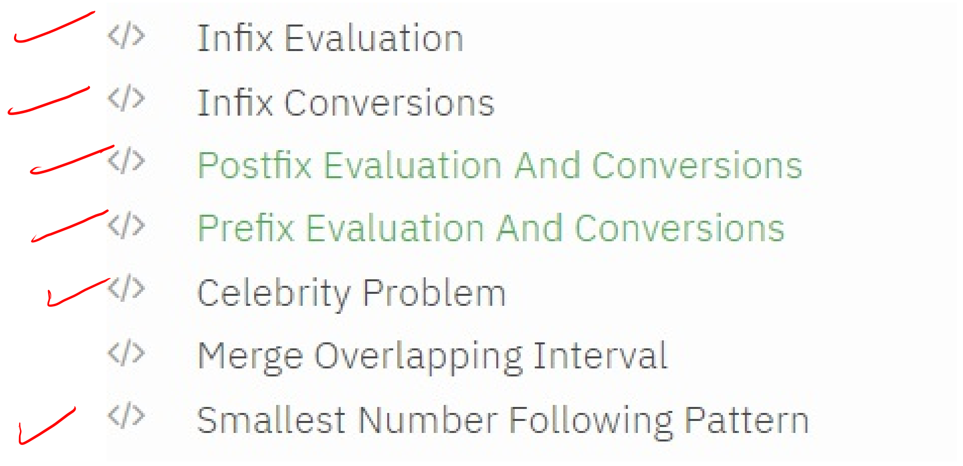


- ✓ Infix Evaluation
- ✓ Infix Conversions
- ✓ Postfix Evaluation And Conversions
- ✓ Prefix Evaluation And Conversions
- ✓ Celebrity Problem
- Merge Overlapping Interval
- ✓ Smallest Number Following Pattern

$1+2\times3/6+8\times2$

$17 \rightarrow$ BODMAS

SAMD

SADM

SDAM

precedence

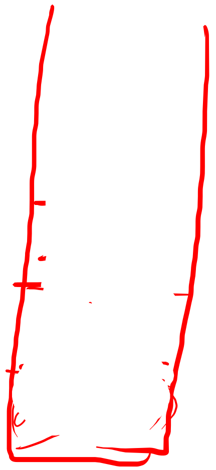
$((a+(b\times c))/d)$

$$\begin{tikzcd}
& \text{Infix (Human Readable)} \arrow[dl] \arrow[dr] & \\
\text{Postfix} & & \text{Prefix}
\end{tikzcd}$$

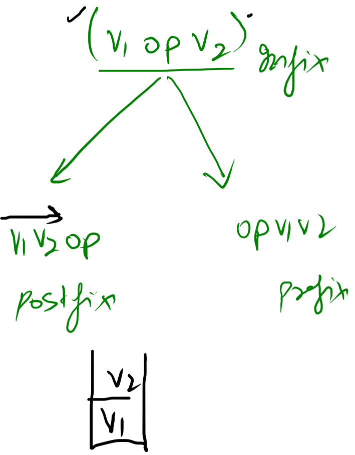
$a\ b\ c\ \boxed{*}\ +\ d\ /$

$/\ d\ +\ a\ \times\ b\ c$

$$\begin{tikzcd}
& (v_1\ op\ v_2)\ \text{Infix} \arrow[dl] \arrow[dr] & \\
v_1 v_2\, op\ \text{Postfix} & & op\, v_1 v_2\ \text{Prefix}
\end{tikzcd}$$

