Assignment 5

2019-12-11 15:44
08d925d

due *Tue 8 Oct*

```haskell
module A05 where

import Control.Monad

{- These declarations below are called "type synonyms." They say we're
   going to represent user names and passwords as strings, but we can
   use the type `User` interchangeably with `String`.
-}

type User = String
type Password = String

{- Here is an enumeration of the different kinds of errors that can
   occur in a financial account transaction.
-}

data Error
  = UnknownUser
  | IncorrectPassword
  | InsufficientFunds
  deriving (Eq, Show)

{- To represent an account, we'll just pair an account password with its
   balance (in dollars).
-}

data Account = Account
  { password :: Password
  , balance :: Int
  }
  deriving Show

{- TODO: complete this helper function for modifying the balance field
   within an account object. Here's an example of how it should work:

        ghci> modifyBalance (+10) (Account "secret" 70)
        Account {password = "secret", balance = 80}
```

```
-}

modifyBalance :: (Int -> Int) -> Account -> Account
modifyBalance adj acct = error "TODO"

{- Here is a little "database" of account data for a few users.
-}

bank :: [(User, Account)]
bank = [("alice", Account "$113" 350),
        ("bob", Account "b01!" 290),
        ("chad", Account "c9K2" 700),
        ("dora", Account "3xpM" 980)]

{- TODO: Retrieve the named account from the database, or report the
   `UnknownUser` error. (Note that the function uses an `Either` type,
   so you'll have to report the error tagged with `Left`, or the
   desired account with `Right`. You may want to use the built-in
   function `lookup`, which is helpful for treating a list of pairs as
   a lookup table. It returns a `Maybe` result, so you'll have to
   match on that and convert to `Either`.
-}

getAccount :: User -> Either Error Account
getAccount user = error "TODO"

{- In addition to retrieving the named account (for which it can call
   `getAccount`), this function should verify whether the specified
   password matches what was in the database. If it matches, it should
   return the account tagged with `Right`, otherwise it should return
   `IncorrectPassword` tagged with `Left`. (It can also report
   `UnknownUser`.) Examples:

       ghci> authenticate "barb" "secret"
       Left UnknownUser
       ghci> authenticate "bob" "secret"
       Left IncorrectPassword
       ghci> authenticate "bob" "b01!"
       Right (Account {password = "b01!", balance = 290})
-}

authenticate :: User -> Password -> Either Error Account
```

```
authenticate user pw = error "TODO"

{- After authenticating (for which it should call `authenticate`), this
   function should modify the balance by adding the indicated amount,
   and then return the account record tagged with `Right`. (It can
   still fail, with `UnknownUser` or `IncorrectPassword`.) You can use
   the `modifyBalance` helper to change the balance field. Example:

       ghci> deposit "mob" "b01!" 90
       Left UnknownUser
       ghci> deposit "bob" "b01!" 90
       Right (Account {password = "b01!", balance = 380})
-}

deposit :: User -> Password -> Int -> Either Error Account
deposit user pw amt = error "TODO"

{- This one is very similar to `deposit`, but it can additionally report
   the error `InsufficientFunds`, so that the account balance never
   becomes negative. Examples:

       ghci> withdraw "alice" "secret" 100
       Left IncorrectPassword
       ghci> withdraw "alice" "$l13" 100
       Right (Account {password = "$l13", balance = 250})
       ghci> withdraw "alice" "$l13" 400
       Left InsufficientFunds
-}

withdraw :: User -> Password -> Int -> Either Error Account
withdraw user pw amt = error "TODO"
```