Expression Evaluation {Stack & queue}

Level (1)

→ Infix Evaluatn + Conversion

→ Postfix Evaluatn + Conversion

→ Prefix Evaluatn + Conversion (HW)

Level (2)

→ Basic Calculator - I, II, III

→ Expression Tree - Constructn & Evaluatn

Infix Evaluation

Infix: → operand1 operator operand2

-1
2
-3
8
5
-2 -1 0

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

operand: 5, 2, 3, 4, 2, 5

operator: +, *, (, -, +, /, *, +

Precedence

(0) Brackets

(1) Division & Multiplication (/, *)

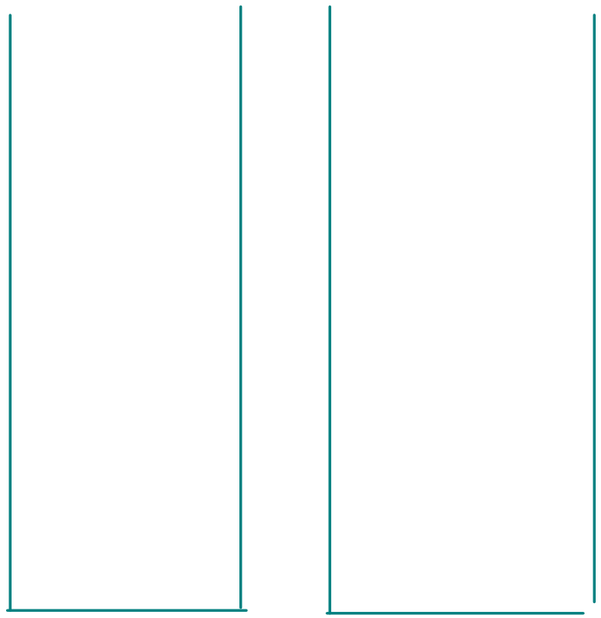
(2) Addition & Subtraction (+, -)

Equal Precedence

"Left to right associativity"

(+, -, /, *)

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



```
if (ch >= '0' && ch <= '9')
    operand.push(ch)
else if(ch == '(')
    operator.push(ch)
else if (ch == ')')
{   pop until '(' on top, perform, res push operand
    pop '(' also
}
else {
while ( stk.size>0 && stk.peek() != '('
        && prec(stk.peek) >= prec(ch))
    perf & push
push yourself;
}
```

```java
public static int precedence(char ch){
    if(ch == '/' || ch == '*') return 2;
    if(ch == '+' || ch == '-') return 1;
    return 0;
}
public static int performOp(int a, char op, int b){
    if(op == '+'){
        return a + b;
    } else if(op == '-'){
        return a - b;
    } else if(op == '*'){
        return a * b;
    } else {
        return a / b;
    }
}
```

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

    if(ch >= '0' && ch <= '9'){
        // operand -> push in operand wali stack
        operand.push(ch - '0');

    } else if(ch == '('){
        operator.push(ch);
```

```java
} else if(ch == ')'){

    while(operator.peek() != '('){
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(performOp(a, op, b));
    }

