Prefix, Postfix, Infix Notation

VKS-LEARNING HUB

Infix Notation

- To add A, B, we write

  A+B
- To multiply A, B, we write

  A*B
- The operators ('+' and '*') go in between the operands ('A' and 'B')
- This is *"Infix"* notation.

VKS-LEARNING HUB

Prefix Notation

- Instead of saying "A plus B", we could say "add A,B " and write

  + A B
- "Multiply A,B" would be written

  * A B
- This is *Prefix* notation.

VKS-LEARNING HUB

Postfix Notation

- Another alternative is to put the operators after the operands as in

  A B +

and

  A B *

- This is *Postfix* notation.

VKS-LEARNING HUB

Pre A In B Post

- The terms infix, prefix, and postfix tell us whether the operators go between, before, or after the operands.

VKS-LEARNING HUB

Parentheses

- Evaluate 2+3*5.
- + First:

  (2+3)*5 = 5*5 = 25
- * First:

  2+(3*5) = 2+15 = 17
- Infix notation requires Parentheses.

VKS-LEARNING HUB

What about Prefix Notation?

- + 2 * 3 5 =
  = + 2 * 3 5
  = + 2 15 = 17
- * + 2 3 5 =
  = * + 2 3 5
  = * 5 5 = 25
- No parentheses needed!

VKS-LEARNING HUB

Postfix Notation

- 2 3 5 * + =
  = 2 3 5 * +
  = 2 15 + = 17
- 2 3 + 5 * =
  = 2 3 + 5 *
  = 5 5 * = 25
- No parentheses needed here either!

VKS-LEARNING HUB

Conclusion:

- Infix is the only notation that requires parentheses in order to change the order in which the operations are done.

VKS-LEARNING HUB

Fully Parenthesized Expression

- A FPE has exactly one set of Parentheses enclosing each operator and its operands.
- Which is fully parenthesized?

(A + B) * C

((A + B) * C)

((A + B) * (C))

VKS-LEARNING HUB

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

((A + B) * (C + D))

VKS-LEARNING HUB

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

(+ A B * (C + D))

VKS-LEARNING HUB

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)

VKS-LEARNING HUB

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C D

Order of operands does not change!

VKS-LEARNING HUB

Infix to Postfix

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

A B + C * D E + F / -

- Operand order does not change!
- Operators are in order of evaluation!

VKS-LEARNING HUB

Summary of the rules follows:

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

VKS-LEARNING HUB

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

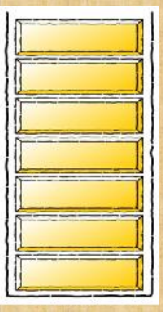
6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

