DCC909 – Programação Funcional

AULA 04

Carlos Bruno Oliveira Lopes

Engenheiro de Computação

Mestre em Ciência da Computação

HASKELL

Entradas e Saídas

- São valores que representam um interação com o “mundo”.
- Uma ação de E/S pode ser executada pelo sistema computacional para interagir com o mundo e retornar um valor obtido através desta interação.
- Em Haskell IO a é o tipo das ações de entrada e saída que interagem com o mundo e retornam um valor do tipo a.
  - IO a é um tipo abstrato, logo sua representação não está disponível nos programas.

HASKELL

Tipo unit

- é comumente usado para indicar o tipo de uma expressão que não possui um valor interessante, irrelevante para o cálculo da expressão maior que a contém.
  - Corresponde à noção do tipo **void** da linguagem C.
  - Ele representa um tipo cujo conjunto de valores é vazio ou unitário, ou seja, não há nenhum valor, ou há exatamente um valor desse tipo.
- Ele ocorre com frequência nos tipos das ações de entrada e saída que, quando executadas, não têm nenhum valor interessante para o retorno.

  Ex.:

  - Uma ação para exibir uma mensagem na saída padrão produz um efeito, porém não tem nenhum valor de retorno útil. Por este motivo o valor retornado por esta ação é a tupla vazia.

HASKELL

Ações de saída padrão

– Função **putChar**

```
putChar :: Char -> IO ()
```

- é uma função que recebe um caractere e resulta em uma ação de E/S que, quando executada, interage com o mundo inserindo o caractere na saída padrão e retorna a tupla vazia ().

Ex.:

```
putChar 'H'
```

HASKELL

Ações de saída padrão

- Função **putStr**

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

  - Recebe uma string e resulta em uma ação de E/S que, quando executada, interage com o mundo inserindo a string na saída padrão e retorna a tupla vazia.

- Função **putStrLn**

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

  - Recebe uma string e resulta em uma ação de E/S que, quando executada, interage com o mundo inserindo a string seguida do caractere '\n' na saída padrão e retorna a tupla vazia.

HASKELL

Ações de saída padrão

– Função **print**

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

- Recebe um valor e resulta em uma ação de E/S que, quando executada, insere na saída padrão o valor convertido para string, seguido de mudança de linha, e retorna a tupla vazia.

HASKELL

Ações de entrada padrão

- Função **getChar**

```
getChar :: IO Char
```

  - Quando executada, interage com o mundo extraindo o próximo caractere disponível da entrada padrão e retorna este caractere.

- Função **getLine**

```
getLine :: IO String
```

  - Quando executada, interage com o mundo extraindo a próxima linha disponível na entrada padrão e retorna esta linha.

HASKELL

Ações de entrada padrão

- Função **getContents**

```
getContents :: IO String
```

  - Quando executada, interage com o mundo extraindo todos os caracteres da entrada padrão e retorna a string formada pelos caracteres.

- Função **readLn**

```
readLn :: Read a => IO a
```

  - Quando executada, interage com o mundo extraindo a próxima linha disponível na entrada padrão e retorna um valor obtido dessa string.

HASKELL

Programa em Haskell

- Um programa em Haskell é uma ação de E/S.
- Um programa é organizado como uma coleção de módulos.
  - Um dos módulos deve ser chamado Main e deve exportar a variável main, do tipo IO t, para algum tipo t.
  - Quando o programa é executado pelo sistema operacional, a ação main é executada, e o seu resultado (do tipo t) é descartado.

HASKELL

Programa em Haskell

Ex.:

```
module Main (main) where

main :: IO ()
main = putChar 'A'
```

- Quando o programa é executado:
  1. **main** recebe (automaticamente) como argumento o mundo existente antes de sua execução,
  2. realiza ações de entrada e saída,
  3. resultando em uma tupla vazia (nenhum valor interessante é produzido), e
  4. produzindo um novo mundo que reflete o efeito das ações de entrada e saída realizadas.

HASKELL

Compilando e executando um programa em Haskell

1. Grave o código fonte do programa em um arquivo texto, digamos “aula04_ex.01.hs”.

```
module Main (main) where
main :: IO ()
main = putChar 'A'
```

2. Compile o programa no terminal:

```
$ ghc --make putchar-a
```

3. Execute o programa no terminal:

```
$ .\aula04_ex.01.exe
```

HASKELL

- Combinar ações de entrada e saída
- Expressão **do**
  - Permite combinar várias ações de E/S de forma sequencial.
  - Uma expressão **do** é da forma:

$$\text{do } \{ \text{ ação}_1 \text{ ; ...; ação}_n \text{ ; expressao } \}$$

onde $n \geq 0$, e expressão é uma ação de E/S.