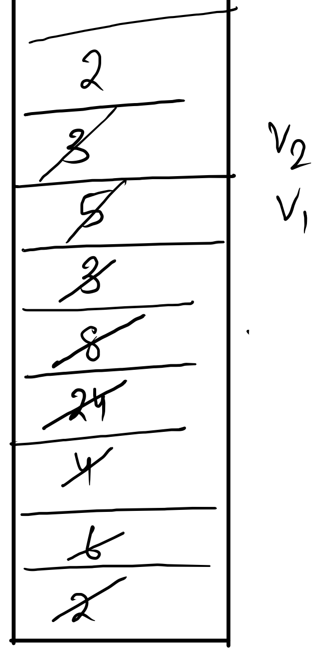


$(a+(b\times c))\ /\ d$

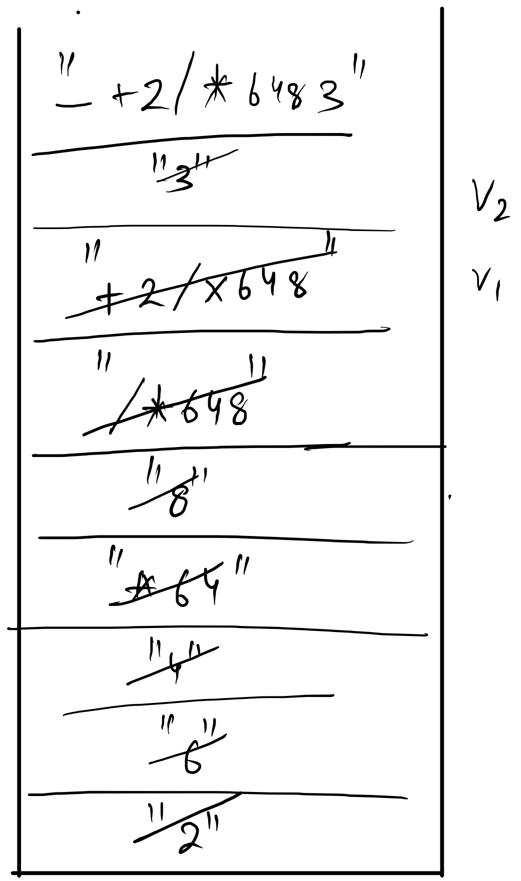


2 6 4 * 8 / + 3 −

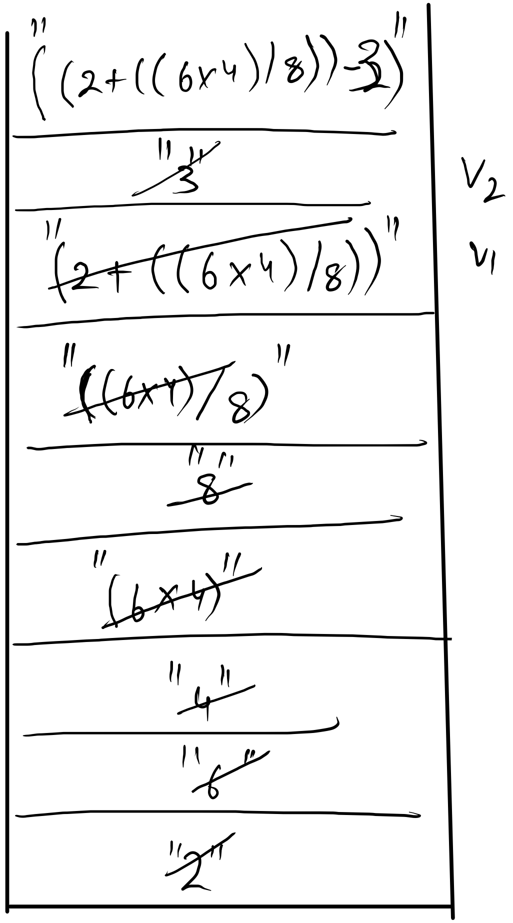
op =: −



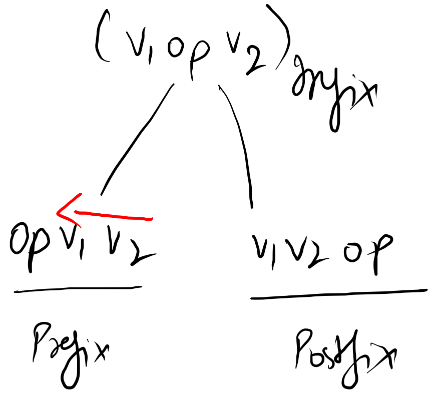
Evaluation <Integer>

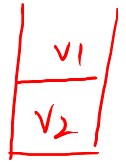


Prefix <String>



Infix <String>

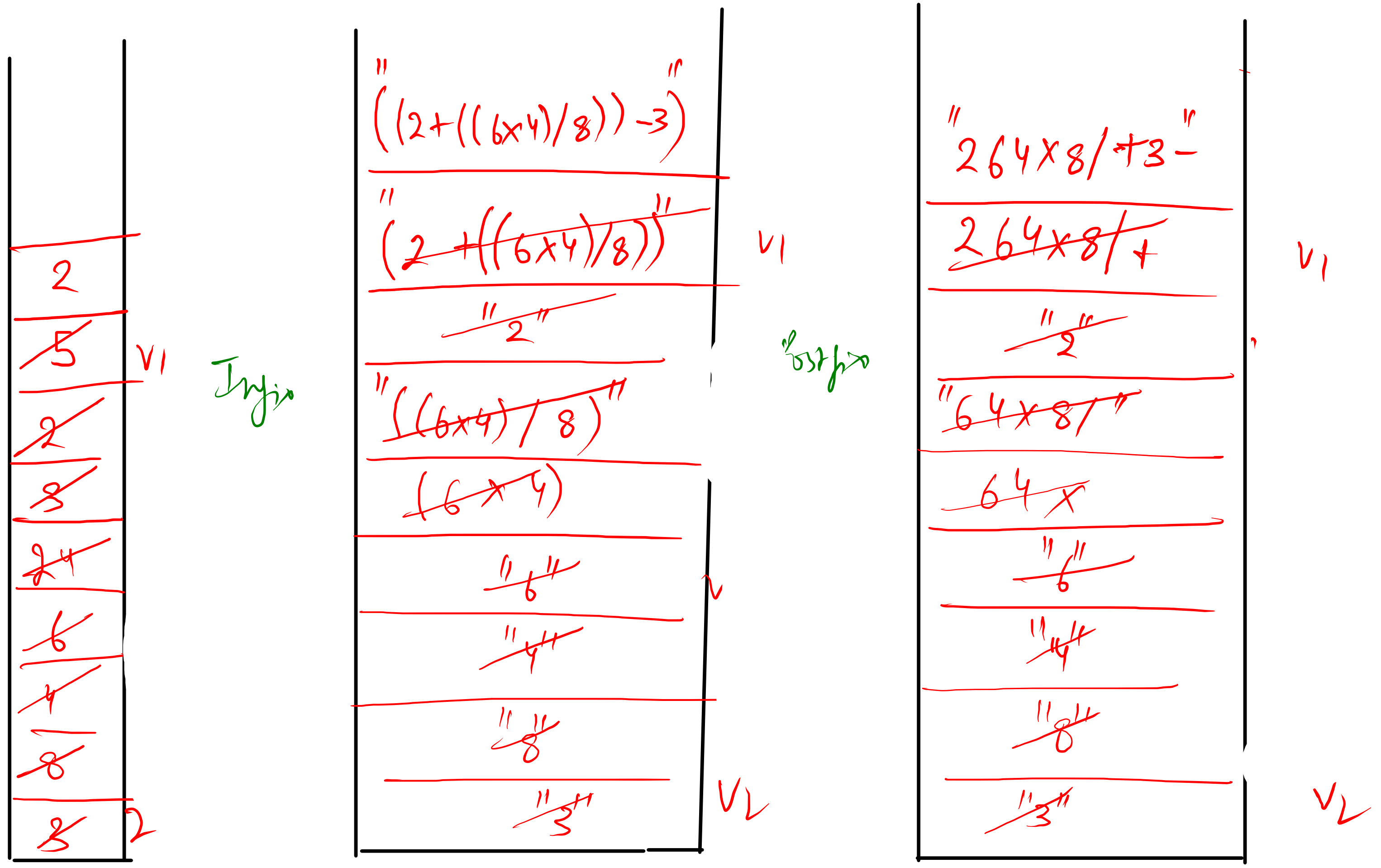




Eval =

- + 2 / × 6 4 8 3

op= -



1. Expression is balanced
2. The only operators used are +, -, *, /
3. Opening and closing brackets - () - are used to impact precedence of operations
4. + and - have equal precedence which is less than * and /. * and / also have equal precedence.
5. In two operators of equal precedence give preference to the one on left.
6. All operands are single digit numbers.

