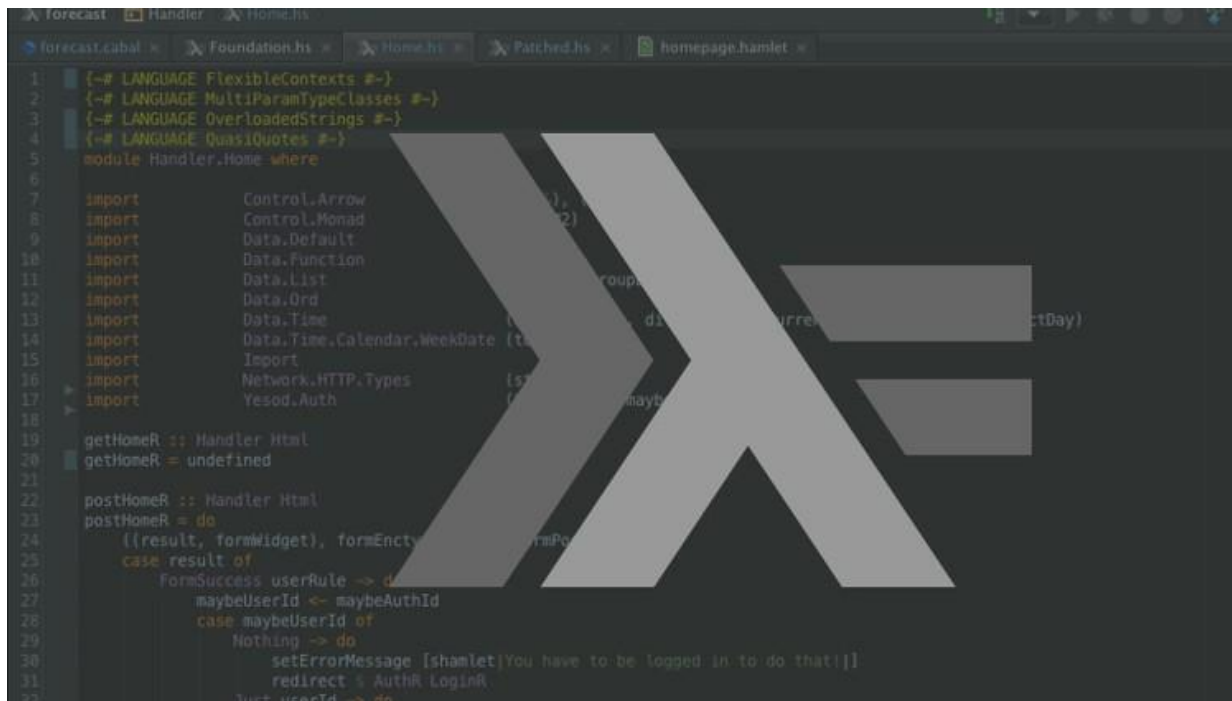


Binary Trees Problems



```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate
15 import Import
16 import Network.HTTP.Types
17 import Yesod.Auth
18
19 getHomeR :: Handler Html
20 getHomeR = undefined
21
22 postHomeR :: Handler Html
23 postHomeR = do
24   ((result, formWidget), formEnctype) <- formPost
25   case result of
26     FormSuccess userRule -> do
27       maybeUserId <- maybeAuthId
28       case maybeUserId of
29         Nothing -> do
30           setErrorMessage [shamlet|You have to be logged in to do that!|]
31           redirect % AuthR.LoginR
32         Just userId -> do
```

Binary Trees Problems



The definition of the trees is given by:

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

That is, a tree with elements of type *a* is, either an empty tree, either a node with an element (of type *a*) and two other trees of the same type. The *deriving (Show)* statement simply enables an visualization of trees.

Problem 7



Write a function *inOrder* :: *Tree a* -> [*a*] that, given a tree, return its in-order traversal.

Public test cases

Input

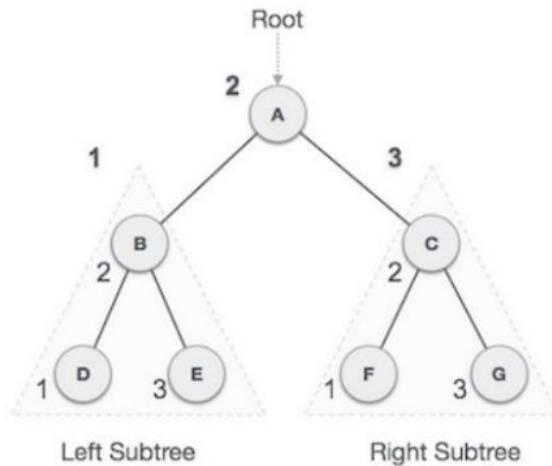
```
let t7 = Node 7 Empty Empty
let t6 = Node 6 Empty Empty
let t5 = Node 5 Empty Empty
let t4 = Node 4 Empty Empty
let t3 = Node 3 t6 t7
let t2 = Node 2 t4 t5
let t1 = Node 1 t2 t3
let t1' = Node 1 t3 t2
```

Output

inOrder t1

[4,2,5,1,6,3,7]

Problem 7



Inorder: $D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

Instructor Youtube Channel: Lucas Science