VKS-LEARNING HUB

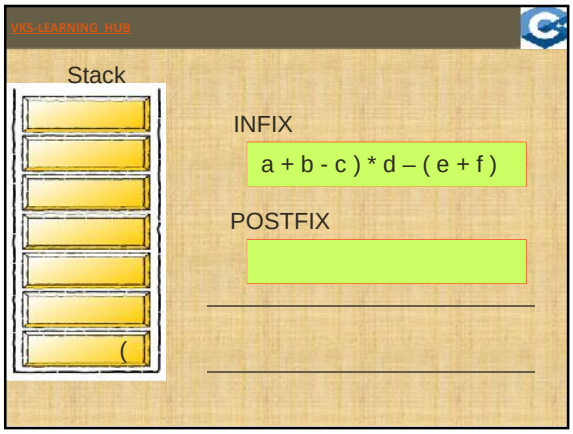
Infix to postfix conversion

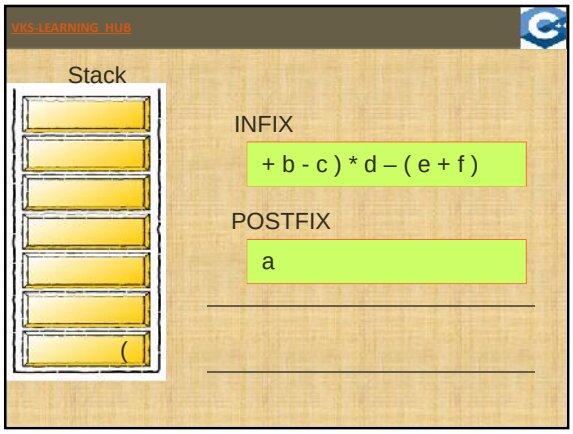


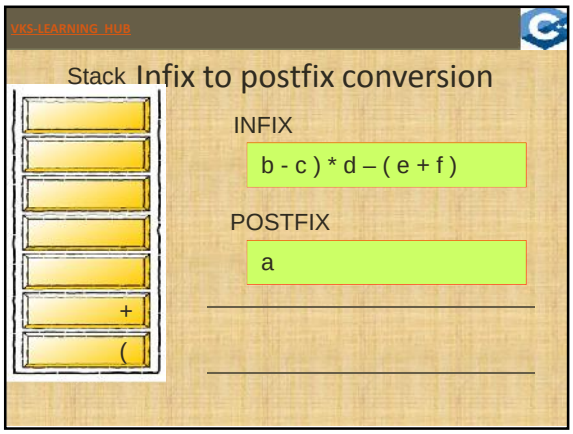
INFIX

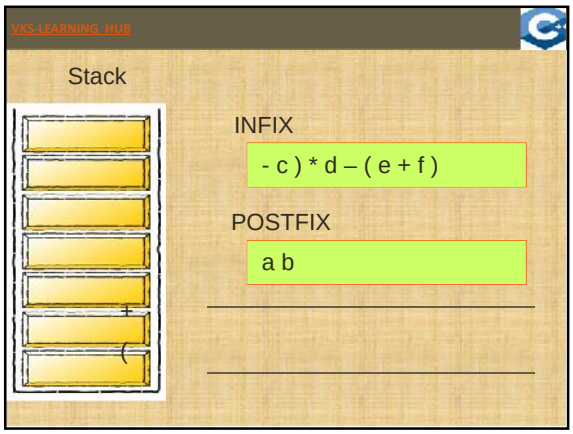
(a + b - c) * d – (e + f)

