Programação Funcional
Input/Output em Haskell

Sérgio Soares
sergio@dsc.upe.br

Até aqui

- Programas (funções) auto-contidos
  - sem interação alguma com o usuário
- Mas programas também devem
  - ler e escrever no terminal
  - ler e escrever arquivos
  - controlar dispositivos

ou seja, realizar operações de entrada e saída

O tipo `IO t`

- Imagine tais tipos como programas que executam entrada/saída (IO) e retornam um valor do tipo `t`
- Lê uma linha do teclado

```
getLine :: IO String
```

O tipo `IO t`

- Escreve uma String na tela.

```
putStr :: String -> IO ()
```

o resultado desta interação tem tipo `()`, que é uma tupla vazia. Neste caso significa dizer que a função não retorna nenhum resultado interessante, apenas faz I/O.

O tipo `IO t`

- Lê um caracter do teclado.

```
getChar :: IO Char
```

- Caso especial da função `putStr`, que insere um enter no final.

```
putStrLn :: String -> IO ()
```

Sequenciando ações de IO

- A operação `do`

```
putStrLn :: String -> IO ()
putStrLn str = do putStr str
                  putStr "\n"
```

Sequenciando ações de IO

```
put4times :: String -> IO ()
put4times str = do putStr str
                   putStr str
                   putStr str
                   putStr str
```

Sequenciando ações de IO

- A operação **`if-then`** (programação imperativa?)

```
putNtimes :: Int -> String -> IO ()
putNtimes n str
  = if n <= 1
       then putStr str
       else do putStr str
               putNtimes (n-1) str
```

Sequenciando ações de IO

- Lendo informações do teclado

```
getNput :: IO ()
getNput = do line <- getLine
             putStr line
```

- O comando **`<-`** funciona "como" uma atribuição
  - mas não é possível "atribuir" outro valor a **`line`** em outra parte do código
  - nomeia a saída da função **`getLine`**

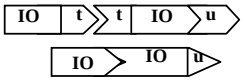
Outra abordagem

A operação 'then', simbolizada por **`>>=`** dá seqüência a duas operações, uma após a outra.

```
(>>=) :: IO t -> (t -> IO u) -> IO u
```

Esta operação combina um **`IO t`** com uma função que pega o resultado dessa expressão (do tipo **`t`**) e retorna algo de tipo **`IO u`**.

Podemos combinar as expressões, passando o resultado da primeira como primeiro argumento da segunda.



Outras Interações

- O opeador **`>>`** é igual ao **`>>=`**, mas ignora o resultado da primeira para a segunda interação

```
(>>) :: IO t -> IO u -> IO u
```

- Retorna um valor do tipo **`IO t`** (converte do tipo **`t`** para o tipo **`IO t`**)

```
return :: t -> IO t
```

Exemplo

```
main :: IO()
main = putStr "Digite seu nome:"  >>
       getLine >>= \ st ->
       putStr "Ao contrario e':"  >>
       putStr (reverse st)
```

Resumo das funções do tipo `IO t`

```
getLine :: IO String
getChar :: IO Char
putStr :: String -> IO ()
putStrLn :: String -> IO ()
(>>=) :: IO t -> (t -> IO u) -> IO u
(>>) :: IO t -> -> IO u -> IO u
return :: t -> IO t
```

Manipulação de arquivos

- Leitura de arquivos:

```
type NomeArquivo = String
readFile:: NomeArquivo -> IO String
```

- Escrever em arquivos:

```
writeFile:: NomeArquivo -> String -> IO()
```

Exemplo

```
main :: IO ()
main =
  putStrLn "Escrevendo" >>
  writeFile "a.txt" "Hello\nworld" >>
  putStrLn "Lendo o arquivo" >>
  readFile "a.txt" >>=
  \x -> putStrLn x
```