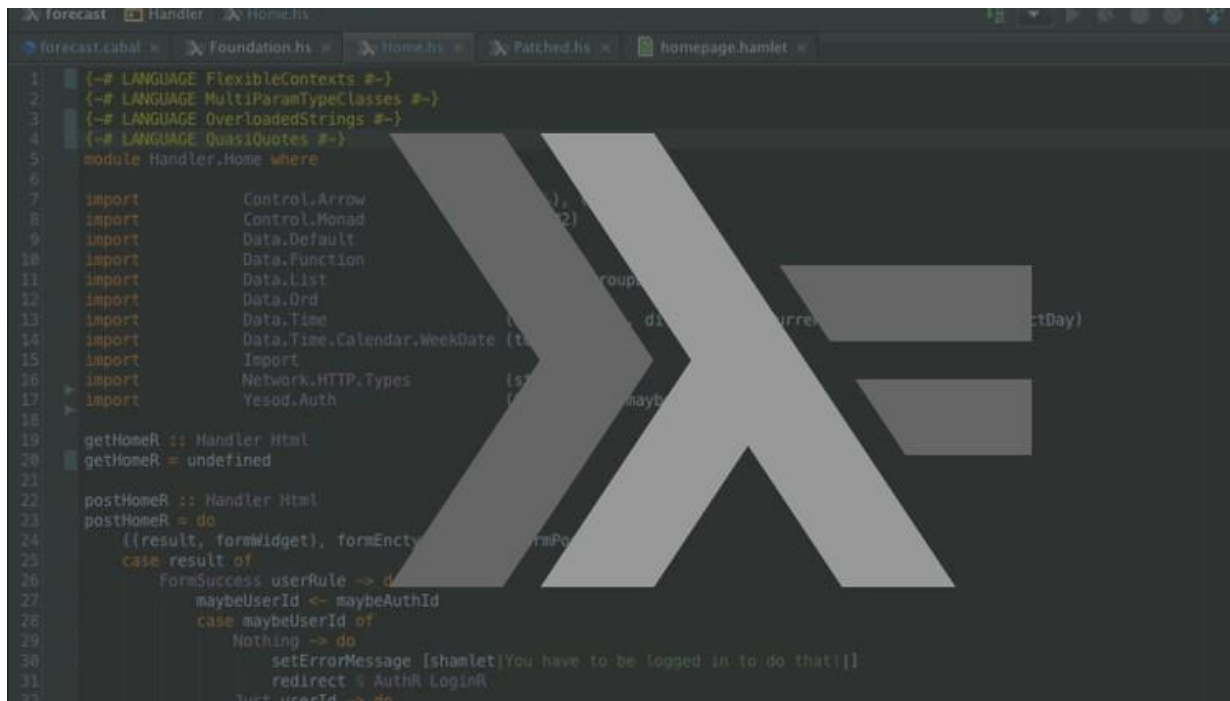


Final Exams



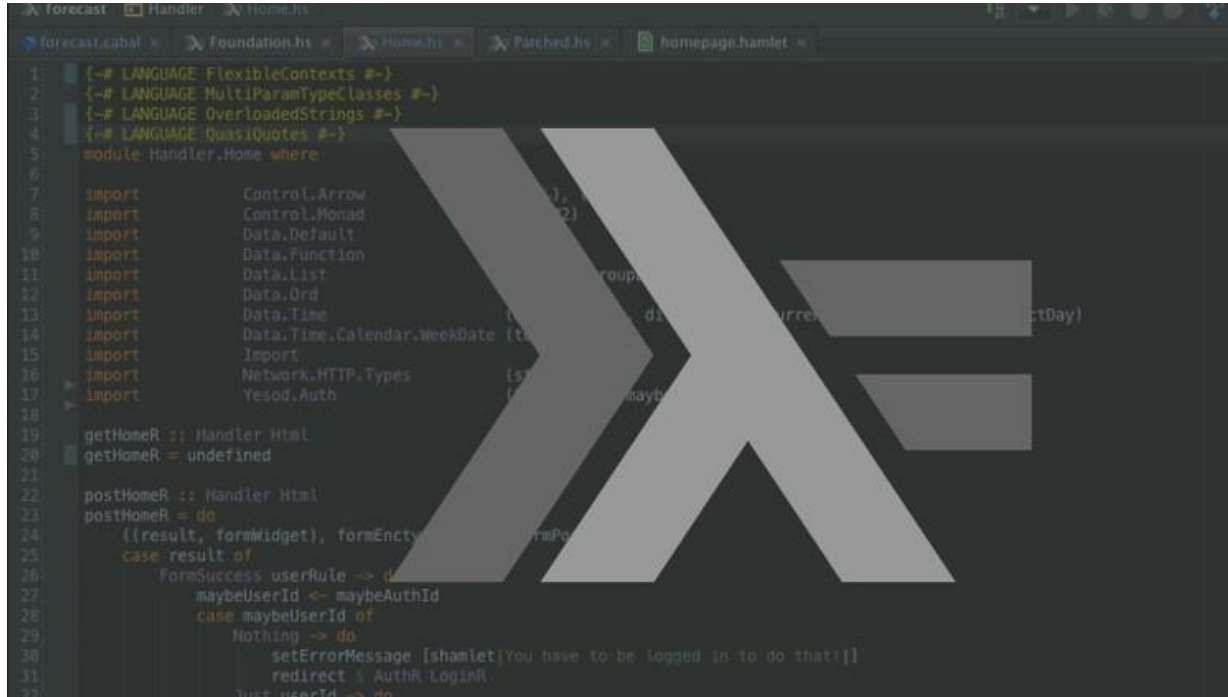
```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import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

Exam 1



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

Problem 1



```
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 (toDayOfYear, dayToWeek)
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) <- 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 1

Define a function $fsmmap :: a \rightarrow [a \rightarrow a] \rightarrow a$ which, given an element x of type a and a list fs of functions of type $a \rightarrow a$, causes $fsmmap\ x\ fs$ to return the application (from left right) of all functions from fs to x

Input

Output

`fsmmap 3 [(+2), (*3), (+4)]`

`-> 19`

`fsmmap "e" [(++ "llo"), (:) 'h', (++ "!")]`

`-> "hello!"`

`fsmmap False []`

`-> False`

Instructor Youtube Channel: Lucas Science