(✓ { ✗ [✗

Precedence

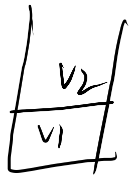
↳ Priority of operators

OP	Precedence	
+	x	1
−	x	1
×	y	2
/	y	2

$x < y$

$((a+b)-c)+d$

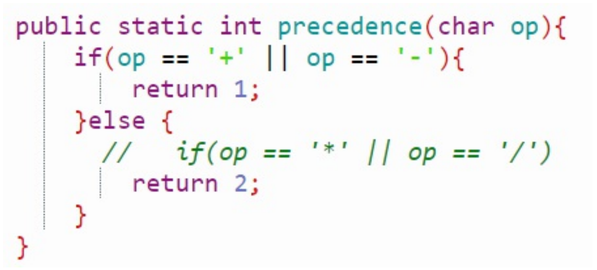
$\overline{V_1 \; op \; V_2}$



$\overrightarrow{(2 + 6 \times 4 / 8 - 3)}$

ch = '-'

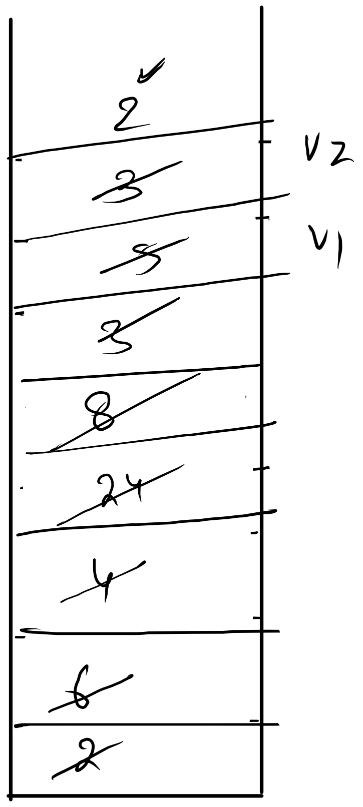
$5 - 3 \rightarrow \boxed{2}$



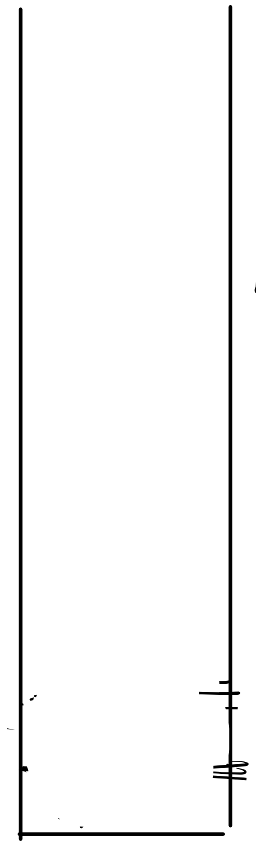
```
public static int precedence(char op){
    if(op == '+' || op == '-'){
        return 1;
    }else {
    //    if(op == '*' || op == '/')
        return 2;
    }
}
```

while(operator.size()>0 &&
    precedence(ch) <= precedence(operator.peek())

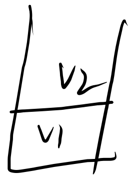
operand :



operator :



