State Monad Example (3H)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

Some Examples (1)

```
module StateGame where

import Control.Monad.State

-- Example use of State monad
-- Passes a string of dictionary {a,b,c}
-- Game is to produce a number from the string.
-- By default the game is off, a C toggles the
-- game on and off. A 'a' gives +1 and a b gives -1.
-- E.g
-- 'ab'    = 0
-- 'ca'    = 1
-- 'cabca' = 0
-- State = game is on or off & current score
--       = (Bool, Int)
```

https://wiki.haskell.org/State_Monad

Some Examples (2)

```
type GameValue = Int
type GameState = (Bool, Int)

playGame :: String -> State GameState GameValue
playGame []     = do
   (_, score) <- get
   return score
```

https://wiki.haskell.org/State_Monad

Some Examples (3)

```
playGame (x:xs) = do
    (on, score) <- get
    case x of
        'a' | on -> put (on, score + 1)
        'b' | on -> put (on, score - 1)
        'c'      -> put (not on, score)
        _        -> put (on, score)
    playGame xs

startState = (False, 0)

main = print $ evalState (playGame "abcaaacbbcabbab") startState
```

https://wiki.haskell.org/State_Monad

Dice Examples

to generate Int dice - result : a number between 1 and 6
throw results from a pseudo-random generator of type StdGen.

the type of the **state processors** will be

State StdGen Int

StdGen -> (Int, StdGen)

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

randomR

the StdGen type : an instance of **RandomGen**

randomR :: (**Random** a, **RandomGen** g) => (a, a) -> g -> (a, g)

assume a is Int (a, a) : range
and g is StdGen a seed

the type of **randomR**

randomR (1, 6) :: **StdGen** -> (Int, **StdGen**)

already have a **state processing function**

A seed of the type **StdGen**
A new seed is generated
by **newStdGen**

(Int, StdGen)
(a random value, a new seed)

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

randomR

If you choose to take a seed,
it should be of type **StdGen**, and you can use **randomR**
to generate a number from it.
Use **newStdGen** to create a new seed
(this will have to be done in IO).

> **import System.Random**
> **g <- newStdGen**
> **randomR (1, 10) g**
(1,1012529354 2147442707)

The result of randomR is a tuple
(a **random value**, a **new seed)**

A seed of the type **StdGen**
A new seed is generated
by **newStdGen**

https://stackoverflow.com/questions/8416365/generate-a-random-integer-in-a-range-in-haskell

randomR

Otherwise, you can use randomRIO
to get a random number directly in the IO monad,
with all the StdGen stuff taken care of for you:

```
> import System.Random
> randomRIO (1, 10)
6
```

https://stackoverflow.com/questions/8416365/generate-a-random-integer-in-a-range-in-haskell

randomR

```
randomR (1, 6) :: StdGen -> (Int, StdGen)
```

```
rollDie :: State StdGen Int
rollDie = state $ randomR (1, 6)
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

randomR

```
import Control.Monad.Trans.State
import System.Random

-- The StdGen type we are using is an instance of RandomGen.
randomR :: (Random a, RandomGen g) => (a, a) -> g -> (a, g)

randomR (1, 6) :: StdGen -> (Int, StdGen)
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

randomR

```
rollDie :: State StdGen Int
rollDie = state $ randomR (1, 6)

rollDie :: State StdGen Int
rollDie = do generator <- get
             let (value, newGenerator) = randomR (1,6) generator
             put newGenerator
             return value



GHCi> evalState rollDie (mkStdGen 0)
6
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

randomR

```
rollDice :: State StdGen (Int, Int)
rollDice = liftA2 (,) rollDie rollDie

GHCi> evalState rollDice (mkStdGen 666)
 (6,1)
```

https://en.wikibooks.org/wiki/Haskell/Understanding_monads/State

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf