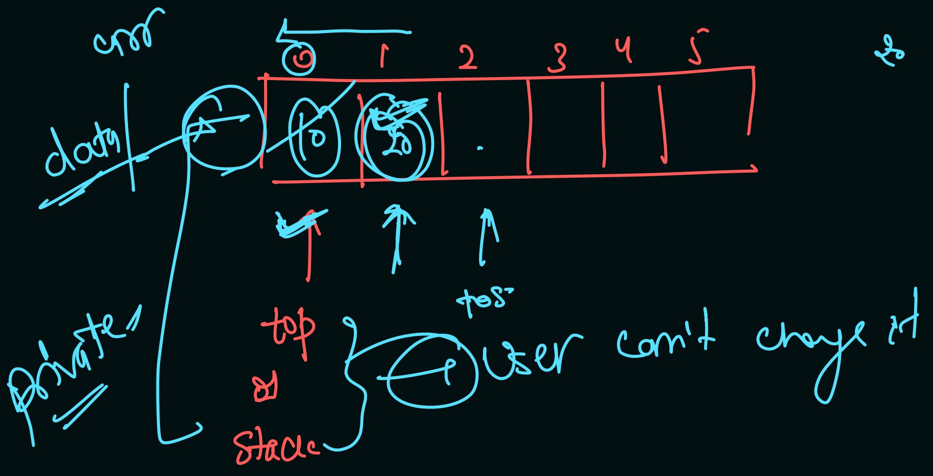
Stack - Implementation!

max size.

Stack → object creation → capacity of Stack

capacity → 5.



&

public

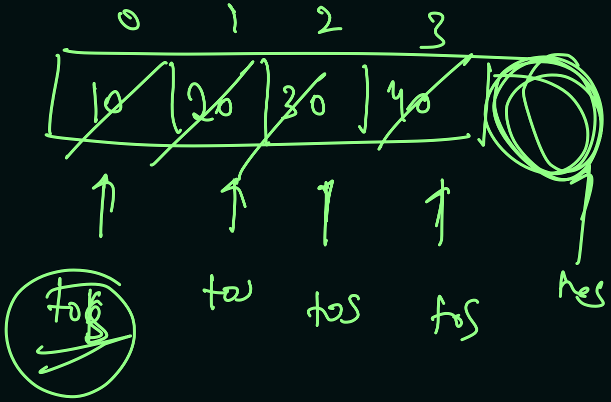
push → Add data at 'tos' index
tos ++

pop → val = arr[tos-1];
tos - - ;

top → val = arr[tos];
return val.

size → return tos;

display.



push - 10 pop
" - 20 pop
- 30 pop
- 40 pop
- 50 pop

Self work →

- → (1) Nomal Stack
- → (2) Dynomic Stack
- → (3) Minimum stack I
- → (4) Adapters

Class work

- Queue → Nomal Queu
  - → Dynomic Queu.
- Min stack → constant space
- Evaluation → Infix
  - prefix
  - postfix

10:00 - 1:00

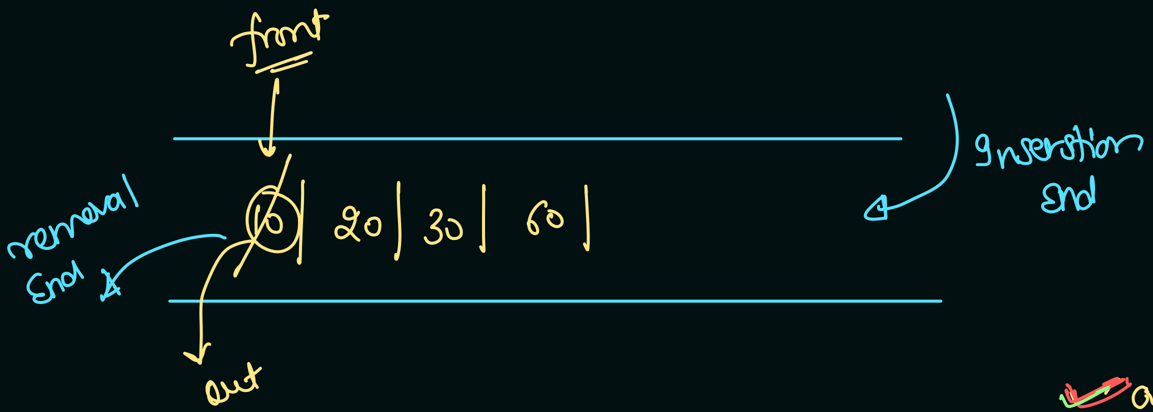
Normal Queue : (using array)

