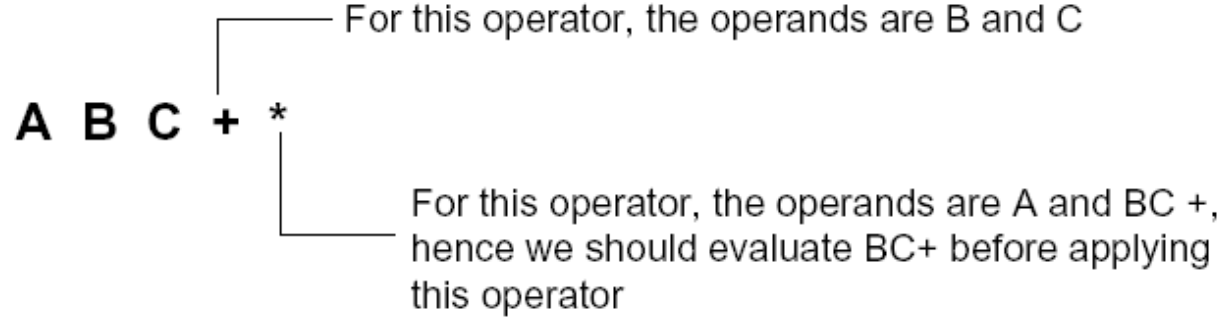
Evaluating the Postfix expression

Each operator in a postfix expression refers to the previous two operands in the expression.



To evaluate the postfix expression, we use the following procedure: Each time we read an operand we push it onto a stack. When we reach an operator, its operand s will be the top two elements on the stack. We can then pop these two elements perform the indicated operation on them and push the result on the stack so that it will be available for use as an operand of the next operator.

Consider an example

3 4 5 * +
=3 20 +
=23 (answer)

Evaluating the given postfix expression:

6 2 3 + - 3 8 2 / + * 2 $ 3 +
=6 5 - 3 8 2 / + * 2 $ 3 +
=1 3 8 2 / + * 2 $ 3 +
=1 3 4 + * 2 $ 3 +
=1 7 * 2 $ 3 +
=7 2 $ 3 +
=49 3 +
= 52

Algorithm to evaluate the postfix expression

Here we use only one stack called vstack(value stack).
1. Scan one character at a time from left to right of given postfix expression
1.1 if scanned symbol is operand then
read its corresponding value and push it into vstack

1.2 if scanned symbol is operator then
– pop and place into op2
– op and place into op1
– compute result according to given operator and push result into vstack
2. pop and display which is required value of the given postfix expression
3. return

Trace of Evaluation:

Consider an example to evaluate the postfix expression tracing the algorithm

ABC+*CBA-+*

123+*321-+*

Scanned character	value	Op2	Op1	Result	vstack
A	1	……	…….	…….	1
B	2	……	……	…..	1 2
C	3	……	……	…..	1 2 3
+	…….	3	2	5	1 5
*	……	5	1	5	5
C	3	…….	……	……	5 3
B	2	…	…….		5 3 2
A	1	…….	…….	……	5 3 2 1
-	…….	1	2	1	5 3 1
+	…….	1	3	4	5 4
*	…….	4	5	20	20

Its final value is 20.

*Evaluate the postfix expression AB + C * D/if A = 2, B = 3, C = 4 and D = 5.*

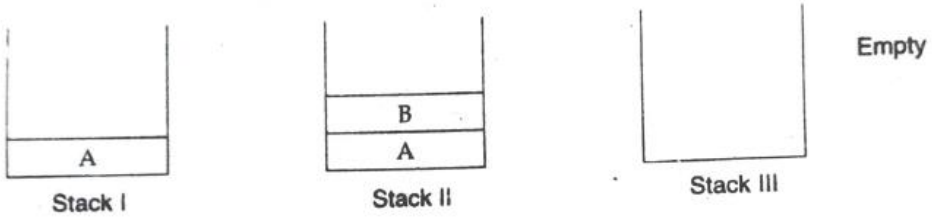
SOLUTION

I. First element is operand A, push A into stack, (see Stack I)

II. Second element is also operand B, push B also into the stack, (see Stack II)

III. Third element '+' is an operand, pop 2 operands from the stack *i.e.*, A and B (see Stack III) and evaluate the expression, *i.e.*,

$$A + B = 2 + 3 = 5$$

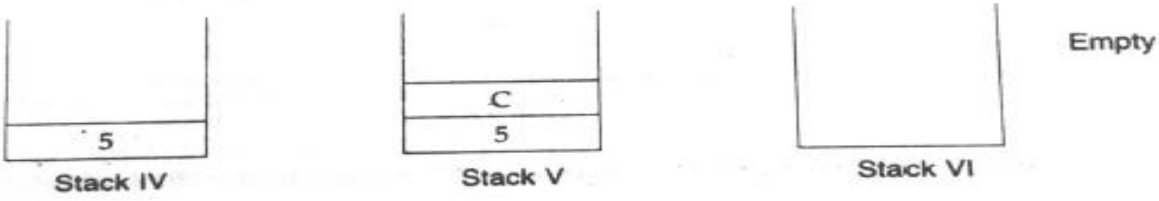


IV. Push the result (5) into the stack, (see Stack IV)

V. Next element C is operand ; pushed into the stack, (see Stack V)

VI. Next 'D' is operator, pop 2 operands from the stack *i.e.*, 5 and C (see Stack VI) and

$$5 \times C = 5 \times 4 = 20$$



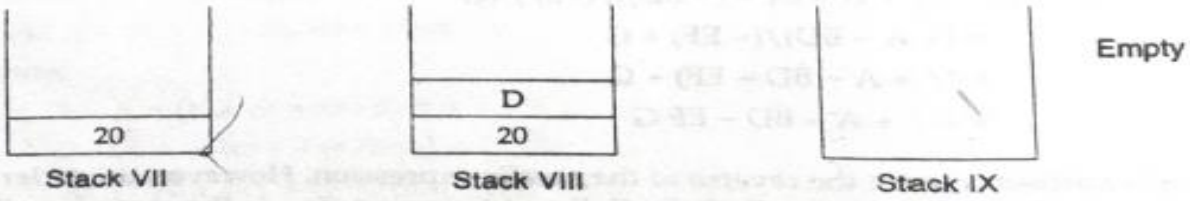
VII. Push the result (20) into the stack, (Stack VII)

VIII. Next 'D' is operand ; pushed into the stack, (Stack VIII)

IX. Next '/' is operator ;

2 operand popped *i.e.*, D and 20 ; (Stack IX)

Expression evaluated as follows : $\frac{20}{D} = \frac{20}{5} = 4$



X. End of the expression. Thus the result 4.

Converting an Infix expression to Prefix expression

The precedence rule for converting from an expression from infix to prefix are identical. Only changes from postfix conversion is that the operator is placed before the operands rather than after them. The prefix of
A+B-C is -+ABC.
A+B-C (infix)
=(+AB)-C
=-+ABC (prefix)
Example Consider an example:
A $ B * C – D + E / F / (G + H) infix form
= A $ B * C – D + E / F /(+GH)
=$AB* C – D + E / F /(+GH)
=*$ABC-D+E/F/(+GH)
=*$ABC-D+(/EF)/(+GH)
=*$ABC-D+//EF+GH
= (-*$ABCD) + (//EF+GH)
=+-*$ABCD//EF+GH which is in prefix form.