Agenda

i) stack basics

ii) Double character trouble

iii) Expression evaluation **

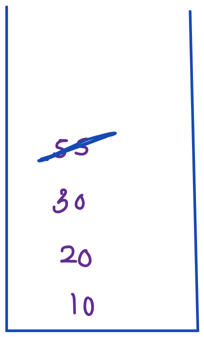
Stack : It follows LIFO

↳ Last In first out

```
Stack < Integer > st = new Stack <> ();
```

↓ name of variable

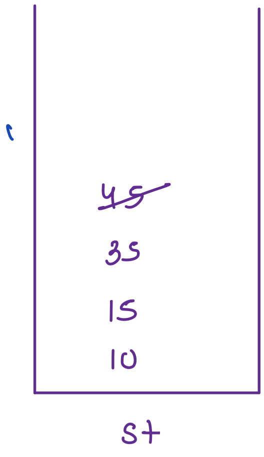
```
st.push(10);
st.push (20);
st.push (30);
st.push (55);
sopln (st.peek ( )); → 55
st.pop( );
Sopln (st.size ( )); → 3
```



all the above 4 functions are TC: O(1)

st. peek( )	st. pop( )
It gives us the topmost element of the stack.	It removes the topmost element of stack and also return us that removed value.

```
Stack < Integer > st = new Stack < > ( );
st. push (10);
st- push (15);
st. push (35);
st. push (45);
sopln (st. size()); → 4
sopln (st. peek()); → 45
sopln (st. size()); → 4
sopln (st. pop()); → 45
sopln (st. size()); → 3
```



Real life examples of stack

i) undo / redo

ii) ← (back) browsing

iii) Expression evaluate

Q. Double character trouble

⇒ Given a string, keep removing the same consecutive char until no more occurrence of same consecutive char remains. Return the final answer string.

Note: same consecutive chars are coming in doubled manner.

Expected TC: O(n)

str = ab~~cc~~bc

↓

a~~bb~~c

↓

ac

str = kmb~~aa~~bmgj

↓

km~~bb~~mgj

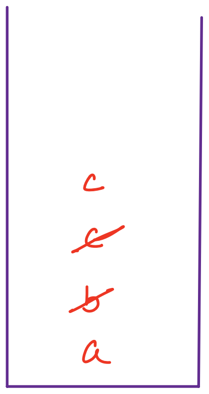
↓

k~~mm~~gj

↓

kgj

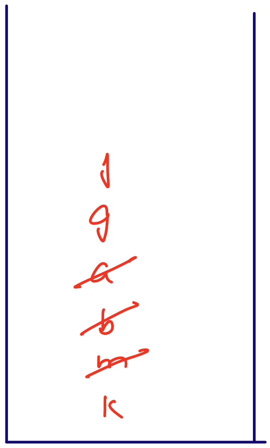
a b c c b c



str = c a

ans = ac

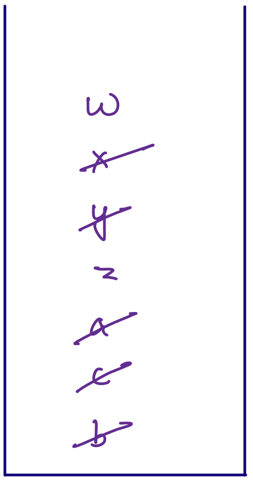
k ~~m~~ ~~b~~ ~~a a~~ ~~b~~ ~~m~~ g j



str = jgk

ans = kgj

b c a a c b z y x x y w



str = wz

ans = zw

```
for ( ch of str) {
    if( st.size() ==0 || st.peek() != ch ) {
        st.push (ch);

    }
    else if ( st.peek() == ch) {
        st.pop();

    }
}
```

// travel stack and calculate final answer.

Expression Evaluation and conversion

Infix expression

2 + 3 — operands

+ ↓ operator

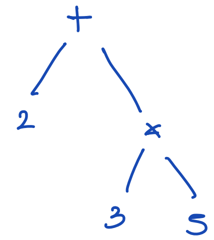
Postfix expression

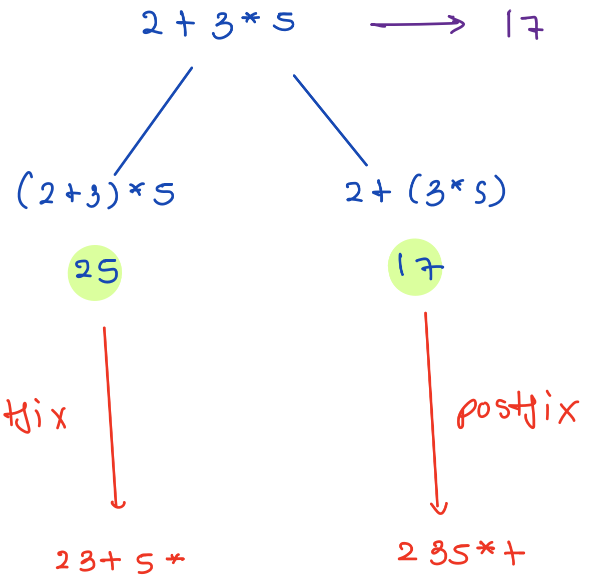
2 3 + ⟶ operator

2 3 — operands

Infix exp : 2 + 3 * 5 => 17

Postfix exp : 2 3 5 * +




2 + 3 * 5 ⟶ 17

(2+3) * 5 — 25 — postfix ↓ 2 3 + 5 *

2 + (3 * 5) — 17 — postfix ↓ 2 3 5 * +

In postfix exp the order of operators is exactly same as the order in which the operators should get evaluated.

