Haskell: I/O

Lidando com Entradas e Saídas em Haskell

Hello world

Código:

// Criar um arquivo chamado helloWorld.hs

main = putStrLn “Hello world”

Como executar:

$ ghc helloWorld.hs //compilando

$ ./helloWorld //executando

//maneira alternativa, compila e executa

// Não gera os arquivos .o .hi nem o executável

$ runhaskell helloWorld.hs

Saída:

Hello world

Principais funções de leitura

putStrLn - (String -> IO ()) - Recebe uma string e exibe na saída padrão com quebra de linha;
putStr - (String -> IO ()) - Recebe uma string e exibe na saída padrão, sem quebra de linha;
putChar - (Char -> IO()) - Recebe um char e exibe na saída padrão, sem quebra de linha;
print - (a -> IO()) - Recebe um objeto qualquer e exibe na saída padrão a representação em string do objeto recebido, invoca a função **show** por debaixo dos panos;

show - (a -> String) - Gera uma representação em String do objeto recebido;

Como Ler da Entrada

getLine - (IO String) - Lê uma sequência de caracteres informado através da entrada padrão até encontrar uma quebra de linha;
Para usar o valor recebido pela entrada, é preciso extrair tal valor do IO. Para isso é preciso usar o operador “<-”.

Exemplo:

```
main = do
        nome <- getLine
        putStrLn nome
```

readLn - (IO a) - Tem a mesma função do **getLine,** porém é permitido especificar o tipo da entrada.

Exemplo:

```
main = do
        idade <- readLn :: Io Int
        putStrLn $ “Sua idade eh: ” ++ show idade
```

---

read - (String -> a) - Faz um cast de string para outro tipo.

Informações Importantes

- As funções do IO são impuras, ou seja, o valor de retorno dela pode ser diferente para cada invocação;
- Para receber o valor do de uma função do tipo IO, é necessário usar o operador “**<-**”;
- Toda função que utiliza o IO no seu escopo deve ter um retorno do tipo **IO a;**

Agora, mão na massa... Ou melhor, no teclado!

Exercício 1

Multiplicação de potências

Um aluno de matemática, cansado de calcular a multiplicação das i-ésimas potências dos i-ésimos números menores de um número, decidiu pedir a você, de computação, para fazer um programa que faça isso, o cálculo é feito da seguinte forma:

P(n) = 1^1 * 2 ^ 2 * ... * n^n

Escreva um programa que solucione esta função.

Exercício 1 - Solução

```
calculo numero controle parcial
        |numero == controle = parcial * (numero ^ numero)
        |otherwise = calculo numero (controle+1) (parcial ^ controle)


main = do
        n <- readLn
        print(calculo n 1 1)
```

Exercício 2

Números Coprimos

Dois números são coprimos se não existir um número diferente de 1 que divida os dois números, ou seja, se os dois tiverem apenas 1 como divisor em comum.
Escreva um programa que leia dois números e determine se eles são coprimos ou não.

Entrada:
2
3
Saída:
True

Entrada:
3
9
Saída:
False

Exercício 2 - Solução

```
Coprimos :: Int -> Int -> Bool
coprimos n1 n2 divisor
  | divisor > menor_numero n1 n2 = True
  | (mod n1 divisor) /= 0 || (mod n2 divisor) /= 0 = coprimos n1 n2 (divisor + 1)
  | otherwise = False

main :: IO()
main = do
  n1 <- readLn
  n2 <- readLn
  print(coprimos n1 n2 2)
```

```
menor_numero :: Int -> Int -> Int
menor_numero n1 n2
  | n1 > n2 = n2
  | otherwise = n1
```

Exercício 3

Restaurante da Dona Bel

Dona Bel possui um pequeno restaurante e gostaria de fazer um aplicativo para facilitar a compra de seus clientes. Os custos de suas refeições são fixas em:
Café da manhã: R$ 8,00
Almoço: R$ 12,00
Jantar: R$ 10,00

O aplicativo deverá receber a qunatidade de refeições que o usuário fará e, em ordem, qual o tipo de refeição que o usuário irá querer e, logo em seguida, retornar qual o valor total do pedido completo. Por exemplo, se o usuário quiser pedir 5 cafés da manhã, o programa deverá mostra 40, pois 5 * 8 = R$ 40,00.

Exercício 3

entrada:

5

cafe

cafe

almoco

jantar

jantar

Saída:

48

entrada:

6

jantar

jantar

jantar

jantar

jantar

jantar

Saída:

60

Exercício 3 - Solução

```
main :: IO()
main = do
pedidos <- readLn :: IO Int
calculaPedidos pedidos 0

calculaPedidos :: Int -> Int -> IO()
calculaPedidos x total
|x > 0 = do
pedido <- getLine
calculaPedidos (x-1) (total + (refeicao pedido))
|otherwise = do
putStrLn (show (total))
```

```
refeicao :: String  -> Int
refeicao x
| x == "cafe" = 8
| x == "almoco" = 12
| x == "jantar" = 10
| otherwise = 0
```

Exercício 4

Faça um programa que dado um inteiro n e leia em seguida n inteiros e calcule a soma desses números.

Entrada:

4

1

2

3

4

Saída:

10

Entrada:

1

1000

Saída:

1000

Exercício 4 - Solução

```haskell
loop :: Int -> Int -> IO()
loop sumAcc 0 = putStrLn $ show sumAcc
loop sumAcc count = do
  n <- readLn :: IO Int
  loop (sumAcc + n) (count - 1)

main = do
  n <- readLn :: IO Int
  loop 0 n
```

Haskell: Aleatoriedade

Gerando e manipulando aleatoriedade em Haskell

Aleatoriedade

```haskell
import System.Random (randomRIO)

main :: IO ()
main =
  do putStrLn "Lançamento de dois dados"
     x <- randomRIO (1,6::Int)
     y <- randomRIO (1,6::Int)
     putStrLn ("Faces obtidas: " ++ show x ++ " e " ++ show y)
```

Aleatoriedade

Exemplo do uso de random:

https://gist.github.com/filipegl/29a98c34b195e325adfd3ad2785bf1db

Como instalar random no ubuntu:

```
$ sudo apt install cabal-install
$ cabal update
$ cabal install random
```

Referências

- http://haskell.tailorfontela.com.br/input-and-output
- https://www.haskell.org/tutorial/io.html
- https://wiki.haskell.org/Introduction_to_IO
- http://hackage.haskell.org/package/random-1.1/docs/System-Random.html
- https://github.com/wesleymonte/PLProject/blob/master/Genetic%20Algorithm%20-%20Haskell/GA.hs#L36
- http://www.decom.ufop.br/romildo/2013-2/bcc222/practices/10p1-valores-aleatorios.pdf