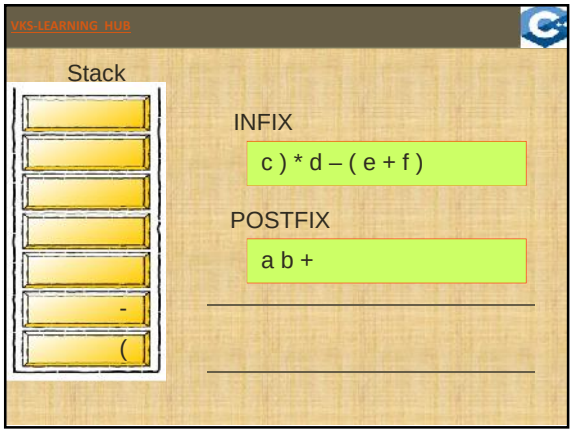
POST

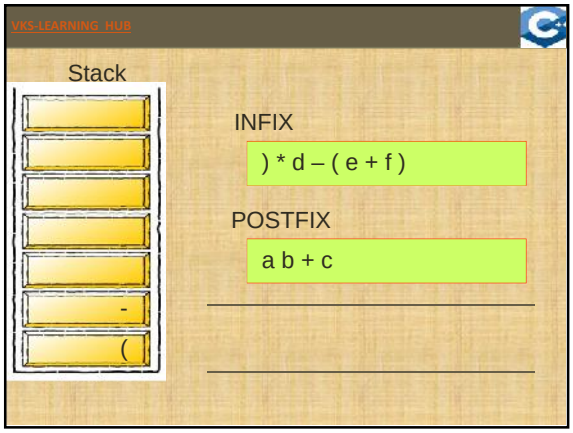

VKS-LEARNING HUB
Stack
(
INFIX
a + b - c) * d – (e + f)
POSTFIX

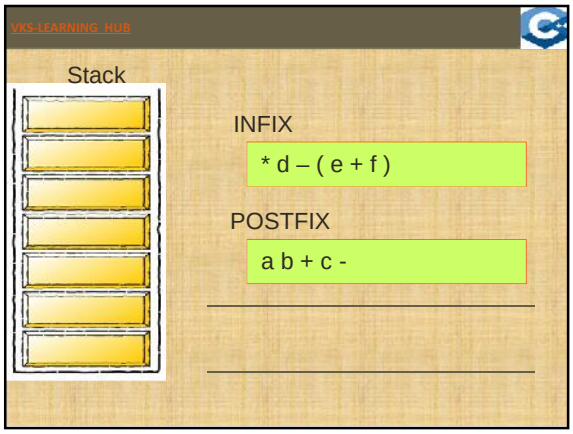

VKS-LEARNING HUB
Stack
(
INFIX
+ b - c) * d – (e + f)
POSTFIX
a

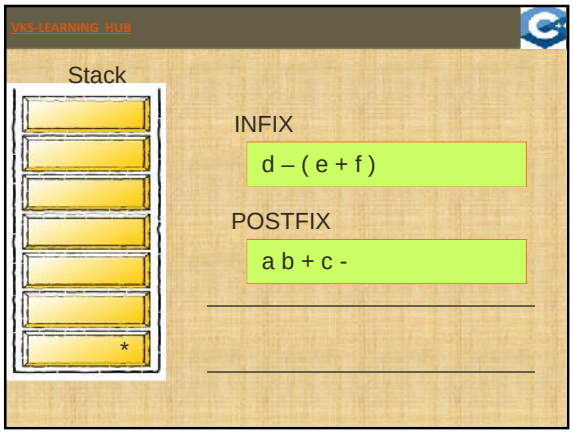

VKS-LEARNING HUB
Stack
Infix to postfix conversion
+
(
INFIX
b - c) * d – (e + f)
POSTFIX
a