$\overline{V_1 \ op \ V_2}$

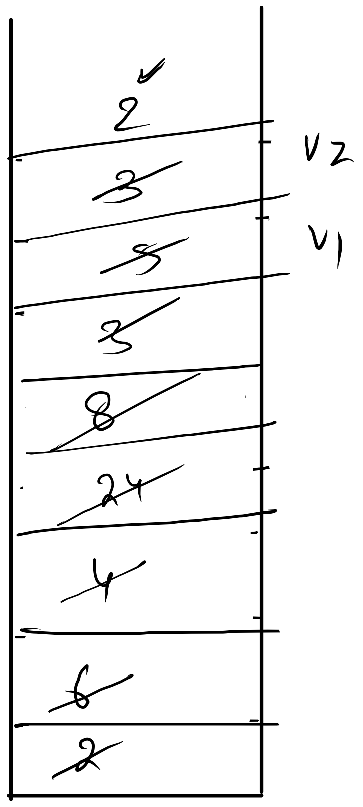


$2 + 6 \times 4 / 8 - 3 \longrightarrow$

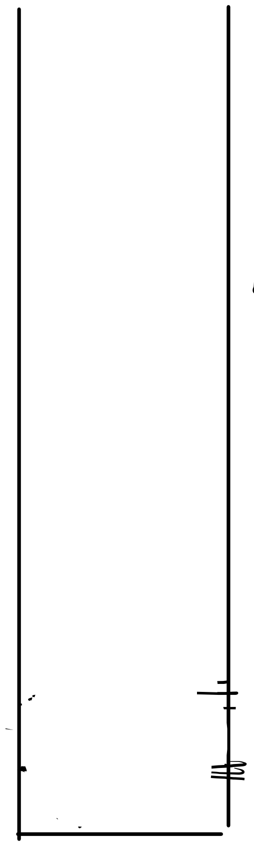
ch = '-'

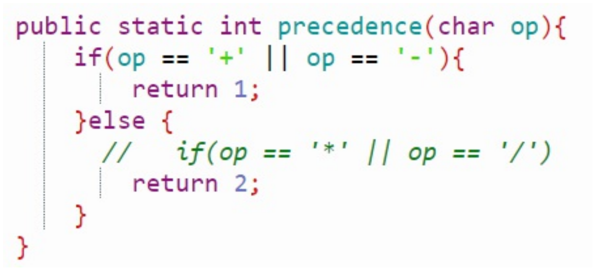
$5 - 3 \rightarrow \textcircled{2}$

operand :



operator :

