Input Output

MONDADS

O conceito de mónade é usado para sintetixar a ideia de computação. Uma computação é algo que se passa dentro de uma "caixa negra"
e da qual conseguimos ver apenas os resultados.

Monad é uma classe de construtores de tpos do Haskell.

```
class Monad m where
    return :: a -> m a {-return corresponde a uma computação nula-}
    (>>=)  :: m a -> (a -> m b) -> m b {-conpõe computações aproveitando o valor devolvido pela primeira para o cálculo da segunda. ("bind") -}
    (>>)   :: m a -> m b -> m b
    fail   :: String -> m a
    -- Minimal complete definition: (>>=), return
    p >> q = p >>= \ _ -> q {-compõe computações ignorando o valor devolvido pela primeira no cálculo da segunda -}
    fail s = error s
     |-> t :: m a
         {-significa que t é uma computação que retorna um valor do tipo a. Ou seja, t é um valor do tipo a com efeito adicional captado por m.Este efeito pode ser, por exemplo, uma acção de IO.-}
```

INPUT / OUTPUT

Como conciliar o princípio de "computação por cálculo" com o IO?
Exemplo: Qual será o tipo de uma função `lerChar` que lê um caracter do teclado?
`lerChar :: Char` (?) --> Se assim fosse, `lerChar` seria uma constante do tipo Char (!!)

--> As funções do Haskell são funções matemáticas puras
--> Ler do teclado é um efeito lateral
--> Os programas interativos têm efeitos laterais
--> As funções interativas podem ser escritas em Haskell usando o construtor de tipos IO, para distinguir puras de ações impuras que podem envolver efeitos laterais.
--> (IO a) é o tipo das ações de input / output que retornam um valor tipo a
--> IO é a instância da classe Monad
--> A função que lê do teclado um caracter é getChar :: IO Char getChar é um valor do tipo Char que resulta de uma ação de input/ output.

Algumas funções IO do Prelude

Para "Ler" do standard input (por omissão, o teclado)
`getChar :: IO Char` --> lê um caracter
`getLine :: IO String` --> lê uma string

Para "Escrever" no standard output (por omissão, o ecrã)

`putChar :: Char -> IO ()`

`putStr :: String -> IO ()`

`putStrLn :: String -> IO ()`

`print :: Show a => a -> IO ()`

Para lidar com "Ficheiros de Texto"

`writeFile :: FilePath -> String -> IO ()` --> Escreve uma string no ficheiro

`appendFile :: FilePath -> String -> IO ()` --> Acrecenta no final do ficheiro

`readFile :: FilePath -> IO String` --> Lê o conteúdo do ficheiro para uma string

`type FilePath = String` --> É o nome do ficheiro (pode incluir a path do file system)

Monad IO

O monade IO agrega os tipos de todas as computações onde existem ações de input/ output.

- return :: a -> IO a não faz nenhuma ação de IO. Apenas faz a conversão de tipo.
- (>>=) :: IO a -> (a -> IO b) --> IO b compõe duas ações de IO podendo utilizar o valor devolvido pela primeira para o cálculo da segunda
- (>>) :: IO a -> IO b -> IO b --> compõe duas ações de IO de forma independente

Exemplos já definidos no Prelude:

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

getLine :: IO String
getLine = getChar >>= (\x-> if x=='\n'
                            then return []
                            else getLine >>= (\xs-> return (x:xs)))
```

Notação "do"

O Haskell fornece uma construção sintática (do) para escrever de forma simplificada cadeias de operações monádicas.

Exemplos:

```
do e1 e2   {-ou-}    do {e1;e2}   {-em vez de-}   e1 >> e2
                     do x <- e1   {-em vez de-}   e1 >>= (\x -> e2)
                        e2
                     do x1 <- e1  {-em vez de-}   e1 >>= (\x1-> e2 >>= (\x2->
e3))
                        x2 <- e2
                                                      e3
```

#haskell
#SoftwareEngineering