1 Problem 2 - Higher Order Functions

(a) Given the function `foldr` as below

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f v [] = v
foldr f v (x:xs) = f x (foldr f v xs)
```

Using the function `foldr`, define a function

```
lengthsum ::  (Num a, Num b) => [a] -> (b, a)
```

That takes a list of numbers as input, then return the length and the sum of the list as a pair.

Svar.

```
lengthSum :: (Num a, Num b) => [a] -> (b, a)
lengthSum = foldr (\n (x, y) -> (1 + x, n + y)) (0, 0)
```

(b) Given the function `foldl` as below:

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f v [] = v
foldl f v (x:xs) = foldl f (f v x) xs
```

Using the function `foldl` define a function

```
inList ::  (Eq a) => a -> [a] -> Bool
```

that takes a value and a list of the same type as inputs, then checks wheter the value is an element of the list.

Svar.

```
inList :: (Eq a) => a -> [a] -> Bool
inList x = foldl (\acc y -> (x == y) || acc) False
```

(c) What do the following expressions return?

(i) `foldr (:)  "hello" "world!"` = ”world!hello”

(ii)

```
foldl (\xs -> \x -> x:xs) "INF122" "exam" ->
```

2 Problem 3 - An evaluator

Consider the following type declaration:

```
data Expr = V Int | M Expr Expr | D Expr Expr
```

You are asked to implement an evaluator

```
eval ::  Expr -> Maybe Int
```

which evaluate an expression of type Expr defined above