SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

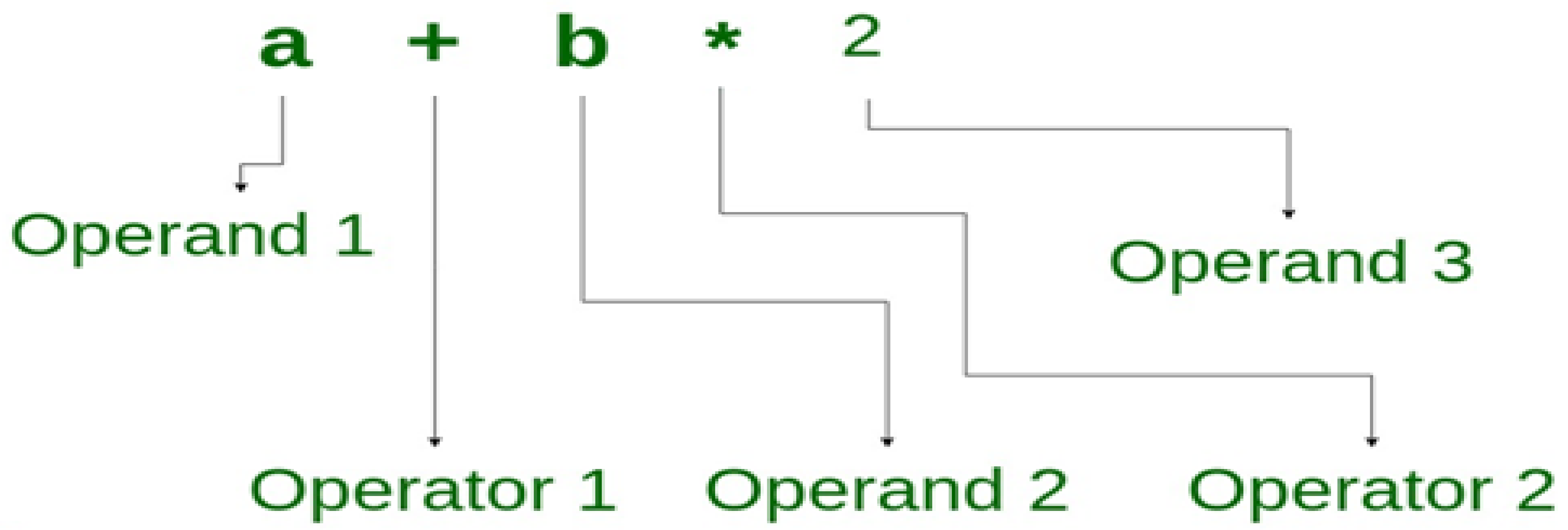
23ITT101-PROGRAMMING IN C AND DATA STRUCTURES

I YEAR - II SEM

UNIT-IV

Expression

An expression is a formula in which operands are linked to each other by the use of operators to compute a value



Expression

Three types of expression

1. Infix expression: X + Y

Operators are written in-between their operands

2. Postfix expression /Reverse Polish notation: X Y +

Operators are written after their operands

3. Prefix expression /Polish notation: + X Y

Operators are written before their operands

Example 1:
Convert Infix expression to A * B + C / D Postfix Expression

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

(A B*) +(C / D)

(AB *) +(C D/)

(AB *) (C D/)+

AB * C D/+

Example 2:
Convert Infix expression to A * B / C + D Postfix Expression

(A * B) / C + D

((A * B) / C) + D

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

i. (((AB*) / C) + D)
ii. ((AB* C/) + D)
iii. AB* C/ D+

Example 3:
Convert Infix expression to A + B - C / D Postfix Expression

A + B - C / D

A + B - (C / D)

(A + B) - (C / D)

((A + B) - (C / D))

i. ((A B+) - (C D/))

ii. A B+ C D/-

Example 4:

Convert Infix expression to A + (B *(C-D)/E) Postfix Expression

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

i. A + (B *(CD-)/E)

ii. A + (BCD-*/E)

iii. A + (BCD-*E/)

iv. A (BCD-*E/)+

ABCD-*E/+

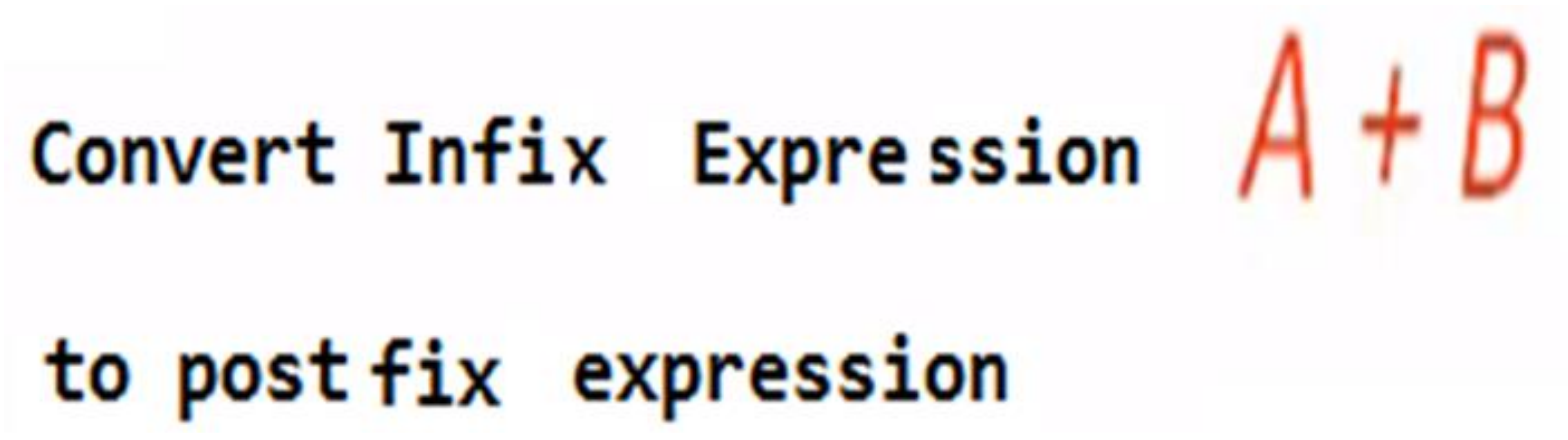
Algorithm to convert Infix To Postfix

1.Get a **Infix expression and Empty stack as input**
2.Scan the infix expression from left to right
3.If the scanned character is an **operand, output** it as postfix expression
4.If the scanned character is **an operator**

1. If the precedence of the scanned operator is **greater than** the precedence of the operator in the stack(or the **stack is empty** or the **stack contains a '('**), **push** it on to stack.
2. Else, **Pop all the operators from the stack which are greater than or equal** to in precedence than that of the scanned operator.
3. After doing that, **Push the scanned operator** to the stack

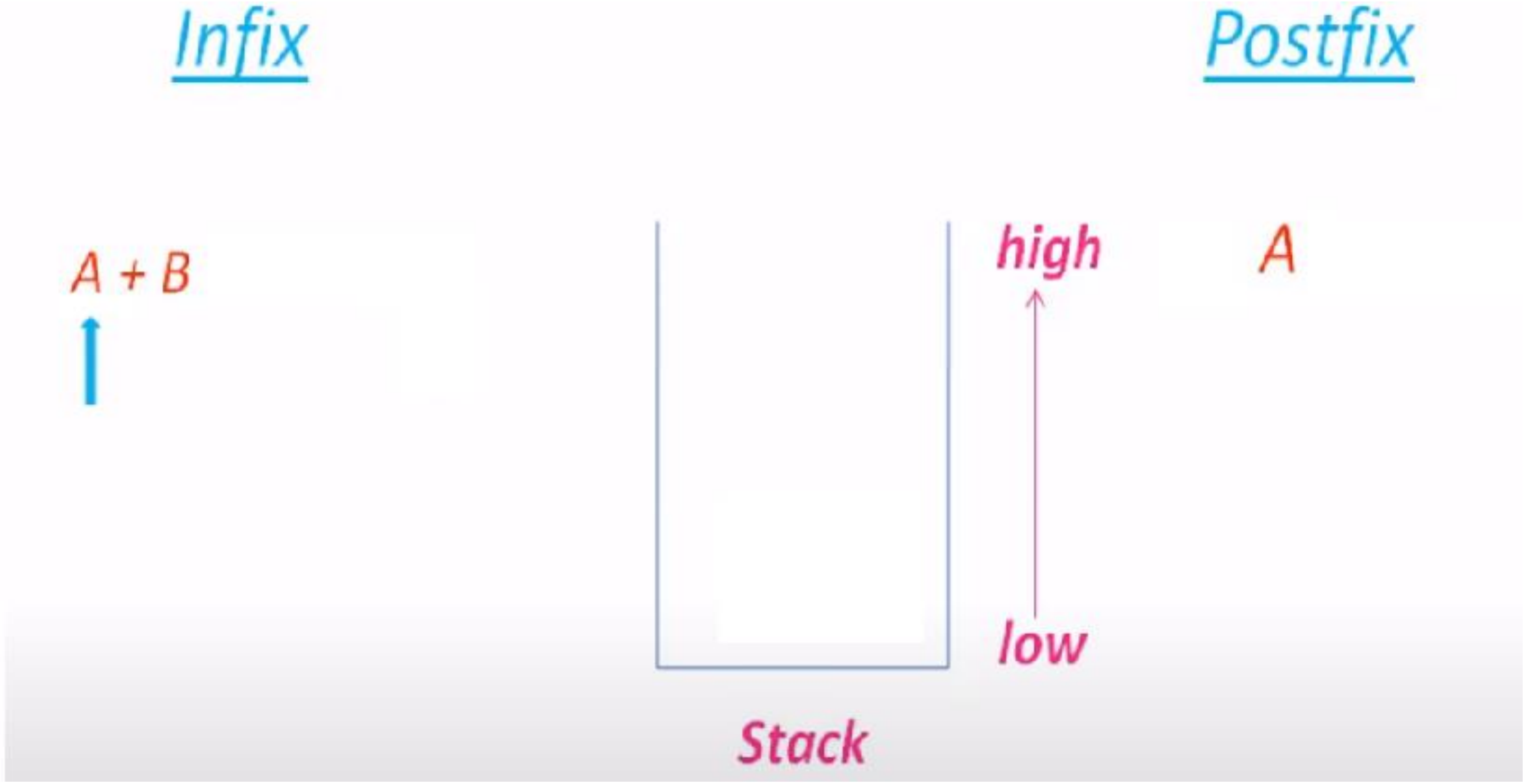
4. If the scanned character is an **'(', push** it to the stack.
5. If the scanned character is an **')', pop** the stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is fully scanned
7. Print the **output as postfix expression**
8. Pop and output from the stack until it is not empty.