1. Add
2. remove
3. peek
4. Size
5. display

FIFO → First in first out

BFS
graph

Linked List



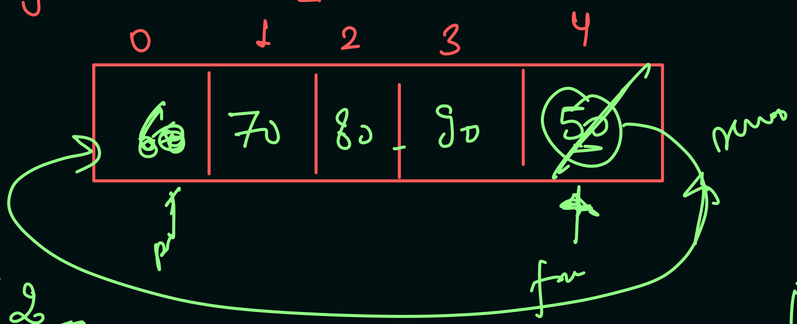
~~add → 10~~
~~add → 20~~
~~add → 30~~ →
~~front / peek~~ → 10
~~remove()~~ → 10
add → 60 → rear
front() → 20
size() → 2
add → 50
add →

add 40
add 50

pointer → front] → peek()

Initial capacity → 5

adding index
↳ front + size
↓
0 + 1



front = ~~0~~ ~~1~~ 2

size = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~4~~ ~~5~~ 4

circular

modulo operation

(2 + 4) % 5 = 1

index = (front + size) % capacity → 0

Infix Evaluation :

Infix Statement]→ Human]→ way of solving an equation

Bracket
Divide = = Multiply
Addition = = subtraction

BDMAS

Priority order
→ Bracket
→ Divide
→ Multiply
→ Addition
→ subtraction

derca

Statement → $10 - 4 * 2$

$\underbrace{10 - 4} * 2$

↓

$6 * 2$

||

$\cancel{12}$

[Hit & try]

$10 - \underbrace{4 * 2}$

↓

$10 - 8$

||

$\cancel{2}$

Correct Approach] → check priority of operation

(, {, },],), [
*, /
+, -

$10 - \underline{4 * 2}$

↓

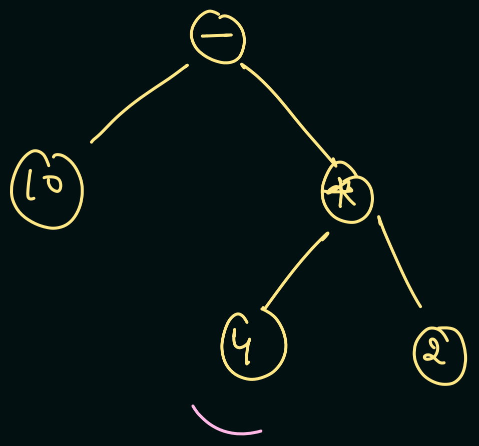
$\underline{10 - 8}$

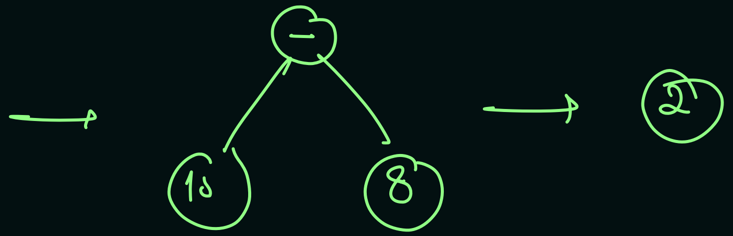
$\boxed{2}$ || ✓

10 - 4 * 2 → Infix Statement / Infix Equation

↓

Convert infix tree





→ In order of tree 10 - 4 * 2

→ Infix

postorder → [10 4 2 * -] → postfix

preorder → [- 10 * 4 2]