    operator.pop(); // pop opening braces as well
```

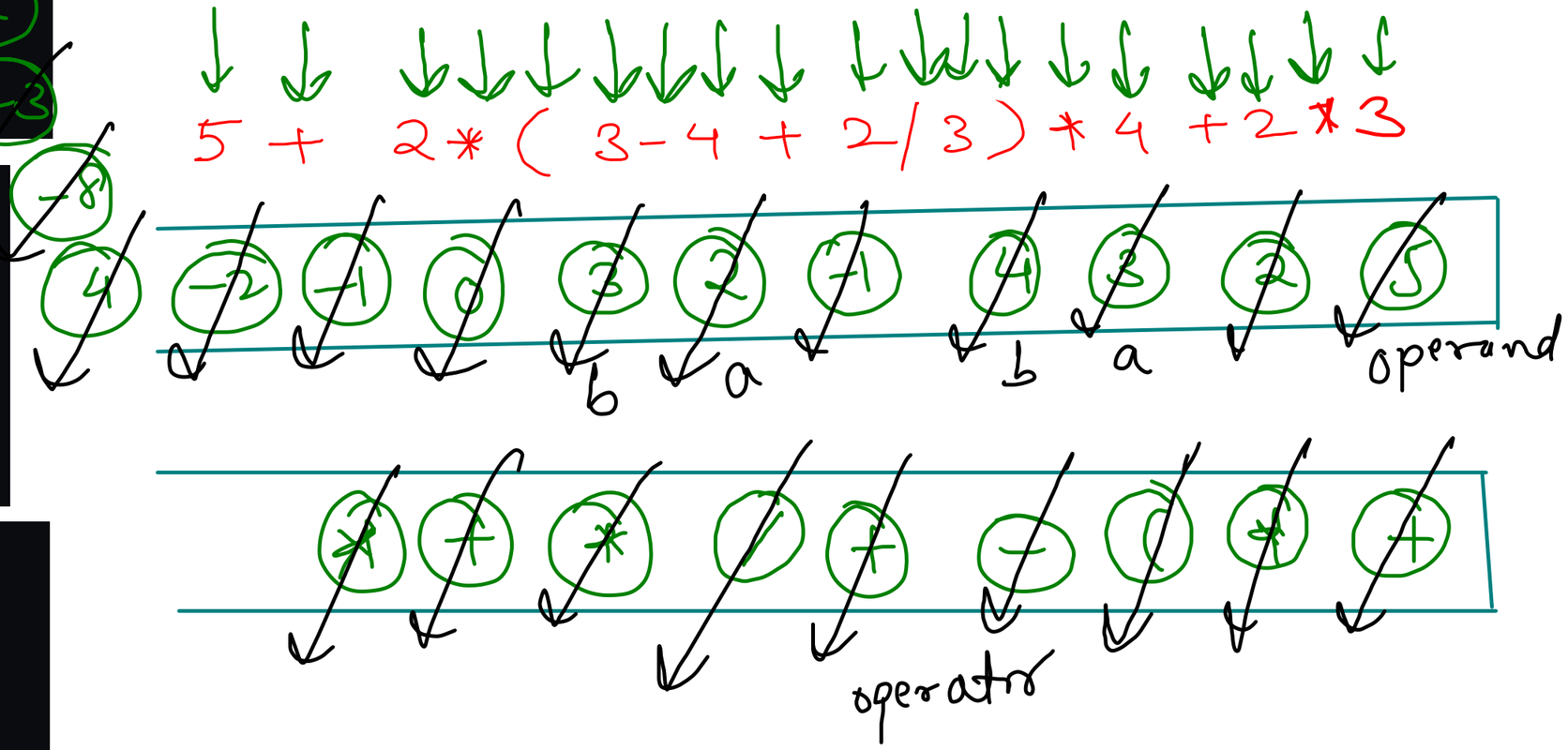
```java
} else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
    // operator -> + or - or * or /
    while(operator.size() > 0 && operator.peek() != '('
        && precedence(operator.peek()) >= precedence(ch)){
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(performOp(a, op, b));
    }