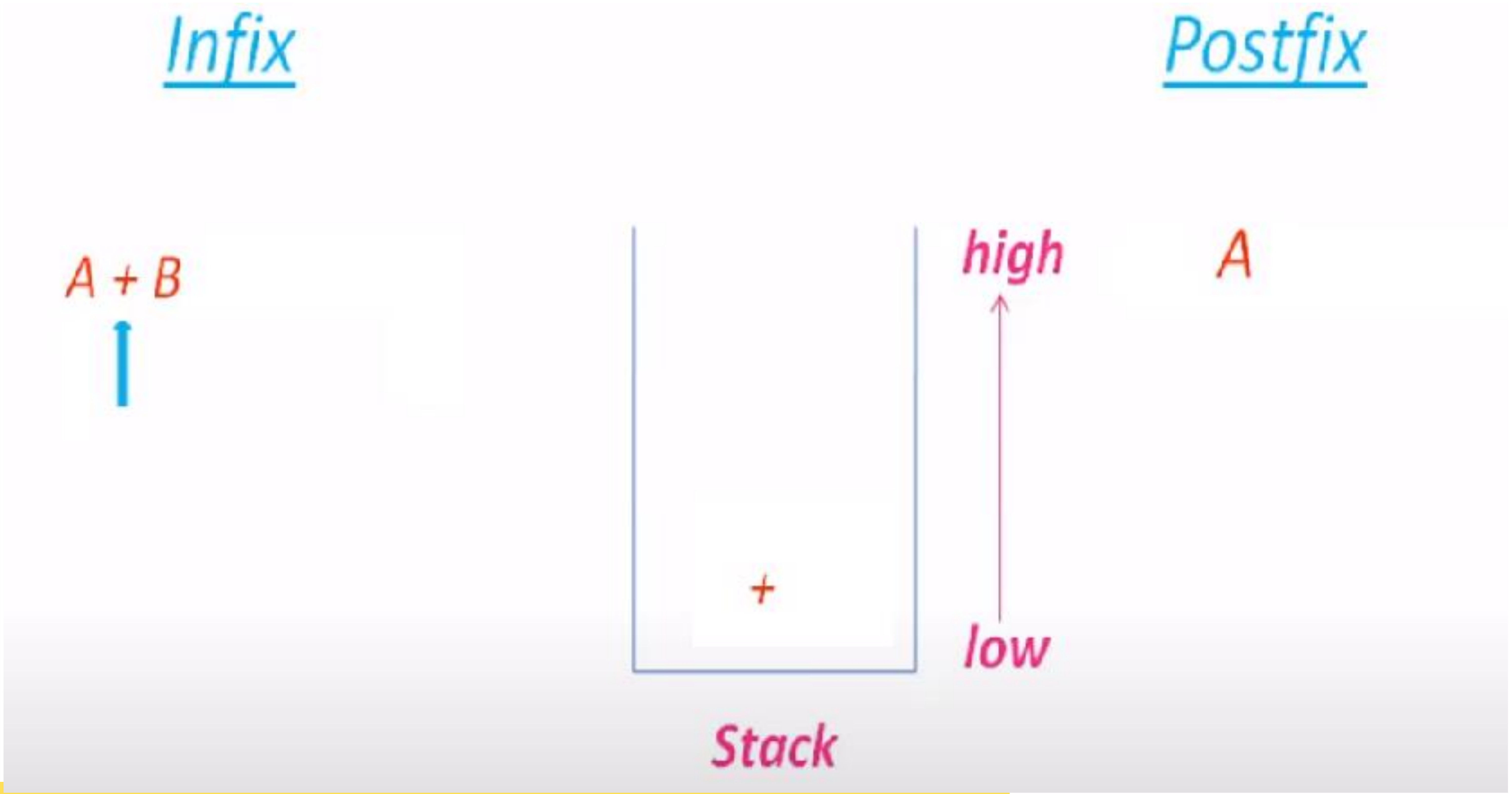
Example 1



Convert Infix Expression $A + B$

to postfix expression


Infix
Postfix
A + B
high
low
A
Stack

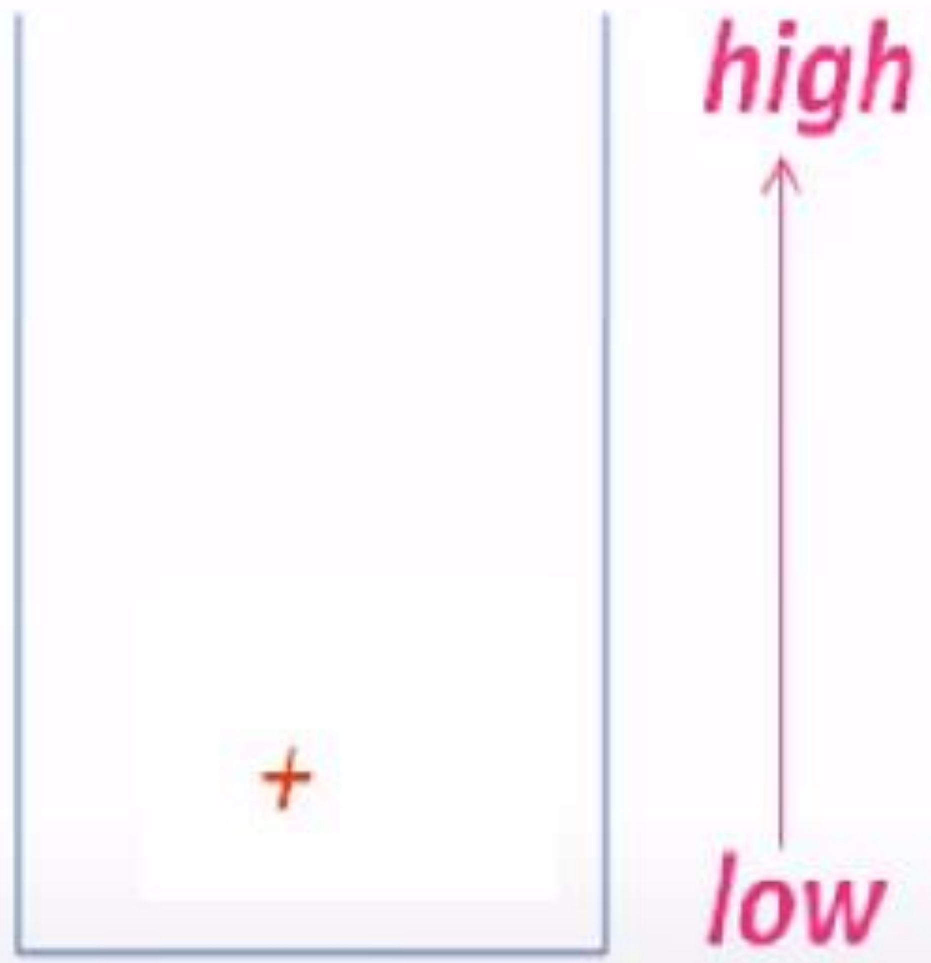

Infix
Postfix
A + B
high
low
+
A
Stack

Infix

A + B

Postfix

A B



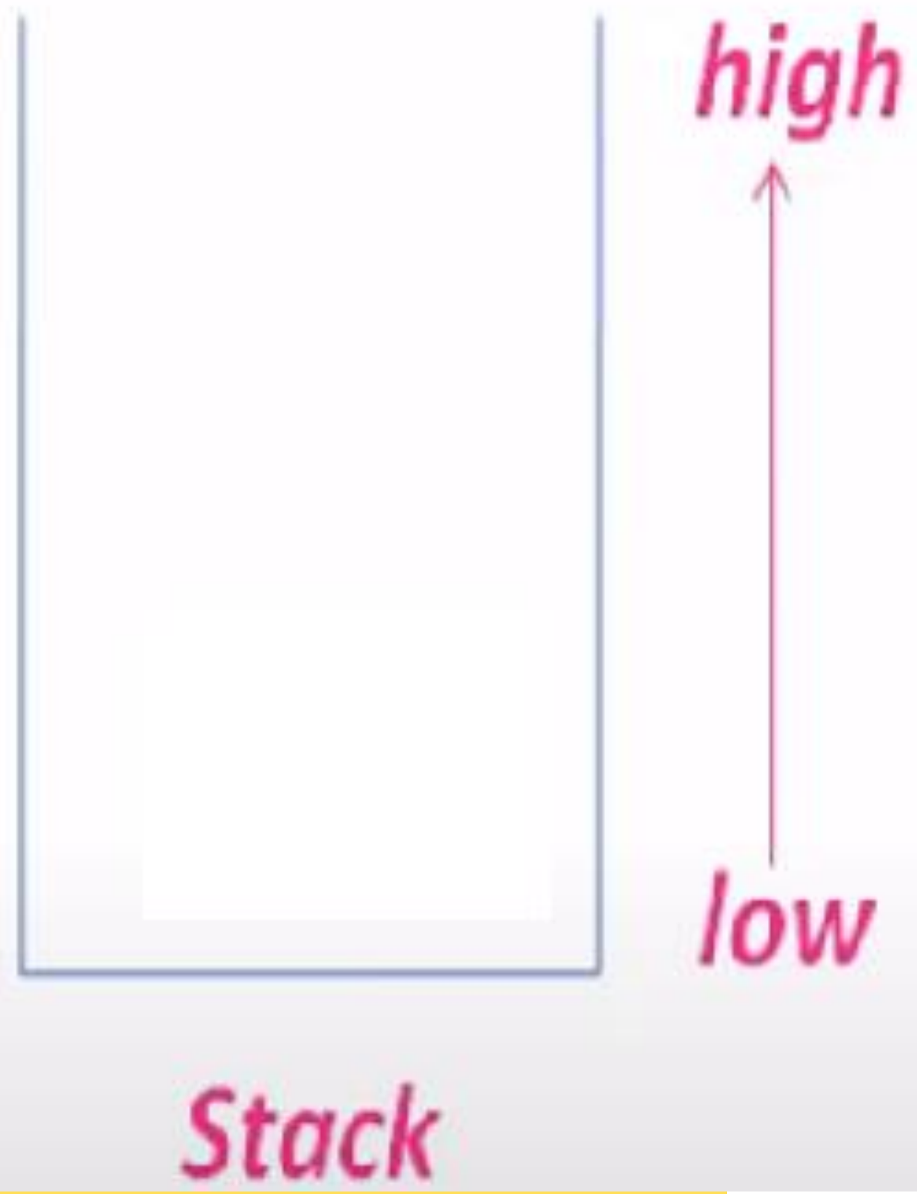
Stack

Infix

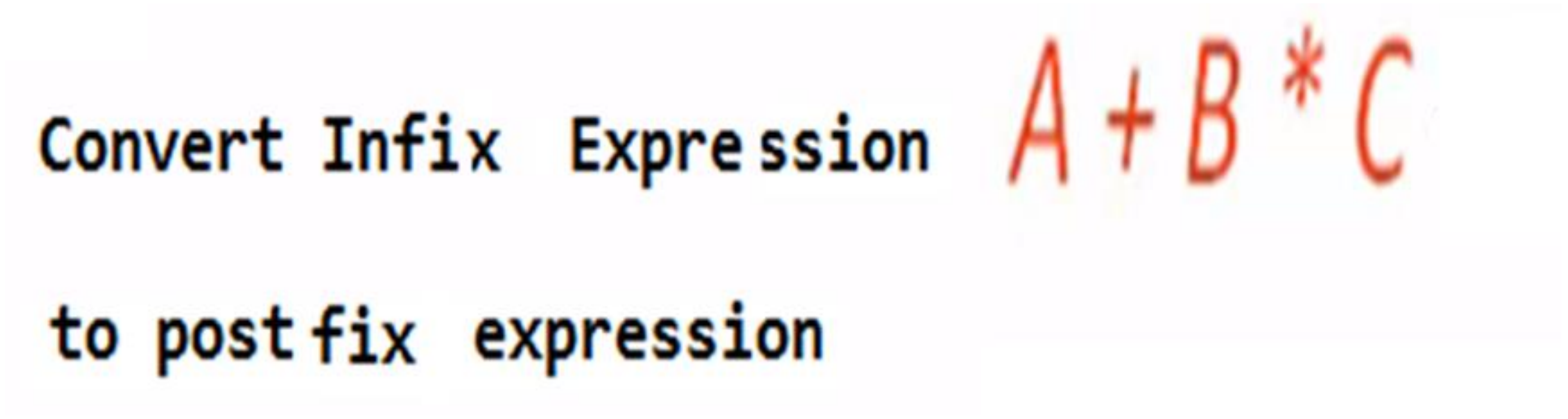
Postfix

A + B

A B +

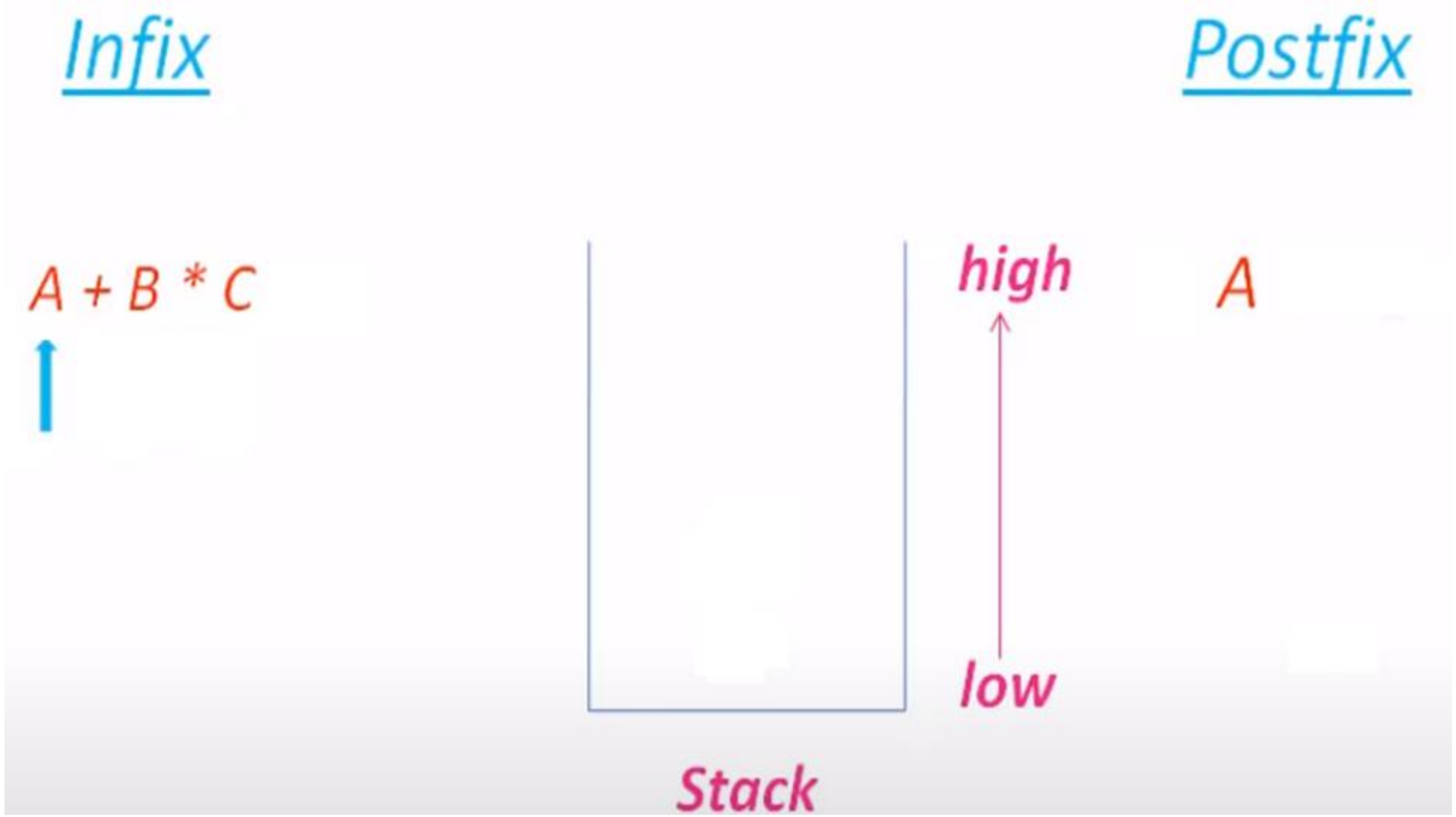


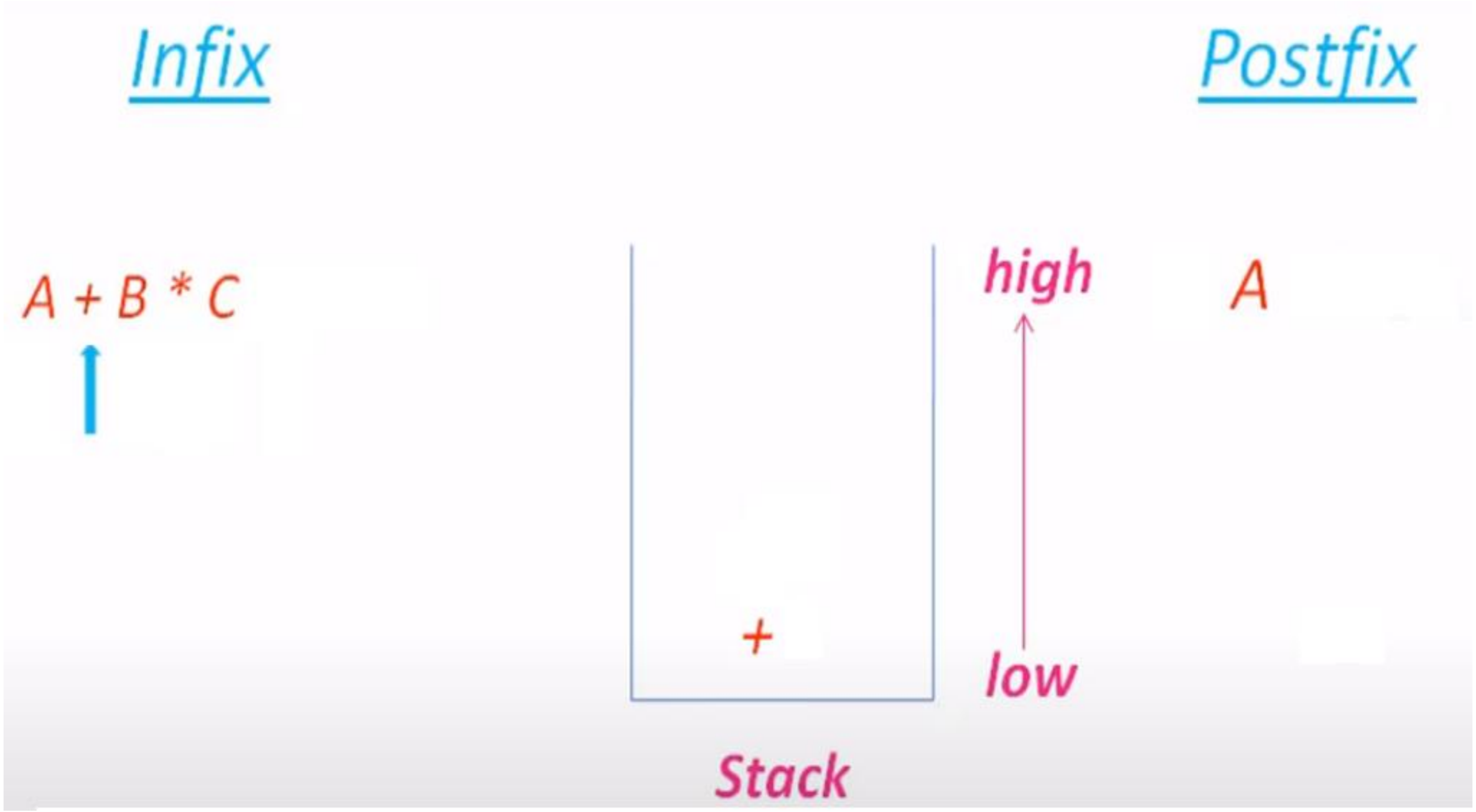
Example 2

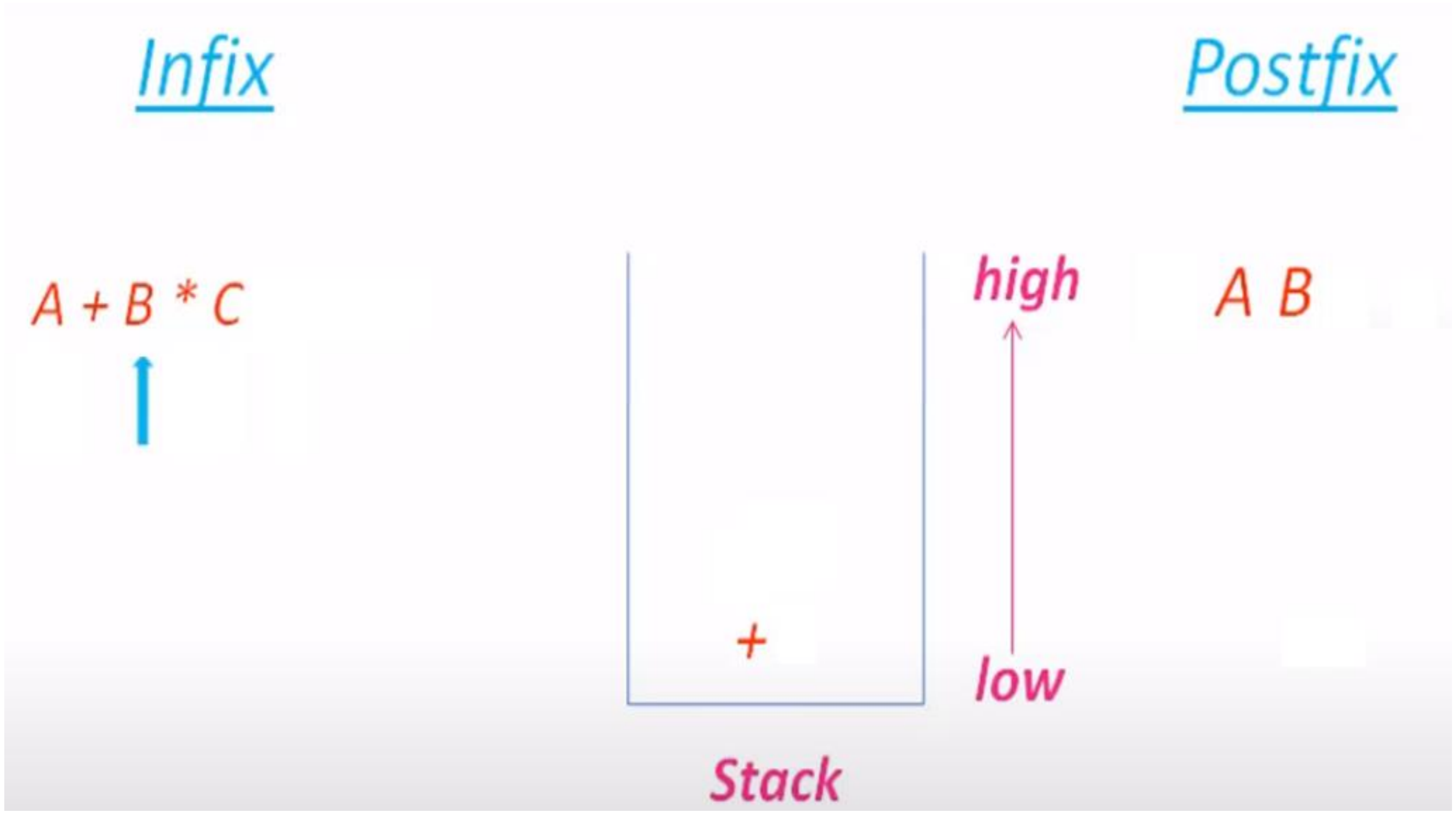


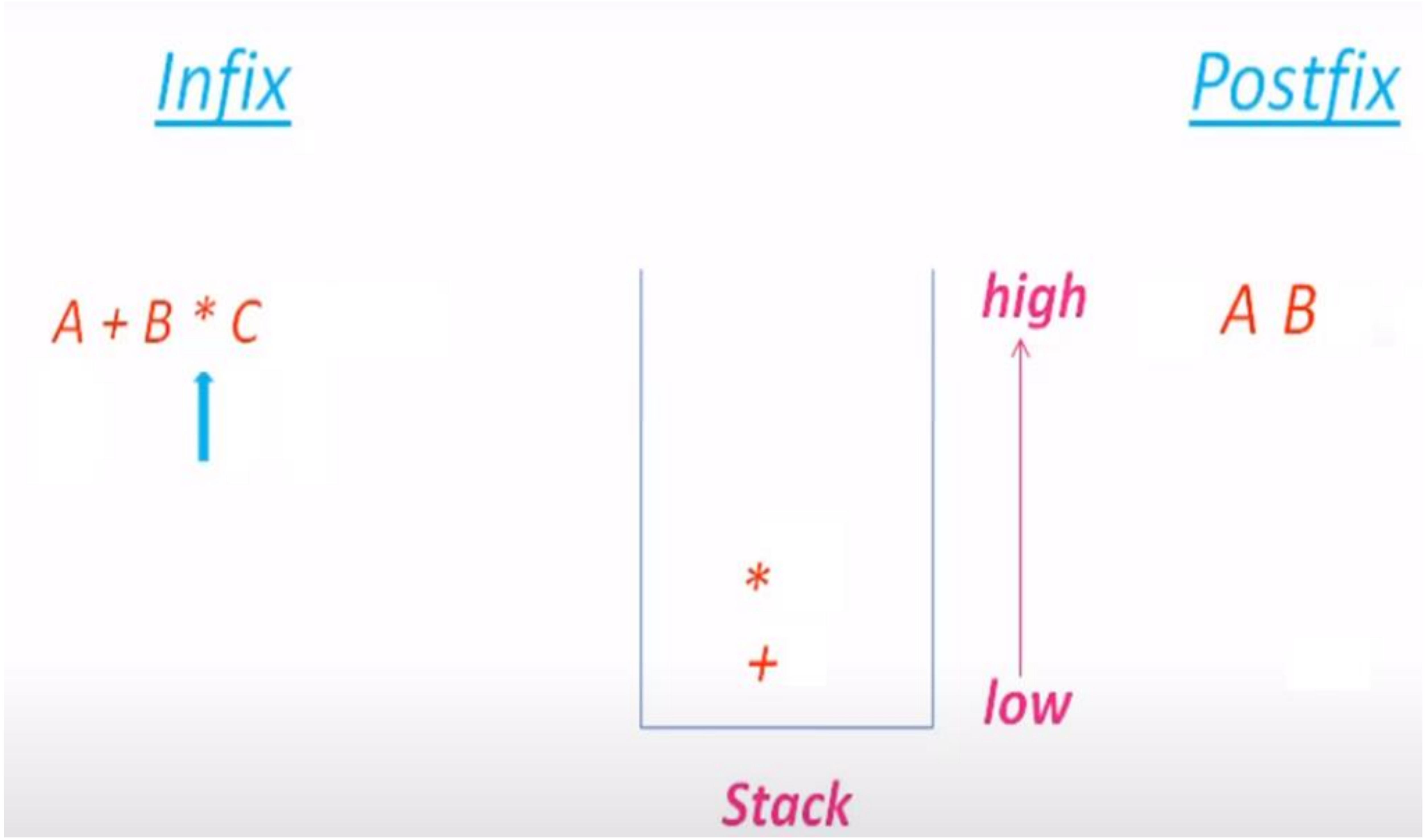
Convert Infix Expression $A + B * C$

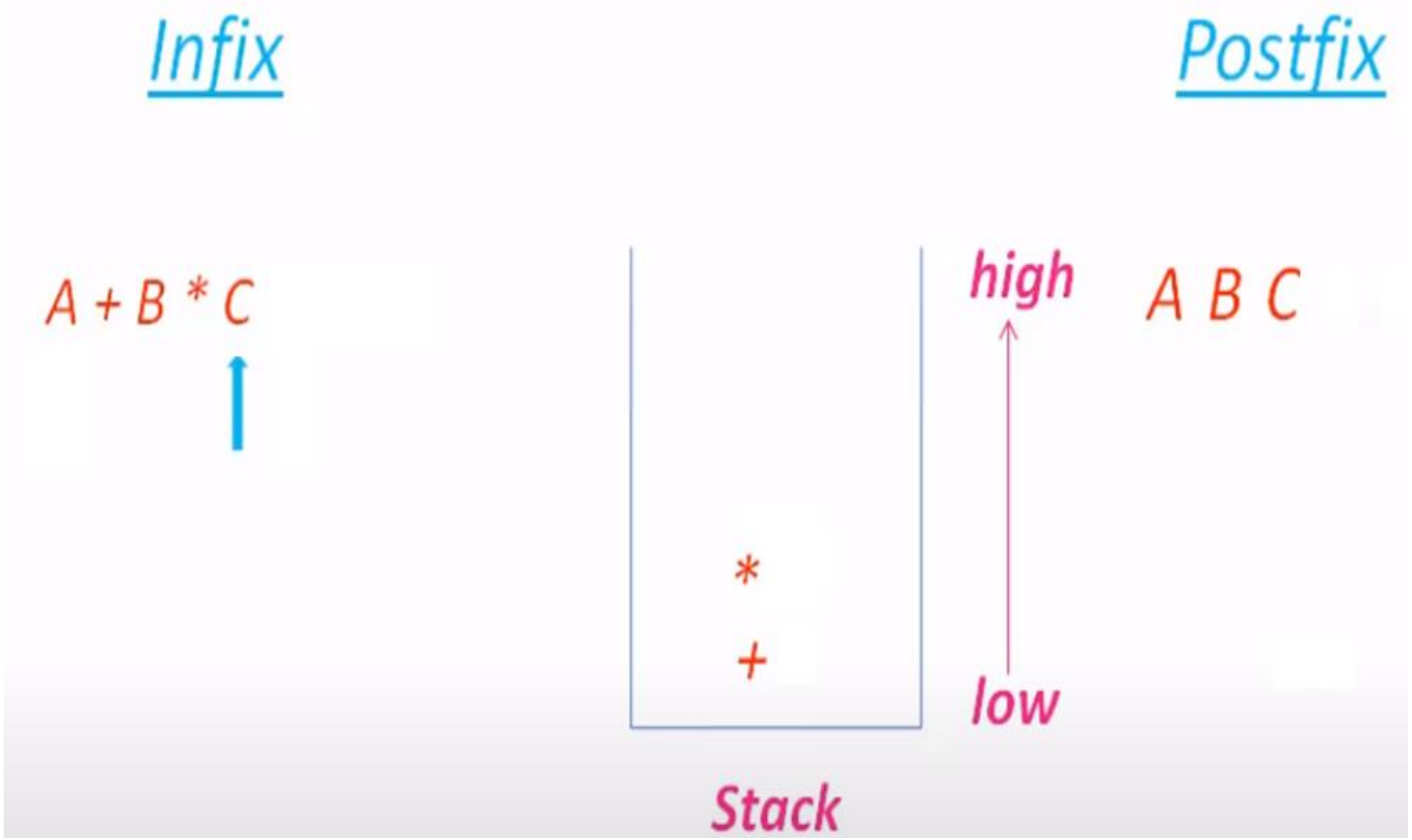
to postfix expression

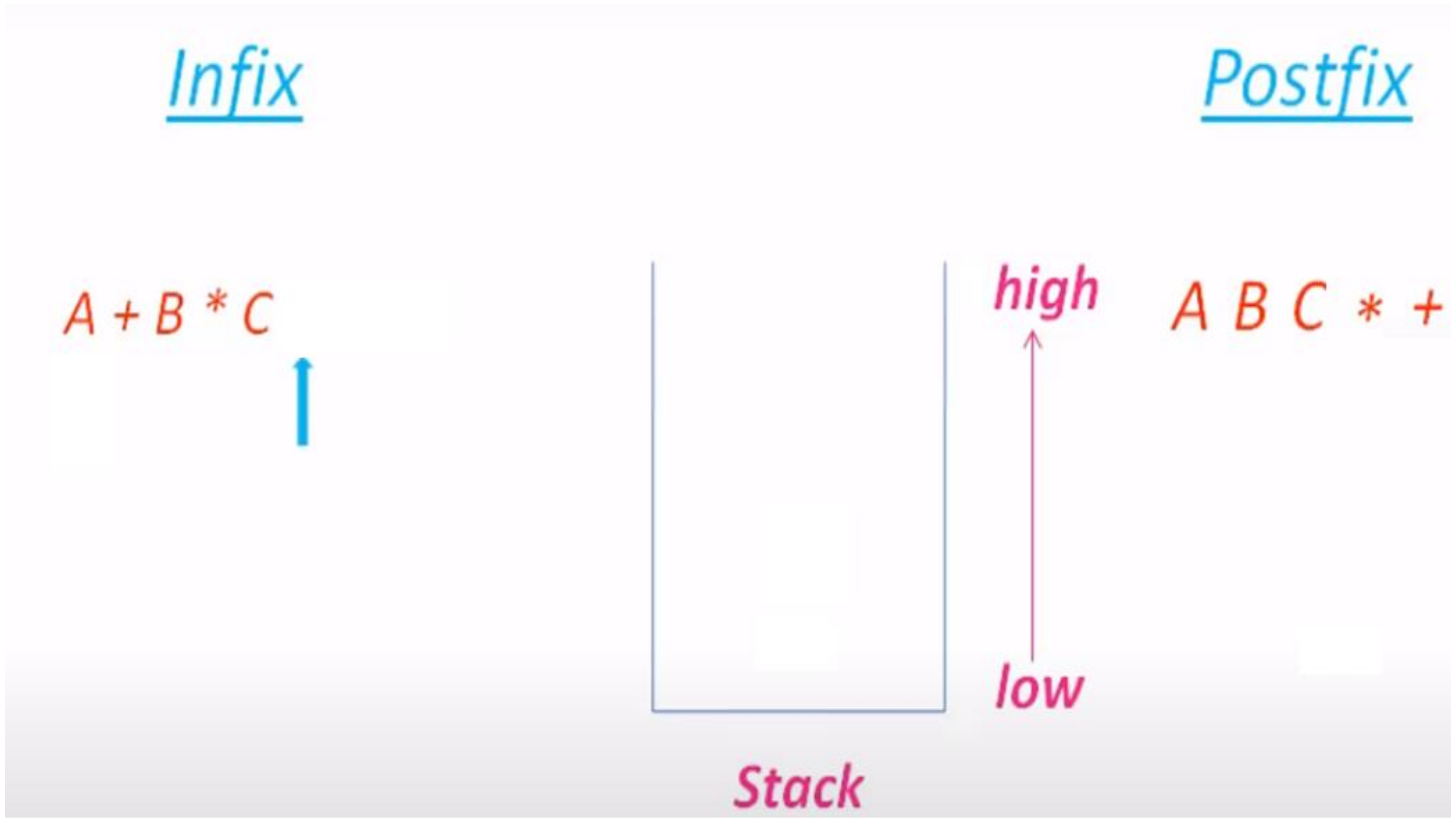

Infix
Postfix
A + B * C
high
low
A
Stack


Infix
Postfix
A + B * C
high
low
+
A
Stack