→ prefix

Infix

Infix Evaluation

Infix to prefix

Infix to postfix

Postfix

postfix Evaluation

postfix to prefix

postfix to infix

prefix

prefix Evaluation

prefix to infix

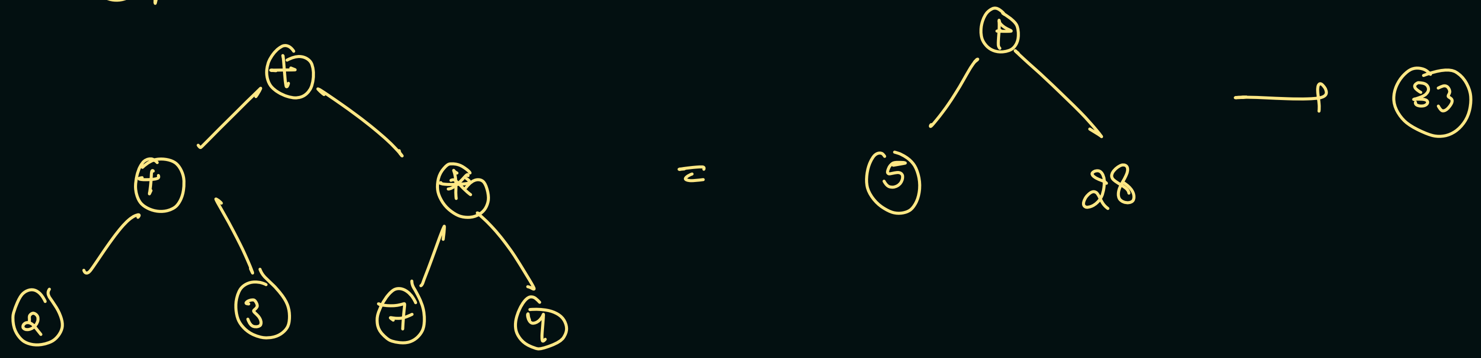
prefix to postfix

Ex $\underbrace{2+3} + \underbrace{7*4}$

↓

$2 + 3 + 28$] → left to Right for same priority

$5+28 = 33$ ←



Infix → $2+3+7*4$

prefix → $+ + 2 \ 3 * 7 \ 4$

postfix → $2 \ 3 + 7 \ 4 * +$

How to manage priority of operation

HashMap{

B → 3] → closing bracket

/ → 2

* → 2

+ → 1

- → 1

opening]

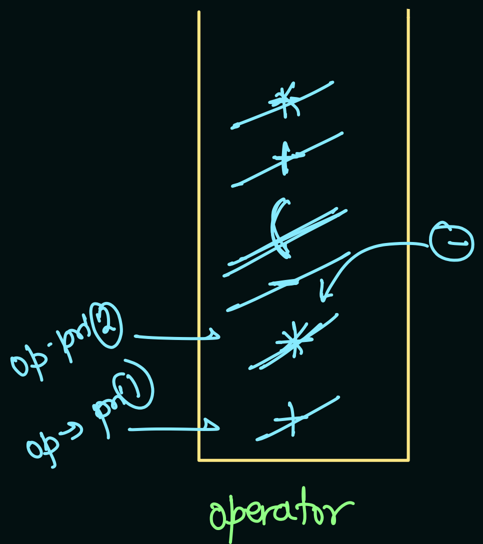
Infix Evaluation:

