Fakultät für Informatik und Mathematik Summer semester 2023

Prof. Dr. Martin Griebl

Exercises for Lecture: Functional Programming

Exercise Sheet 10 (Monads)

Problem 1 (Monadic Variants of Standard Functions)

Implement the following standard Haskell functions using the functions `(>>=)` and `return`. You can also use `do`-notation if you like. What is the more general type of your monadic version of the standard function?

(a) `concat :: [[a]] -> [a]`

(b) `map :: (a -> b) -> [a] -> [b]`
Here you have two possibilities, depending on whether the argument function returns a monadic value or not.

(c) `filter :: (a -> Bool) -> [a] -> [a]`
Here the class `MonadPlus` from module `Control.Monad` may help (we did not consider `MonadPlus` in the lecture, use `:i MonadPlus` to get some information about `MonadPlus`).

Problem 2 (Guess My Number)

Write a program for the game "Guess My Number" (c.f. the file `GuessMyNumber.hs` in Stud.IP). The program shall choose a random number in the range 1 to 100 and the user has to guess which number it is. The program tells the user after each wrong guess whether the number to guess is bigger or smaller than the number entered:

```
I think of a number in the range 1 to 100.
Which number is it?
Try #1 > 50
My number is bigger.
Try #2 > 75
My number is smaller.
Try #3 > 62
My number is smaller.
Try #4 > 56
My number is bigger.
Try #5 > 59
My number is smaller.
Try #6 > 58
My number is smaller.
Try #7 > 57
Correct!
```

You can use the following functions in your implementation:

```
-- randomRIO (x,y) return a random number in the range [x .. y].
randomRIO :: Random a => (a,a) -> IO a

putStr :: String -> IO ()    -- Output a string without an additional newline
getLine :: IO String         -- Read the user's input until the next newline
```

To ensure that the output of `putStr` is visible immediately (and does not stay in the output buffer) even when no newline is output, the program should disable the buffering of standard output at the beginning:

```
main :: IO ()
main = do
   hSetBuffering stdout NoBuffering
   ...
```

You can load your program into GHCi and call the function “main” or use

```
ghc -o game GuessMyNumber.hs -main-is GuessMyNumber.main
```

(assuming that “`GuessMyNumber`” is your module’s name) to compile your program to the binary “`game`”.

It may be necessary (depending on the compiler version) to call GHC(i) with the option `-package random` to make `System.Random` available. In the CIP pool, you may need to run the commands

```
/home/groessli/bin/cabal update
/home/groessli/bin/cabal install --lib random
```

to install package `random` first.

Problem 3 (Counting Monad)

Define a `Monad` instance for the given type constructor `CountM`:

```
data CountM a = C a Int  deriving (Show)
```

The `Int` field of the constructor shall count, how often the bind operator `(>>=)` has been used in a computation; `return` initializes the counter with 0. For example,

```
do { x <- return "abc"; y <- return "def"; return (x++y) }  :: CountM String
```

produces the output

```
C "abcdef" 2
```

because `(>>=)` is used twice in the computation (syntactically hidden using `do` notation).

Note: To define an instance of `Monad`, you also need an instance of `Functor` and of `Applicative`. You can use

```
import Control.Monad (ap)
instance Applicative CountM where
    pure  = return
    (<*>) = ap
```

to define the `Applicative` instance.

Due date: Tuesday, June 27, 2023 at 16:00

Upload your solutions to the `Submissions` folder for this exercise in Stud.IP.