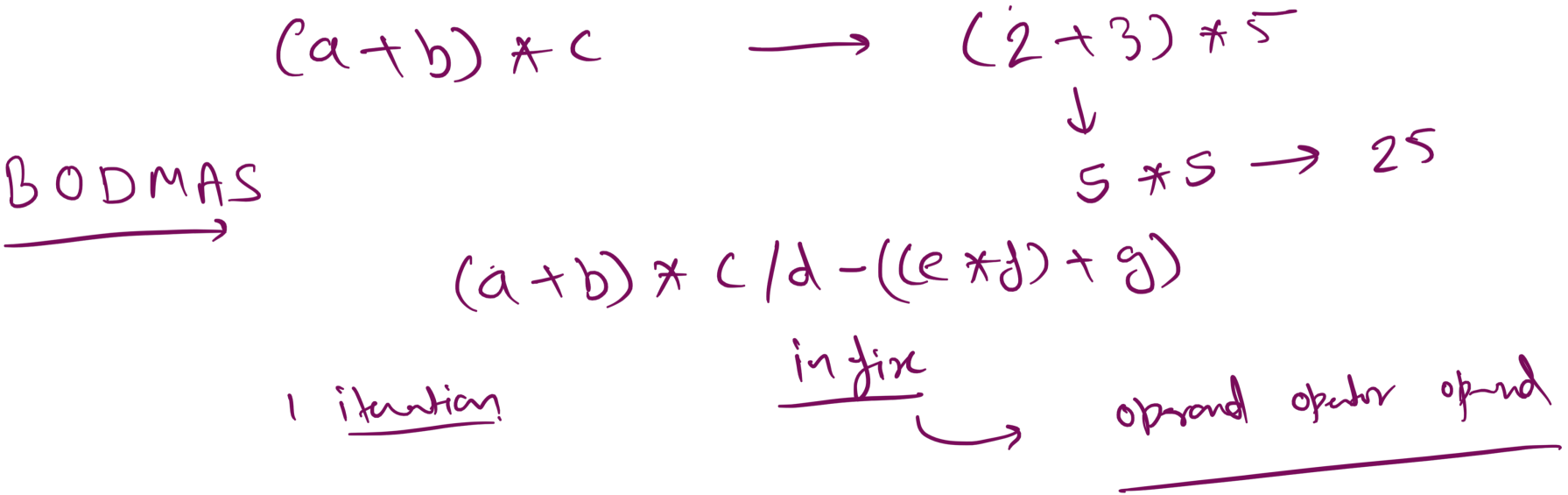
Stack Problems - I

Infix, Postfix and Prefix Expressions



Precedency of operators

right → ^

left associative

*, /
+, -

3 * 4 - 2

→ 3 - 4 * 2

BODMAS

(3 ^ (4 ^ 5))

3^{4^5}

((3 * 4) / 5)

* = left to right precedence when same precedence

$3 + 4 * ((5 - 2) / 7) - 3 * (2 - 3)$

$(3 / 7)$ $\quad$ $(2 - 3)$

$3 + 4 * 0 - 3 * (-1)$

$3 + 0 + 3$

$= \boxed{6}$

infix → Postfix

$a \circ b$

$\downarrow$

$\longrightarrow a\ b\ \circ$

left to right

No brackets

Infix to Postfix Conversion (Logic)

a ∘ b → ab∘

((a + b) * c)

↓

ab+ c *

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

ab-c+ de+* fe/ -

a+b → ab+

2 + 3 * 4

2 3 4 * +

4 5 9

$(a + b/c) * ((g/e) - g)$

a bc/+ ge/g- *

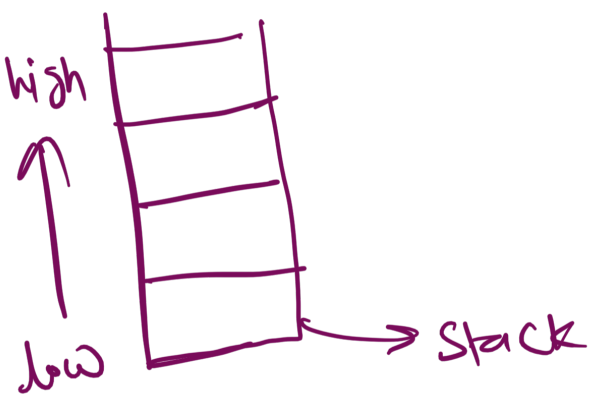
/, *

+, -

45+9 → 45 9+

4 + 59

4 59 +



operators / (bracket

① → (

② →)