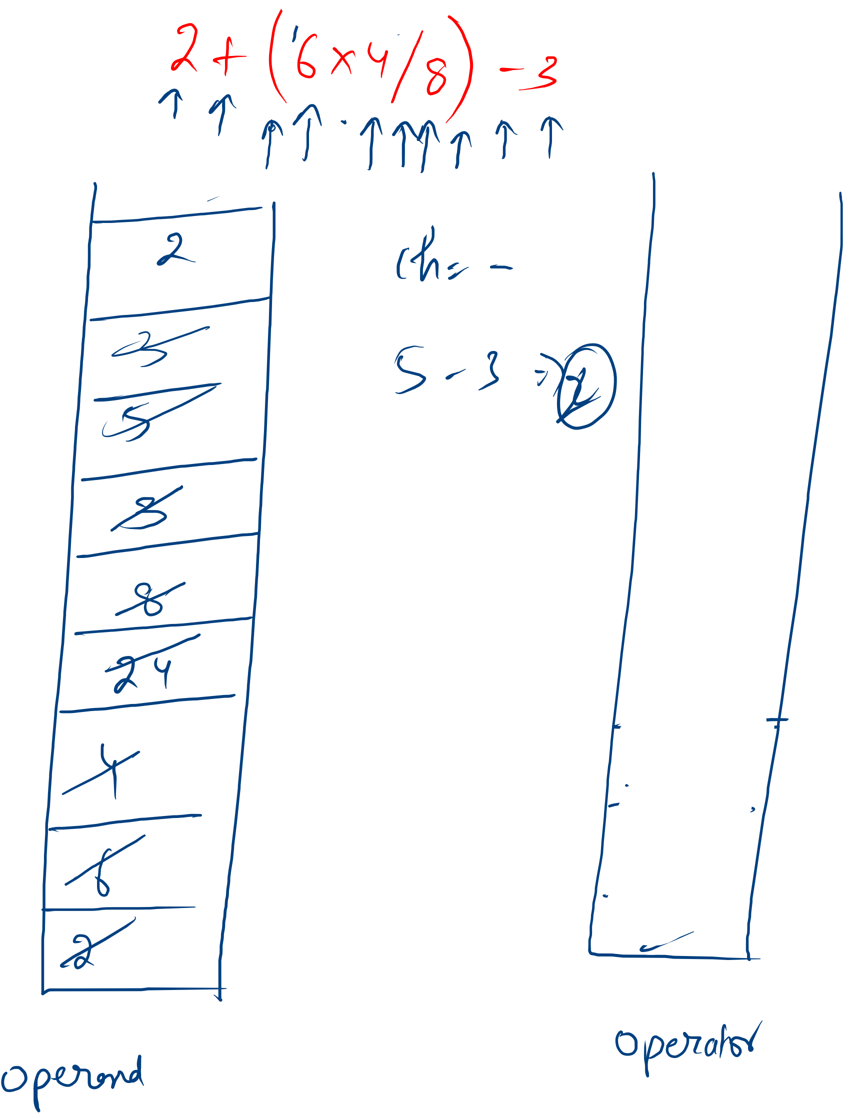


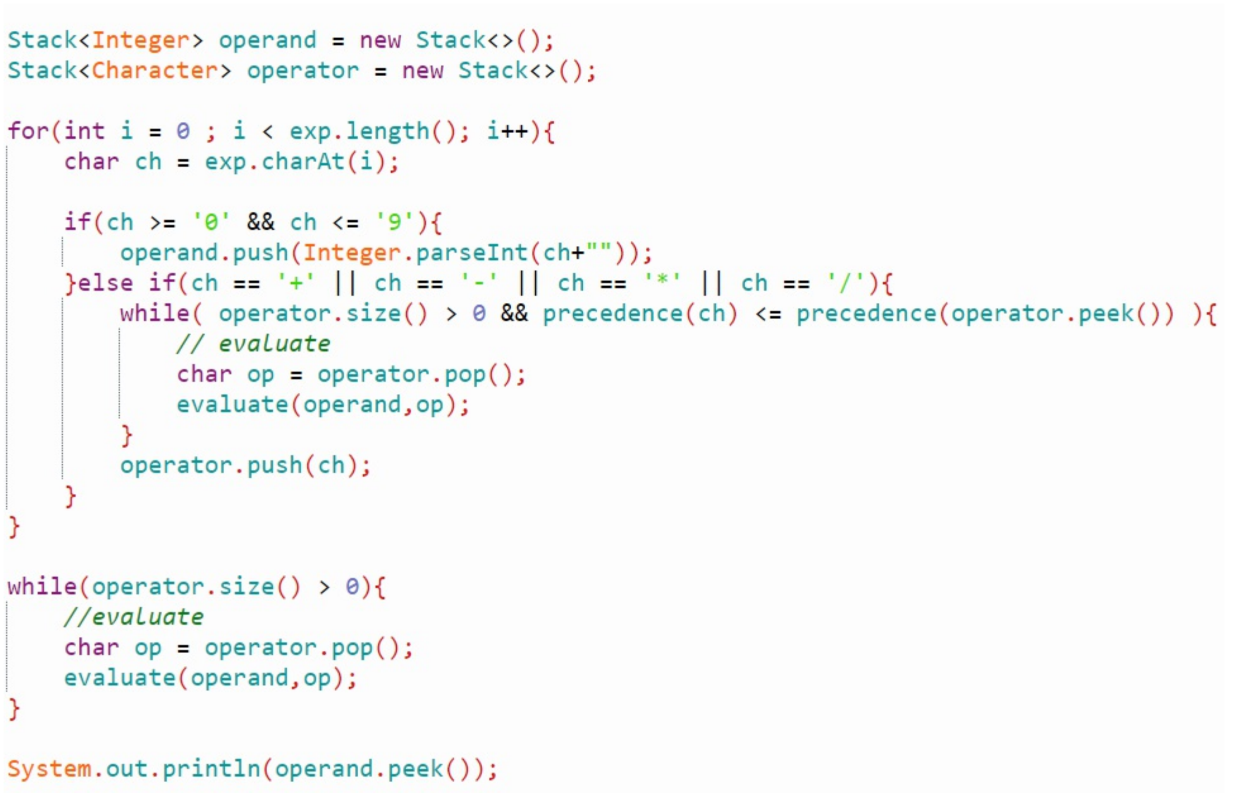


```
public static int precedence(char op){
    if(op == '+' || op == '-'){
        return 1;
    }else {
      //   if(op == '*' || op == '/')
        return 2;
    }
}
```

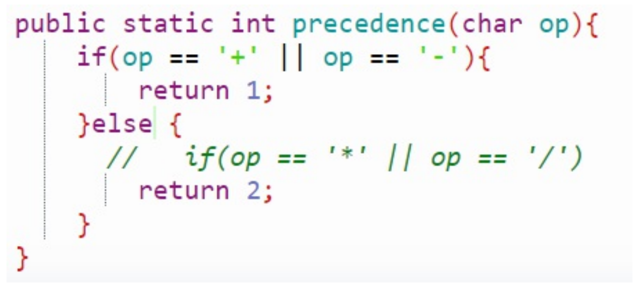
while(operator.size()>0 &&

precedence(ch) <= precedence(operator.peek())