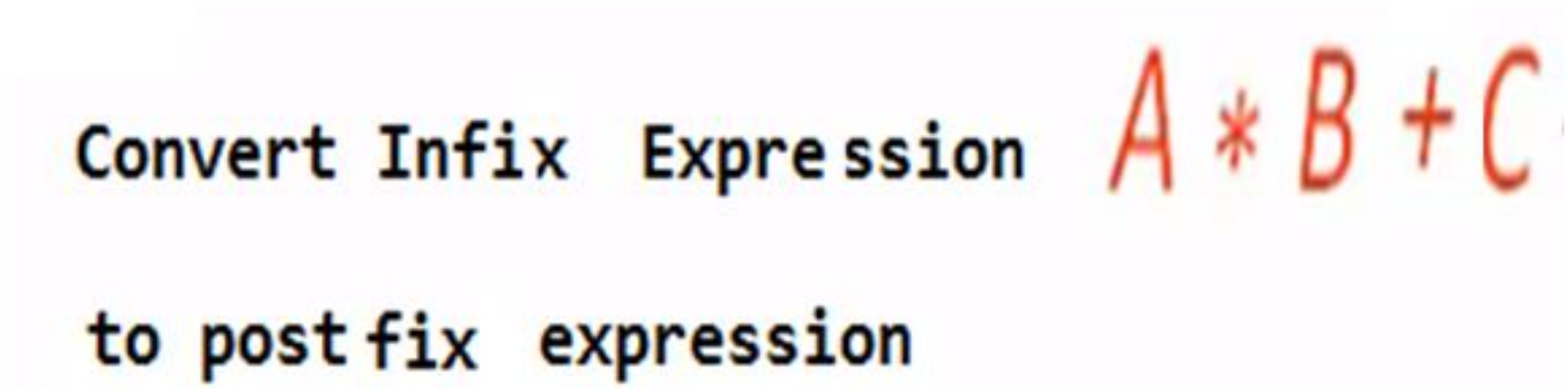

Infix
Postfix
A + B * C
high
low
+
Stack
A B


Infix
Postfix
A + B * C
high
low
A B
*
+
Stack

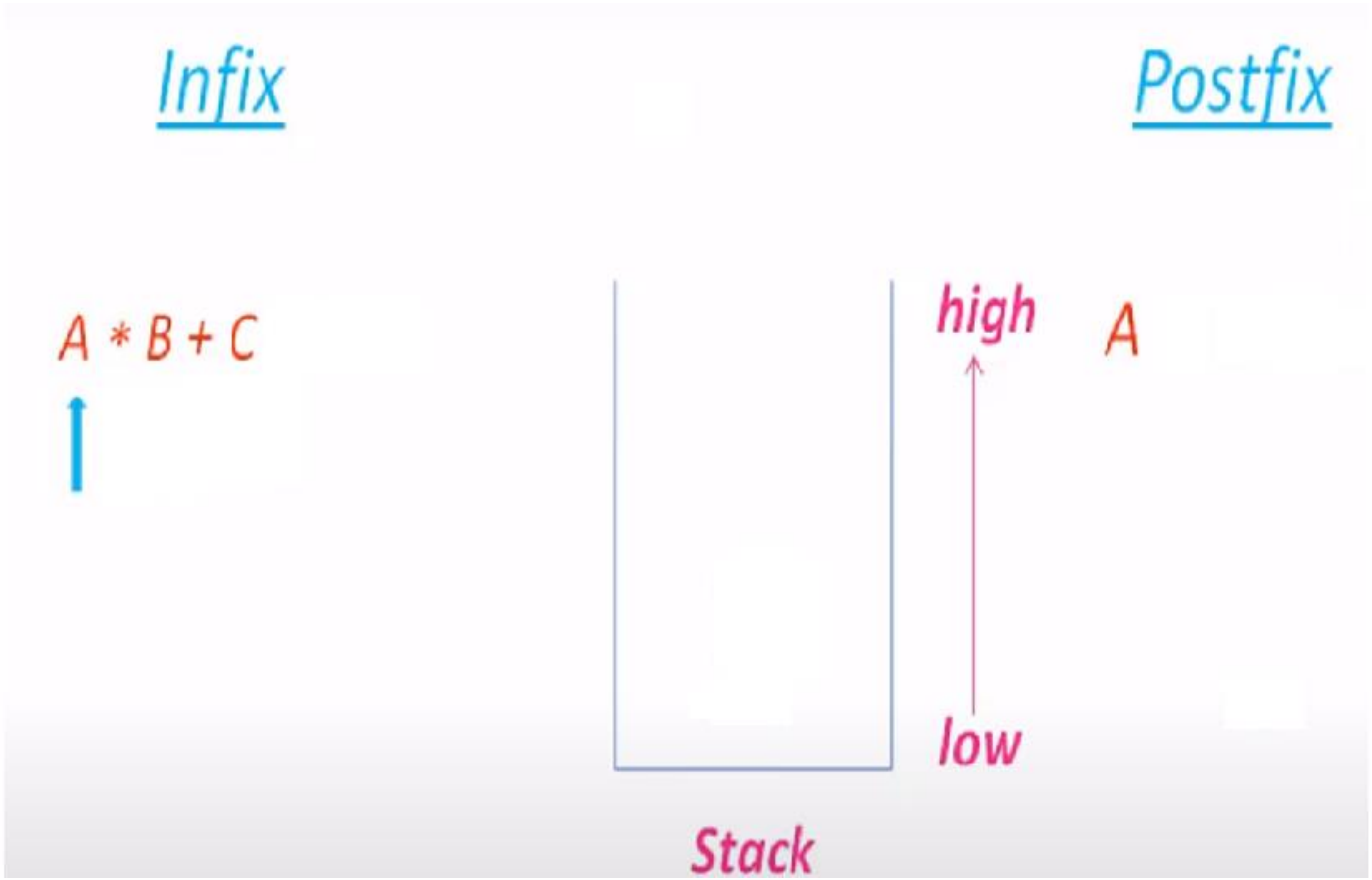

Infix
Postfix
A + B * C
high
low
A B C
*
+
Stack

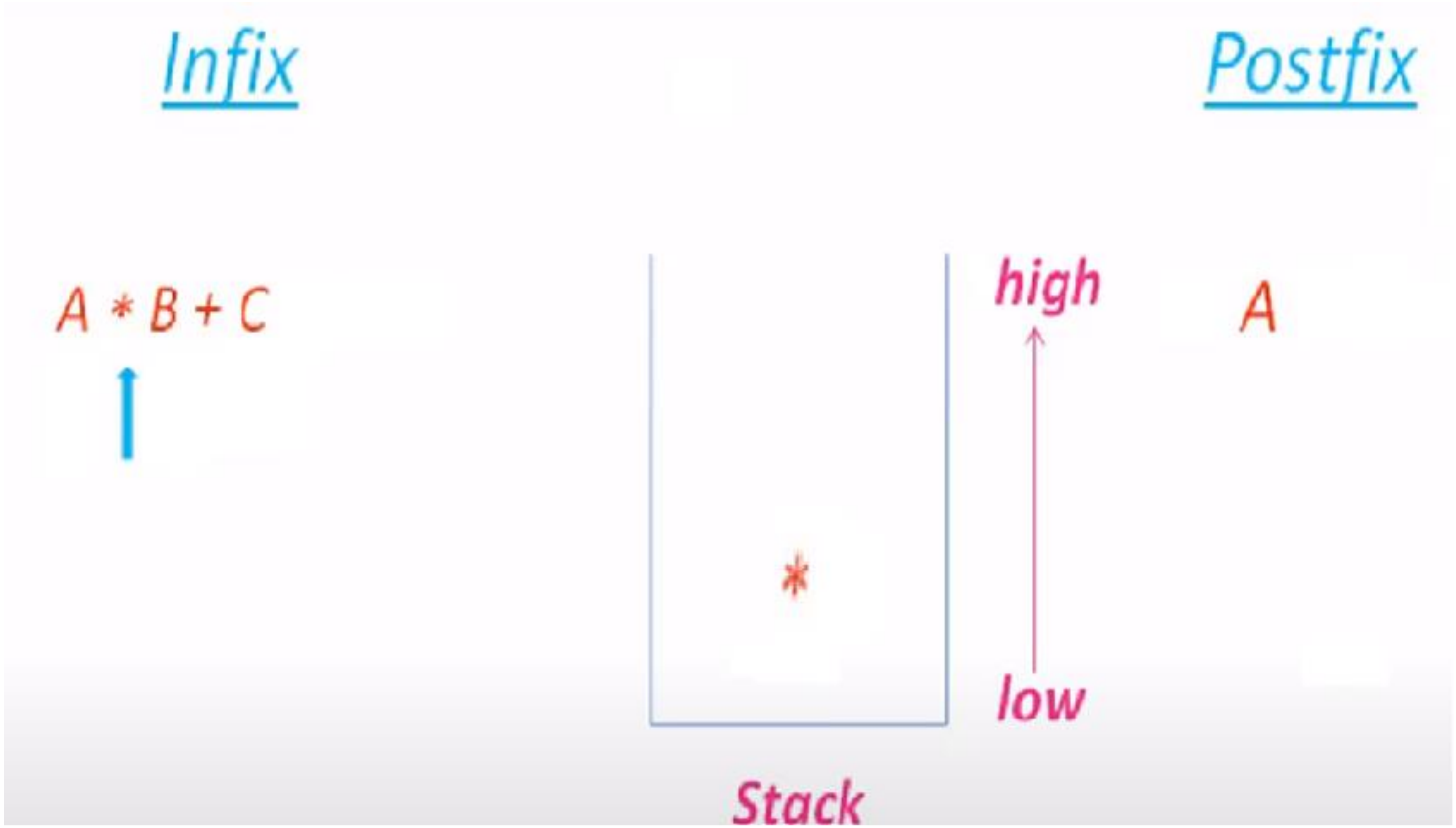

Infix
Postfix
A + B * C
high
low
Stack
A B C * +

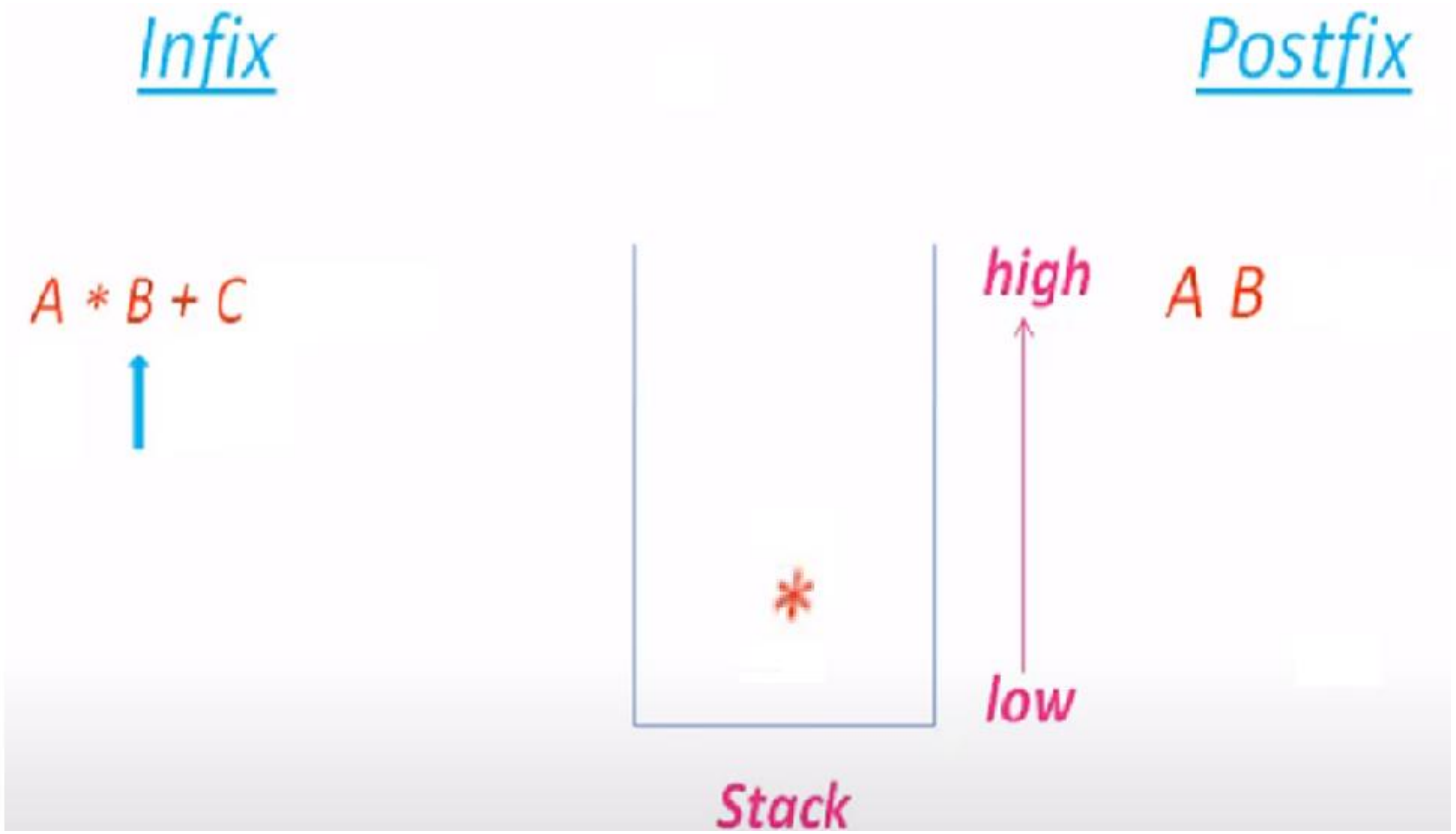
Example 3

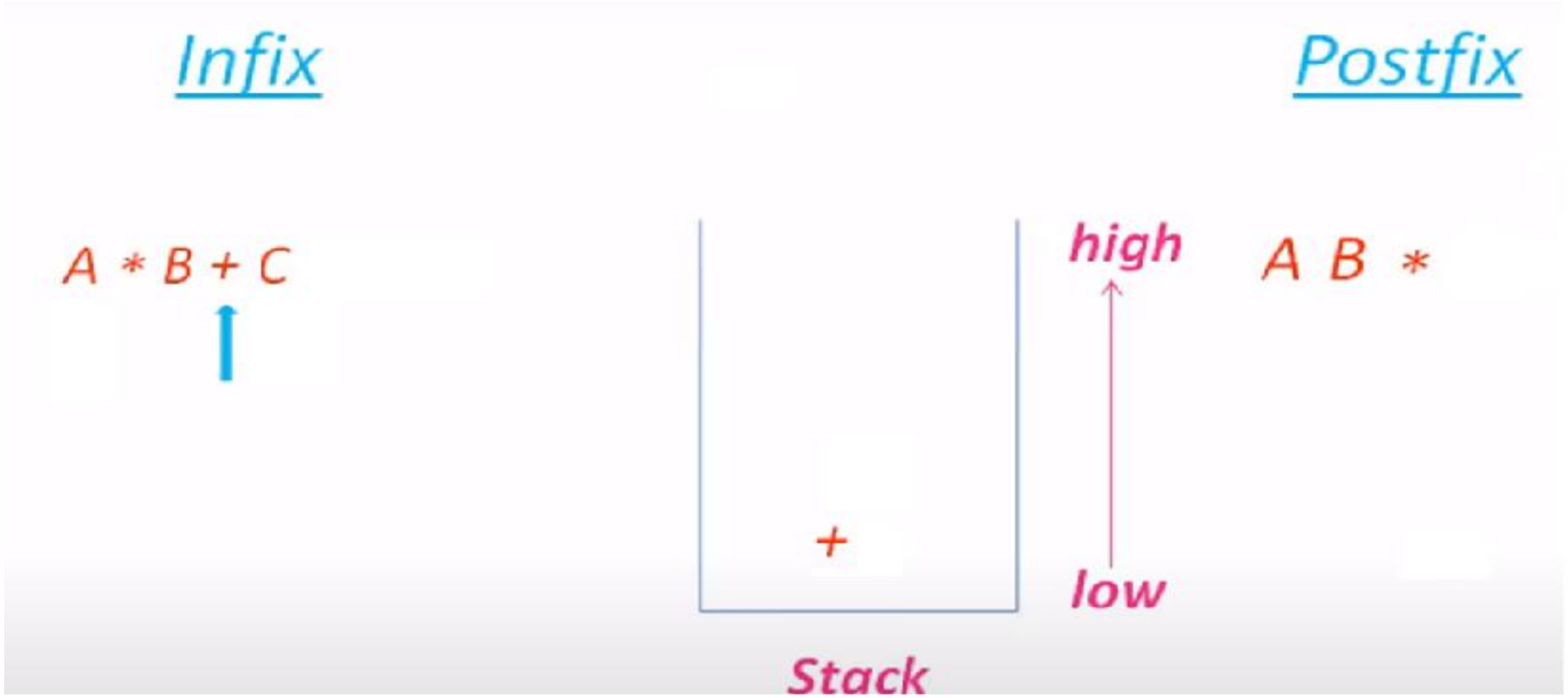

Convert Infix Expression $A * B + C$

to post fix expression


Infix
Postfix
A * B + C
high
low
A
Stack


Infix
Postfix
A * B + C
high
low
*
A
Stack

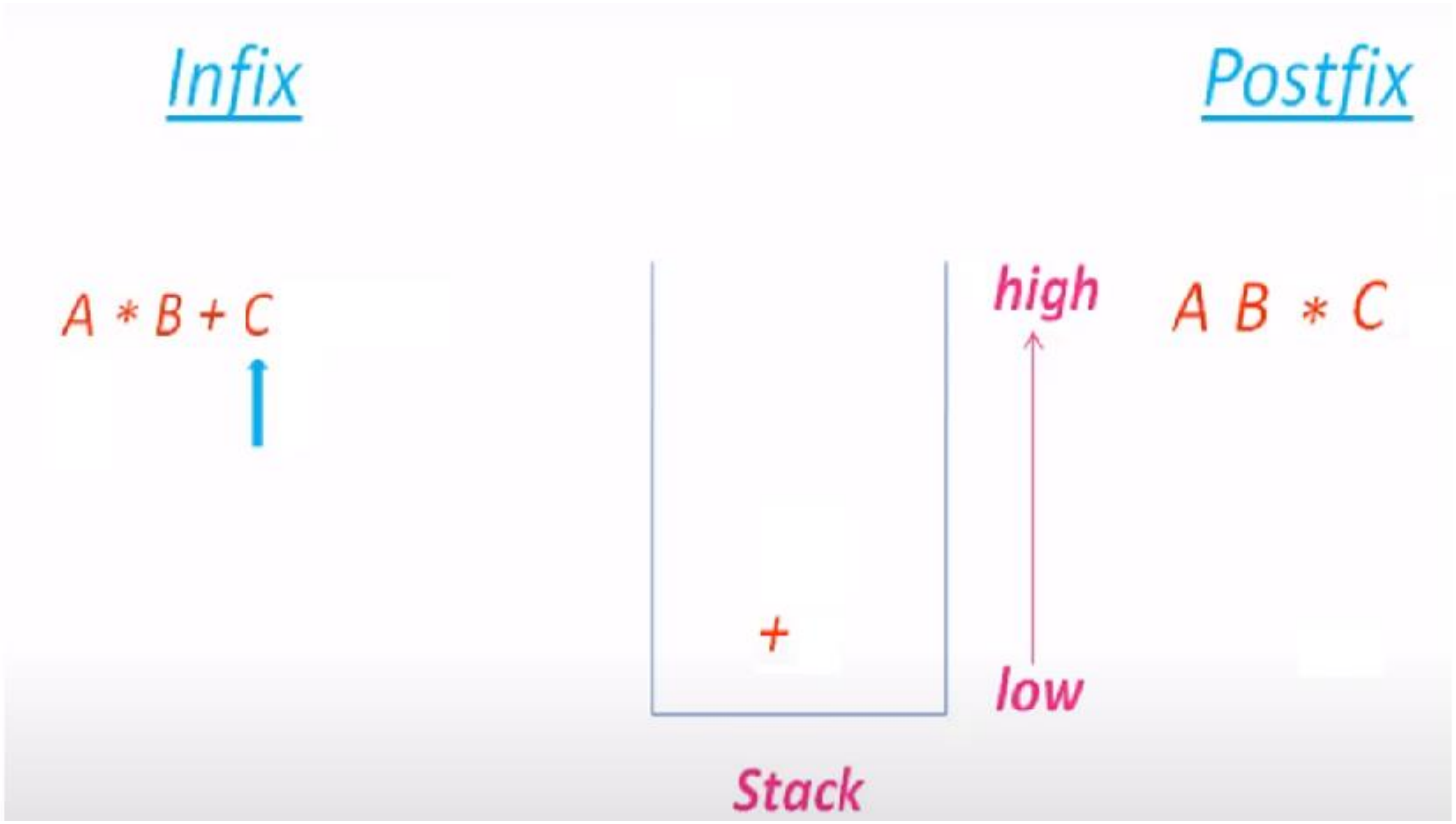

Infix
Postfix
A * B + C
high
low
*
A B
Stack

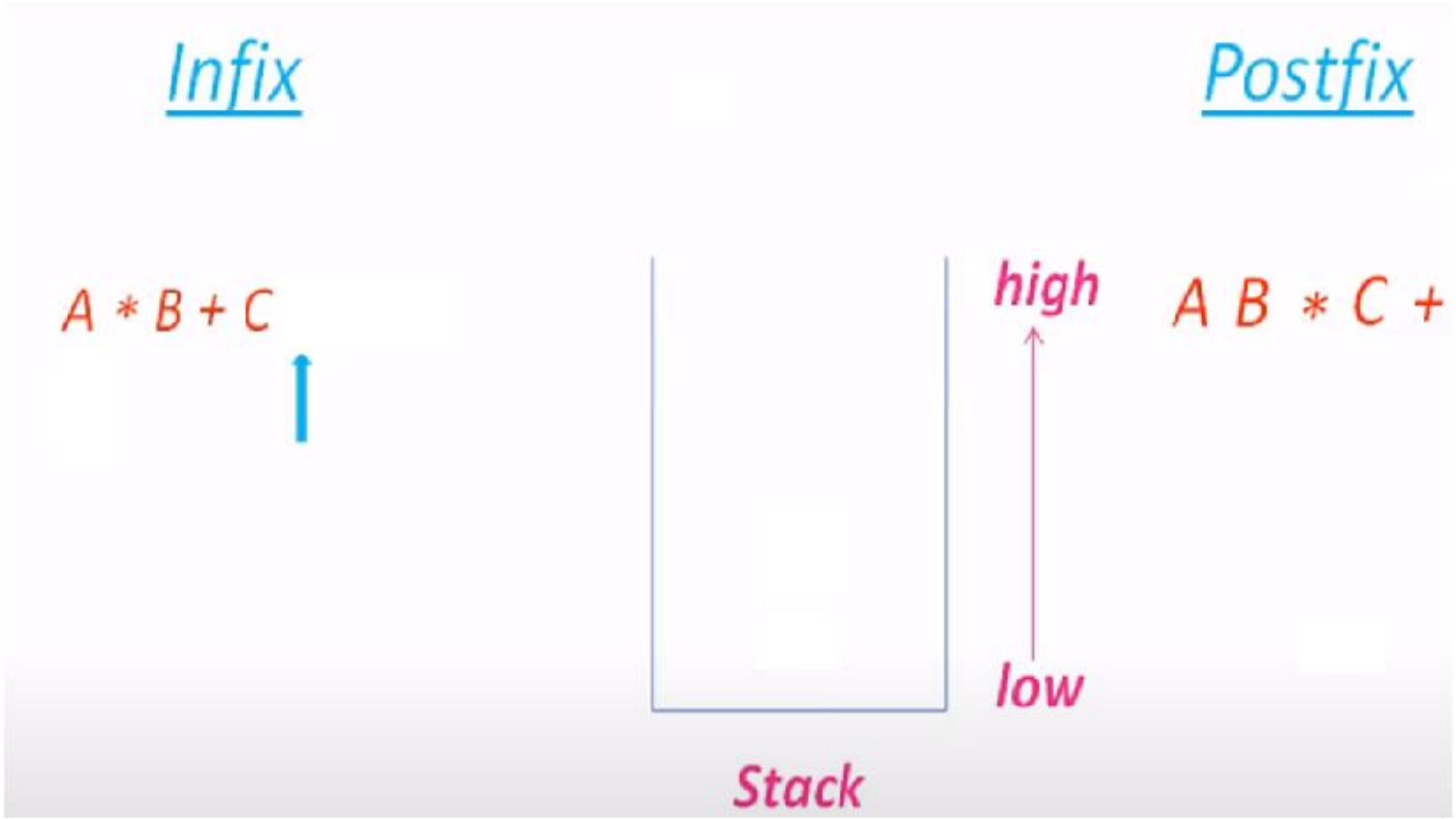


+ is lower precedence than *

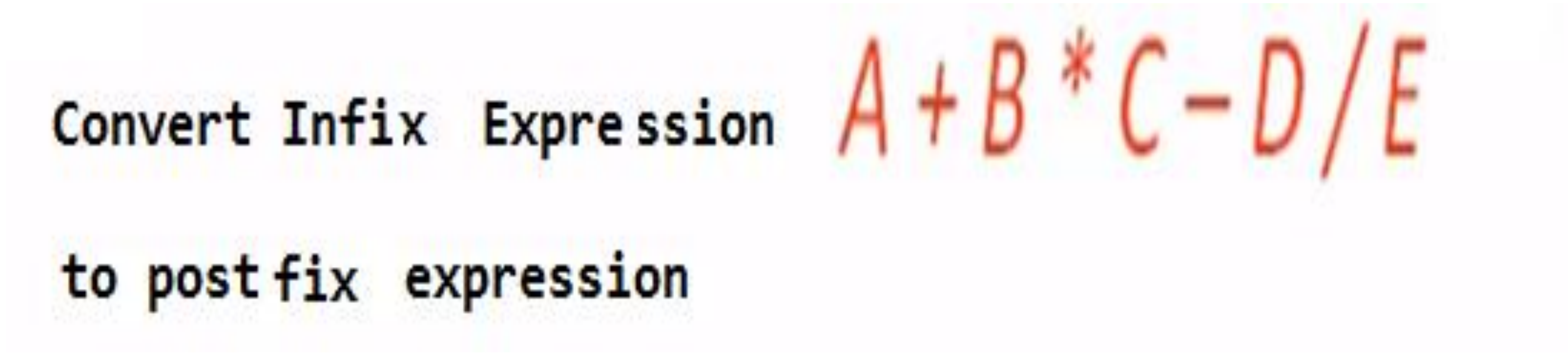
Pop * from stack to postfix

