Capítulo 5

Leitura e Escrita

A leitura e a escrita em *Haskell* é feita de uma forma puramente funcional através da Monade `IO`. Não vou aqui descrever as monades nem a forma como o *Haskell* formaliza as entradas e saídas à custa das monades, vou limitar-me a descrever as funções de leitura e escrita presentes no `Prelude`.

5.1 Funções de Escrita

As funções de escrita vão ter todas co-domínio na monade `IO`. Temos assim as seguintes funções de escrita de valores.

`putChar :: Char -> IO ()`	Escreve um caracter
`putStr :: String -> IO ()`	Escreve uma sequência de caracteres
`putStrLn :: String -> IO ()`	Escreve uma sequência de caracteres e muda de linha
`print :: Show a => a -> IO ()`	Escreve um valor.

5.2 Funções de Leitura

As funções leitura vão ter diferentes co-domínios conforme o seu objectivo.

`getChar :: IO Char`	Lê um caracter
`getLine :: IO String`	Lê uma linha e converte-a numa só sequência de caracteres
`getContents :: IO String`	Lê todo o conteúdo da entrada e converte-a numa só sequência de caracteres
`interact :: (String -> String) -> IO ()`	recebe uma função de sequências de caracteres para sequências de caracteres como argumento. Todo o conteúdo da entrada é passado como argumento para essa função, e o resultado dessa aplicação é visualizado.
`readIO :: Read a => String -> IO a`	Lê uma sequência de caracteres.
`readLine :: Read a -> IO a`	Lê uma sequência de caracteres e muda de linha.

5.3 Sequenciação

A forma de combinar vários comandos de escrita (ou de leitura) é feita no âmbito dos operadores monádicos. Temos então dois operadores que nos permitem combinar várias instruções de entrada/saída.

`>>`	Combina dois comandos sequencialmente.
`>>=`	Combina dois comandos sequencialmente, passando o resultado do primeiro como argumento para o segundo.

5.3.1 A notação `do`

Uma sequência ordenada de instruções pode ser escrita de uma forma simplificada através da notação `do`. Temos então:

```
do <padrão associado ao resultado de uma acção> <- <acção>
   <definições locais>
```

Por exemplo:

```
le :: IO()
le = do c <- getChar
        putChar c
```

5.4 Um exemplo

Torres de Hanói Defina uma função que calcule, para um dado número de discos, a solução do problema das Torres de Hanói.

```
-- Move n discos da Torre A para a Torre C, com a Torre B como auxiliar
--
hanoi :: IO ()
hanoi = do putStr "Introduza um Inteiro (de 0 a 10)"
           c<-getChar
           putStrLn "\n\nOs movimentos necessários são:"
           hanoi1(fromEnum(c)-48)

hanoi1 :: Int -> IO()
hanoi1 n = putStr ("\n"++(move n 'A' 'C' 'B'))
          where
          move :: Int -> Char -> Char -> Char -> String
          move 1 orig dest aux =
              "Move um disco de "++[orig]++" para "++[dest]++"\n"
          move n orig dest aux =
              (move (n-1) orig aux dest)++
              (move 1 orig dest aux)++
              (move (n-1) aux dest orig)
```