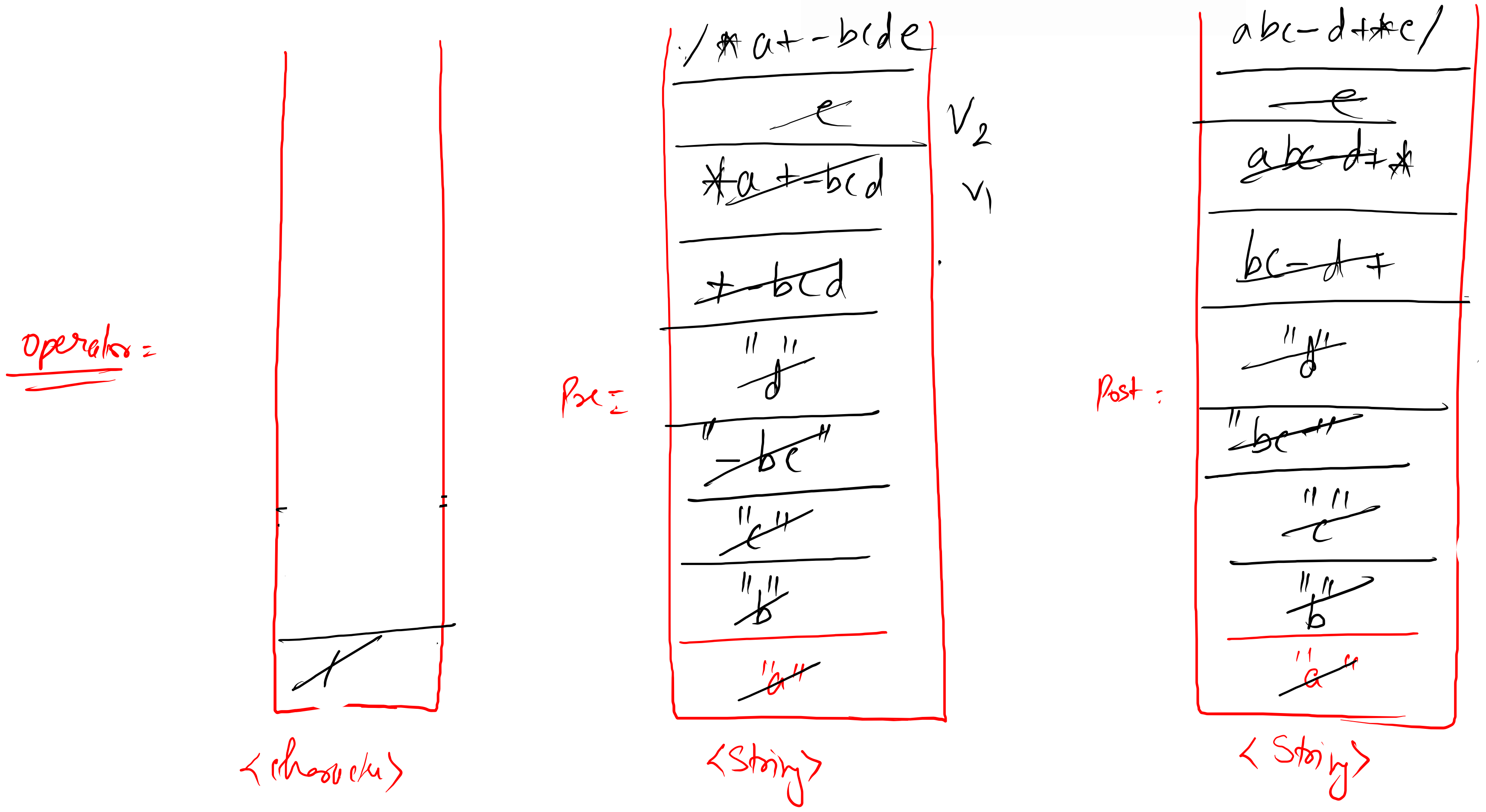
```
Stack<Integer> operand = new Stack<>();
Stack<Character> operator = new Stack<>();

for(int i = 0 ; i < exp.length(); i++){
    char ch = exp.charAt(i);

    if(ch >= '0' && ch <= '9'){
        operand.push(Integer.parseInt(ch+""));
    }else if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){
        while( operator.size() > 0 && precedence(ch) <= precedence(operator.peek()) ){
            // evaluate
            char op = operator.pop();
            evaluate(operand,op);
        }
        operator.push(ch);
    }
}

while(operator.size() > 0){
    //evaluate
    char op = operator.pop();
    evaluate(operand,op);
}

System.out.println(operand.peek());
```



