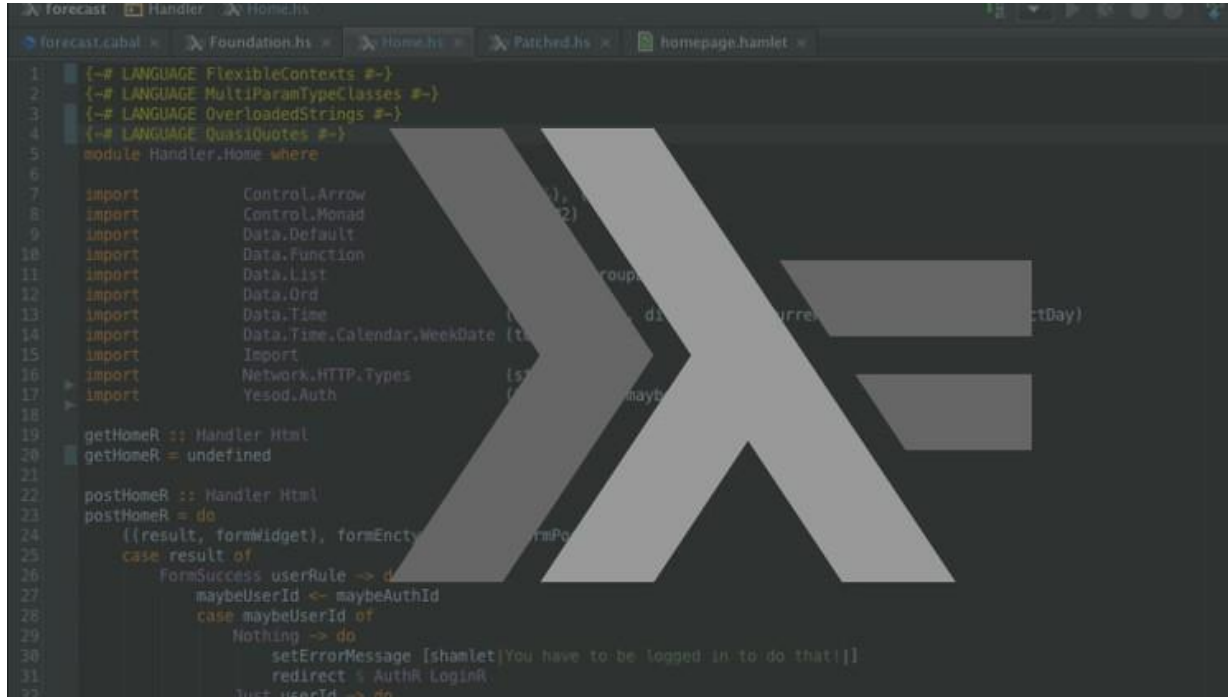


Multiway Tree Problems



```
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 Network.HTTP.Types
16 import Yesod.Auth
17
18 getHomeR :: Handler Html
19 getHomeR = undefined
20
21 postHomeR :: Handler Html
22 postHomeR = do
23   ((result, formWidget), formEnctype) <- runPost
24   case result of
25     FormSuccess userRule -> do
26       maybeUserId <- maybeAuthId
27       case maybeUserId of
28         Nothing -> do
29           setErrorMessage [shamlet|You have to be logged in to do that!|]
30           redirect % AuthR.LoginR
31         Just userId -> do
```

Problem 2



```
forecast.cabal x Handler x Home.hs
forecast.cabal x Foundation.hs x Home.hs x Patched.hs x homepage.hamlet x

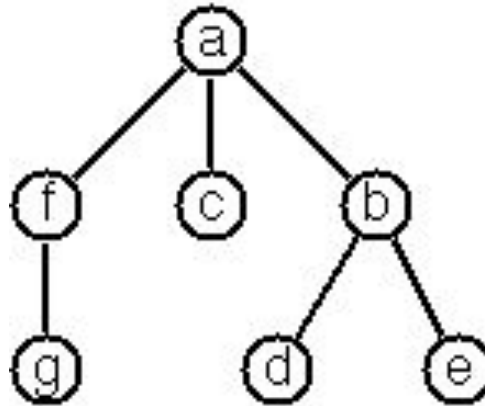
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import Data.Ord
13 import Data.Time
14 import Data.Time.Calendar.WeekDate (toDayOfWeek)
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) <- runFormPost
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
```

Problem 2



Write a function *stringToTree* :: *String* -> *Tree a* that, given a string, it builds the tree. We suppose that the nodes of a multiway tree contain single characters. In the depth-first order sequence of its nodes, a special character ^ has been inserted whenever, during the tree traversal, the move is a backtrack to the previous level.

By this rule, the tree below is represented as: `afg^^c^bd^e^^^`



Problem 2



Examples:

```
stringToTree "afg^^c^bd^e^^"
```

```
-> Node 'a' [Node 'f' [Node 'g' []],Node 'c' [],Node 'b' [Node 'd' [],Node 'e' []]]
```

```
stringToTree "ka^b^c^^"
```

```
-> Node 'k' [Node 'a' [],Node 'b' [],Node 'c' []]
```

Problem 2



For this problem you will have to return a multiway tree of characters, so use:

stringToTree :: String -> Tree Char

Haskell Multiway Trees



In Haskell, we define multiway trees as a datatype:

```
data Tree a = Node a [Tree a]
              deriving (Eq, Show)
```

Example



```
tree1 = Node 'a' [  
    Node 'f' [Node 'g' []],  
    Node 'c' [],  
    Node 'b' [Node 'd' [], Node 'e' []]  
]
```