Push + to the stack


Infix
Postfix
A * B + C
high
A B * C
+
low
Stack

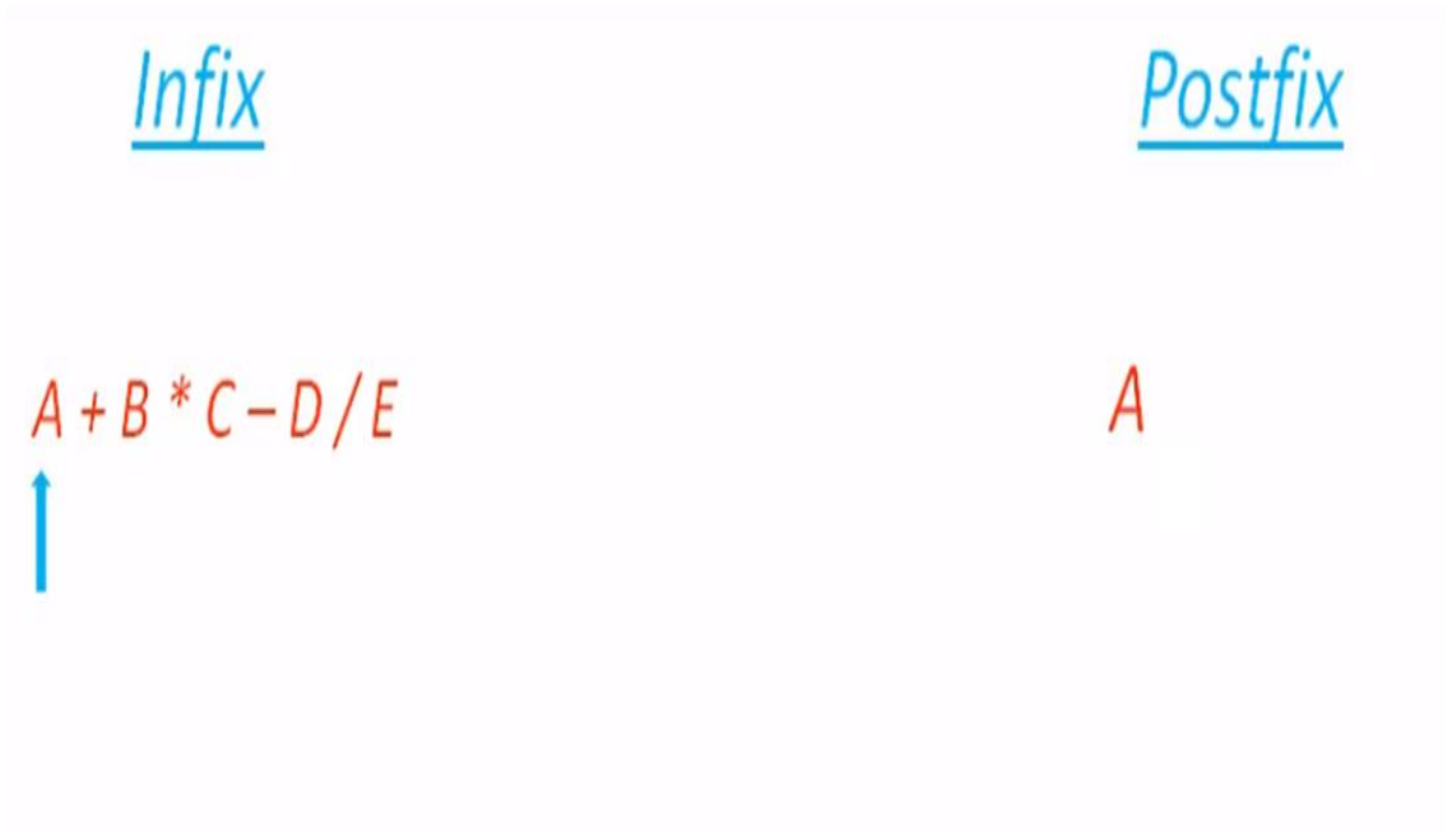

Infix
Postfix
A * B + C
high
A B * C +
low
Stack

Example 4



Convert Infix Expression $A+B*C-D/E$

to postfix expression



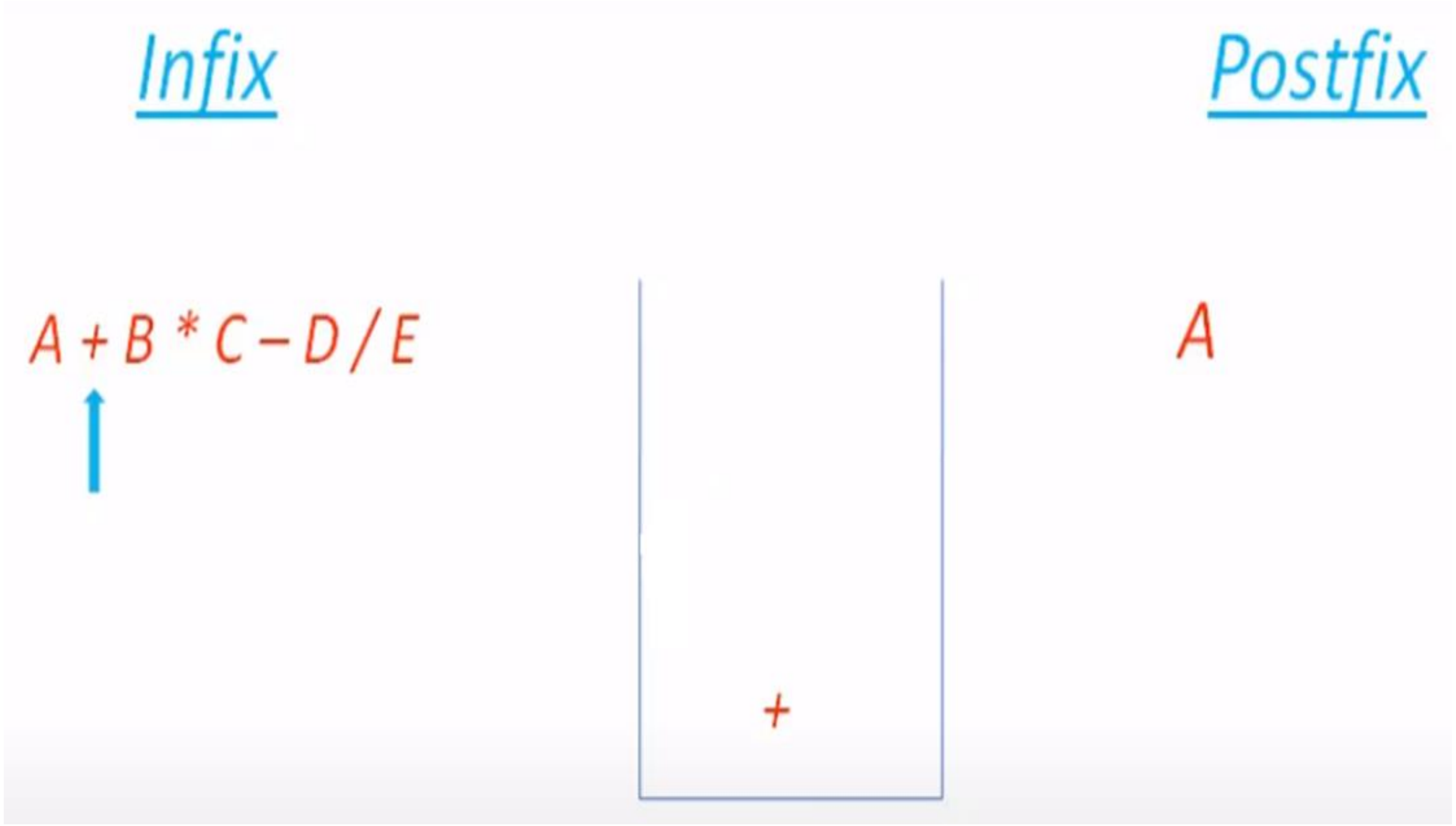
Infix

Postfix

$A + B * C - D / E$

$\uparrow$

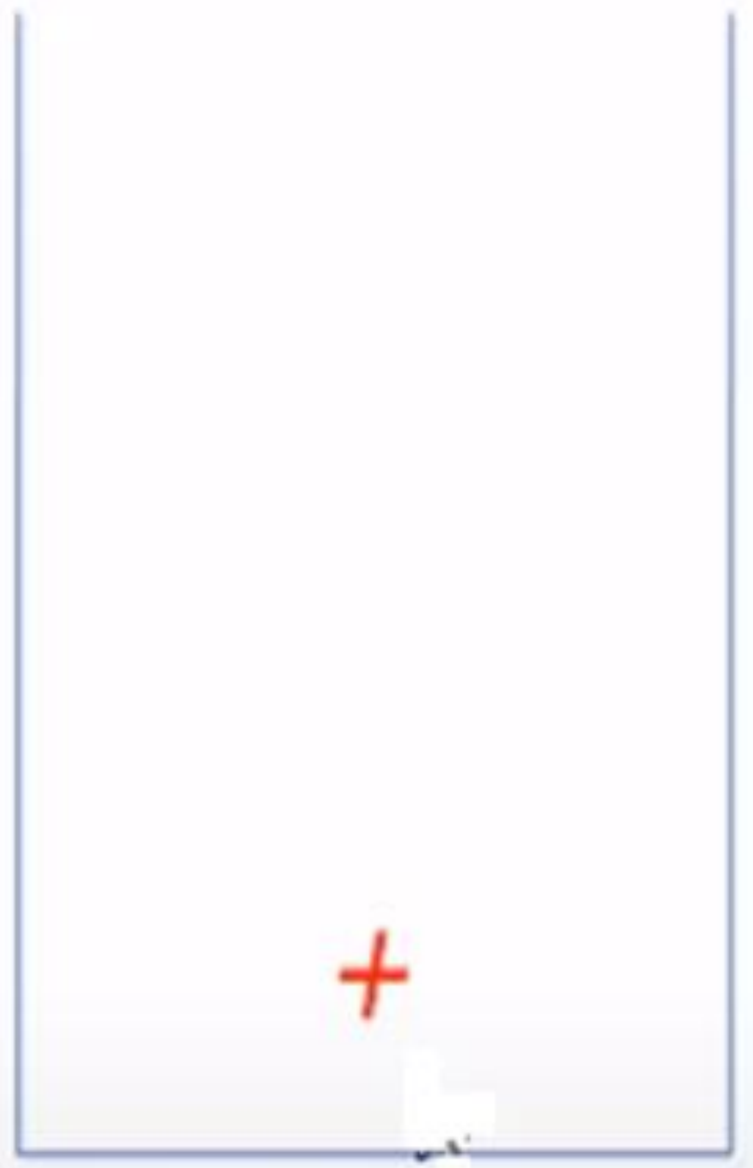
A


Infix
Postfix
A + B * C − D / E
+
A

Infix

Postfix

A + B * C – D / E

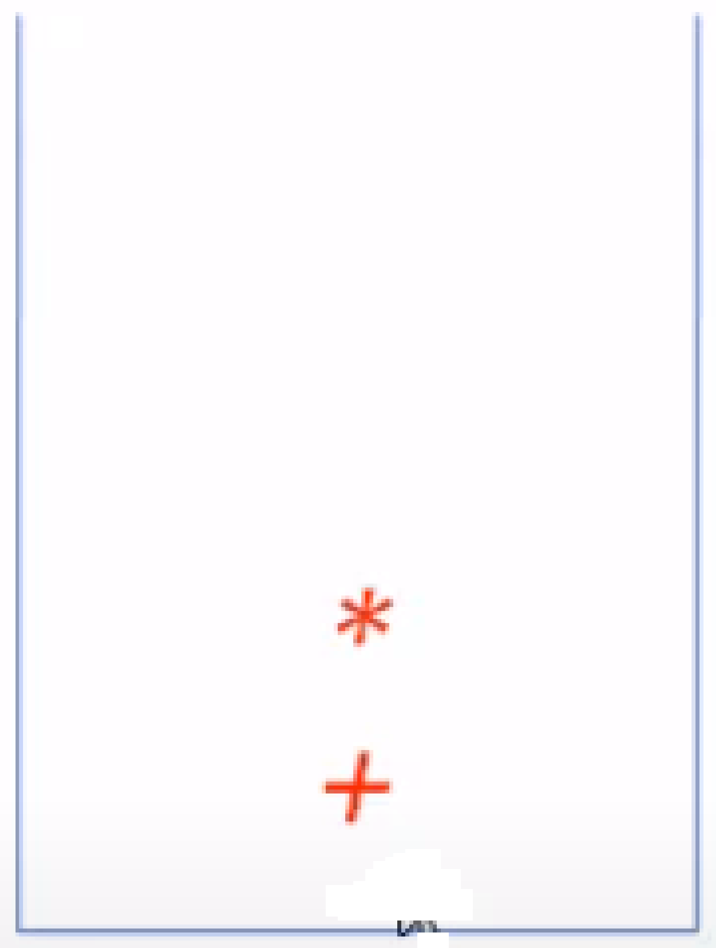


A B

Infix

Postfix

A + B * C – D / E



Stack

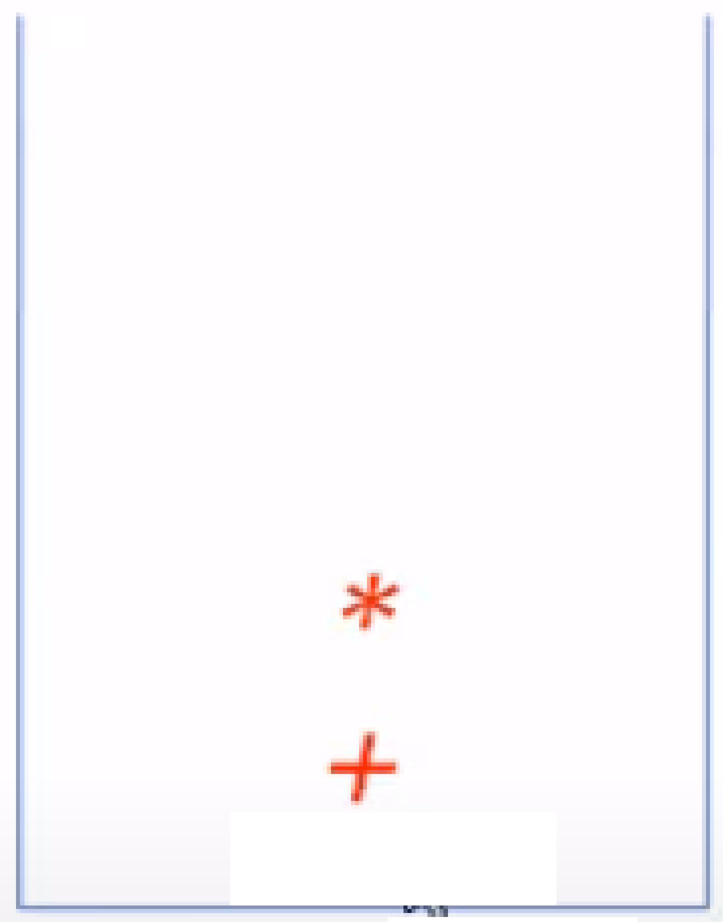
A B

Infix

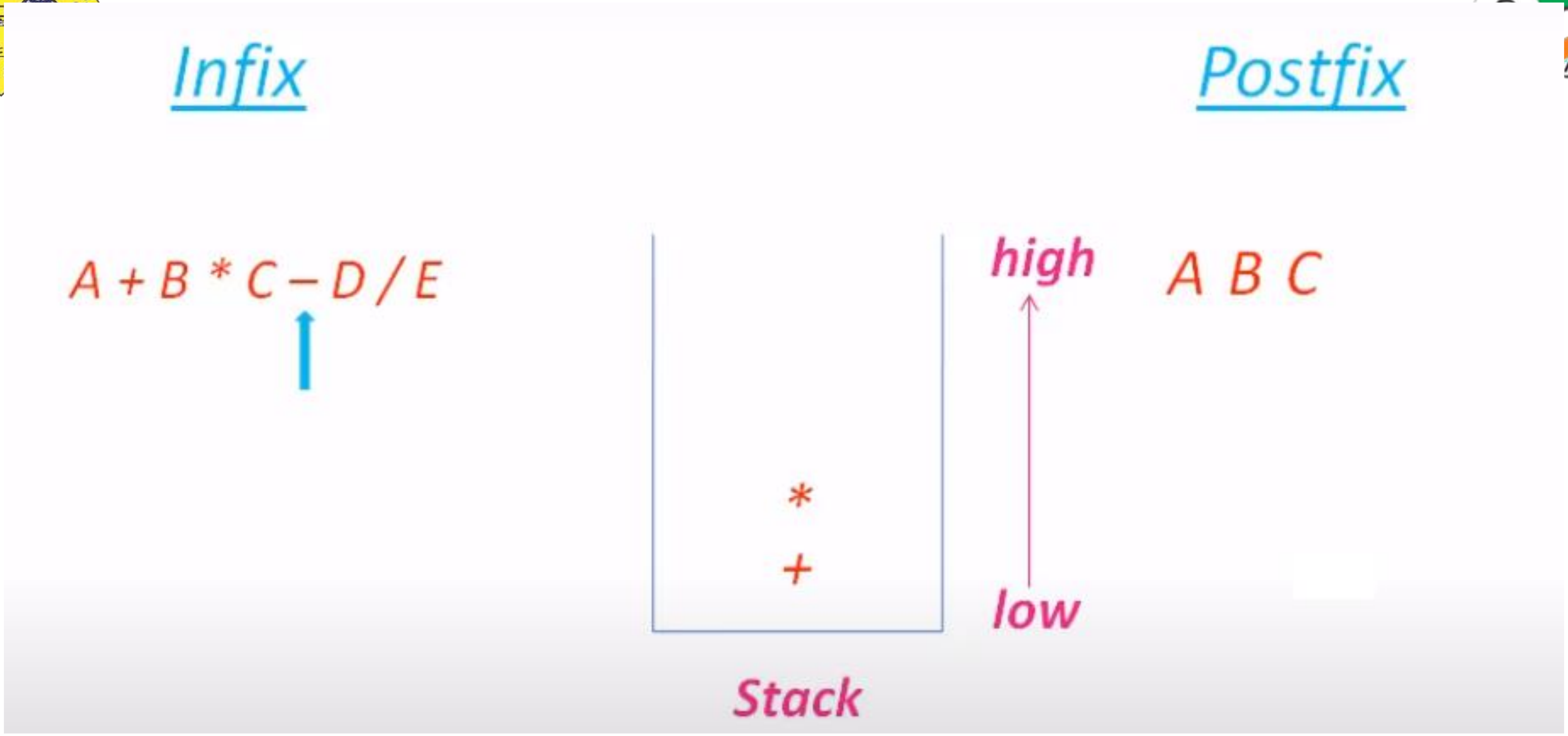
Postfix

A + B * C – D / E

A B C

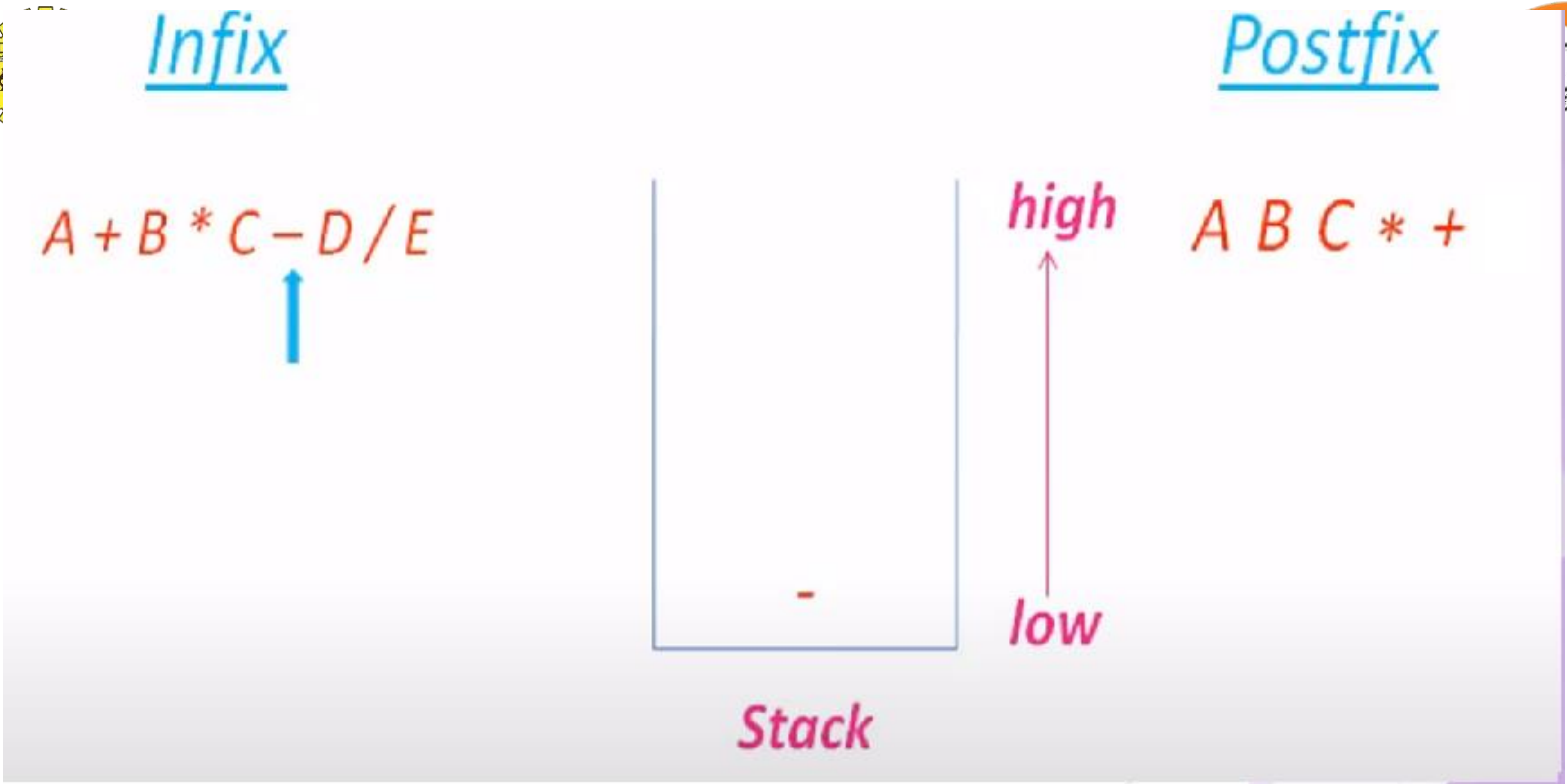


Stack



- Have low precedence than *

Pop *, + from stack