Ex → $\underline{2} + \underbrace{\underline{3} * \underline{2}} - (\underline{3} + \underbrace{\underline{4} * \underline{1}})$

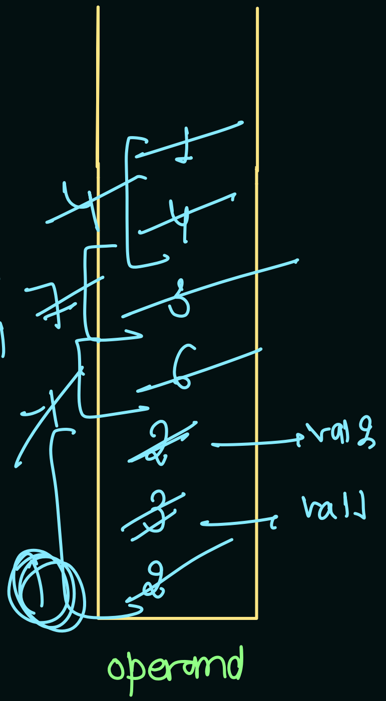
$\underbrace{2+6} - (\underbrace{3+4})$

$8 - 7 = \boxed{1}$

max priority operator can stand on min priority operator



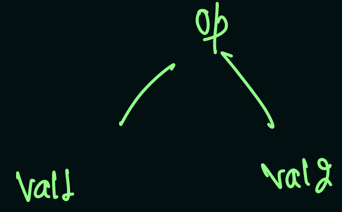
if closing bracket encounter pop until opening bracket not nil



operator =	~~*~~	~~+~~	~~−~~	+
val2 =	~~1~~	~~4~~	~~7~~	−1
val1 =	~~4~~	~~3~~	~~6~~	2
result =	~~4~~	~~7~~	~~−1~~	1

= 1

3 * 2] = 6



→

Infix → val1 op val2

prefix op val1 val2

postfix val1 val2 op]

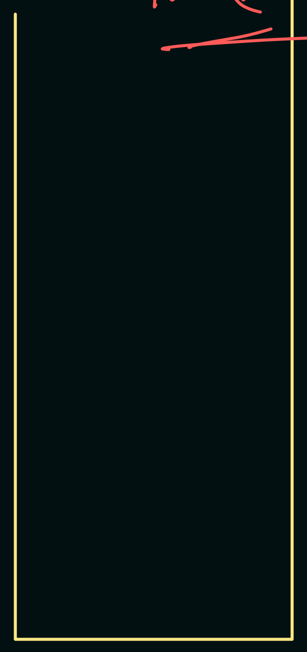
Infix to prefix :

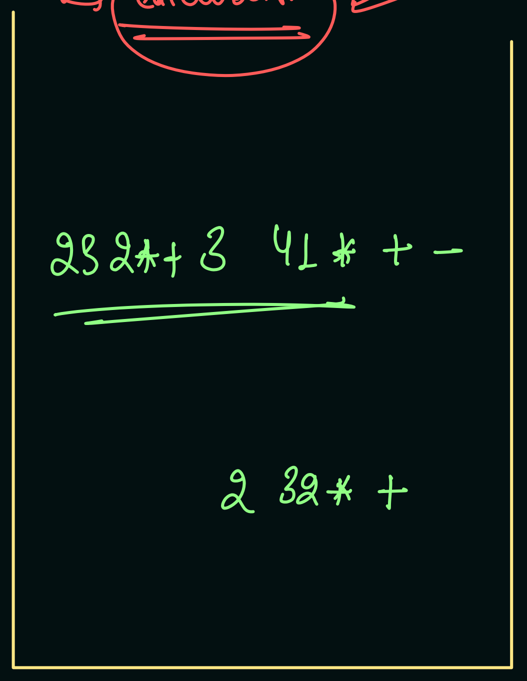
2 + 3 * 2 − (3 + 4 * 1) → equation

Infix → prefix

12+(3*7)

calculator


operator stack



value stack

op = −

val2 = +3*41

val1 = +2*32

Result = op val1 val2] pr

= *32

= +2*32

= *41

= − +2*32+3*41

~~prefix~~

How to solve postfix Expression ?