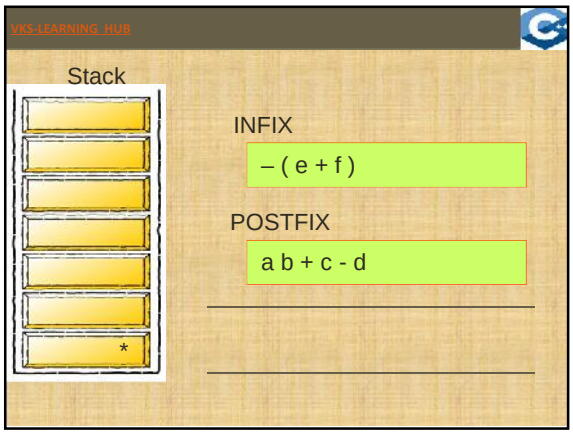

VKS-LEARNING HUB
Stack
+
(
INFIX
- c) * d – (e + f)
POSTFIX
a b

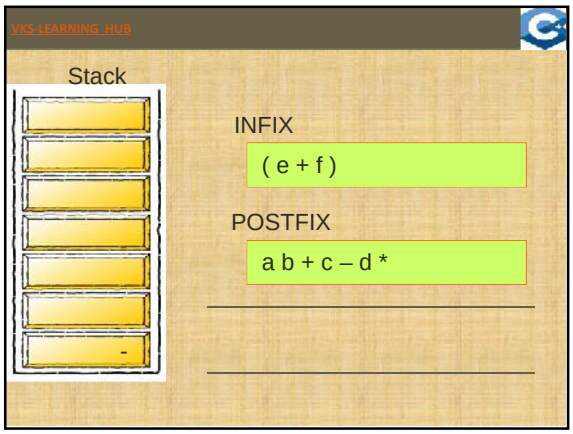

VKS-LEARNING HUB
Stack
-
(
INFIX
c) * d – (e + f)
POSTFIX
a b +

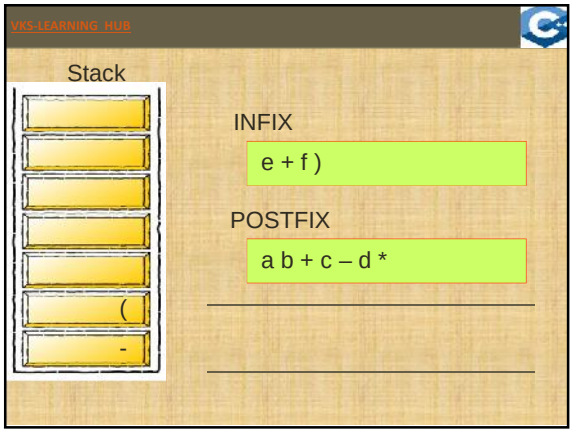

VKS-LEARNING HUB
Stack
-
(
INFIX
) * d – (e + f)
POSTFIX
a b + c

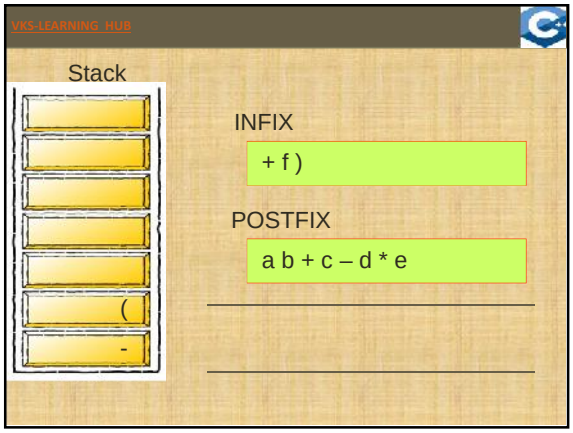

VKS-LEARNING HUB
Stack
INFIX
* d – (e + f)
POSTFIX
a b + c -


VKS-LEARNING HUB
Stack
*
INFIX
d – (e + f)
POSTFIX
a b + c -


VKS-LEARNING HUB
Stack
*
INFIX
– (e + f)
POSTFIX
a b + c - d

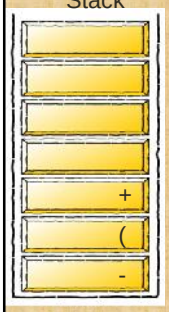

VKS-LEARNING HUB
Stack
-
INFIX
(e + f)
POSTFIX
a b + c – d *


VKS-LEARNING HUB
Stack
(
-
INFIX
e + f)
POSTFIX
a b + c – d *


VKS-LEARNING HUB
Stack
(
-
INFIX
+ f)
POSTFIX
a b + c – d * e

VKS-LEARNING HUB

Infix to postfix conversion



INFIX

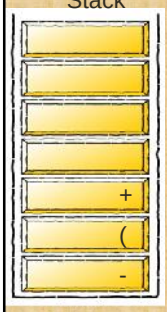
f)

POSTFIX

a b + c – d * e

VKS-LEARNING HUB

Infix to postfix conversion



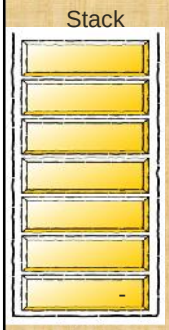
INFIX

)

POSTFIX

a b + c – d * e f

VKS-LEARNING HUB

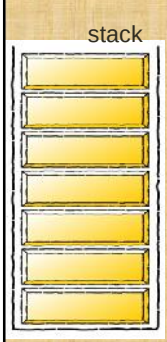


INFIX

POSTFIX

a b + c – d * e f +

VKS-LEARNING HUB



INFIX

POSTFIX

a b + c – d * e f + -

VKS-LEARNING HUB

INFIX	POSTFIX
(A + B) * C + D / (E + F * G) - H	
(A * B + C) / D - E / (F + G)	
A - B - C * (D + E / F - G) - H	
A + ((B - C * D) / E) + F - G / H	
(A * B - (C - D)) / (E + F)	
A * B ^ C + D	

