PROBLEM SOLVING

Functional style

FOLDL

Sometimes you want to fold a list up from the left

```
foldr (op) z [ ] = z
foldr (op) z (x : xs) = x op (foldr (op) z xs)
```

```
foldl (op) z [ ] = z
foldl (op) z (x : xs) = foldl (op) (z op x) xs
```

REVERSE POLISH NOTATION

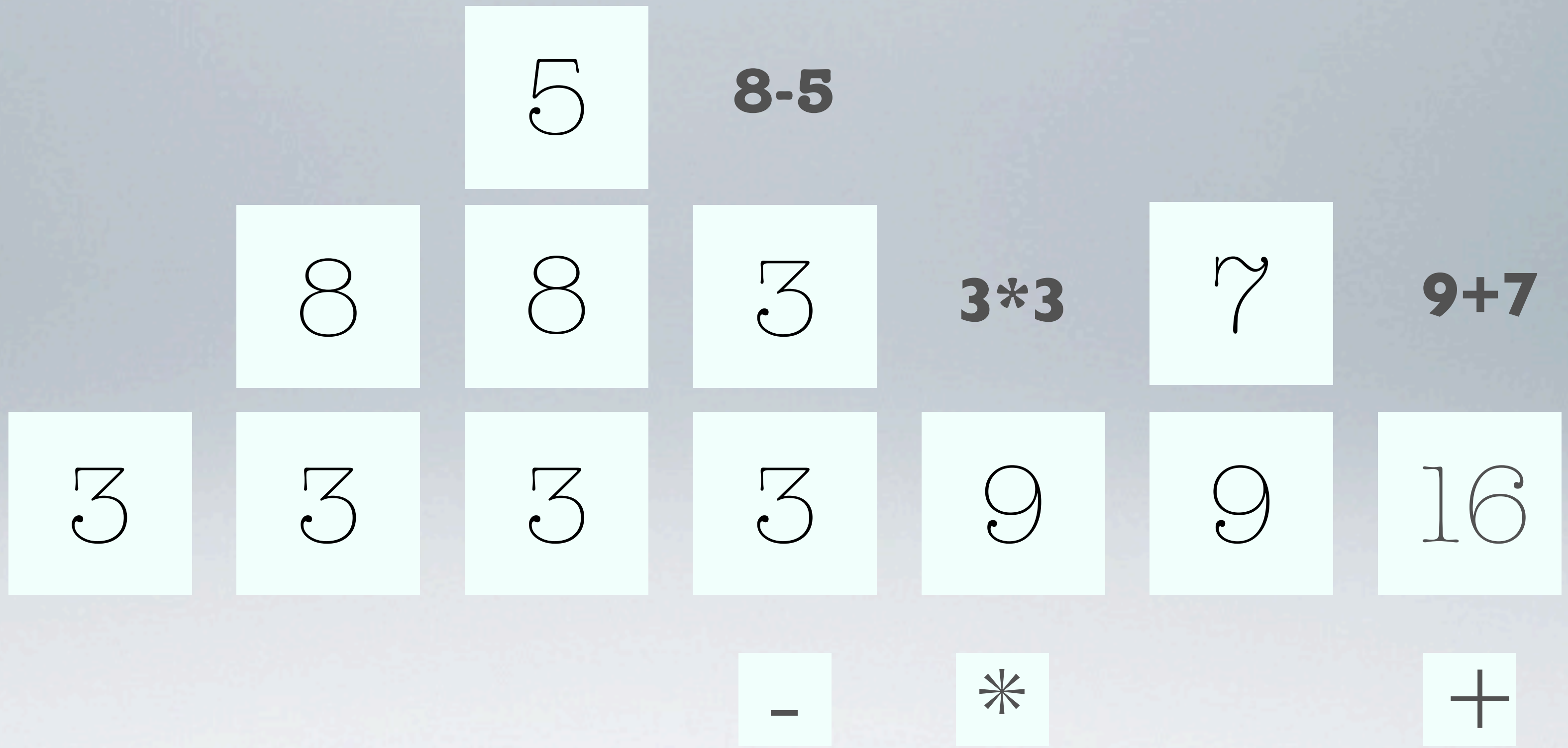
3 * (8 - 5) + 7

↓

3 8 5 - * 7 +

Huh?