prefix → − + 2 * 3 2 + 3 * 4 1

postfix → 2 3 2 * + 3 4 1 * + −

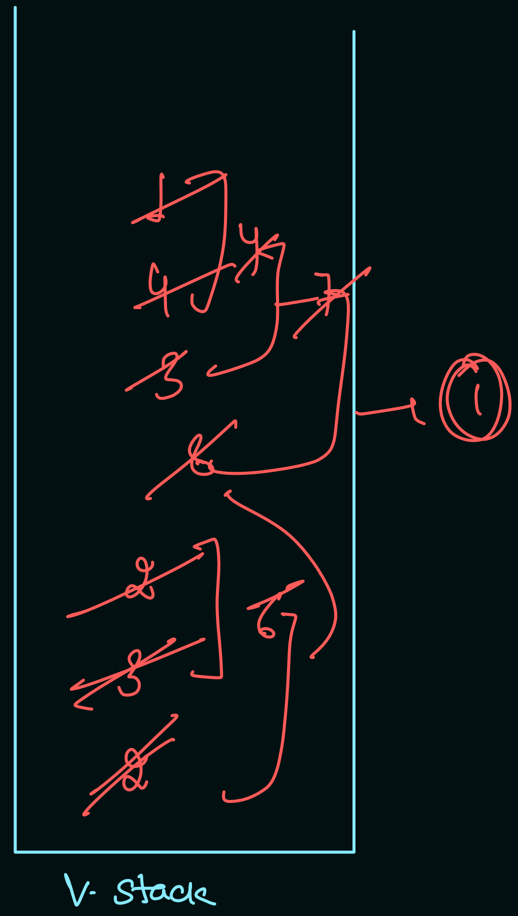
postfix Evaluation:

↳ val1 val2 operator

Exp → 2 3 2 * + 3 4 1 * + -

values

operator



operator =	~~*~~	~~+~~	~~*~~	~~+~~	−
val2 =	~~2~~	~~6~~	~~1~~	~~4~~	7
val1 =	3	~~2~~	~~4~~	3	8

res = val1 op val2 =

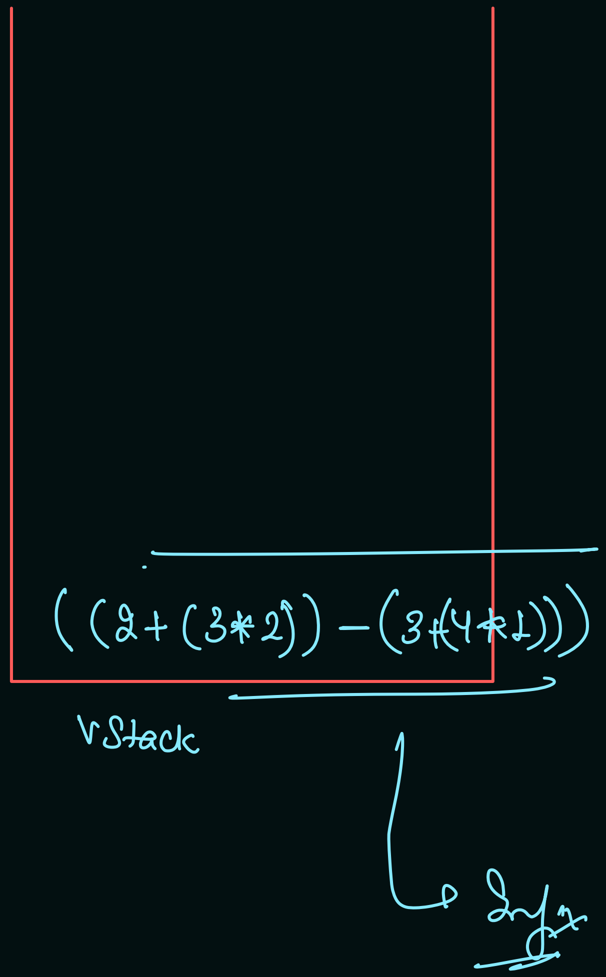
val1 val2 op

op val1 val2

val1 op val2

postfix to Infix:

