Lesson 7

interactive programming

I/O

Chapter 10

so far we have seen batch programs in Haskell

input is given together with the program that executes in order to print the result

interactive programs ask input and give output possibly several time during the execution

since Haskell is functional we want to see also I/O as a function of type IO
type IO = World -> World

but I/O actions may return a value

type IO a = World -> (a,World)

IO Char returns a Char

IO () pure side effect

interactive programs may need input

use currying

Char -> IO ()

what is World ?

in reality IO is a built-in type whose details are hidden

data IO a = ...

we start with some basic I/O actions
we will compose them to make more sophisticated interactive programs

```
--getChar :: IO Char
--putChar :: Char -> IO ()
--return :: a -> IO a
```

these actions are built into the GHC system

return transform any expression into a IO action that delivers that expression the type IO a is a monad and therefore we use a special notation for composing I/O actions:

```
do v1 <- a1
   v2 <- a2
    ........
   vn <- an
   return (f v1 v2  ... vn)
```

mind the layout

each v <- a is a generator

we use a alone when v doesn't matter

example: an action that reads 3 Char and returns the 1° and 3° ones

```
act :: IO (Char, Char)
act = do x <- getChar
         getChar
         y <- getChar
         return (x,y)
```

omitting the return would result in a type error

Derived primitives

```
getLine :: IO String
getline = do x <- getChar
             if x == ‘\n’ then
               return []
             else
               do xs <- getline
                   return (x:xs)
```

```
putStr :: String -> IO ()
putStr []      = return ()
putStr (x:xs) = do putChar x
                   putStr xs

putStrLn :: String -> IO ()
putStrLn xs = do putStr xs
                 putChar ‘\n’
```

example: an I/O action that prompts for a string and displays its length

```
strlen :: IO ()
strlen = do putStr «Enter a string: »
            xs <- getLine
            putStr «The string has »
            putStr (show (length xs))
            putStrLn « characters»
```

Hangman
is a game as follows.
-one player secretly enters a word
-another player tries to find the word through a series of guesses
-for each guess the program indicates which letters in the secret word occur in the guess and also in which positions of the secret word

```
hangman :: IO ()
hangman = do putStrLn «Think of a word:»
             word <- sgetLine
             putStrLn «Try to guess it :»
              play word
```

```
sgetLine :: IO String
sgetLine = do x <- getCh
              if x == ‘\n’ then
                do putChar x
                   return []
               else
                 do putChar ‘_’
                 xs <- sgetLine
                 return (x:xs)
```

getCh reads a Char without echo to the screen

```haskell
import System.IO
getCh :: IO Char
getCh = do hSetEcho stdin False
           x <- getChar
           hSetEcho stdin True
           return x
```

```
play :: String -> IO ()
play word = do putStr «?»
               guess <- getLine
               if guess == word then
                 putStrLn «You got it!!»
                else
                 do putStrLn (match word guess)
                     play word

match :: String -> String -> String
match xs ys = [if elem x ys then x else ‘-’ | x <- xs]
```