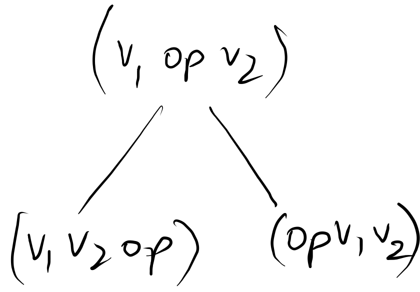
```
public static int precedence(char op){
    if(op == '+' || op == '-'){
        return 1;
    }else {
    //   if(op == '*' || op == '/')
        return 2;
    }
}
```

```
Stack<Integer> operand = new Stack<>();
Stack<Character> operator = new Stack<>();

for(int i = 0 ; i < exp.length(); i++){
    char ch = exp.charAt(i);

    if(ch >= '0' && ch <= '9'){
        operand.push(Integer.parseInt(ch+""));
    }else if(ch == '+' || ch == '-' || ch == '*' || ch == '/'){
        while( operator.size() > 0 && operator.peek() != '(' && precedence(ch) <= precedence(operator.peek()) ){
            // evaluate
            char op = operator.pop();
            evaluate(operand,op);
        }
        operator.push(ch);
    }else if(ch == '('){
        operator.push(ch);
    }else if(ch == ')'){
        while(operator.peek() != '('){
            // evaluate
            char op = operator.pop();
            evaluate(operand,op);
        }
        operator.pop(); // opening bracket
    }
}

while(operator.size() > 0){
    //evaluate
    char op = operator.pop();
    evaluate(operand,op);
}

System.out.println(operand.peek());
```

a*(b-c+d)/e



Operator =
<character>
Pre =
/*a+-bcde
~~e~~
~~*a+-bcd~~
~~+-bcd~~
~~"d"~~
~~"-bc"~~
~~"c"~~
~~"b"~~
~~"a"~~
v_2
v_1
<String>
Post :
abc-d+*e/
~~e~~
~~abc-d+*~~
~~bc-d+~~
~~"d"~~
~~"bc"~~
~~"c"~~
~~"b"~~
~~"a"~~
v_2
v_1
<String>



$(v_1\ op\ v_2)$
$(v_1 v_2\ op)$
$(op\ v_1 v_2)$

data ⇒

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline ⇒ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 1 & 1 & 1 & 0 \end{array}$$

3 → H.W.

data[i][j] = 0 [pi × knows pj]

= 1 [pi knows pj]

find Celebrity ?

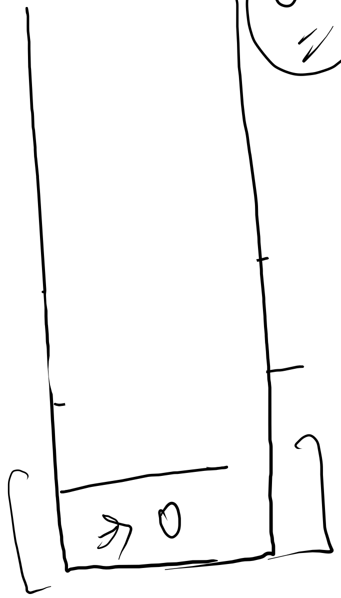
↳ known by everybody + knows nobody

O(n)

Celebrity ⇒ 0

⇒ 2n

O(n)



O(n)

P1	P2
1	0

P1 knows P2 ?

$$\begin{tikzcd} & \text{data}[P_1][P_2] \arrow[dl, "0"'] \arrow[dr, "1"] & \\ \text{P.Celeb: } P_1 & & \text{P.Celeb: } P_2 \end{tikzcd}$$