Exp → 2 3 2 * + 3 4 1 * + -



(val1 op val2)

val2 = ~~2~~ ~~(3*2)~~ ~~1~~ ~~(4*1)~~ (3+(4*1))

val1 = ~~3~~ ~~2~~ ~~4~~ ~~3~~ (2+(3*2))

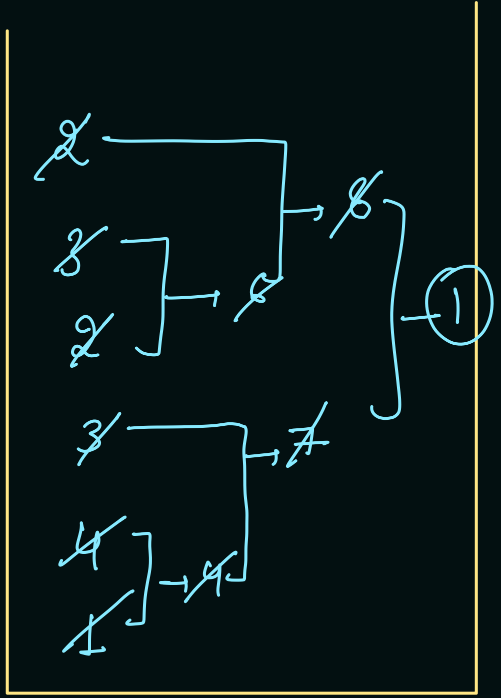
op = ~~*~~ ~~+~~ ~~*~~ ~~+~~ -

Prefix

Exp→ - + 2 * 3 2 + 3 * 4 1

- + 2 * 3 2 + 3 * 4 1

op val1 val2



op = ~~*~~ ~~+~~ ~~*~~ ~~+~~ -

val1 = ~~4~~ ~~8~~ ~~3~~ ~~2~~ 8

val2 = ~~1~~ ~~4~~ ~~2~~ ~~6~~ 7

res = ~~7~~ 1

Infix → (val1 op val2)

Postfix → val1 val2 op

'0' → as character

Int num = '0'; ← ASCII

48 → numerical

num = '0' - 48 = 0

nu = '1' - 48 = 1

1 49 OR

'1' - '0' = 1

→ computer can store binary Number

How to store 'a' in computer? → ASCII] Assign a decimal to Every character.

Primitive data type

① byte

② Short

③ int

④ long

⑤ double

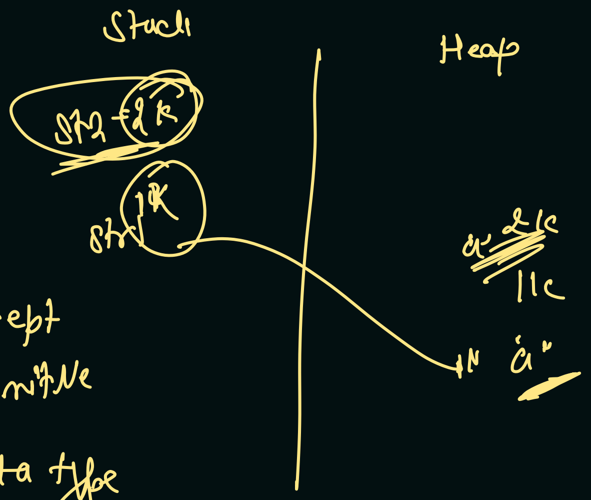
⑥ boolean

⑦ float

⑧ char

} Memory on Stack

Str2 'a'



Except primitive data type

Everything is create on Heap → string create on String pool

Str.equals(str2)

compare To → comparable

equals

compare → comparators