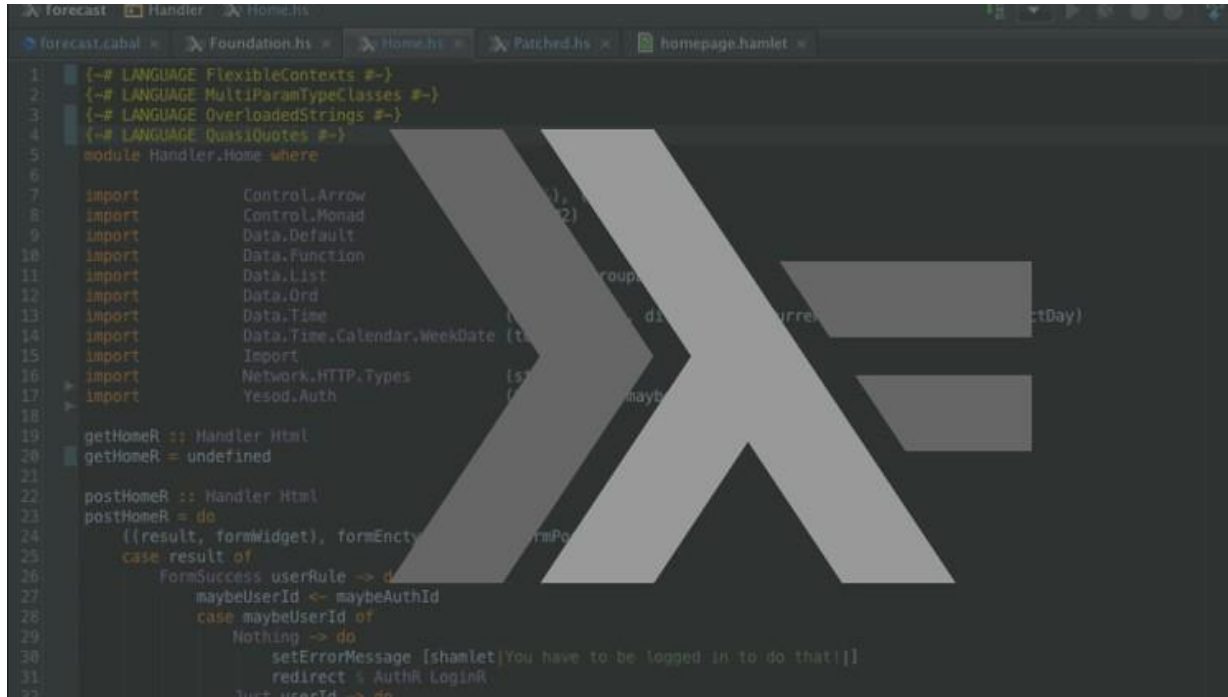


Final Exams



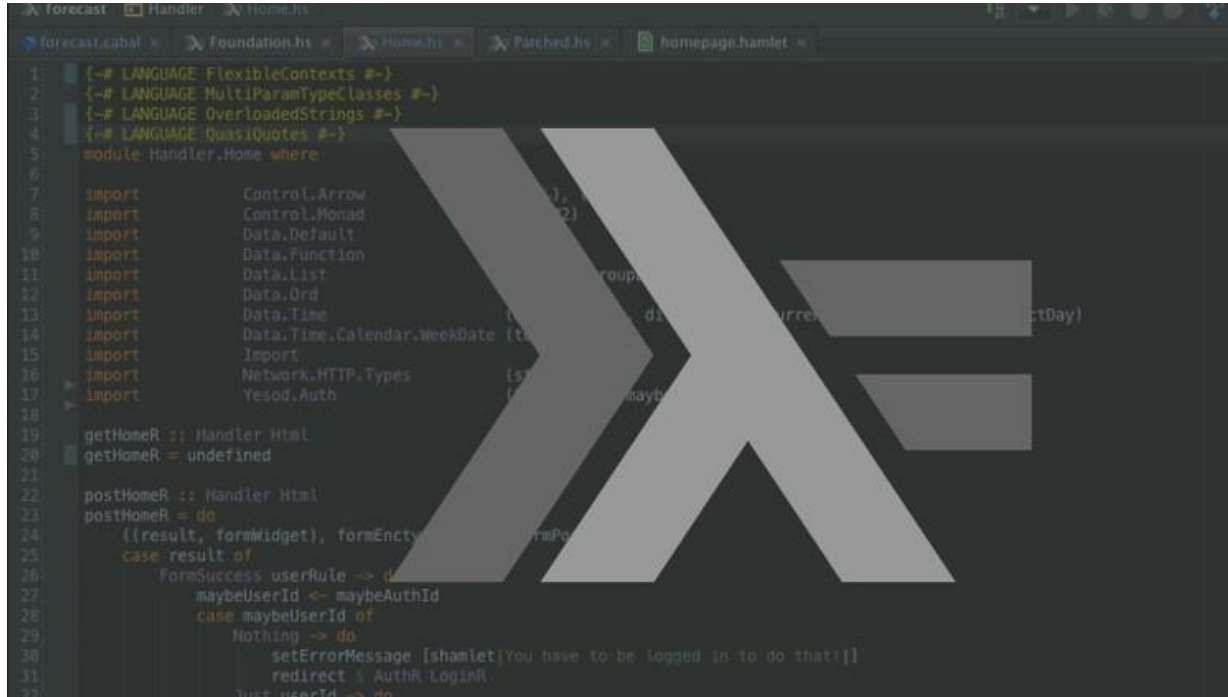
```
1 {-# LANGUAGE FlexibleContexts #-}
2 {-# LANGUAGE MultiParamTypeClasses #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 {-# LANGUAGE QuasiQuotes #-}
5 module Handler.Home where
6
7 import Control.Arrow
8 import Control.Monad
9 import Data.Default
10 import Data.Function
11 import Data.List
12 import 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 2



```
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) <- 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 3



```
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
```

Problem 3

Consider a generic symbol table that converts texts (Strings) to values of type `a` defined by

```
type SymTab a = String -> Maybe a
```

The symbol table returns a *Maybe a* and not an *a* in order to indicate unsuccessful searches.

Maybe a

```
data Maybe a = Just a | Nothing
```

Problem 3

The operations on the symbol table are:

```
empty                ::                SymTab                a  
get    ::    SymTab    a    ->    String    ->    Maybe    a  
set :: SymTab a -> String -> a -> SymTab a
```

where *empty* creates an empty symbol table, *get* returns the value of a text to the symbol table (with *Just* if present or *Nothing* if not), and *set* returns a new symbol table defining a new value for a symbol (and overwriting the old value if the symbol was already in the table).

Implement these three operations on the given type (which you can't change).

Problem 3

Input

```
get (set empty "a" 1) "a"  
get (set empty "a" 1) "b"  
get (set (set empty "a" 1) "b" 2) "a"  
get (set (set empty "a" 1) "b" 2) "b"  
get (set (set empty "a" 1) "b" 2) "c"  
get (set (set empty "a" 1) "a" 2) "a"
```

Output

```
Just 1  
Nothing  
Just 1  
Just 2  
Nothing  
Just 2
```

Instructor Youtube Channel: Lucas Science