R.P.N. AS A STACK IMPLEMENTATION



5 8-5

8 8 3 3*3 7 9+7

3 3 3 3 9 9 16

- * +

3 8 5 - * 7 +

1 evalRPN :: (Num a) => String -> a

Type

2 “3 8 5 - * 7 +”

words :: String -> [String]

[“3”,“8”,“5”,“-”,“*”,“7”,“+”]

3 Stack?

list

```
import Data.List

evalRPN :: (Num a, Read a) => String -> a
evalRPN = head . foldl procStack [ ] . words
```

from Data.List

procStack?

procStack does to a list of numbers and a string essentially what we did in the stack implementation

```
procStack :: (Num a, Read a) => [a] -> String -> [a]

procStack (x : y : ys) “ * ” = (y * x) : ys
procStack (x : y : ys) “+” = (y+x) : ys
procStack (x : y : ys) “-” = (y-x) : ys
procStack xs numString = read numString : xs
```

```
evalRPN “3 8 5 - *  7 +”
=  head . foldl procStack [ ] . words “3 8 5 - *  7 +”
= head . foldl procStack [ ] [“3”, “8”, “5”, “-”, “*”, “7”, “+”]
= head . foldl procStack (procStack [ ] “3”) [“8”, “5”, “-”, “*”, “7”, “+”]
= head . foldl procStack (procStack [3] “8”) [“5”, “-”, “*”, “7”, “+”]
= head . foldl procStack (procStack [8,3] “5”) [“-”, “*”, “7”, “+”]
= head . foldl procStack (procStack [5,8,3] “-”) [“*”, “7”, “+”]
= head . foldl procStack (procStack [8-5,3] “*”) [“7”, “+”]
= head . foldl procStack (procStack [3 *(8-5)] “7”) [“+”]
= head . foldl procStack (procStack [7, 3 *(8-5)] “+”) [ ]
= head  (procStack [7, 3*(8-5)] “+”)
= head  [3 *(8-5) + 7]
= 3*(8-5)+7
= 16
```

MATRICES

Represent a matrix as a list of lists

$$\begin{pmatrix} 1 & 4 & 9 \\ 3 & 5 & 7 \end{pmatrix} \longrightarrow [[1, 4, 9], [3, 5, 7]]$$

```
type Matrix = [[Int]]
```

Is a list of lists of `Int` a matrix?

Yes, if

1. every list in the list has the same length
2. there is at least one row and one column

1 map `length` over list; check every number is the same

2 the list is non-empty

```
-- every element of a list satisfies a predicate

all :: (a -> Bool) -> [a] -> Bool
all p xs = foldr (&&) True (map p xs)

-- version using function composition
-- all p = foldr (&&) True . map p
```

all is a library function

```
-- every element of a list of Int is the same
uniform :: [Int] -> Bool
uniform [ ] = True
uniform xs = all (== head xs) (tail xs)

-- check the two properties
valid :: Matrix -> Bool
valid [ ] = False
valid (x : xs) = not (null x) && uniform (map length (x : xs))
```

vacuously true

MATRIX ADDITION

$$\begin{pmatrix} 1 & 4 & 9 \\ 3 & 5 & 7 \end{pmatrix} + \begin{pmatrix} 2 & 5 & 0 \\ 3 & 1 & 7 \end{pmatrix} = \begin{pmatrix} 3 & 9 & 9 \\ 6 & 6 & 14 \end{pmatrix}$$

have to be the same width and same height

ZIPWITH

Library function

```
-- our version
zipWith’ :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith’ f xs ys = [f x y | (x, y) <- zip xs ys]
```

calculate matrix width

```
matrixWidth :: Matrix -> Int
matrixWidth xss = length (head xss)
```

calculate matrix height

```
matrixHeight :: Matrix -> Int
matrixHeight xss = length xss
```

ADD TWO MATRICES

```
plusM :: Matrix -> Matrix -> Matrix
plusM m n | ok = zipWith (zipWith (+)) m n
    where ok = valid m && valid n
               && matrixWidth m == matrixWidth n
               && matrixHeight m == matrixHeight n
```