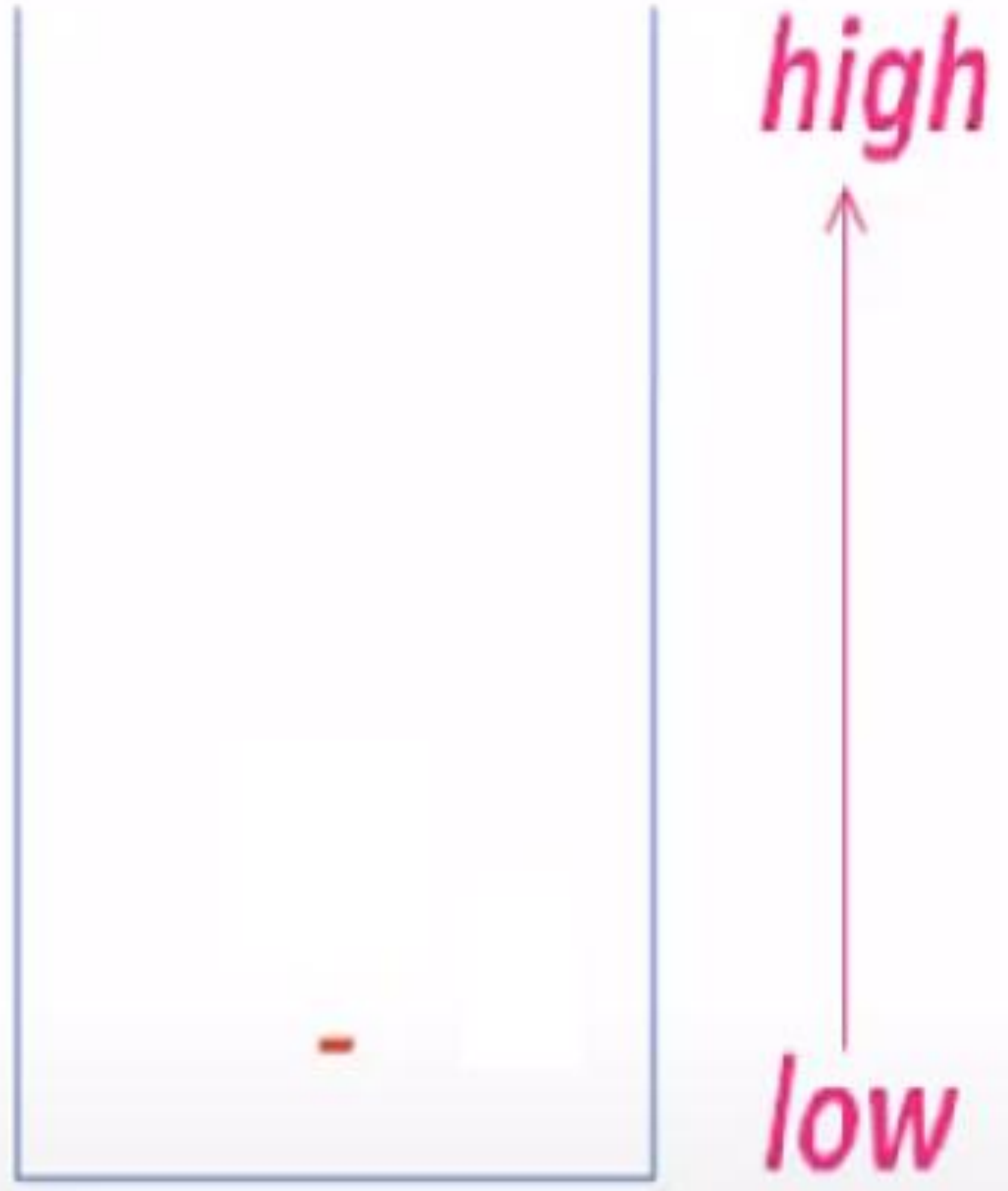


Then Push - to stack

Infix

Postfix

A + B * C – D / E



A B C * + D

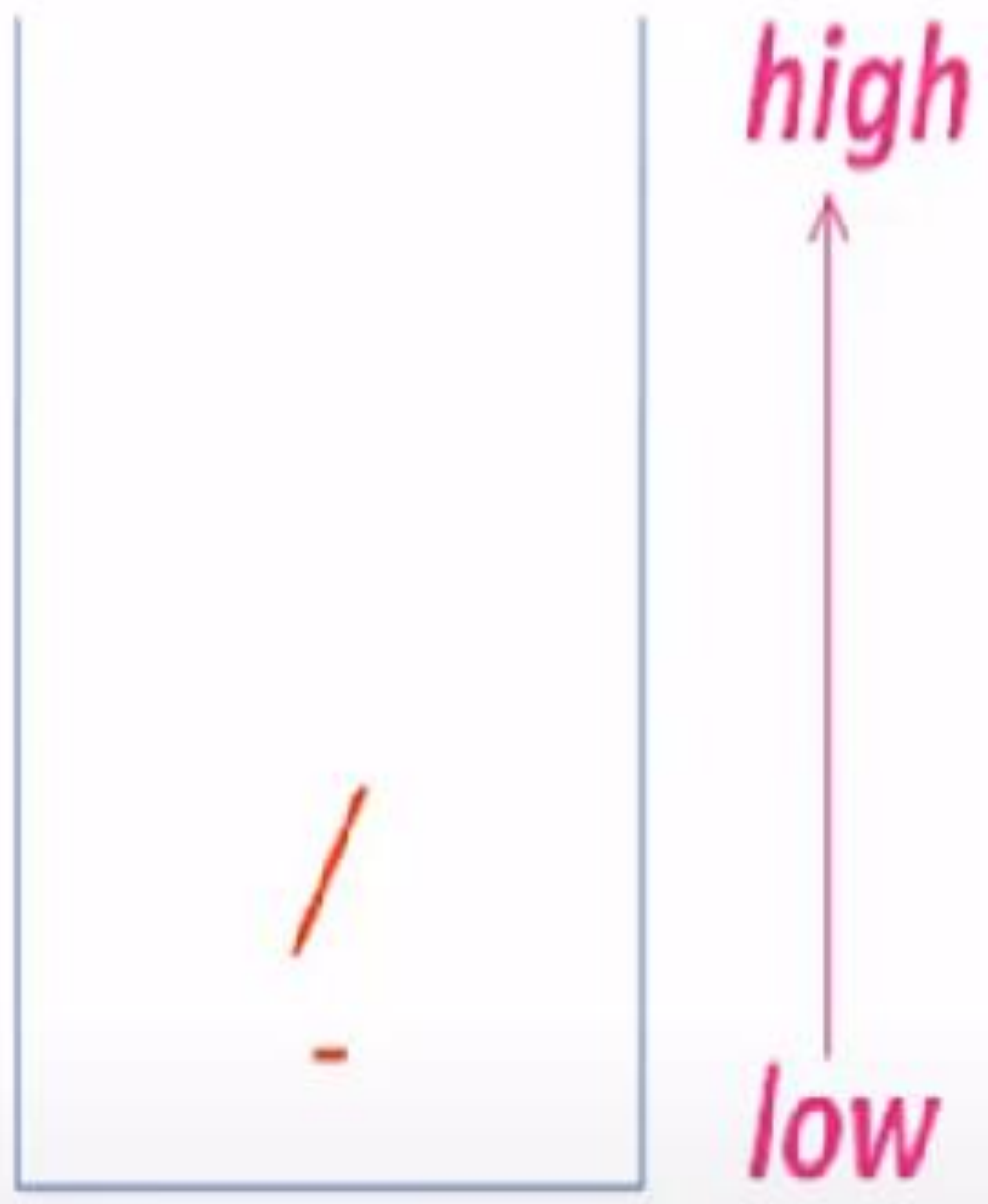
Stack

Infix

Postfix

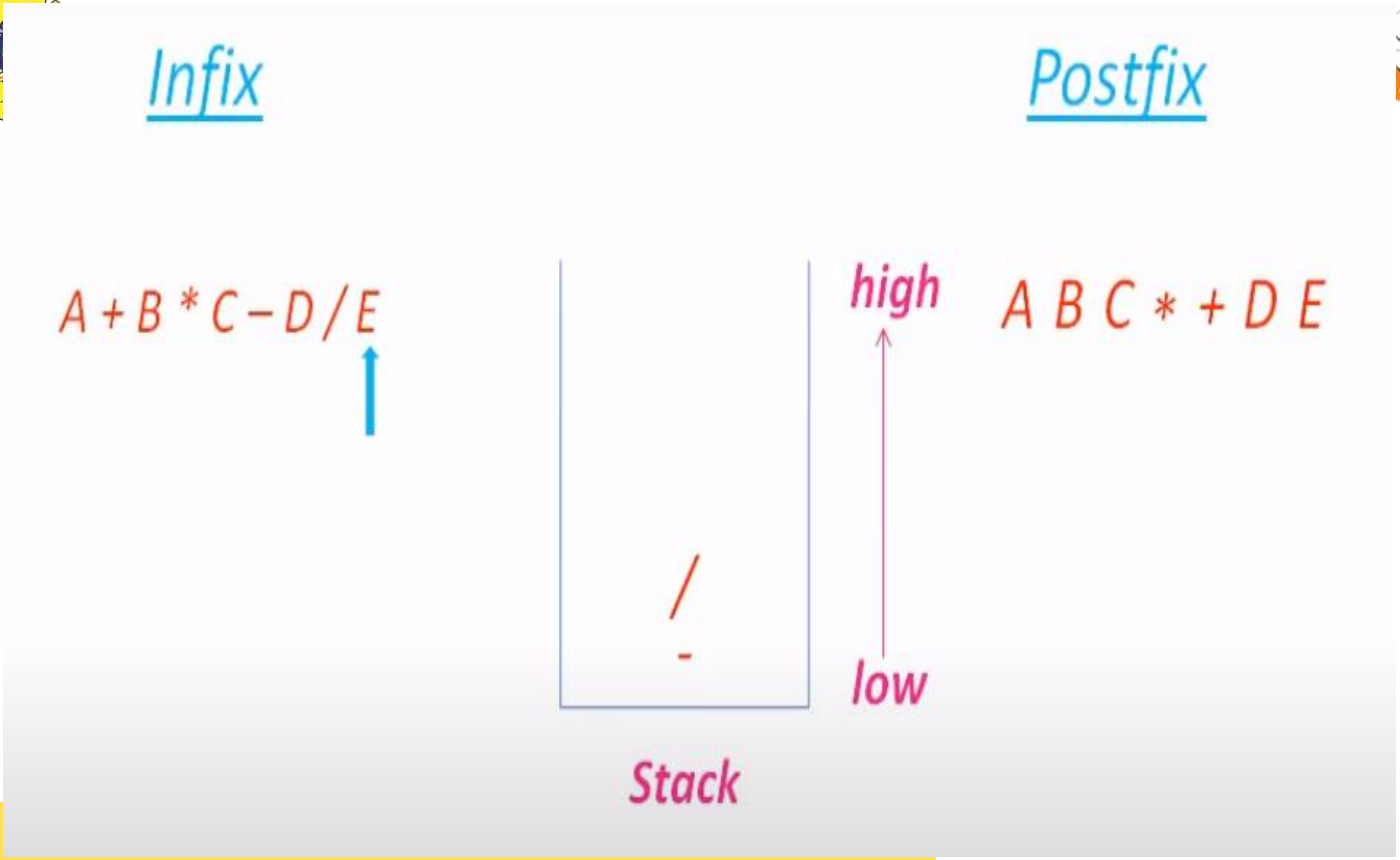
A + B * C − D / E

↑



Stack

A B C * + D



Infix

Postfix

A + B * C – D / E

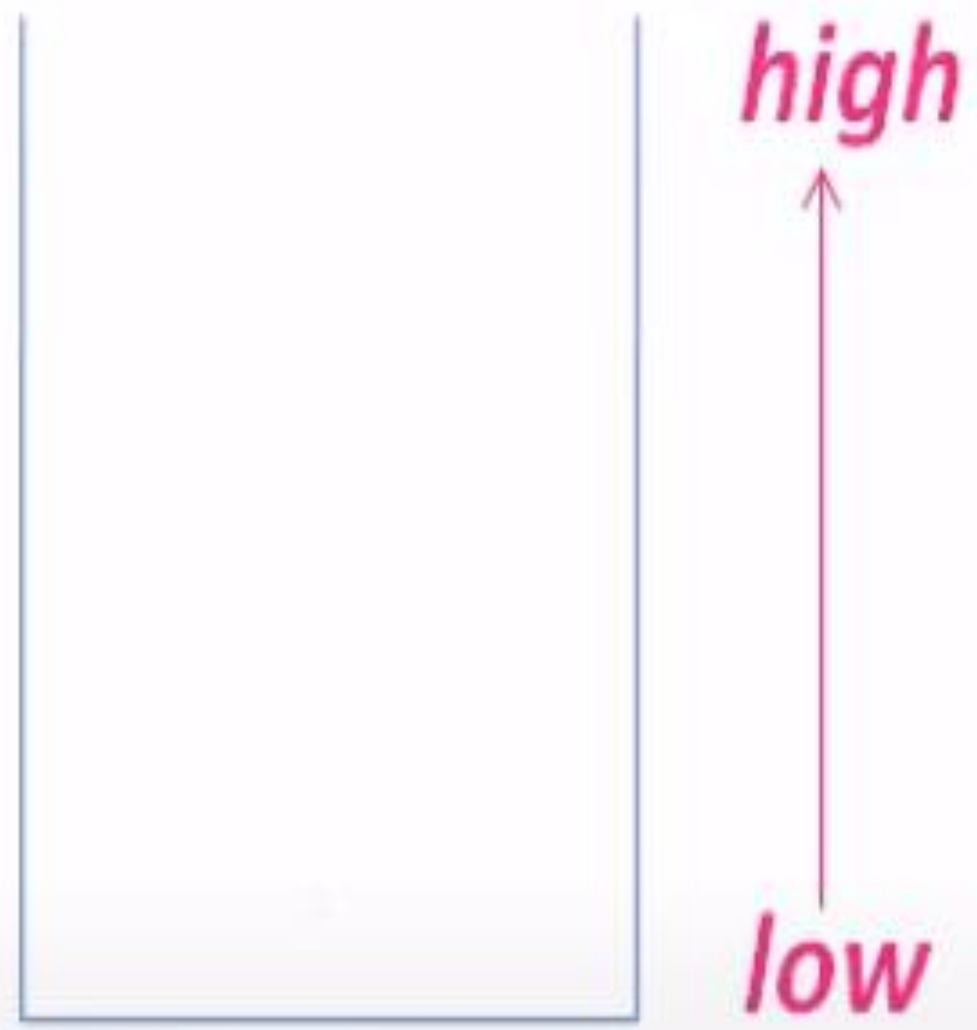
A B C * + D E

Infix

A + B * C – D / E

Postfix

A B C * + D E / -



Stack

Example5

Infix

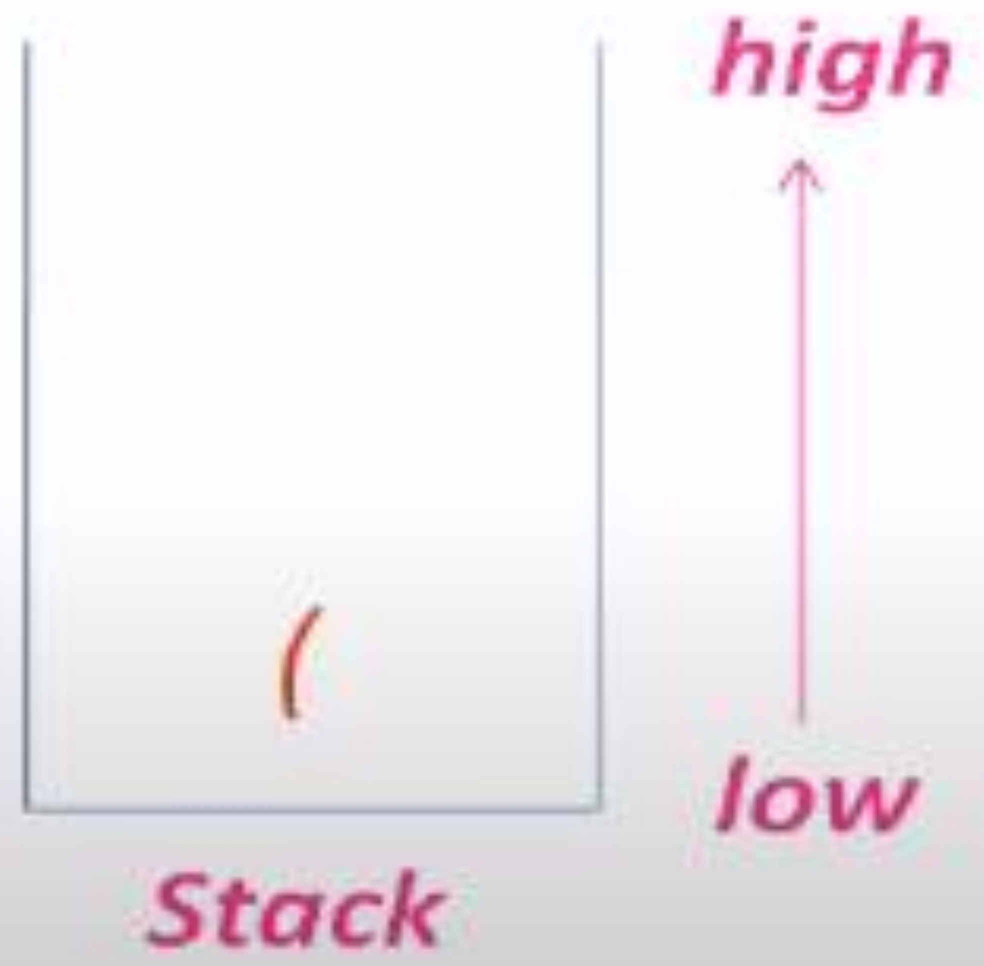
Postfix

((A+B)*(C−D)/E)

Infix

Postfix

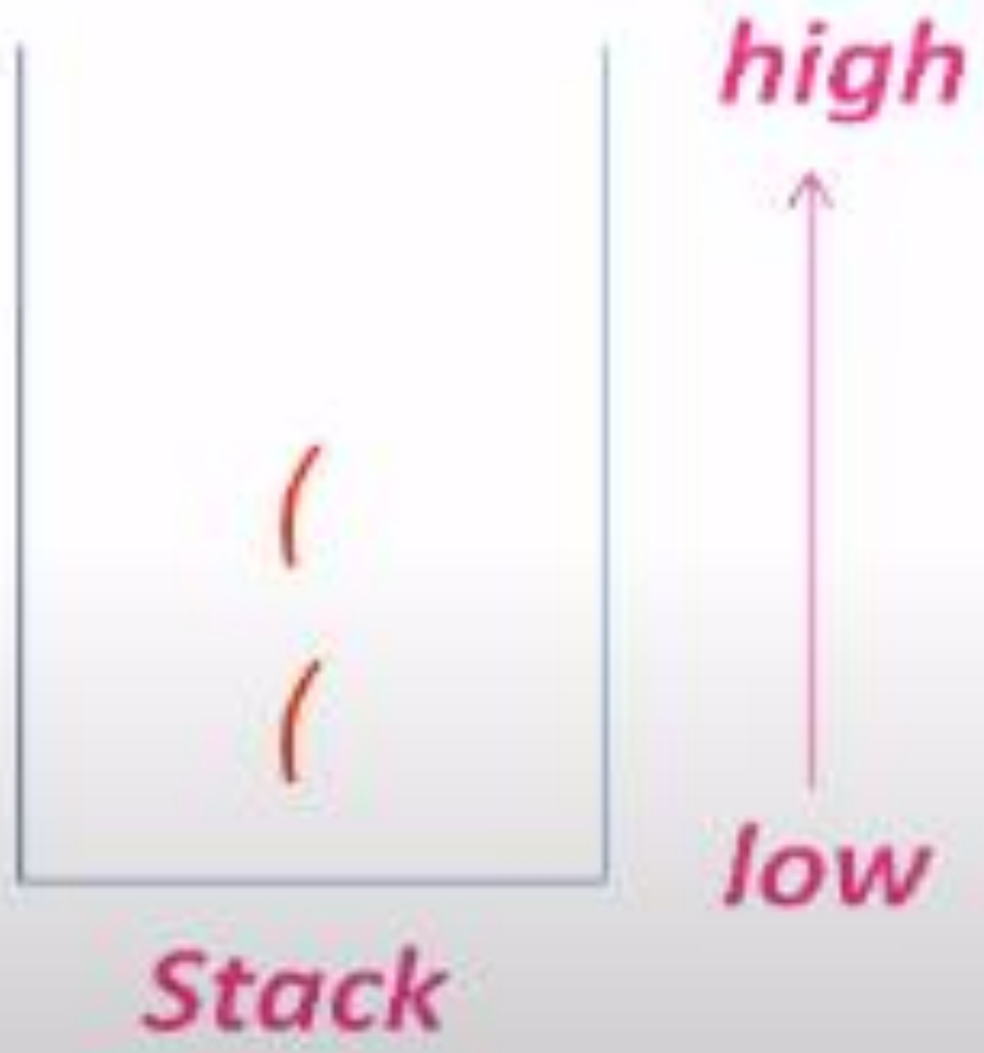
$((A + B) * (C - D) / E)$



Infix

Postfix

((A + B) * (C – D) / E)



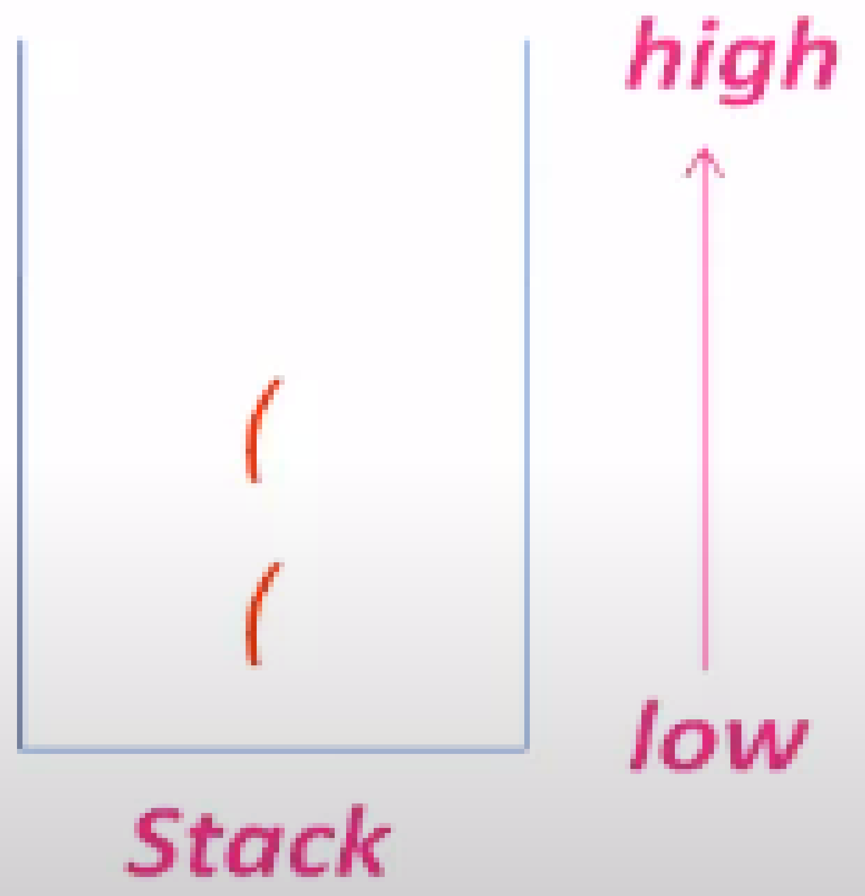
Infix

Postfix

((A + B) * (C – D) / E)

$\uparrow$

A

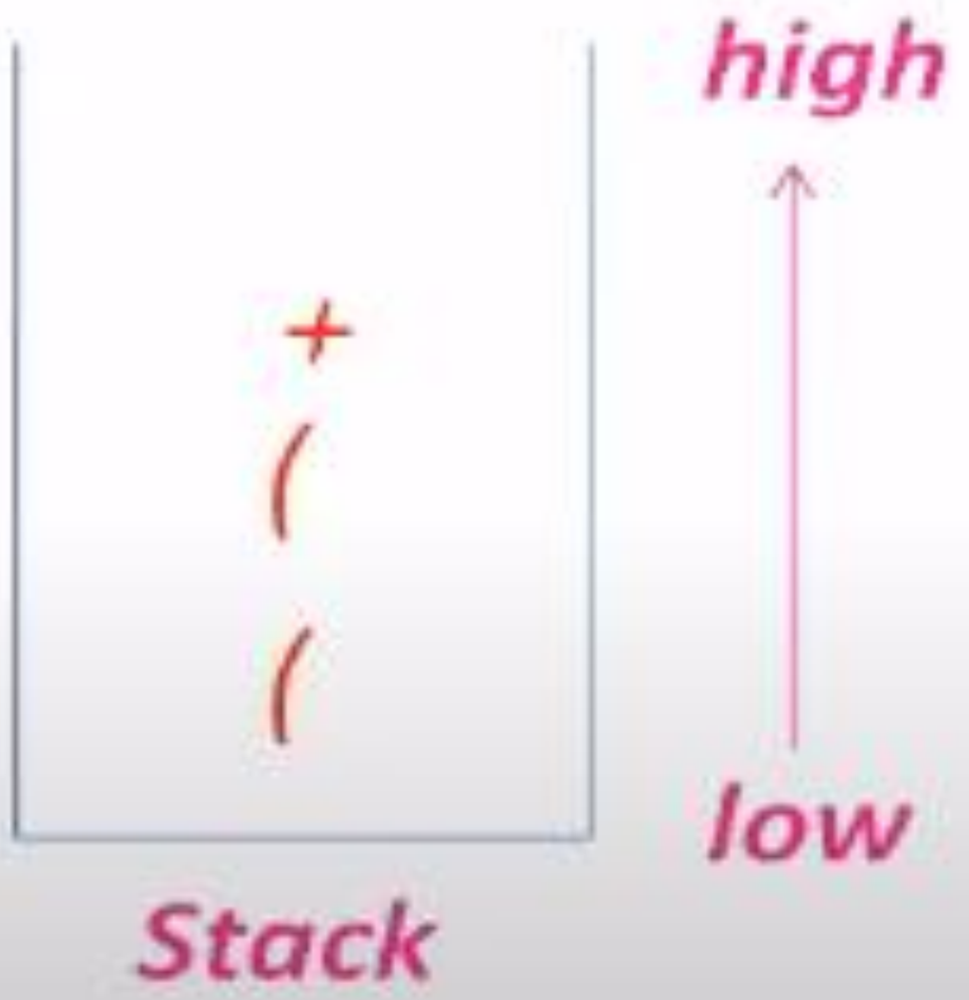


Infix

Postfix

((A + B) * (C – D) / E)

