Infix: 1 + 1

Postfix: 1 1 +

$$2 * \underbrace{(3+4)}_{7}$$

$$2 * 7 \Rightarrow 2\ 7\ *$$

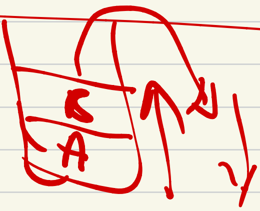
$$\downarrow$$

$$3\ 4\ +$$

$$\Rightarrow 2\ (3\ 4\ +)$$

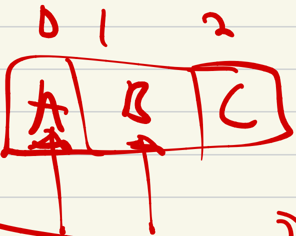
$\boxed{(A + B / C * (A + C) - F)}$

→ Stack →



→ List / Array

↳→



Operators: → Stack

↳→ order → F

Operands: → List

+ / * → Stack

2 + (5 * 2)

$\boxed{(A + B / C * (D + C) - F)}$

Stack (operator)

(
(+
(+/
(+*
(+*(
(+*(+
(+*(+) ⟶ (+*~~(~~
(+*
(+*$\boxed{-}$
① ② ①
(+-
① ①
(-
(-)
⌊ ⌋
~~( )~~
$\boxed{+}$

return

^ ⟶ highest (3)
*, / ⟶ mid (2)
+- ⟶ low (1)

List Postfix / Result / Number

A
A B
ABC
ABC/
ABC/D
ABC/DC
ABC/DC+
* +
ABC/DC+*+F
$\boxed{ABC/DC+*+F-}$

Rules I

① Incoming element w/ higher prec should only be added to the stack
↳ ① = ① ⟹ preserve only current incoming elem.
↳ ② = ① ↳ pop / remove existing

② if ")", get rid of everything b4 until you reach "("

③ Add whatever elements lefts in stack to list as part of the final result.

```
function infixToPostfix (inp) {
        Stack   stack;  ⊔
        ListResult  list;  [ ]
        Prec    pre = { "+" : 1,
                        "-" : 1,
                        "*" : 2,
                        "/" : 2,
                        "^" : 3
                      }
        for (el in inp) {
          if (el is Number)
              list.add(el)
          if (el == "(")
              stack.add(el)
          else if (el == ")") {
              pel = stack.peek()
              while (pel != "(")      !stack.isEmpty() &&
                  popel = stack.pop()
                  list.add(popel)
              if (pel == ")") (optional)
                  stack.pop()

          if (el is in the prec keys) {      pr.haskey(el)
              if (prec[el] > prec[stack.peek()]
                  stack.add(el)
if (prec[el] <= prec[stack.peek()]) [optional]
    while (pre[el] > pre[temp])
        list.add(stack.pop())
!stack.isEmpty() &&
}
```