Advantages of postfix :

→ no brackets

→ no operator priority

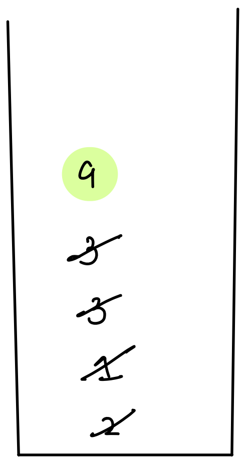
Postfix evaluation

Given postfix expression, evaluate it and return final answer.

opr : *, /, -, +

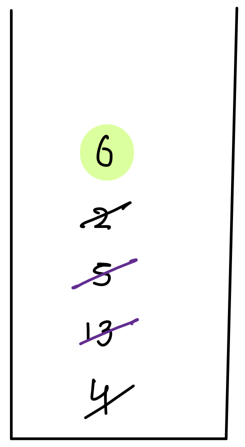
↓

eg1 ["2", "1", "+", "3", "*"] => 9

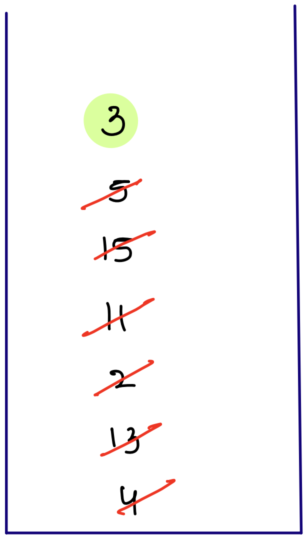


↓

eg2 ["4", "13", "5", "/", "+"] => 6



["4", "13", "2", "-", "+", "5", "/"]



```
if (str is operator) {
      //evaluate
      int v2 = st.pop();
      int v1 = st.pop();
      calculate res of v1 str v2
      st.push(res);
}
else {
        int val = str to int
        st.push(val);
  }
```

Infix to postfix conversion

^ → power

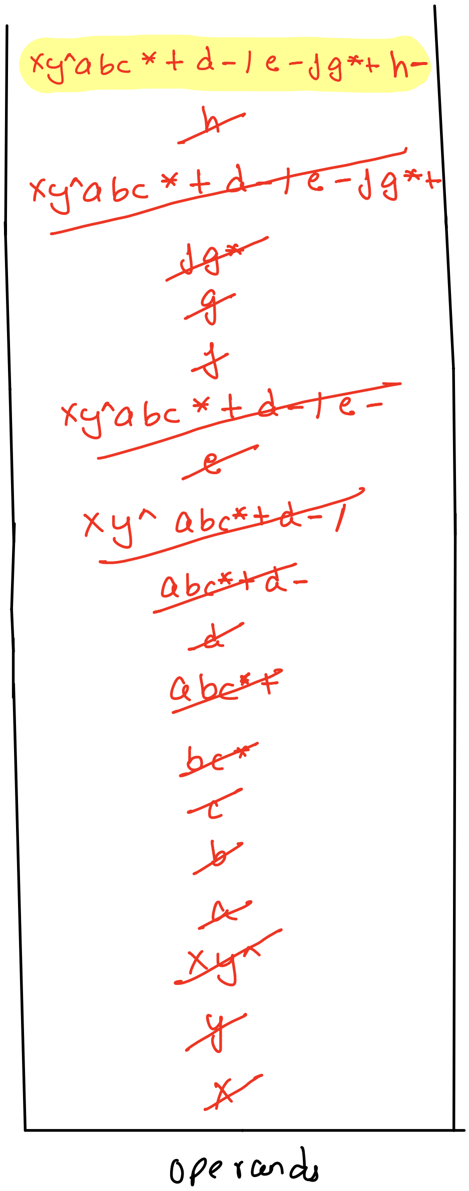
lowercase letters → operands

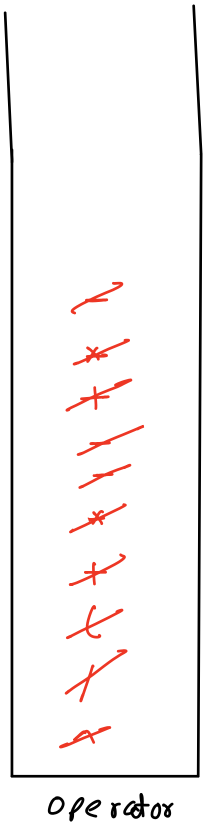
operators → ^, +, -, *, /

brackets

eg1 : "x^y/(a+b*c-d)-e+f*g-h

priority
=
^
/, *
+, -





if ch is operand then push it to operand stack

if ch is an opening bracket push it to operator stack

if ch is closing bracket

- ↳ evaluate till an opening brackets comes on operator.peek()

if ch is an operator

- ↳ evaluate till higher or equal priority operators are coming on operator.peek() but
  - → stop if stack becomes empty
  - → stop if operator.peek() becomes '('
- add ch to operator stack

final evaluation