③ → operator

④ → operand

o/p → abc/+ge/g- *

No brackets

Infix to Postfix Conversion

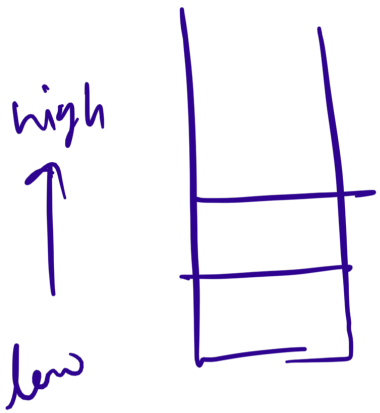
1. If we have an opening parenthesis "(", we push it into the stack.
2. If we have an operand, we append it to our postfix expression.
3. If we have a closing parenthesis ")" we keep popping out elements from the top of the stack and append them to our postfix expression until we encounter an opening parenthesis. We pop out the left parenthesis without appending it.
4. If we encounter an operator:-

   4.1. If the operator has higher precedence than the one on top of the stack (We can compare), we push it in the stack.

   4.2. If the operator has lower or equal precedence than the one on top of the stack, we keep popping out and appending it to the postfix expression.

5. When the last token of infix expression has been scanned, we pop the remaining elements from stack and append them to our postfix expression.*

$(a+b)*c-d$



ab+c*d-

$(2 * 3) - (4 / 5 - 6 + 7)$

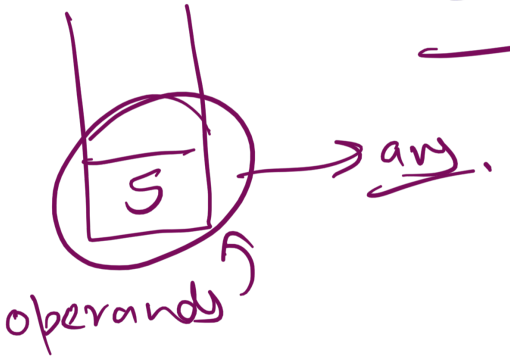
$6 - (0 + 1) = 6 - 1 = 5$.

$23 * 45 / 6 - 7 + -$

left to right

↑ ① → operator

② → operand



$6 - 1 = 5$

Evaluation of Postfix Expression (Logic)

Travel left to right

① operator

second = stack.pop()

first = stack.pop()

eval = first operator second

push back to stack

② operand

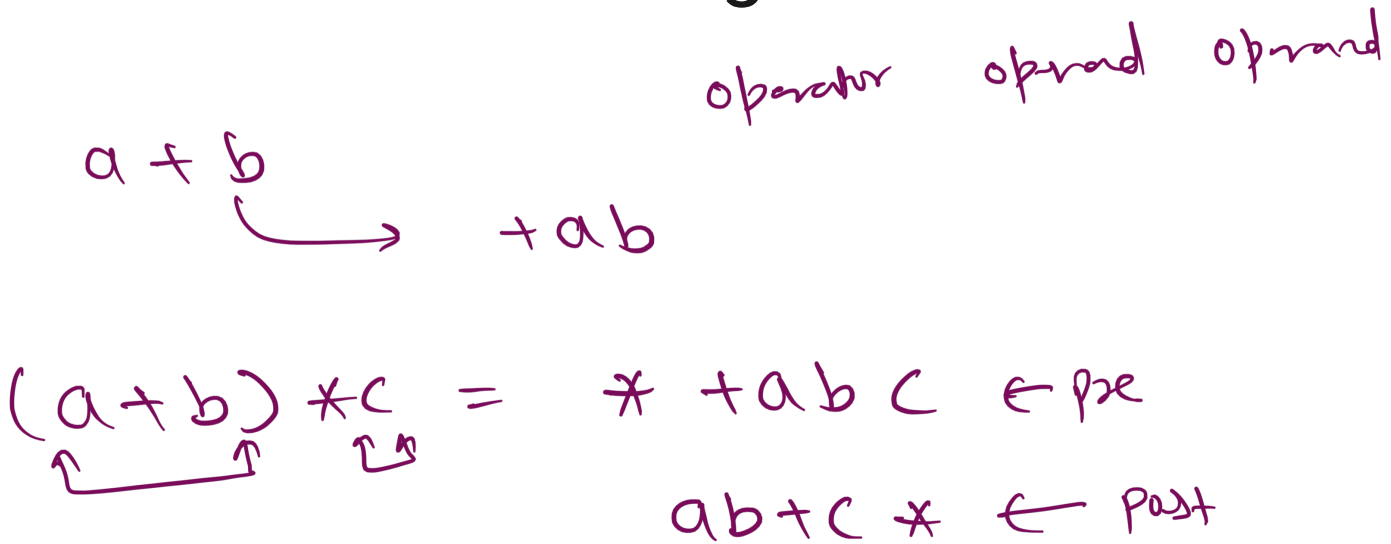
push back to stack.

