

Tipos Básicos em Haskell

Lucas Albertins

Departamento de Computação - UFRPE

Números Inteiros

- Valores

- ..., -3, -2, -1, 0, 1, 2, 3, ... :: Int

- Funções:

- +, *, -, ^, div, mod, abs :: Int -> Int -> Int

- >, >=, ==, /=, <=, < :: Int -> Int -> Bool

Int possui precisão fixa (*bounded*)

Integer possui precisão arbitrária

Números Inteiros - Exemplos

> 2 ^3

~> 8

> div 14 3

~> 4

> 14 'div' 3

~> 4

> mod 14 3

~> 2

> 14 'mod' 3

~> 2

- Valores
 - `True, False :: Bool`
- Funções:
 - `&&, || :: Bool -> Bool -> Bool`
 - `not :: Bool -> Bool`

Booleans - Exemplos

```
eXor :: Bool -> Bool -> Bool
```

```
eXor x y = (x || y) && not ( x && y )
```

—*outra forma*

```
eXor :: Bool -> Bool -> Bool
```

```
eXor True x = not x
```

```
eXor False x = x
```

Booleanos - Exemplos

Verifica se não houve vendas em uma semana n

```
vendasNulas :: Int -> Bool  
vendasNulas n = (vendas n == 0)
```

ao invés de

```
vendasNulas :: Int -> Bool  
vendasNulas n  
| vendas n == 0 = True  
| otherwise     = False
```

- Valores
 - `'a'` , `'b'` , ... :: `Char`
- Funções:
 - `ord` :: `Char` \rightarrow `Int`
 - `chr` :: `Int` \rightarrow `Char`

Podem ser encontradas na biblioteca `Data.Char`

Caracteres - Exemplo

```
fromEnum :: Char -> Int
```

```
toEnum :: Int -> Char
```

```
offset = fromEnum 'A' - fromEnum 'a'
```

```
maiuscula :: Char -> Char
```

```
maiuscula ch = toEnum (fromEnum ch + offset)
```

```
ehDigito :: Char -> Bool
```

```
ehDigito ch = ('0' <= ch) && (ch <= '9')
```


- Valores
 - `"abc", "casa" :: String`
- Funções:
 - `++ :: String -> String -> String`
 - `show :: ? -> String` (overloading)

Observe que `' '`, `"""` e `" "` são diferentes

Strings - Exemplos

```
> "peixe" ++ "\n" ++ "gato"  
"peixe\ngato"
```

String e valores

```
> show (2+3)  
> show (True || False)  
  
> read "True"  
> (read "3") :: Int
```

Float e Double - Ponto flutuante

- Valores
 - 22.3435
 - 23.4e-4
- Funções:
 - `+, -, *, / :: Float -> Float -> Float`
 - `pi :: Float`
 - `ceiling, floor, round :: Float -> Int`
 - `fromIntegral :: Int -> Float`
 - `read :: String -> Float`
 - `show :: Float -> String`

- Defina a função `addEspacos` que produz um string com uma quantidade `n` de espaços.
`addEspacos :: Int -> String`
- Defina a função `paraDireita` utilizando a definição de `addEspacos` para adicionar uma quantidade `n` de espaços à esquerda de um dado `String`, movendo o mesmo para a direita.
`paraDireita :: Int -> String -> String`

Exercícios

Escreva uma função para retornar, em forma de tabela, todas as vendas da semana 0 até a semana **n**, incluindo o total e a média de vendas no período. Usem as funções definidas previamente e defina novas funções que achar necessário.

Semana	Venda
0	12
1	14
2	15
Total	41
Média	13.6667

```
imprimeTabela :: Int -> String
imprimeTabela n = cabecalho
                  ++ imprimeSemanas n
                  ++ imprimeTotal n
                  ++ imprimeMedia n
```

Estruturas de dados - Tuplas

```
intP :: (Int, Int)
```

```
intP = (33,43)
```

```
(True, 'x') :: (Bool, Char)
```

```
(34, 22, 'b') :: (Int, Int, Char)
```

```
addPair :: (Int,Int) -> Int
```

```
addPair (x,y) = x+y
```

```
shift :: ((Int,Int),Int) -> (Int,(Int,Int))
```

```
shift ((x,y),z) = (x,(y,z))
```

Exemplo - Equação do segundo grau

- $ax^2 + bx + c = 0$
 - Duas raízes, se $b^2 > 4 \cdot a \cdot c$
 - Uma raiz, se $b^2 = 4 \cdot a \cdot c$
 - Não possui raízes, se $(-b \pm \sqrt{b^2 - 4ac})/2a$
- Calculando as raízes: $(-b \pm \sqrt{b^2 - 4ac})/2a$

Resolução bottom-up

Definir funções auxiliares

```
oneRoot :: Float -> Float -> Float -> Float
```

```
oneRoot a b c = -b/(2.0*a)
```

```
twoRoots :: Float -> Float -> Float -> (Float, Float)
```

```
twoRoots a b c = (d-e, d+e)
```

```
  where
```

```
    d = -b/(2.0*a)
```

```
    e = sqrt(b^2-4.0*a*c)/(2.0*a)
```

Resolução bottom-up

Definir a função principal

```
roots :: Float -> Float -> Float -> String
roots a b c
  | b^2 == 4.0*a*c = show (oneRoot a b c)
  | b^2 > 4.0*a*c = show f ++ " " ++ show s
  | otherwise = "no roots"
  where (f,s) = twoRoots a b c
— ou
    f = fst(twoRoots a b c)
    s = snd(twoRoots a b c)
```

Sinônimos de tipos

```
type Name = String
type Age = Int
type Phone = Int
type Person = (Name, Age, Phone)
```

```
name :: Person -> Name
name (n,a,p) = n
```

- Defina a função **menorMaior** que recebe três inteiros e retorna uma tupla com o menor e o maior deles, respectivamente.
- Defina a função **ordenaTripla** que recebe uma tripla de inteiros e ordena a mesma.

Uma linha pode ser representada da seguinte forma

```
type Ponto = (Float, Float)
```

```
type Reta = (Ponto, Ponto)
```

- Defina funções que
 - retornem
 - a primeira coordenada de um ponto
 - a segunda coordenada de um ponto
 - indique se uma reta é vertical ou não ($x_1 = x_2$)

Se uma reta é dada por

$$(y - y_1)/(x - x_1) = (y_2 - y_1)/(x_2 - x_1),$$

defina uma função

`pontoY :: Float -> Reta -> Float`

que, dada uma coordenada `x` e uma reta, retorne a coordenada `y`, tal que o ponto `(x, y)` faça parte da reta.

- [1] Simon Thompson.
Haskell: the craft of functional programming.
Addison-Wesley, terceira edition, Julho 2011.