    operator.push(ch);
}
```

① Precedence

② Associativity

O(N)



2^{3^4}

$(2^3)^4 = (8)^4 = 4096$

$2^{(3^4)} = 2^{81} =$ ✓ _ _ _ _ _ _

2 ^ 3 ^ 4

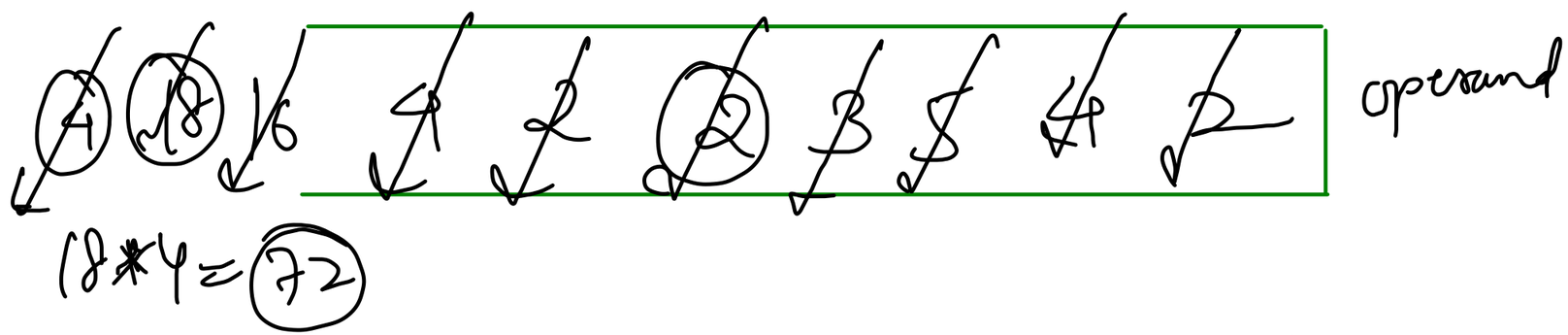
81

3

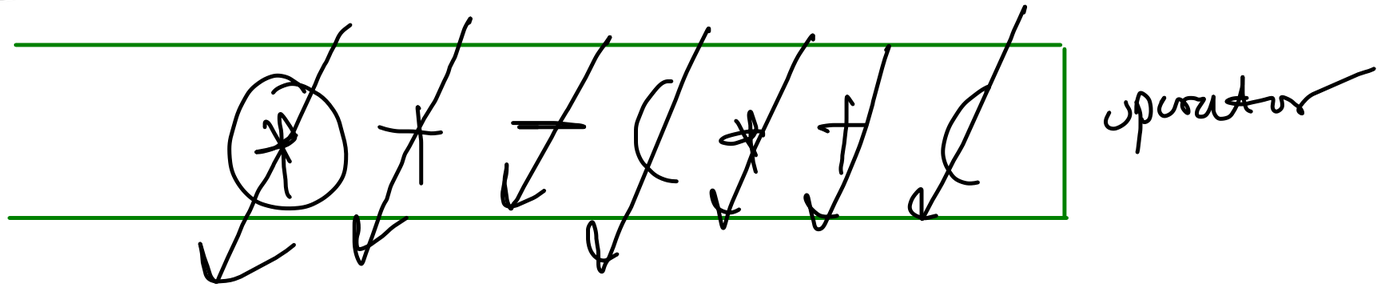
2

operand

operator



18*4=72



Infix Conversion

a operator b

Prefix → operator a b

Postfix → a b operator

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