- Cada ação_i pode ser da forma:
  - expressao. Uma ação de E/S cujo retorno é ignorado;
  - padrao <- expressao. O padrão é uma variável cujo o valor e o retorno da ação.
  - **let** declarações. permite fazer declarações locais de variáveis e funções cujo escopo se estende até o final da expressão do.

HASKELL

Combinar ações de entrada e saída

Ex.:

```
module Main (main) where

main :: IO ()
main = do { putChar 'F' ; putChar 'i' ; putChar 'm' }
```

ou

```
module Main (main) where

main :: IO ()
main = do { putChar 'F'
          ; putChar 'i'
          ; putChar 'm'
          }
```

HASKELL

Combinar ações de entrada e saída

Regra de layout com expressão **do**

- As chaves { e } e os pontos-e-vírgulas ; podem ser omitidos na sequência de ações, sendo substituídos por uso de indentação adequada.
  - Cada ação que compõe a expressão do deve começar na mesma coluna e, se continuar em linhas subsequentes, deve sempre ocupar as colunas à direita da coluna onde iniciou.

Ex.:

```
module Main (main) where

main :: IO ()
main = do putChar 'F'
          putChar 'i'
          putChar 'm'
```

HASKELL

Exemplos de programas interativos

- ler um caracter

```
module Main (main) where

main :: IO Char
main = getChar
```

- Ler e exibir um caracter

```
module Main (main) where

main :: IO ()
main = do caracter <- getChar
          putChar caracter
```

HASKELL

Exemplos de programas interativos

- ler e exibir um caracter (v2)

```haskell
module Main (main) where

import Data.Char (toLower, toUpper)

main :: IO ()
main = do letra <- getChar
          putChar (toLower letra)
          putChar (toUpper letra)
```

HASKELL

Exemplos de programas interativos

– saudação

```
module Main (main) where

main :: IO ()
main = do putStrLn "Qual é o seu nome? "
          nome <- getLine
          putStr nome
          putStrLn ", seja bem vindo(a)!"
```

HASKELL

Exemplos de programas interativos

– soma de dois números

```haskell
module Main where

main :: IO ()
main =
  do putStrLn "Digite um número: "
     s1 <- getLine
     putStrLn "Digite outro número: "
     s2 <- getLine
     putStr "Soma dos números digitados: "
     let n1 = read s1 :: Rational
         n2 = read s2 :: Rational
         s  = fromRational (n1 + n2) :: Double
     putStrLn (show s)
```

HASKELL

Exemplos de programas interativos

– soma de dois números (v2)

```haskell
module Main (main) where

main :: IO ()
main = do putStrLn "Digite um número: "
          n1 <- readLn
          putStrLn "Digite outro número: "
          n2 <- readLn
          putStr "Soma dos números digitados: "
          putStrLn (show (n1 + n2))
```

HASKELL – exercício

Escreva um programa que solicita ao usuário três números em ponto flutuante, lê os números, e calcula e exibe o produto dos números.

Dicas: Provavelmente será necessária uma anotação de tipo para que o programa funcione com números em ponto flutuante, pois a operação de multiplicação é definida para todos os tipos numéricos e, não havendo informações no contexto suficientes para decidir o tipo numérico a ser usado, o tipo **Integer** é escolhido. A anotação de tipo pode ser feita em qualquer subexpressão do programa.

Exemplo de execução da aplicação

```
Digite um número:
10
Digite outro número:
2.3
Digite outro número:
5
Produto dos números digitados: 115.0
```

HASKELL

Saída bufferizada

função **hFlush** (definida no módulo System.IO)

- recebe um manipulador de arquivo (handle) e resulta em uma ação de E/S que, quando executada, faz com que os itens armazenados no buffer de saída do manipulador sejam enviados imediatamente para a saída.

```
hFlush :: Handle -> IO ()
```

  - O tipo **Handle** (definido no módulo System.IO) é um tipo abstrato que representa um dispositivo de E/S internamente para o Haskell.

HASKELL

Saída bufferizada

função **hFlush** (definida no módulo System.IO)

- O módulo **System.IO** define variáveis que representam alguns dispositivos padrões:

```
stdin  :: Handle  -- entrada padrão
stdout :: Handle  -- saída padrão
stderr :: Handle  -- saída de erro padrão
```

HASKELL

Saída bufferizada

função **hFlush** (definida no módulo System.IO)

Ex.:

– Esvazia o buffer da saída padrão antes de fazer a entrada de dados;

```haskell
module Main (main) where

import System.IO (stdout, hFlush)

main :: IO ()
main = do putStr "Digite um número: "
          hFlush stdout                    -- esvazia o buffer de saída
          s1 <- getLine
          putStr "Digite outro número: "
          hFlush stdout                    -- esvazia o buffer de saída
          s2 <- getLine
          putStr "Soma dos números digitados: "
          putStrLn (show (read s1 + read s2))
```

HASKELL

Saída bufferizada

função **hSetBuffering** (definida no módulo System.IO)

- Pode ser utilizada para configurar o modo de bufferização de um dispositivo.

```
hSetBuffering :: Handle -> BufferMode -> IO ()
```

  - O tipo **BufferMode** (definido no módulo System.IO) representa um modo de bufferização:
    - » sem buferização: **NoBuffering**
    - » buferização por linha: **LineBuffering**
    - » buferização por bloco: **BlockBuffering**
  - Normalmente a saída para o dispositivo padrão é feita com *buferização por linha*.