"23*45/6-7+-"

-6+7 = 1

6-1 = 5

Infix to Prefix Conversion (Logic)



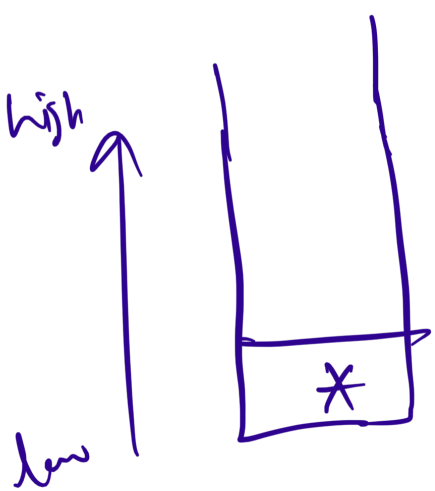
(2 - 3) * (4 - 5 + (6 * 7))

* - 2 3 + - 4 5 * 6 7

(a - b) * c

* - a b c

a b - c *



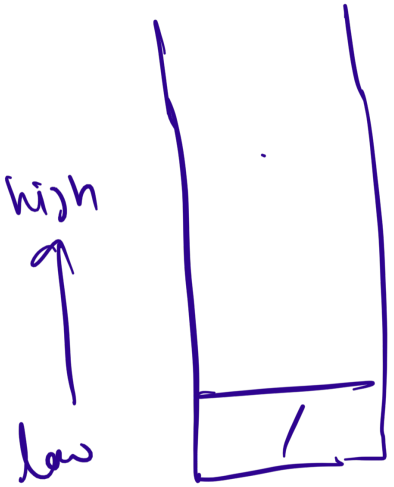
c b a - *

reverse

* - a b c

Infix to Prefix Conversion

1. First, reverse the infix expression given in the problem.
2. Scan the expression from left to right.
3. Whenever the operands arrive, print them.
4. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
5. If the incoming operator has higher precedence than the TOP of the stack, push the incoming operator into the stack.
6. If the incoming operator has the same precedence with a TOP of the stack, push the incoming operator into the stack.
7. If the incoming operator has lower precedence than the TOP of the stack, pop, and print the top of the stack. Test the incoming operator against the top of the stack again and pop the operator from the stack till it finds the operator of a lower precedence or same precedence.
8. If the incoming operator has the same precedence with the top of the stack and the incoming operator is ^, then pop the top of the stack till the condition is true. If the condition is not true, push the ^ operator.
9. When we reach the end of the expression, pop, and print all the operators from the top of the stack.
10. If the operator is ')', then push it into the stack.
11. If the operator is '(', then pop all the operators from the stack till it finds) opening bracket in the stack.
12. If the top of the stack is ')', push the operator on the stack.
13. At the end, reverse the output.



(a - b * c) / ((e + f) * g / h)

/ - a * b c / * + e f g h

h g f e + * / c b * a - /

↓ reverse

/ - a * b c / * + e f g h

Evaluation of Prefix Expression (Logic)

Practice.

Convert a Postfix expression to a Prefix Expression

post : ab+c* → pre ?