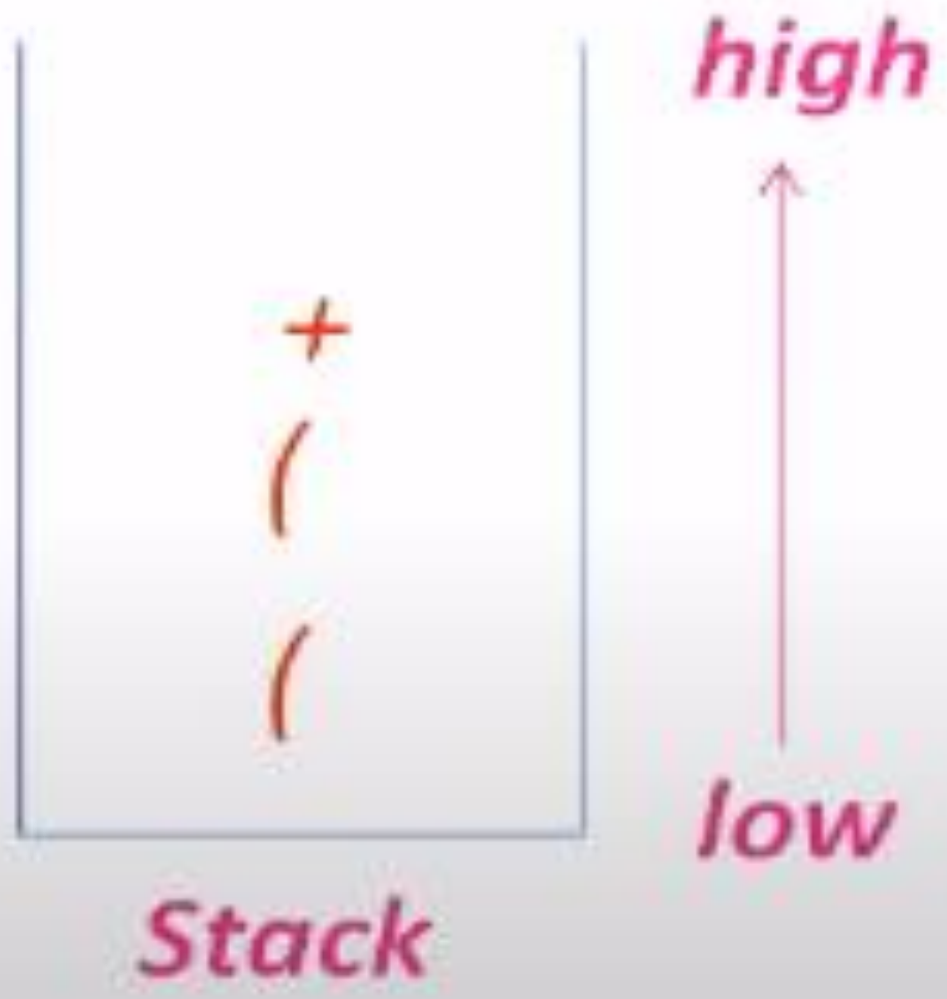
A

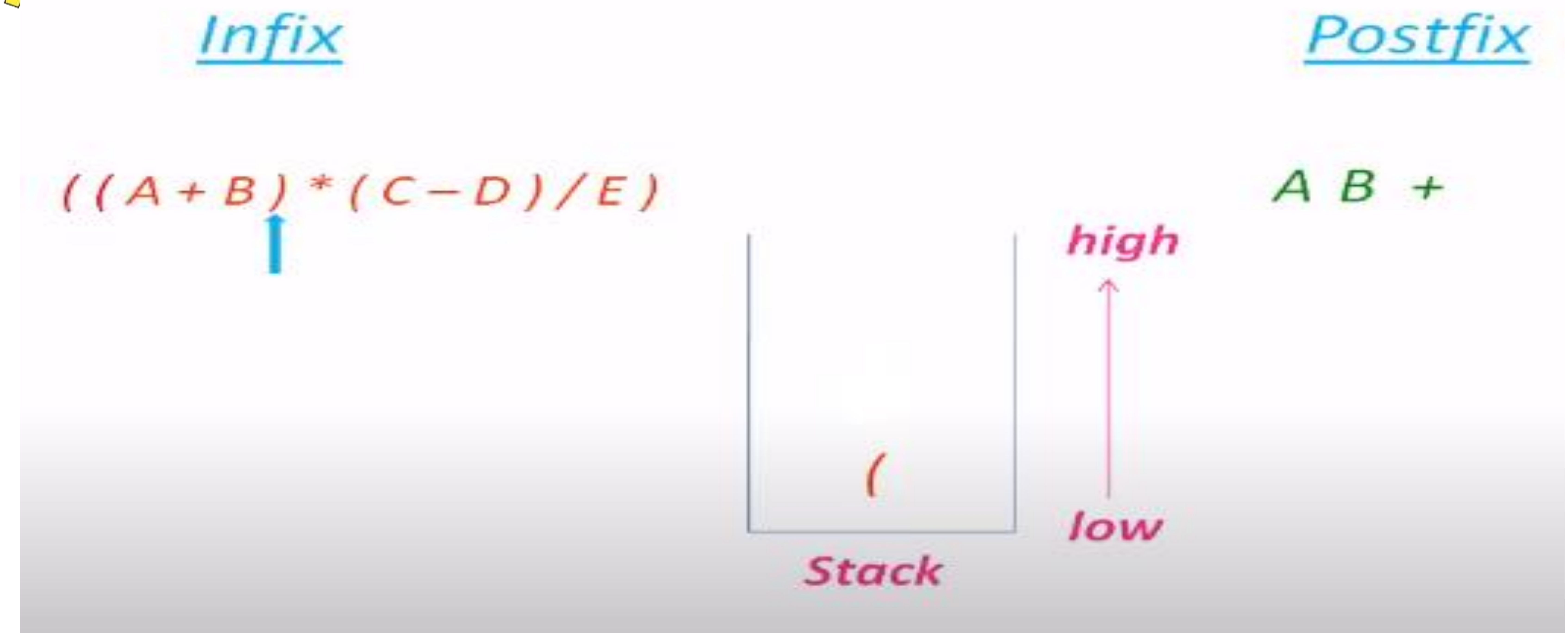


Infix **Postfix**

((A + B) * (C – D) / E)

A B





) , pop + from stack to postfix

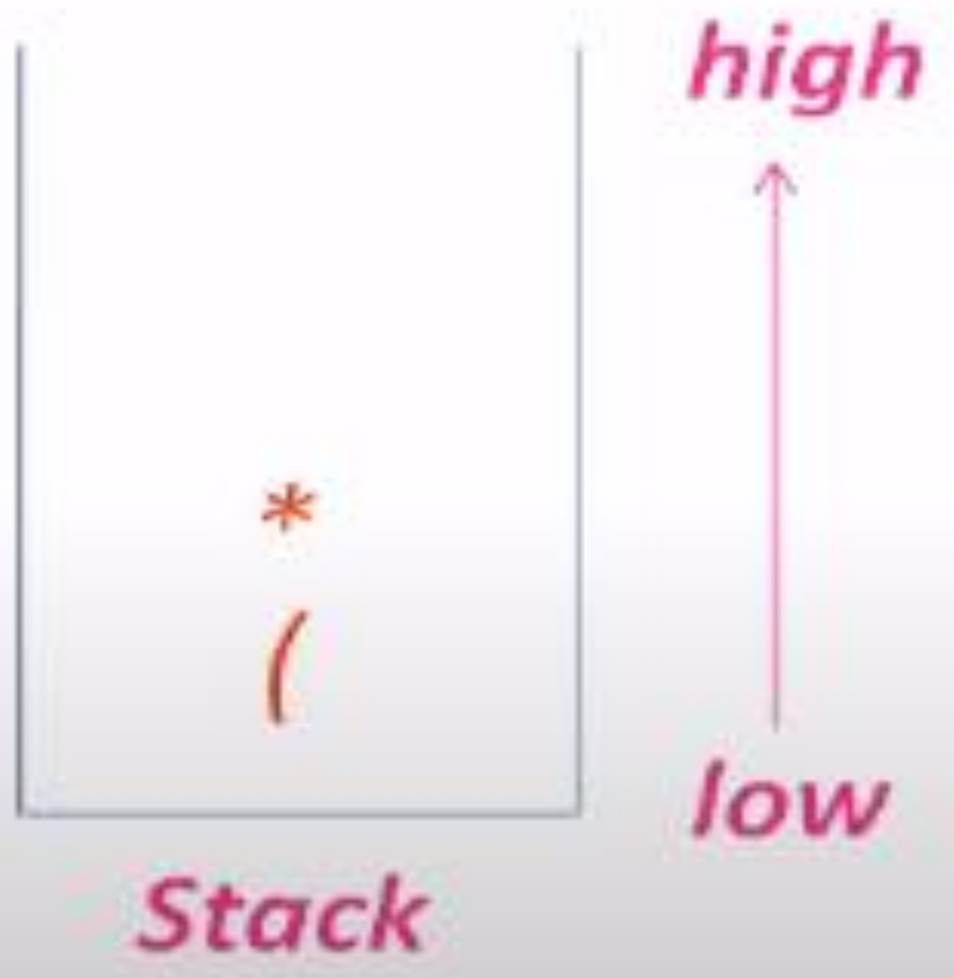
pop ‘(’ from stack and discard both the parenthesis

Infix

Postfix

((A + B) * (C – D) / E)

A B +

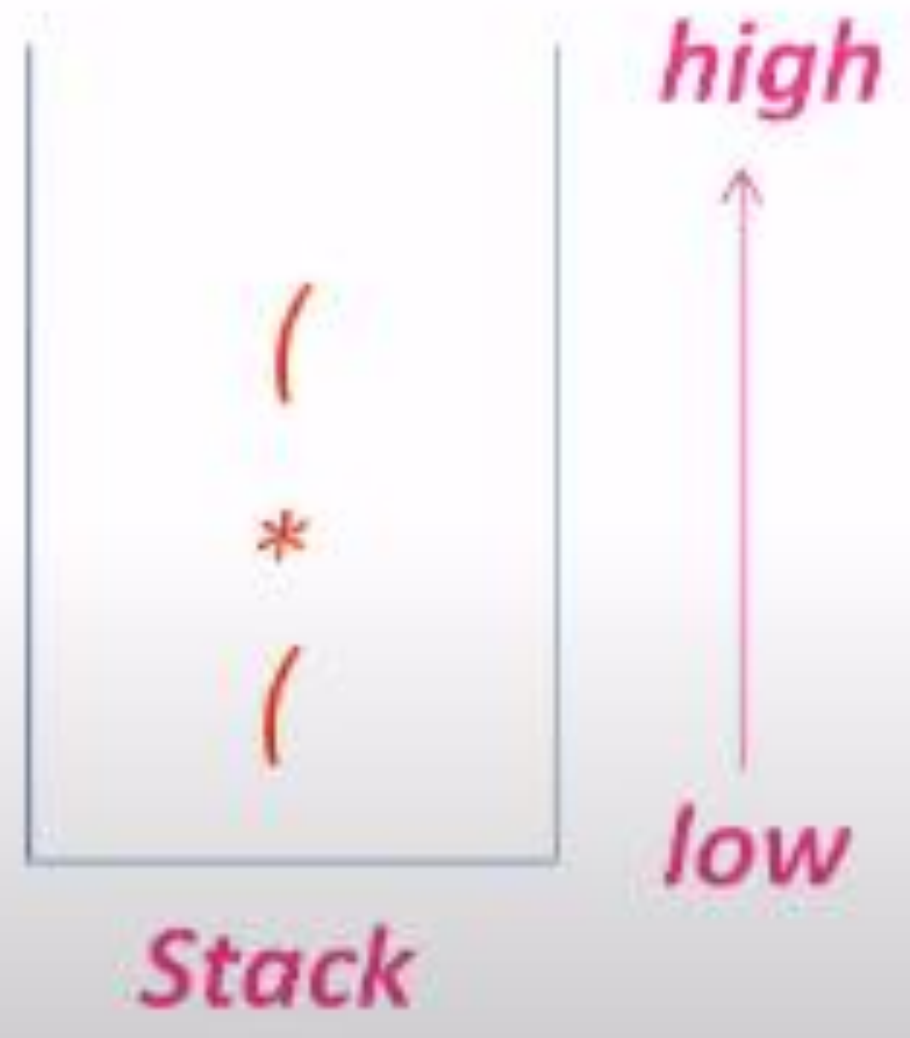


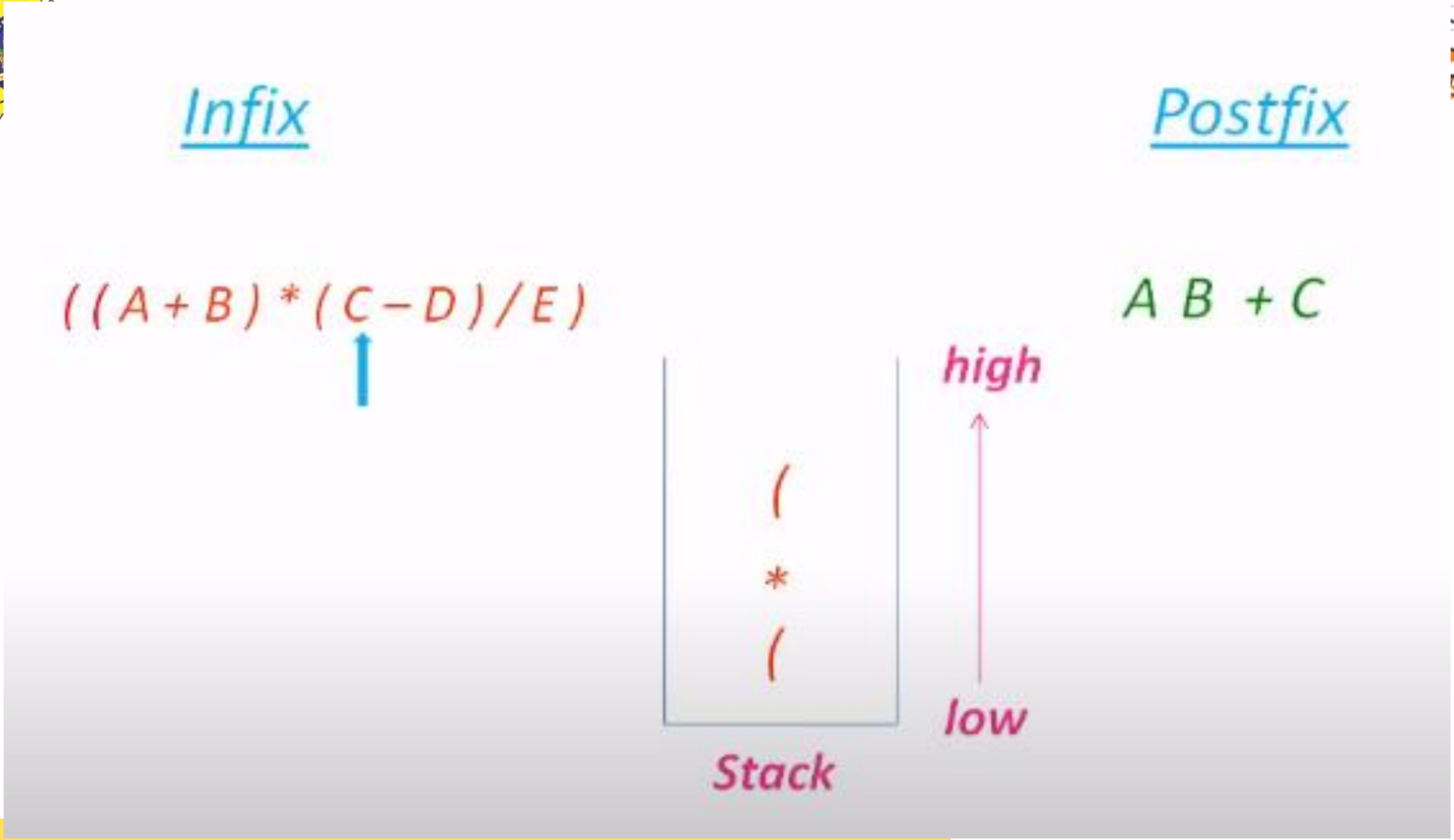
Infix

Postfix

((A + B) * (C – D) / E)

A B +





Infix

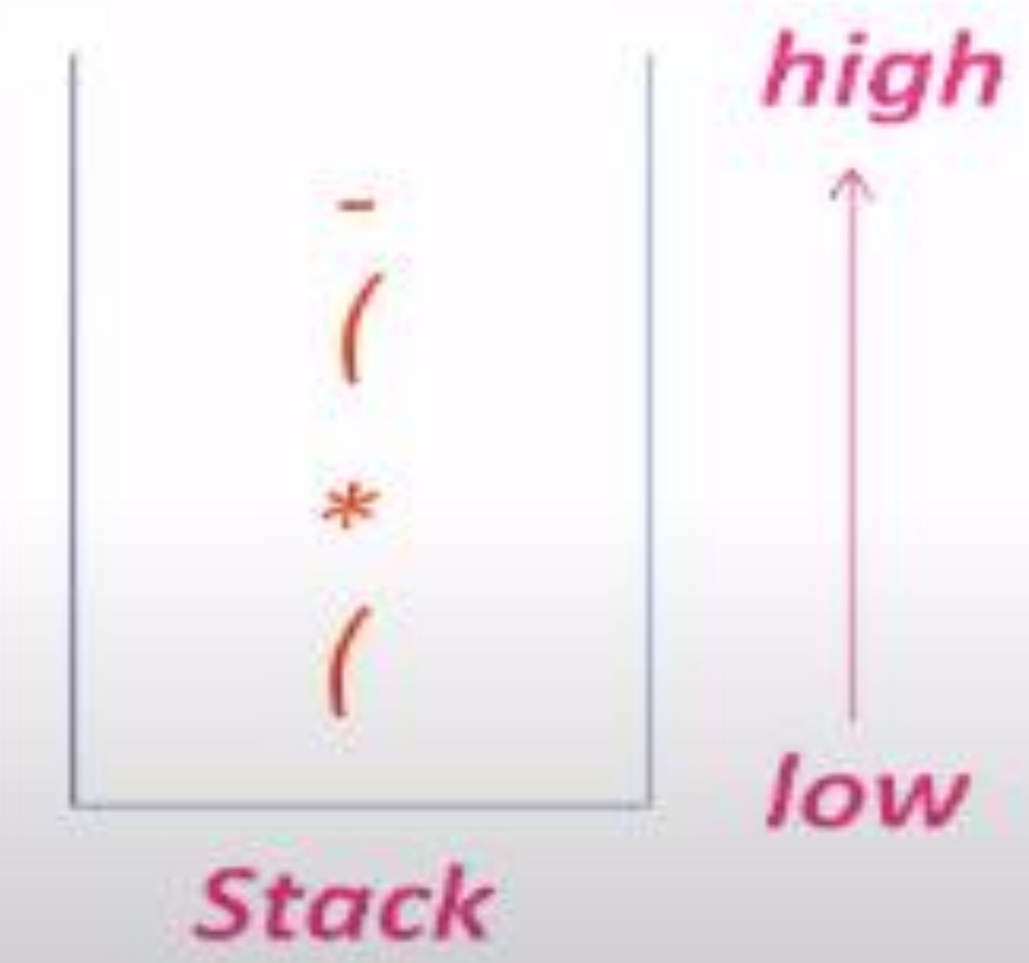
Postfix

((A + B) * (C – D) / E)

A B + C

Infix

((A + B) * (C – D) / E)



Postfix

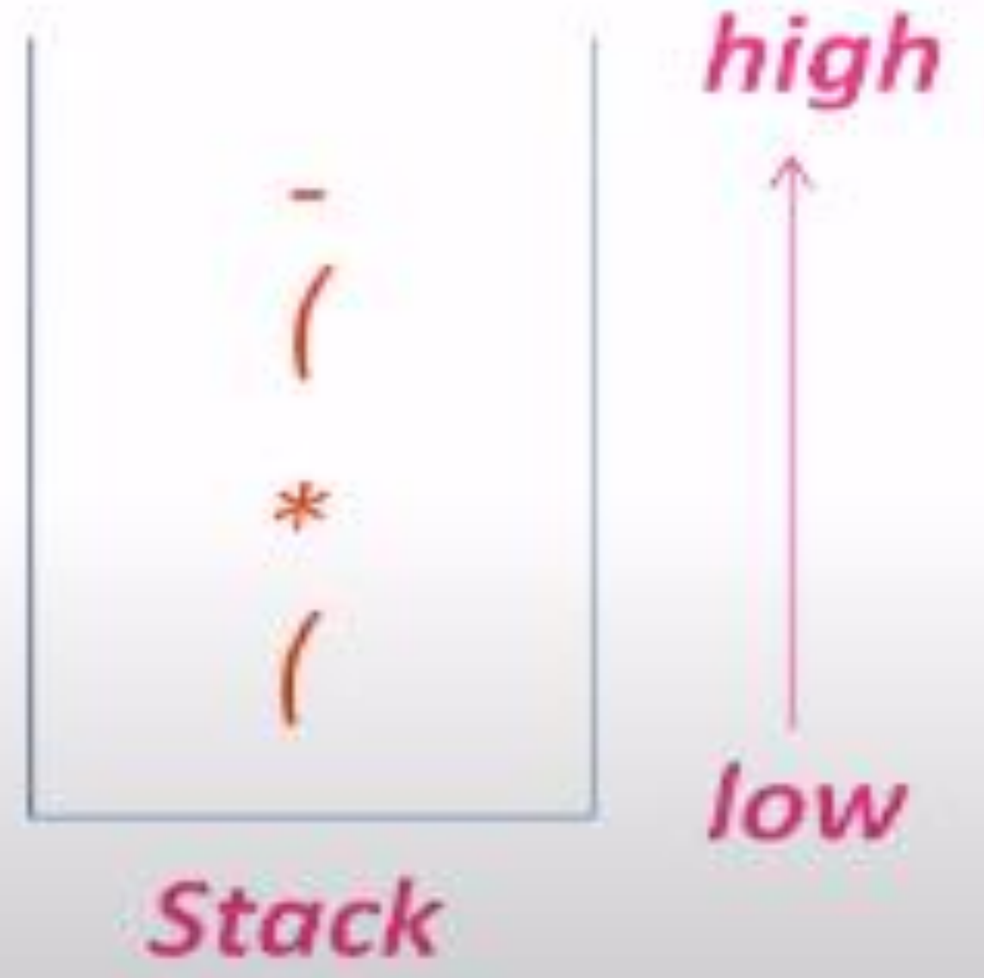
A B + C

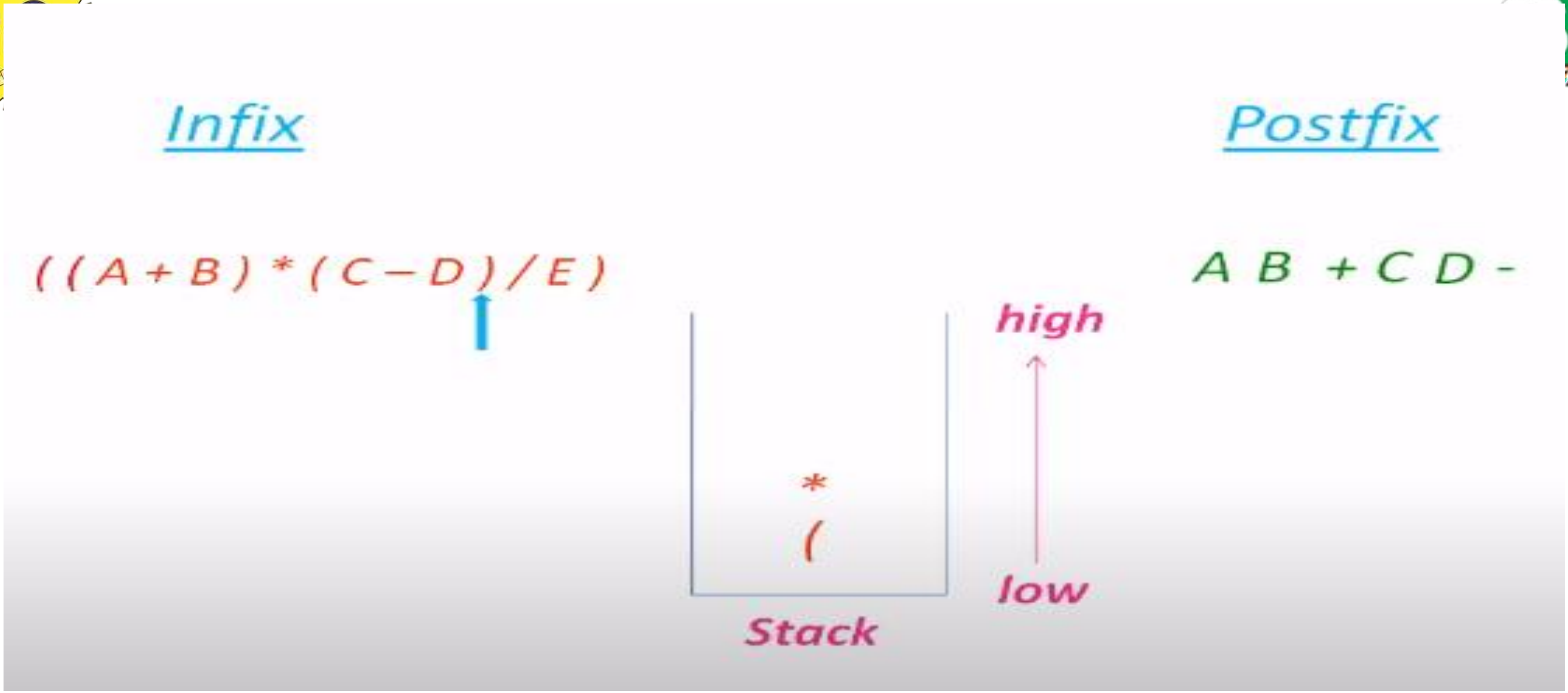
Infix

((A + B) * (C − D) / E)

