Entrada e Saída em Haskell

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Interação com o usuário

- Haskell, como as outras linguagens de programação, possui funções que se comunicam com o sistema operacional para realizar entrada e saída de dados

- Estas operações trabalham com valores do tipo (**IO t**), e durante a sua avaliação requisitam operações de IO ao sistema operacional

```
main = putStr "Olá mundo!“
--putStr :: String -> IO ()
```

Interação com o usuário

- Se o valor devolvido por uma função for do tipo **IO**, o interpretador Haskell não responde simplesmente imprimindo o valor na tela, e sim mandando uma requisição ao sistema operacional para que faça a entrada ou saída de dados

- O tipo **IO** é polimórfico,e quando uma função não retorna nada utiliza-se o tipo **( )**, como em **putStr**

Funções de saída/escrita

- putChar :: Char -> IO
  - Escreve um caractere na saída padrão
- putStr :: String -> IO ()
  - Escreve uma cadeia de caracteres na saída padrão
- putStrLn :: String -> IO ()
  - Idem ao interior, acrescentando um salto de linha ao final
- print :: Show a => a -> IO ()
  - Da classe genérica Show, uma função genérica para escrever um valor, numérico, caracter, etc na saída padrão

Funções de saída/escrita

```
olaMundo :: IO ()
olaMundo = putStr "Olá mundo cruel!"

num_float :: Float
num_float = 3.4567

num_double :: Double
num_double = 9.87654

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

Funções de estrada/leitura

- getChar :: IO Char
  - Lê um caractere na entrada padrão
- getLine :: IO String
  - Lê uma linha na saída padrão, e converte em uma cadeias de caracteres
- getContents :: IO String
  - Lê todo o conteúdo da entrada, como um arquivo, e converte em uma cadeia de caracteres

Funções de estrada/leitura

- interact :: (String -> String) -> IO ()
  - Recebe como parâmetro uma função que que opera sobre um conjunto de caracteres. Todo o conteúdo da entrada é passado como argumento para essa função, e o resultado é visualizado na tela
- readIO:: Read a => String ->IO a
  - Lê um cadeia de caracteres para um variável do tipo IO a
- read :: Read a => String -> a
  - Lê uma sequência de caracteres e converte para uma variável do tipo a

Expressão do

- Em Haskell uma sequência de entrada e saída de dados é expressa através de uma expressão do:

```
main = do
        putStr ("Escreva uma palavra: ")
        palavra <- getLine
        putStr ("Palavra invertida: "++ reverse palavra)
```

- Note que a função main possui o seguinte tipo:

```
main :: IO ()
```

Arquivos

- Existem duas funções principais para se trabalhar com arquivos em Haskell
  - writeFile :: String -> String -> IO ()
  - readFile :: String -> IO String

```
main = do
        putStr ("Escreva uma linha e tecle ENTER: ")
        linha <- getLine
        nome <- criaArq linha
        putStr ("A linha \n" ++ linha ++ "\nesta no arquivo " ++ nome ++ "!")
```

Arquivos

```
criaArq :: String -> IO String
criaArq linha = do
                    putStr ("Nome do Arquivo a ser criado: ")
                    nome <- getLine
                    writeFile nome linha
                    return (nome)
```

- return :: a -> IO a

Arquivos

- Considere a existência de um arquivo haskell.txt, podendo ser do exemplo anterior

```
adiciona = do
            putStr ("Linha para adicionar ao arquivo haskell2.txt:\n")
            linha <- getLine
            arquivo <- readFile "haskell.txt"
            writeFile "haskell2.txt" (arquivo ++ "\n" ++ linha)
            putStr ("Linha adicionada!")
```

Interações

- Usando recursão podemos trabalhar com entrada de dados de forma ilimitada

```
main = do
        nomes <- leNomes
        putStr (unlines (sort nomes))
```

Interações

```
leNomes = do
            putStr ("Escreva um nome: ")
            nome <- getLine
            if nome == ""
                then return []
                else do
                        nomes <- leNomes
                        return ([nome] ++ nomes)


sort [] = []
sort (a:b) = sort [x | x <- b, x<a]
             ++ [a] ++
             sort [x | x <- b, x>= a]
```

Interações

- Estruturas de repetição, tais como for, while, repeat, etc de programação imperativa são implícitas em linguagens funcionais

```
-- exemplo de while
le_imprime = do{
                    entrada <- getLine; -- início do laço
                    if (entrada == [])  -- teste condicional
                          then return ()
                          else do{
                                    putStrLn entrada;
                                    le_imprime;
                                  }
                  } -- fim do laço
```

Programa em Haskell

- Um **programa** em Haskell é uma coleção de módulos

- Um dos módulos deve ser chamado **Main** e deve exportar a variável main , do tipo IO t , para algum t

- Para se criar um módulo, utiliza-se a palavra reservada module, e em seguida o nome do módulo. Após o nome, lista-se todas as funções que se quer utilizar em outros programas. Logo depois vem a palavra where e as implementações

- Quando um outro programa precisar utilizar o módulo utiliza-se a palavra reservada import

Exemplo de programa em Haskell

```
module Main (main) where

main :: IO ()
main = putChar 'H'

Main> main
H
```