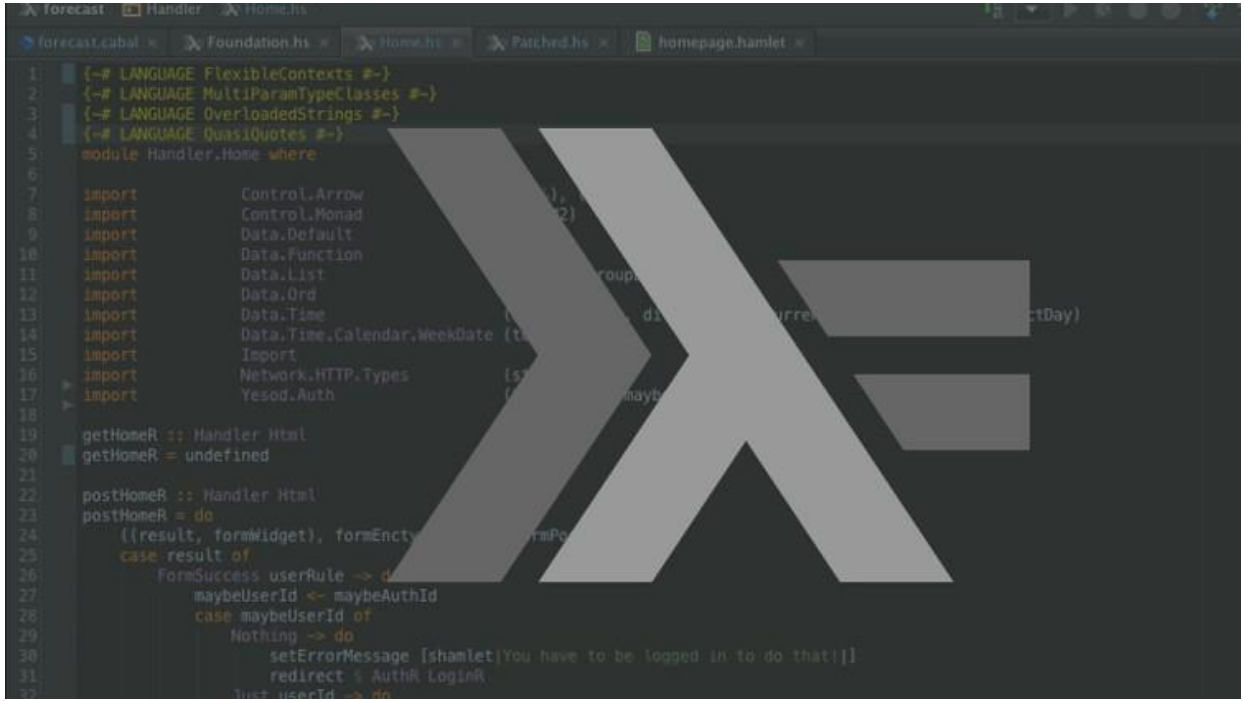
Exam 3 - Problem 2



Problem 2

In Haskell, the standard class *Foldable* is responsible for allowing the use of folds on structured data in order to obtain an aggregate. For example, lists are instances of Foldable, and the sum function is defined in the Foldable class.

To instantiate *Foldable* we only need to define *foldr*, as all other operations have a default definition that uses it.

The goal of this exercise is to make the type of the binary trees an instance of the Foldable class.

Lucas Bazilio

foldr

foldr is a higher-order function used to reduce (or fold) a data structure into a single value by recursively applying a binary function to the elements of the structure, starting from the rightmost element.

Here's an example of how foldr can be used to calculate the sum of a list of integers:

```
sumList :: [Int] -> Int
sumList xs = foldr (+) 0 xs
```

[2,3,6] → 11

Problem 2

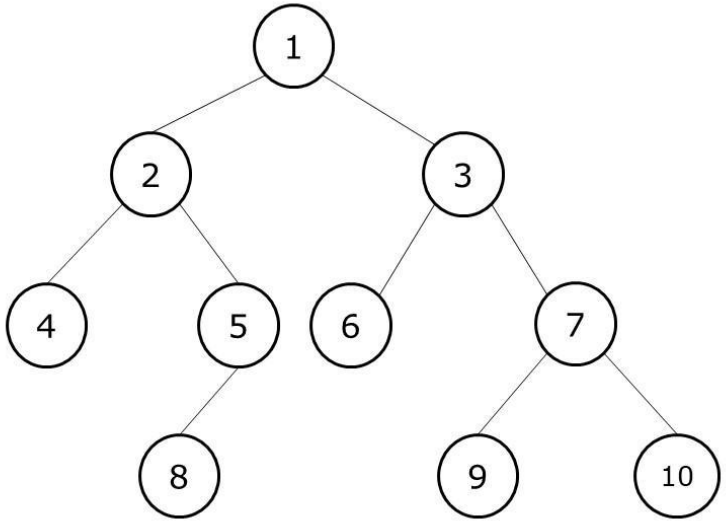
The goal of this exercise is to make the type of the binary trees an instance of the *Foldable* class. Consider this type for binary trees:

```
data Tree a = Empty | Node a (Tree a) (Tree a)
          deriving (Show)
```

It is requested:

1. Make *Tree* an instance of *Foldable*. To do this, implement the foldr function by applying a function to the elements of the tree while following a preorder traversal.
2. Define a function *avg :: Tree Int -> Double* to calculate the average of the elements of a non-empty tree of integers. Use fromIntegral to convert from integer to real.
3. Define a function *cat :: Tree String -> String* to concatenate with spaces all the nodes of a text tree.

Preorder Traversal



Preorder Traversal:

[root, left, right]

1	2	4	5	8	3	6	7	9	10

Problem 2

Public Test Case

Input

```
maximum $ Node 'a' (Node 'c' Empty Empty) (Node 'b' Empty Empty)
avg $ Node 10 (Node 20 Empty Empty) (Node 30 Empty Empty)
cat $ Node "my" (Node "dog" Empty Empty) (Node "likes" (Node "summer" Empty Empty) Empty)
```

Output

```
‘c’
20.0
"my dog likes summer"
```