VKS-LEARNING HUB

INFIX	POSTFIX
(A + B) * C + D / (E + F * G) - H	A B + C * D E F G * + / + H -
(A * B + C) / D - E / (F + G)	A B * C + D / E F G + / -
A - B - C * (D + E / F - G) - H	A B - C D E F / + G - * - H -
A + ((B - C * D) / E) + F - G / H	A B C D * - E / + F + G H / -
(A * B - (C - D)) / (E + F)	AB*CD—EF+/
A * B ^ C + D	A B C ^ * D +

VKS-LEARNING HUB

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

INPUT	STACK	POSTFIX
(	(	
(	((	
A	((	A
+	((+	A
B	((+	AB
)	(	AB+
*	(*	AB+
C	(*	AB+C
+	(+	AB+C*
D	(+	AB+C*D
/	(+/	AB+C*D
(	(+/(	AB+C*D

VKS-LEARNING HUB

INPUT	STACK	POSTFIX
E	(+/(	AB+C*DE
+	(+/(+	AB+C*DE
F	(+/(+	AB+C*DEF
*	(+/(+*	AB+C*DEF
G	(+/(+*	AB+C*DEFG
)	(+/	AB+C*DEFG*+
-	(-	AB+C*DEFG*+/+
H	(-	AB+C*DEFG*+/+H
)	EMPTY	AB+C*DEFG*+/+H-

AB+C*DEFG*+/+H-

VKS-LEARNING HUB

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

INPUT	STACK	POSTFIX
(		
(		
A		
*		
B		
+		
C		
)		
/		
D		
-		
E		

VKS-LEARNING HUB

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

INPUT	STACK	POSTFIX
/		
(		
F		
+		
G		
)		
)		

A B * C + D / E F G + / -

VKS-LEARNING HUB

Evaluation of Postfix notation

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

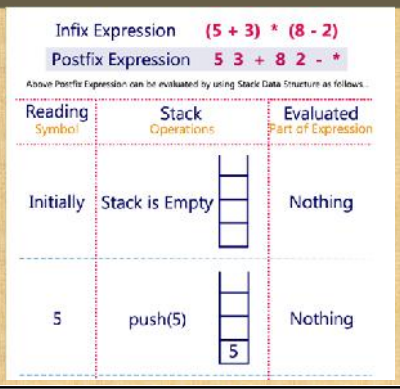
Keep the following points for evaluation postfix expressions.

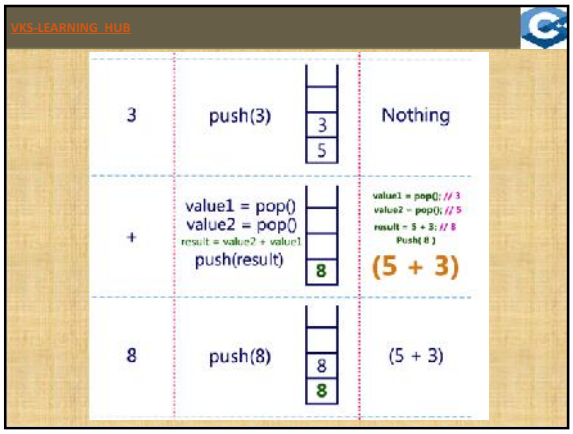
1) Create a stack to store operands (or values).
2) Scan the given expression and do following for every scanned element.

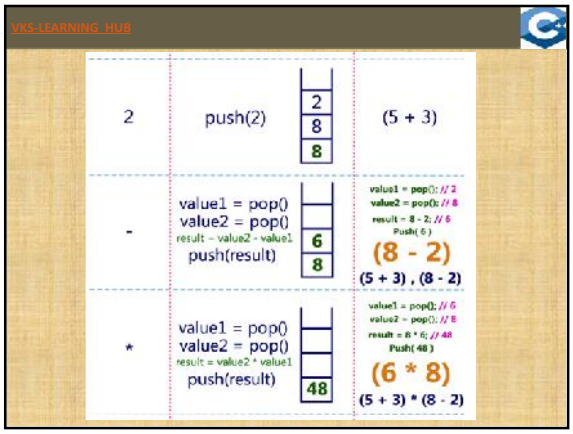
- **If the element is a number, push it into the stack**
- **f the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack**