operator <Character>

prefix <String>

Postfix <String>

* * 2 + -34 / 234

4 | +2+-34/23 | +-34/23

4 | 234-23/+* | 34-23/+

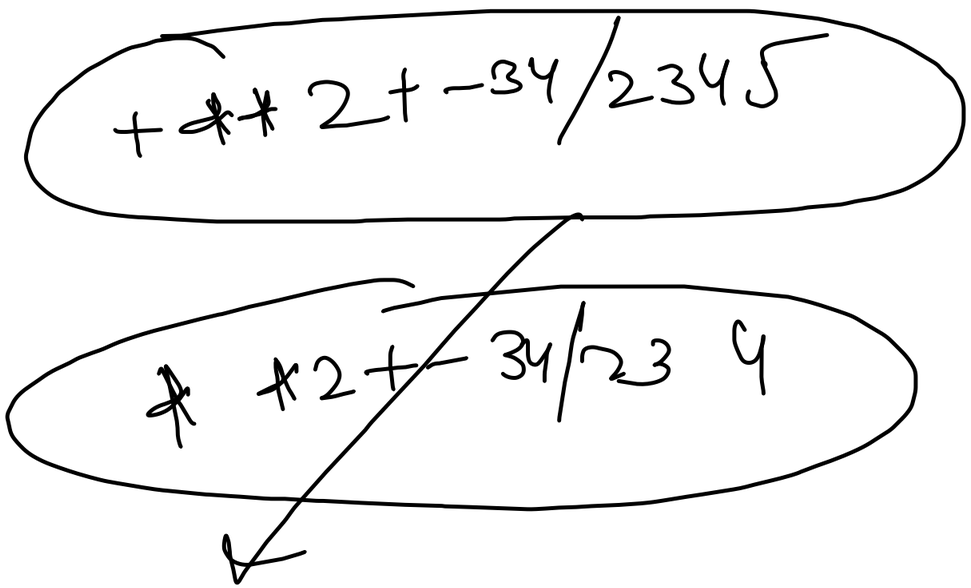
234 - 23/+*4*

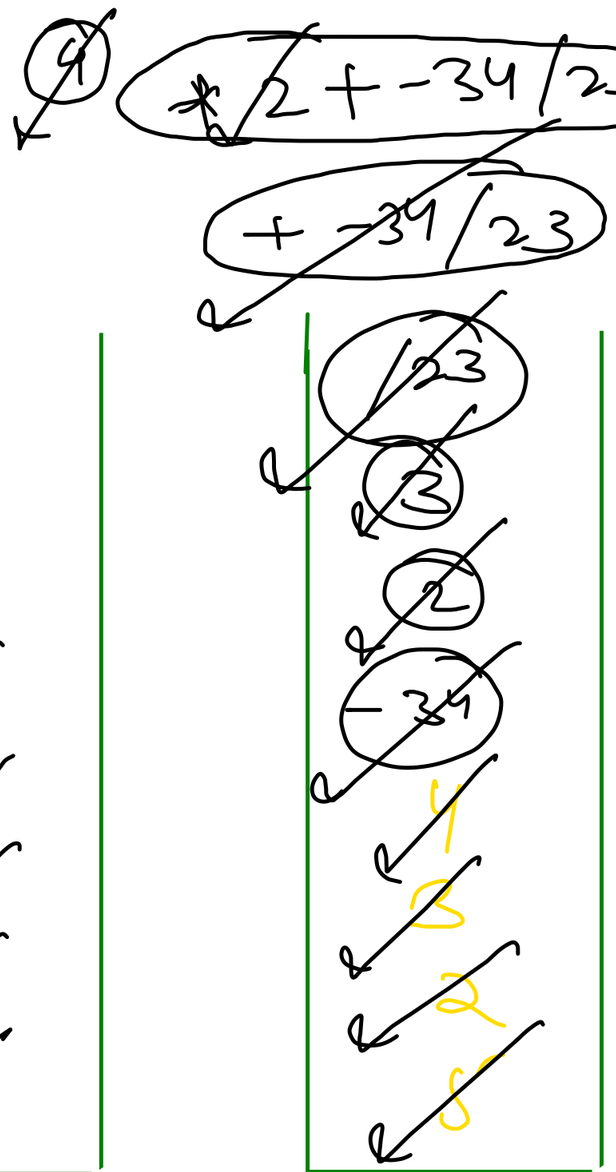
Prefix ⇒ + ∫ * * 2 + - 34 / 234

Postfix ⇒ ∫ 234 - 23 / + * 4 * +

Postfix Evaluation & Conversion

5 2 3 4 - 2 3 / + * 4 * +





Postfix 2 3 4 − 2 3 / + *