HASKELL

Saída bufferizada

função **hSetBuffering** (definida no módulo System.IO)

```
hSetBuffering hdl mode
```

- é uma ação que, quando executada, configura o modo de bufferização para o handler **hdl**.

Ex.:

```
hSetBuffering stdout NoBuffering
```

HASKELL

Saída bufferizada

função **hSetBuffering** (definida no módulo System.IO)

Ex.:

```haskell
module Main (main) where

import System.IO (stdout, hSetBuffering, BufferMode(NoBuffering))

main :: IO ()
main = do hSetBuffering stdout NoBuffering -- desabilita a bufferização
          putStr "Digite um número: "
          s1 <- getLine
          putStr "Digite outro número: "
          s2 <- getLine
          putStr "Soma dos números digitados: "
          putStrLn (show (read s1 + read s2))
```

HASKELL

Mais exemplos de programas interativos Peso ideal

- Peso ideal

sexo	peso ideal
masculino	$72.7 \times h - 58$
feminino	$62.1 \times h - 44.7$

Escrever um programa em Haskell que recebe a altura e o sexo de uma pessoa e calcula e mostra o seu peso ideal, utilizando as fórmulas constantes na tabela a acima.

```haskell
module Main (main) where

import System.IO (stdout, hSetBuffering, BufferMode(NoBuffering))

main :: IO ()
main =
  do hSetBuffering stdout NoBuffering
     putStr "Altura (m): "
     h <- readLn
     putStr "Sexo (f/m): "
     s <- getLine
     if s == "F" || s == "f"
       then putStrLn ("Peso ideal: " ++ show (62.1 * h - 44.7))
       else if s == "M" || s == "m"
            then putStrLn ("Peso ideal: " ++ show (72.7 * h - 58))
            else putStrLn "Sexo inválido"
```

HASKELL

Mais exemplos de programas interativos Peso ideal

- Situação do aluno

  Faça um programa que receba três notas de um aluno, e calcule e mostre a média aritmética das notas e a situação do aluno, dada pela tabela a acima.

média das notas	situação
menor que 3	reprovado
entre 3 (inclusive) e 7	exame especial
acima de 7 (inclusive)	aprovado

```
module Main (main) where
import System.IO (stdout, hSetBuffering, BufferMode(NoBuffering))

prompt msg = do putStr msg
                readLn

main = do hSetBuffering stdout NoBuffering
          n1 <- prompt "Nota 1: "
          n2 <- prompt "Nota 2: "
          n3 <- prompt "Nota 3: "
          let media = (n1 + n2 + n3)/3
          putStrLn ("Média: " ++ show media)
          putStr "Situação: "
          if media < 3
            then putStrLn "reprovado"
            else if media < 7
                   then putStrLn "exame especial"
                   else putStrLn "aprovado"
```

HASKELL – exercícios

Escreva um programa em Haskell que solicita ao usuário para digitar uma frase, lê a frase da entrada padrão, e testa se a frase lida é uma palíndrome, exibindo uma mensagem apropriada.

A frase deve ser lida como a próxima linha da entrada.

Exemplo de execução da aplicação

```
Digite uma frase:
abcddcba
É uma palíndrome.
```

Exemplo de execução da aplicação

```
Digite uma frase:
ABCdCBA
É uma palíndrome.
```

Exemplo de execução da aplicação

```
Digite uma frase:
ouro preto
Não é uma palíndrome.
```

Dicas: Para verificar se uma frase é palíndrome basta verificar se ela é igual à sua reversa. Para reverter uma string pode-se usar a função `reverse` do prelúdio.

HASKELL – Exercícios

1. Escreva um programa em Haskell que solicita ao usuário uma temperatura na escala Fahrenheit, lê esta temperatura, converte-a para a escala Celsius, e exibe o resultado. Use a seguinte equação para a conversão: $C = \frac{5}{9} \times (F - 32)$.
2. Escreva uma função que recebam três números e que devolva a sua soma se forem todos positivos e zero caso contrário.
3. Escreva uma função que receba três números e devolva **True** se a diferença entre os dois primeiros for inferior ao terceiro e **False** caso contrário.
4. Escreva uma função addDigit que receba dois inteiros, o segundo dos quais entre 0 e 9, e que devolva o inteiro resultante de acrescentar o segundo no fim do primeiro. Por exemplo:

```
ghci> addDigit (-123) 4
-1234
```