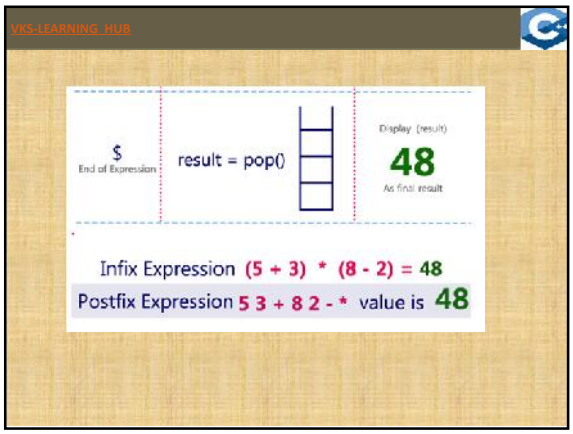
3) When the expression is ended, the number in the stack is the final answer

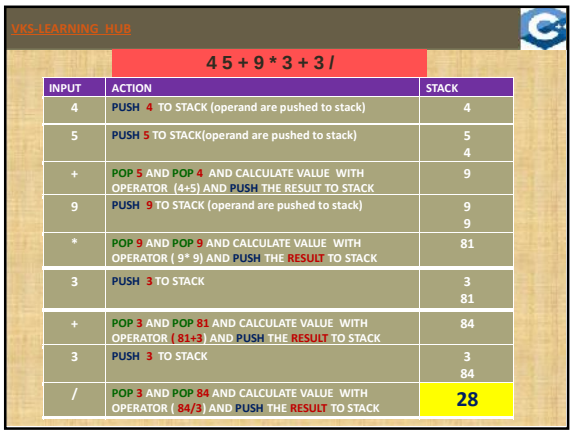
VKS-LEARNING HUB



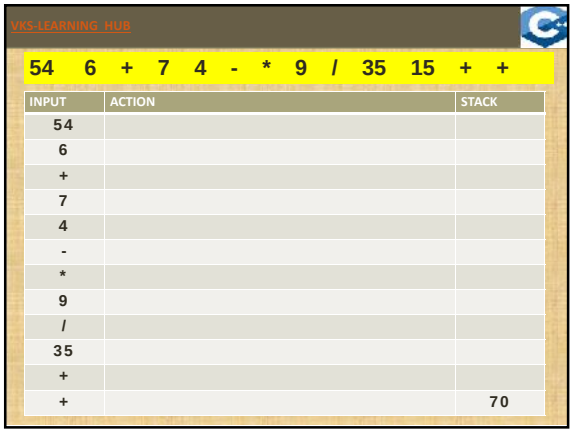








INPUT	ACTION	STACK
4	PUSH 4 TO STACK (operand are pushed to stack)	4
5	PUSH 5 TO STACK(operand are pushed to stack)	5 4
+	POP 5 AND POP 4 AND CALCULATE VALUE WITH OPERATOR (4+5) AND PUSH THE RESULT TO STACK	9
9	PUSH 9 TO STACK (operand are pushed to stack)	9 9
*	POP 9 AND POP 9 AND CALCULATE VALUE WITH OPERATOR (9* 9) AND PUSH THE RESULT TO STACK	81
3	PUSH 3 TO STACK	3 81
+	POP 3 AND POP 81 AND CALCULATE VALUE WITH OPERATOR (81+3) AND PUSH THE RESULT TO STACK	84
3	PUSH 3 TO STACK	3 84
/	POP 3 AND POP 84 AND CALCULATE VALUE WITH OPERATOR (84/3) AND PUSH THE RESULT TO STACK	28



INPUT	ACTION	STACK
54		
6		
+		
7		
4		
-		
*		
9		
/		
35		
+		
+		70