Postfix

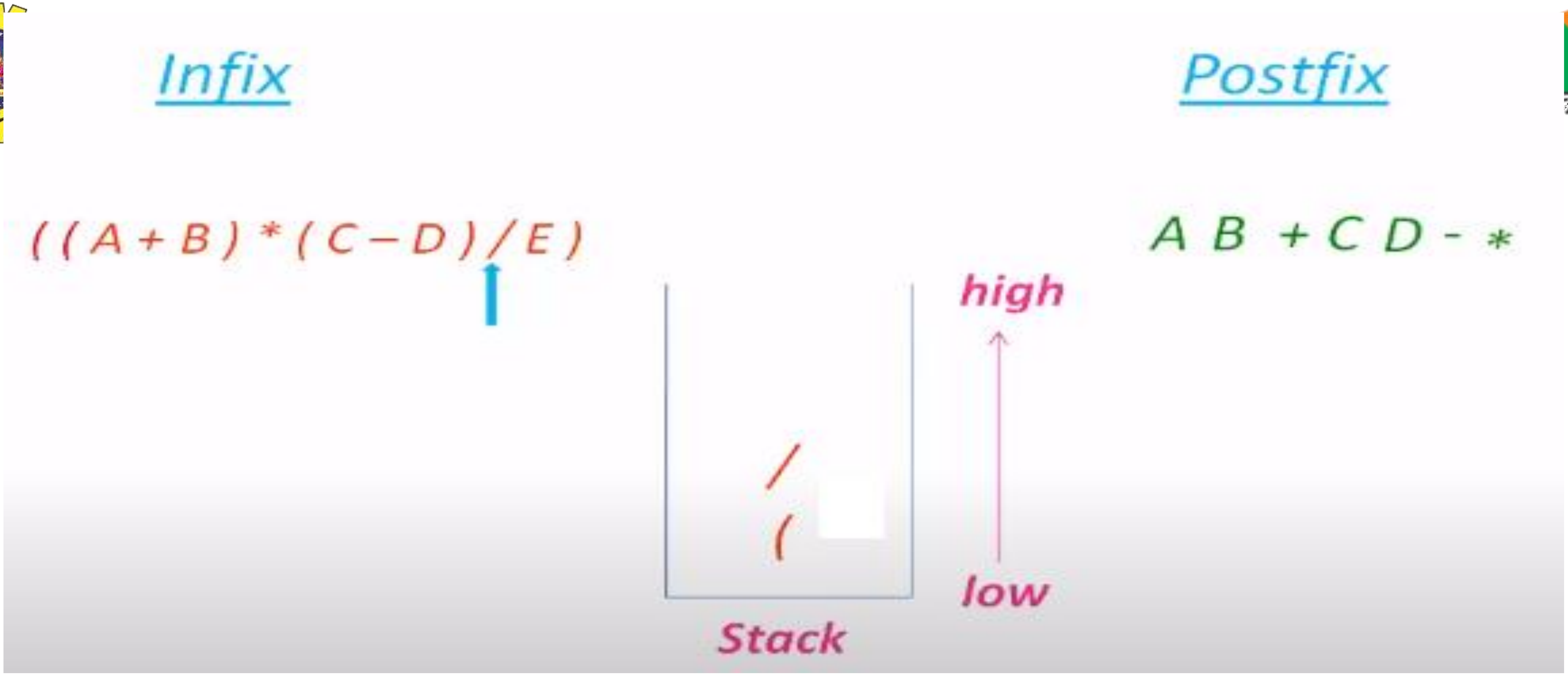
A B + C D





) , pop - from stack to postfix

pop ‘(’ from stack and discard both the parenthesis



/ equal precedence *, pop * from stack to postfix

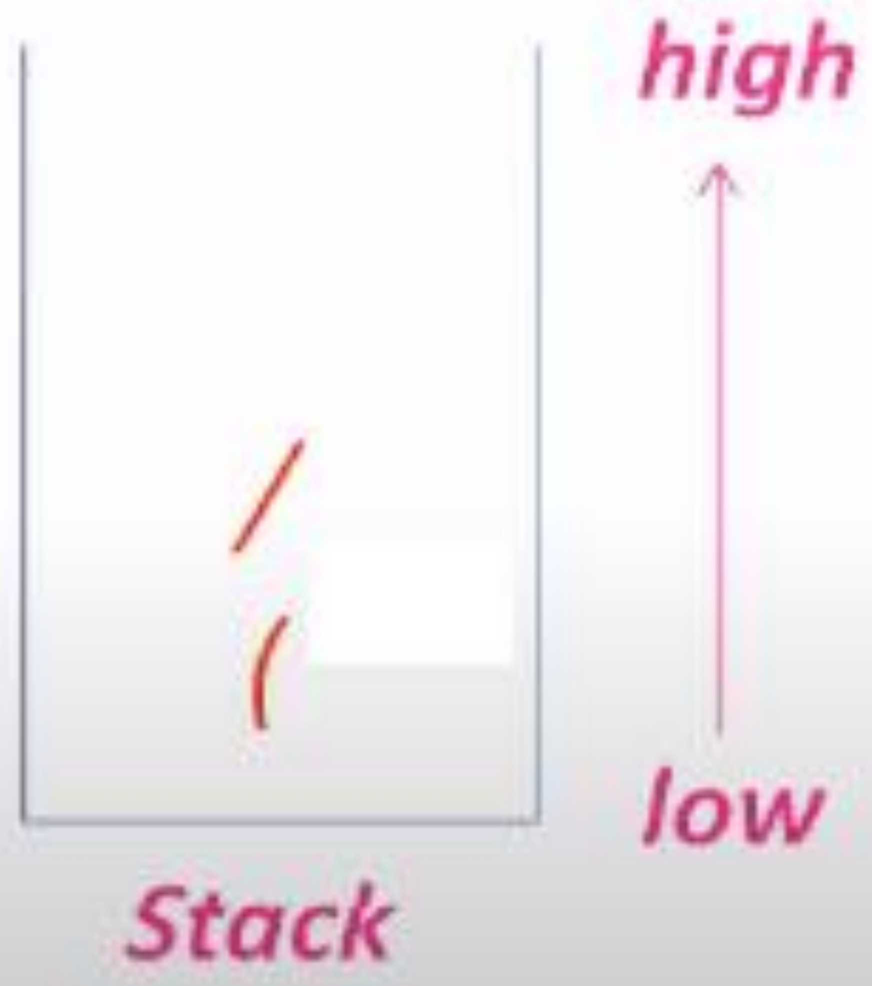
Push / to stack

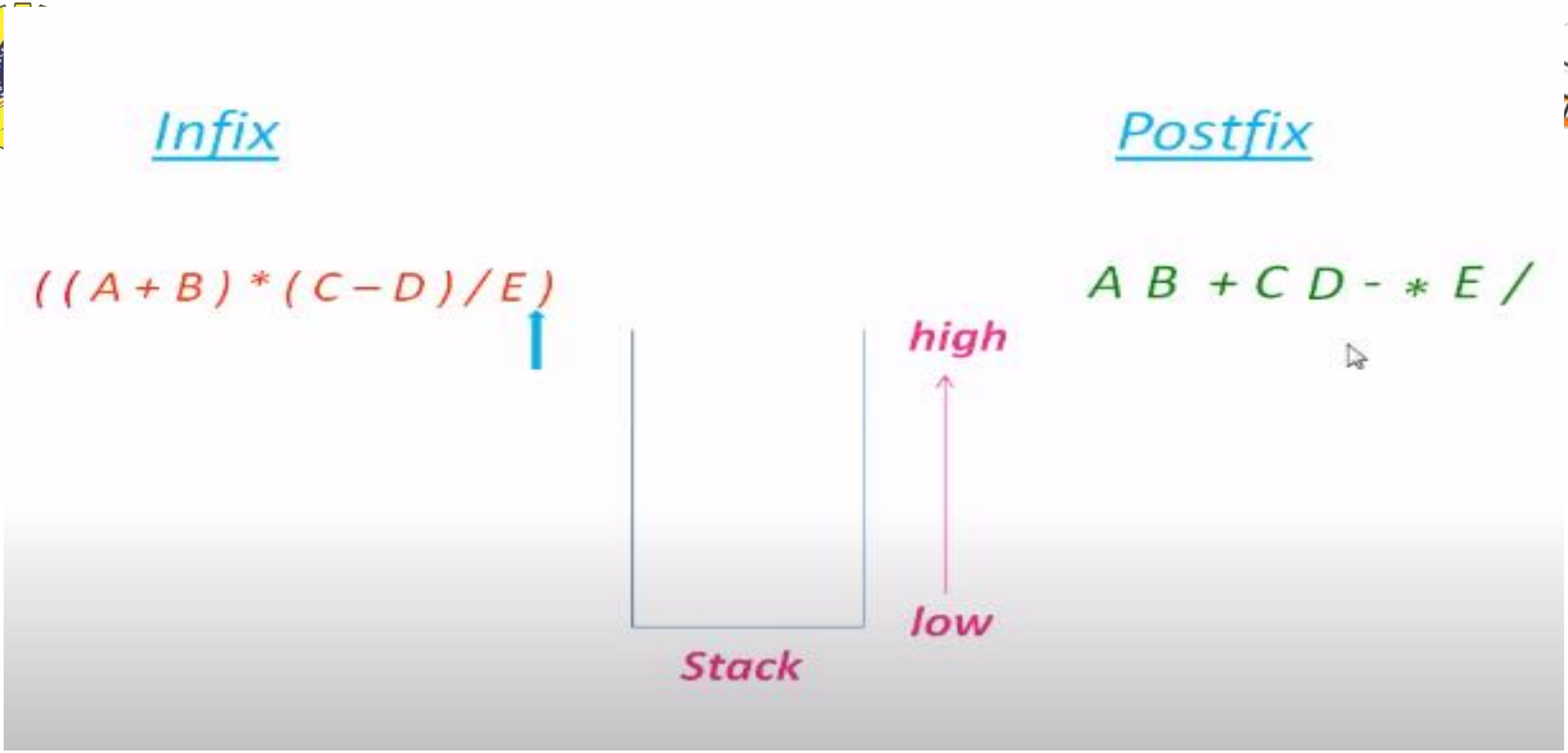
Infix

((A + B) * (C – D) / E)

Postfix

A B + C D - * E





) , pop / from stack to postfix

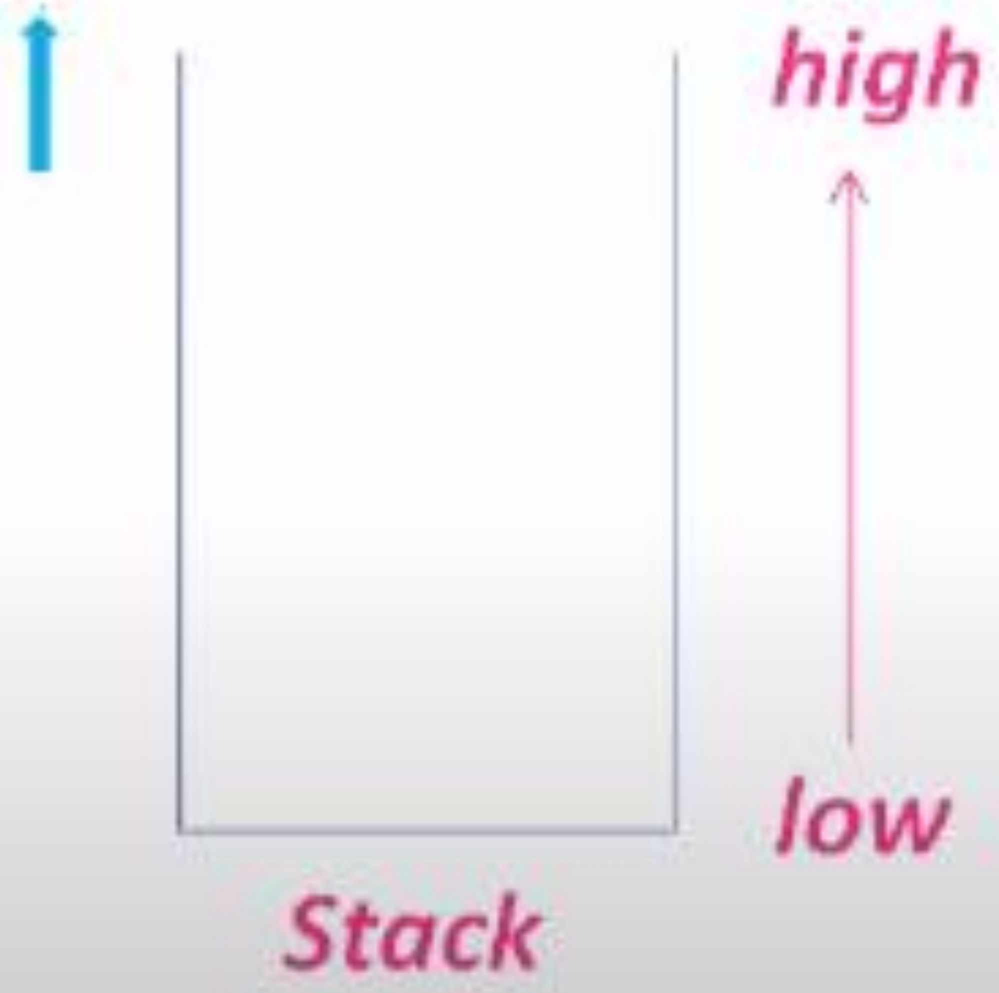
pop '(' from stack and discard both the parenthesis

Infix

((A + B) * (C – D) / E)

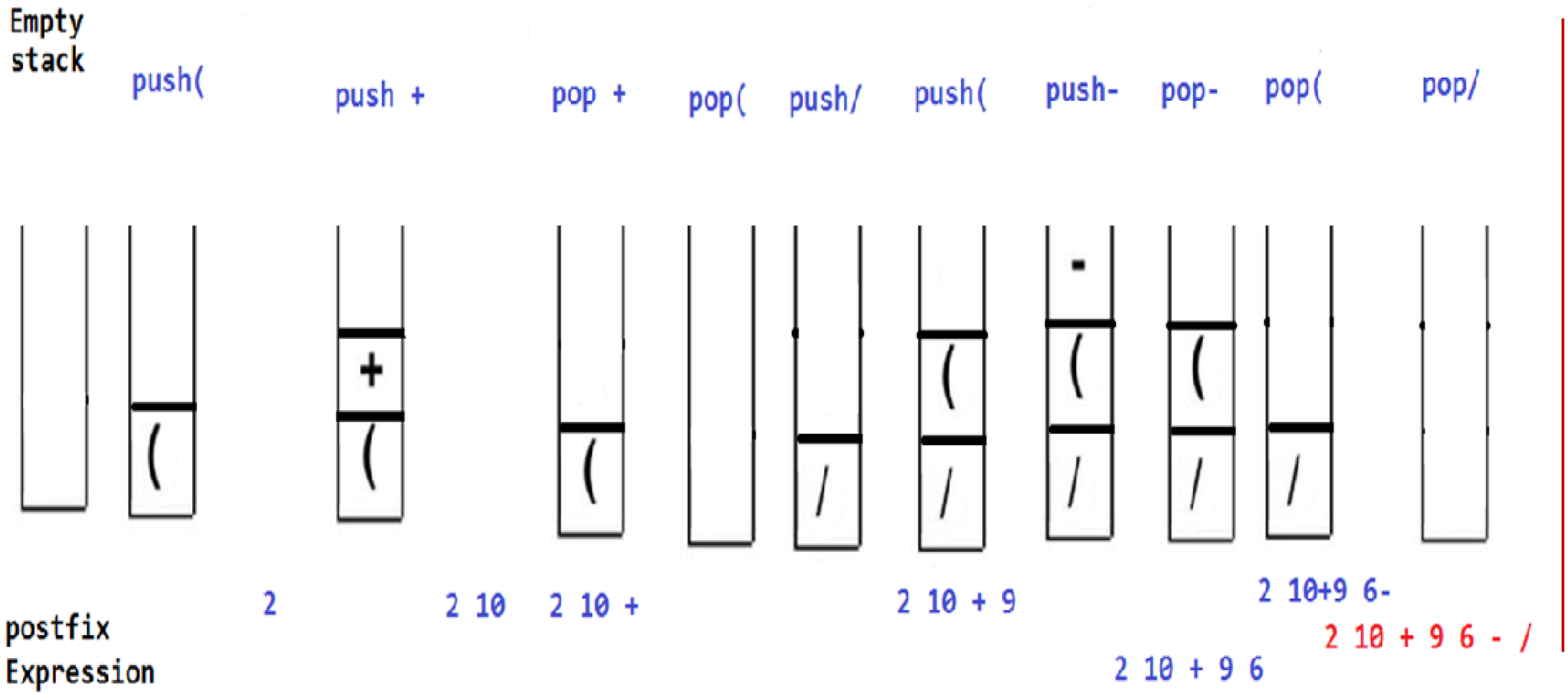
Postfix

A B + C D - * E /



Example 6

(2 + 10) / (9 - 6)   infix expression



4. Evaluate the postfix expression

Evaluate the postfix expression

Other name of postfix expression is reverse polish notation

Algorithm:

1.Get Postfix Expression and an empty stack as input

2.Scan the postfix expression from left to right

3.If element is an **operand, push** it into the stack

4.If the element is an **operator , pop twice**

5.Evaluate expression according to the operator & **push the result** back to the stack

6.Repeat step 2 to 5 until expression is end

7.The value in the stack is the final answer

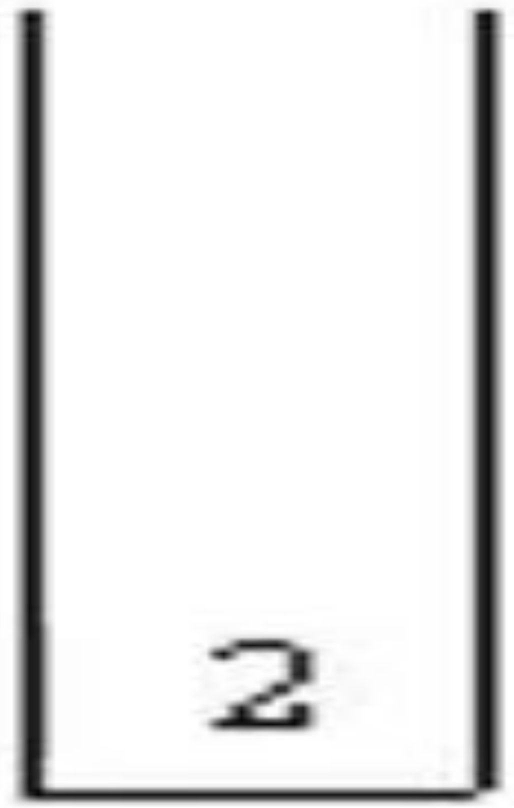
Evaluating Postfix Expression

Example: Consider the postfix expression, **2 10 + 9 6 - /**

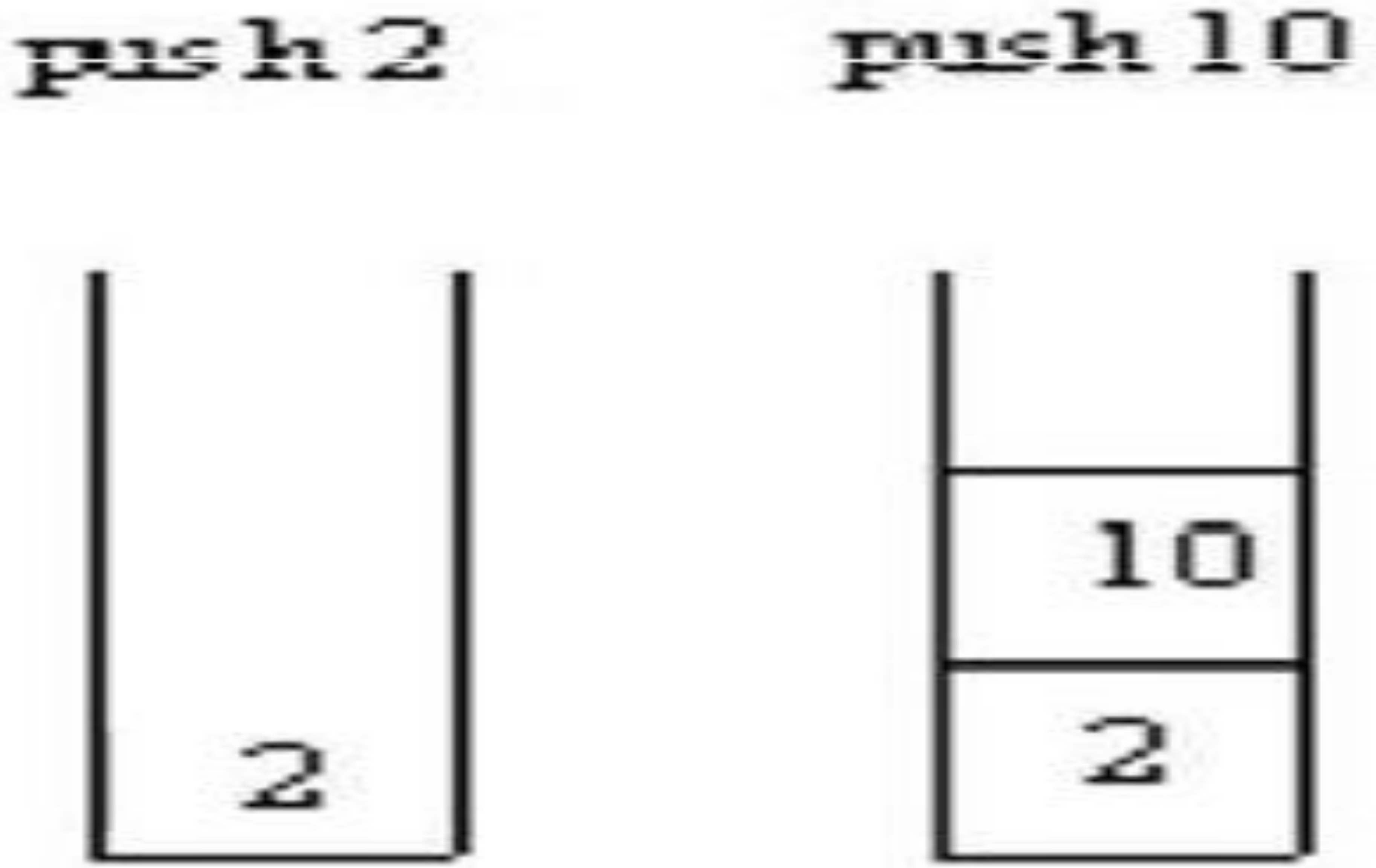
(2 + 10) / (9 - 6) in infix, the result of which is 12 / 3 = 4