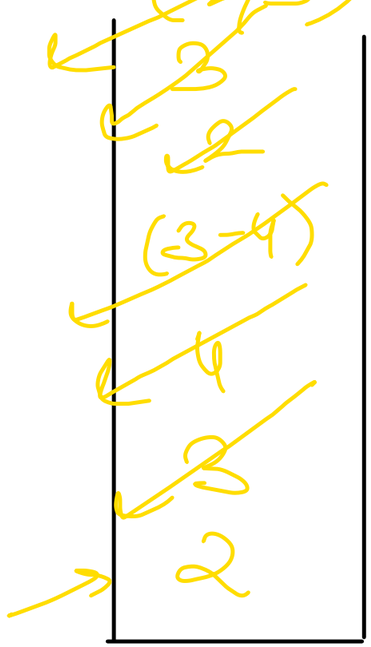
(2*((3-4)+(2/3)))

(3-4)+(2/3)

*2+-34/23

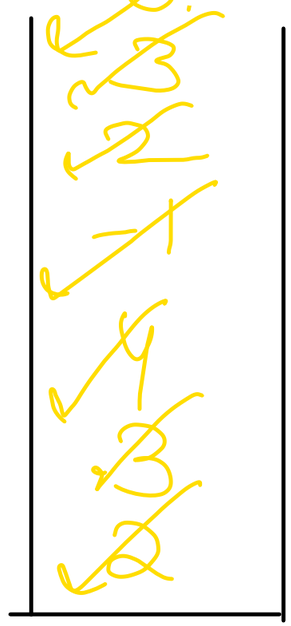
+-34/23

-2



Infix

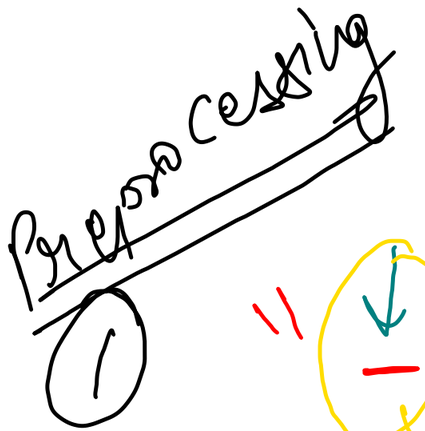
Postfix

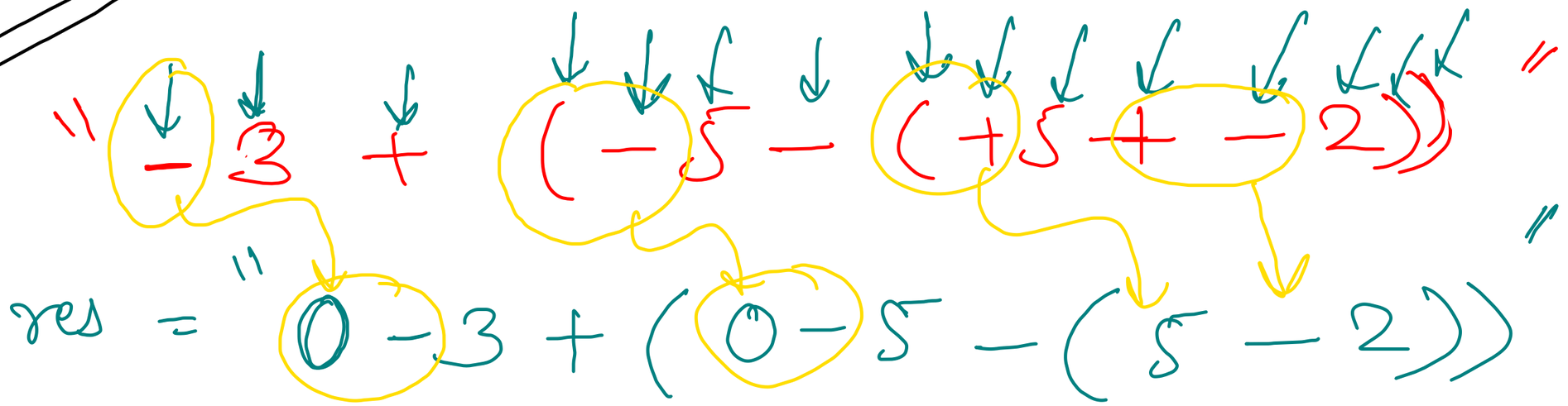


Evaluate

Basic Calculator -I, II, III

Preprocessing





>1 digit

(2)

235 + 478

curr = 2*10 + 3 = 23*10 + 5 = 235

curr = 4*10 + 7 = 47*10 + 8 = 478