2 10 + 9 6 - /

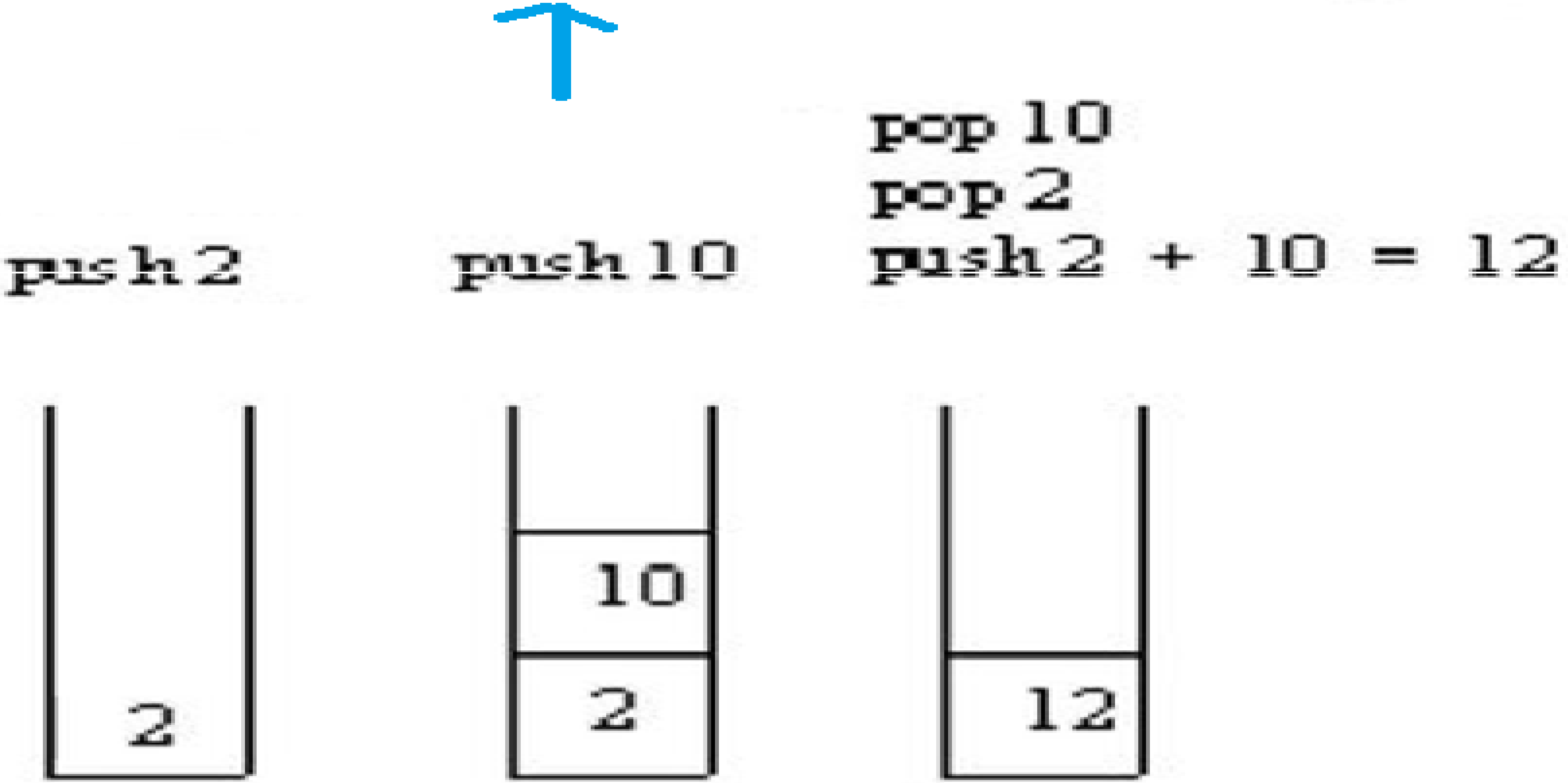
push 2



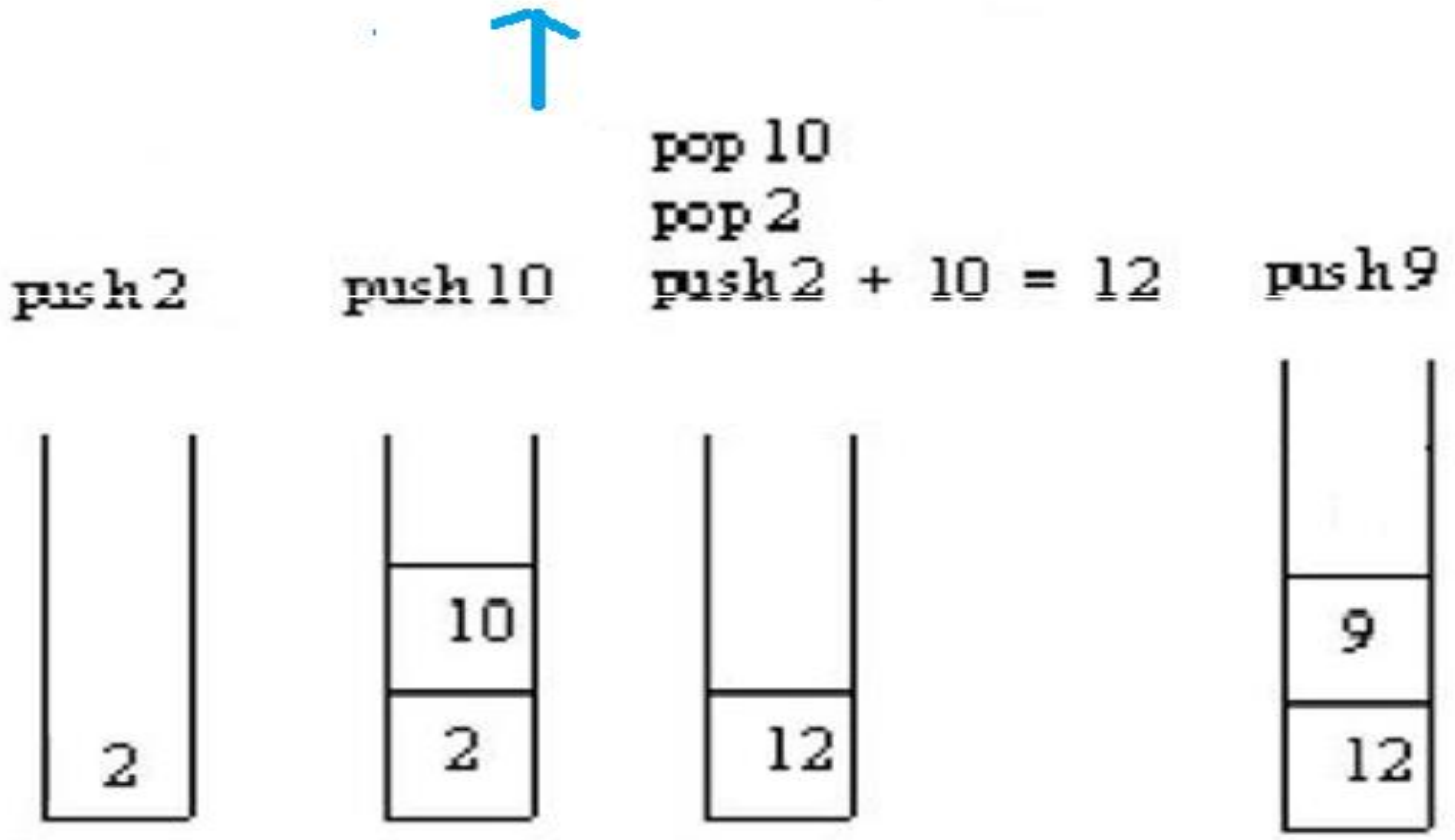
2 10 + 9 6 - /



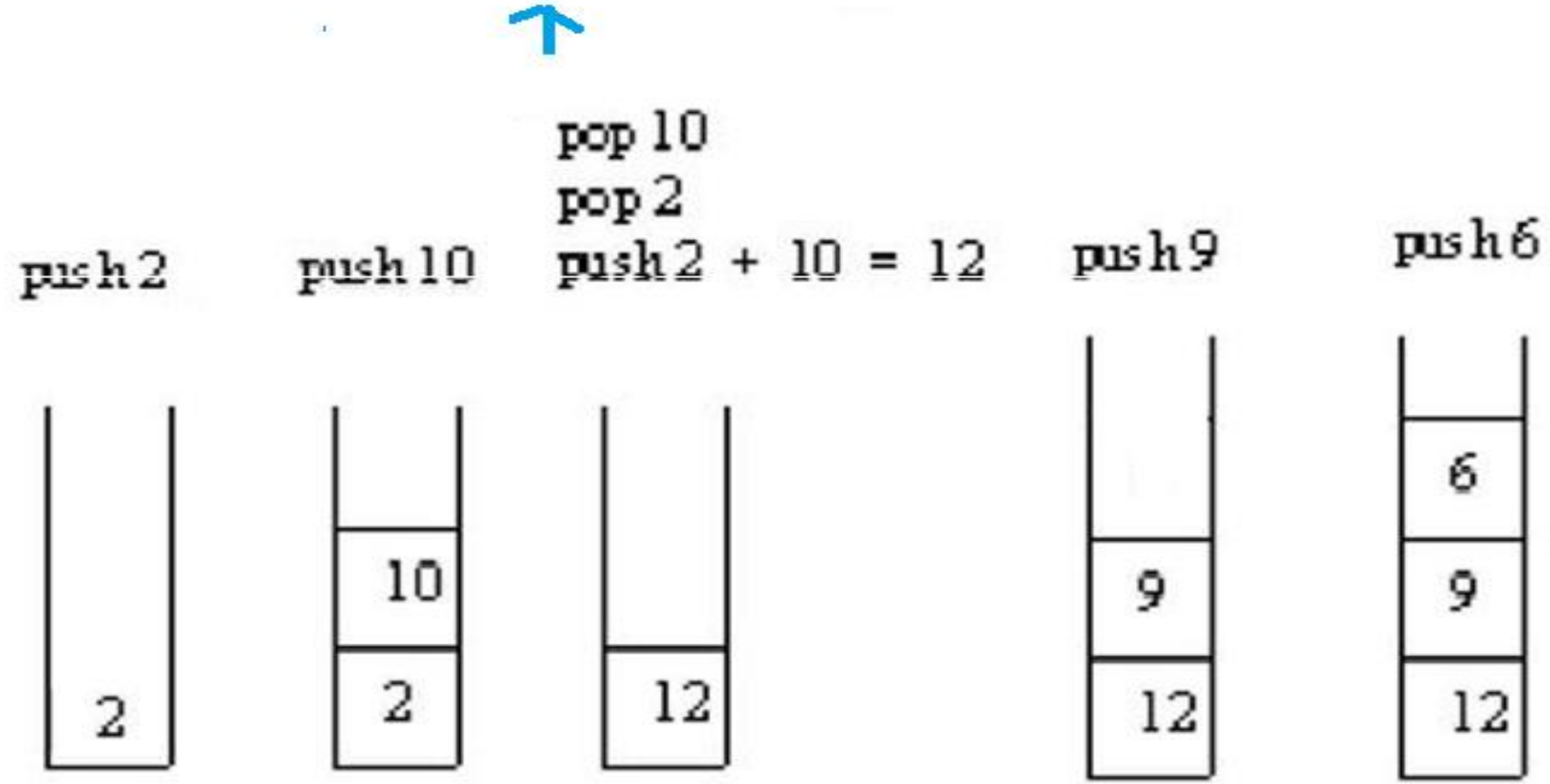
2 10 + 9 6 - /



2 10 + 9 6 - /

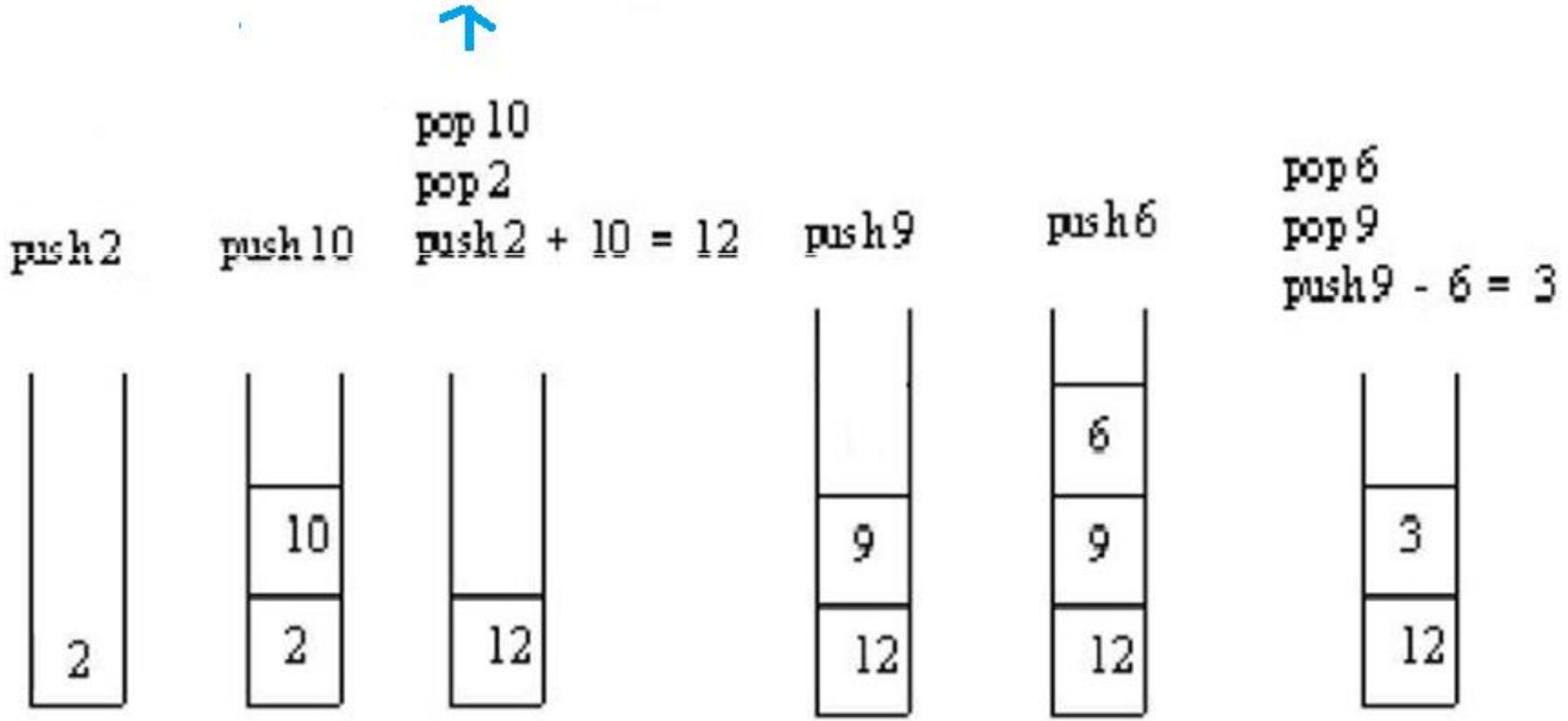


2 10 + 9 6 - /

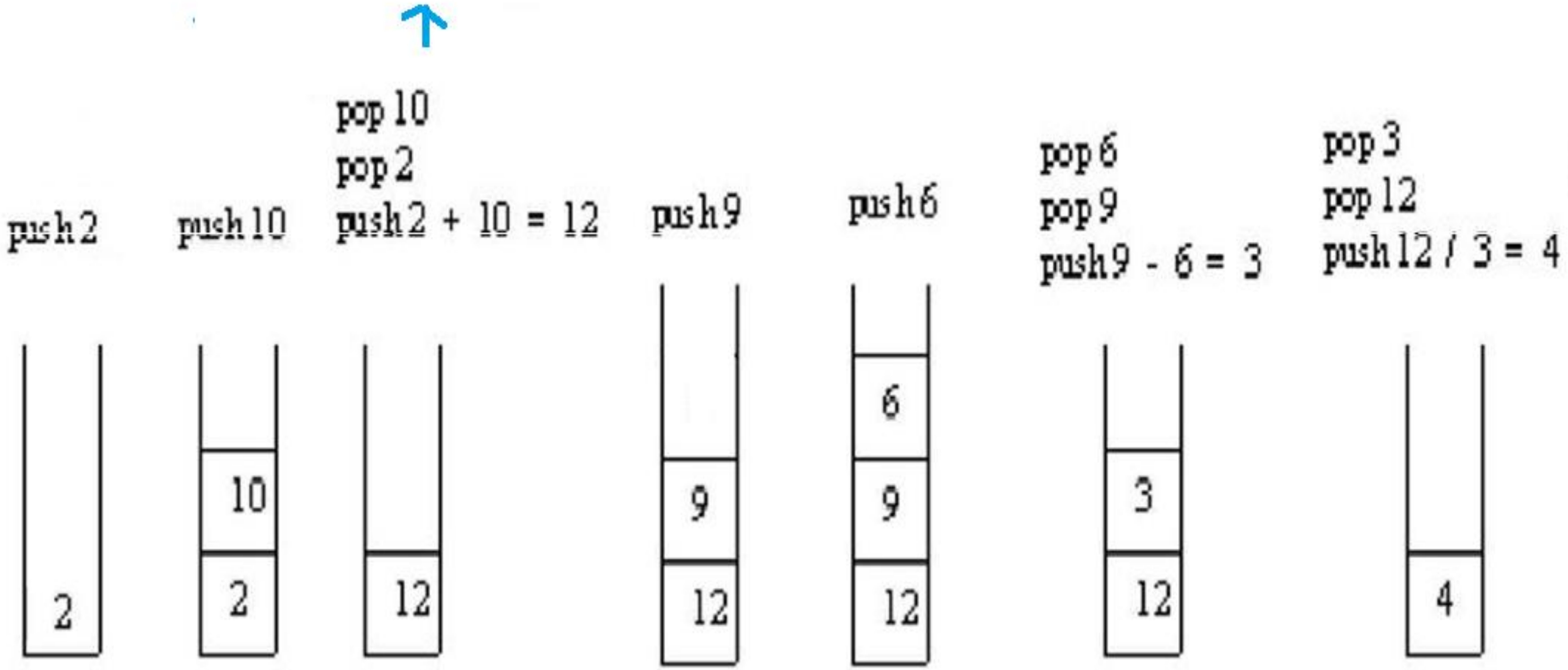


push 2
push 10
pop 10
pop 2
push 2 + 10 = 12
push 9
push 6
2
10
2
12
9
12
6
9
12

2 10 + 9 6 - /

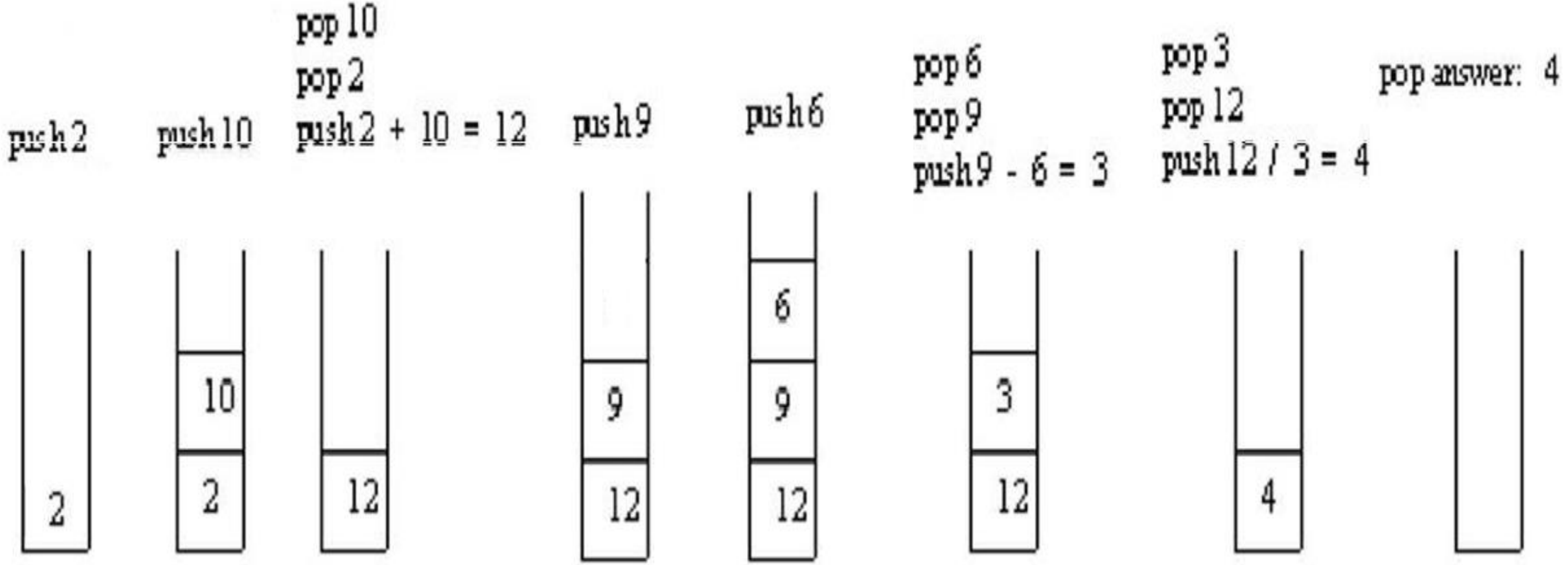

push 2
2
push 10
10
2
pop 10
pop 2
push 2 + 10 = 12
12
push 9
9
12
push 6
6
9
12
pop 6
pop 9
push 9 - 6 = 3
3
12

2 10 + 9 6 - /



push 2
push 10
pop 10
pop 2
push 2 + 10 = 12
push 9
push 6
pop 6
pop 9
push 9 - 6 = 3
pop 3
pop 12
push 12 / 3 = 4
2
10
2
12
9
12
6
9
12
3
12
4

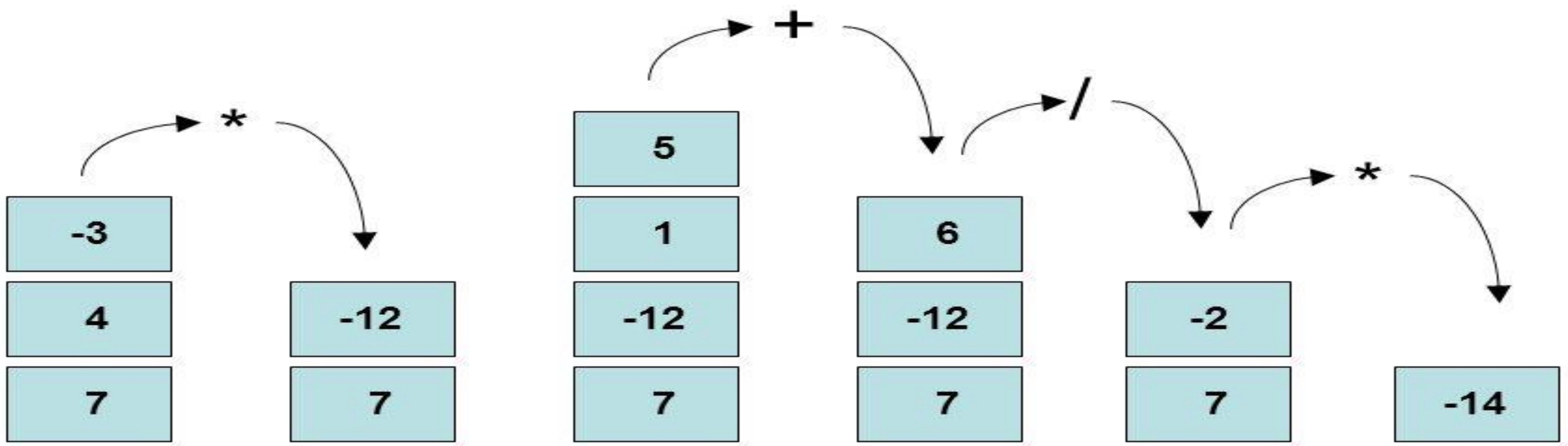
2 10 + 9 6 - /


push 2
push 10
pop 10
pop 2
push 2 + 10 = 12
push 9
push 6
pop 6
pop 9
push 9 - 6 = 3
pop 3
pop 12
push 12 / 3 = 4
pop answer: 4
2
10
2
12
9
12
6
9
12
3
12
4

Example 2

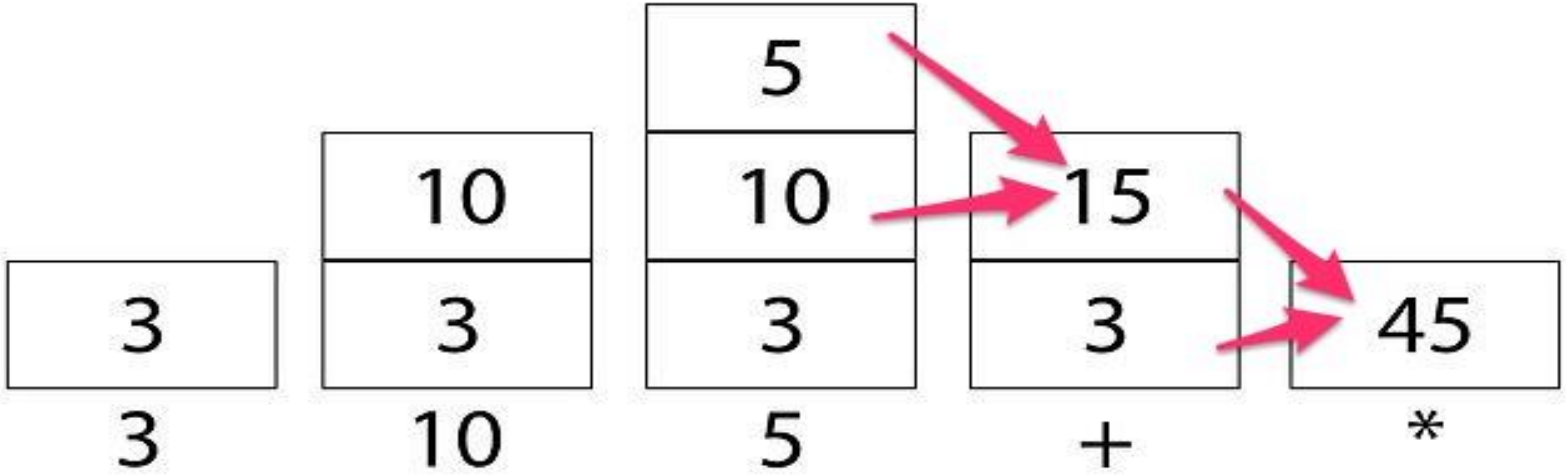
Evaluating Postfix Expressions

- Expression = 7 4 -3 * 1 5 + / *



Example 3

Equation: 3 10 5 + *



Thank you