Tipos Básicos

André Santos

(a partir de *slides* elaborados por **André Santos, Sérgio Soares, Fernando Castor e Márcio Cornélio**)

Números Inteiros

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

- `+, *, -, ^, div, mod :: Int -> Int -> Int`
- `>, >=, ==, /=, <=, < :: Int -> Int -> Bool`

Integer e *Int*

- **Integer: precisão arbitrária**
- **Int: precisão fixa (*bounded*)**

Operadores e funções

- **+, *, ^, -, div, mod, abs, negate**
- **Relacionais: >, >=, ==, /=, <=, <**

Exemplo

```
> 2^3
8
> div 14 3
4
> 14 `div` 3
4
> mod 14 3
2
> 14 `mod` 3
2
```

Booleanos

- `True, False :: Bool`

- `&&, || :: Bool -> Bool -> Bool`
- `not :: Bool -> Bool`

Exemplo

```
-- ou exclusivo
eXor :: Bool -> Bool -> Bool
eXor x y = (x || y) && not (x && y)

-- outra forma de ou exclusivo
eXor :: Bool -> Bool -> Bool
eXor True  x = not x
eXor False x = x
```

Exemplo

```
-- 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
```

Caracteres

- `'a','b',...:: Char`
- `'\t', '\n', '\\', '\'', '\"' :: Char`

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

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‘)
```

Strings

- `"abc", ”casa" :: String`

- `++ :: String -> String -> String`
- `show :: ? -> String` (overloading)

' ', "" e " " são diferentes!

Exemplo

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

Strings e valores

```
> show (2+3)
> show (True || False)

> read “True”
> (read “3”)::Int
```

Números Reais – Ponto Flutuante

Float e Double

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

Exercícios

- 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 adiciconar uma quantidade **`n`** de espaços à esquerda de um dado String, movendo o mesmo para a direita.

```
paraDireita :: Int -> String -> String
```

Exercício

- 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
```

```
vendas
show
totalVendas
maxVendas
vendasNulas
addEspacos
paraDireita
```

Dica

```
imprimeTabela :: Int -> IO()
imprimeTabela n = putStr(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))
```

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
```

Exemplo: equações de segundo grau

$ax^2 + bx + c = 0.0$

- Duas raízes, se $b^2 > 4.0*a*c$
- Uma raiz, se $b^2 = 4.0*a*c$
- Não tem raízes, se $b^2 < 4.0*a*c$
- Calculando as raízes:

$(-b \pm sqrt(b^2-4ac))/2a$

Resolução bottom-up

- Definir as 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)
```

Exercícios

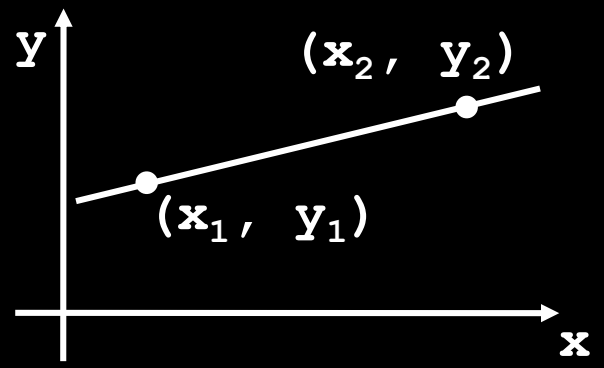
- Defina a função **`menorMaior`** que recebe três inteiros e retorna uma tupla com o menor e o maior deles, respectivamente.

- Defina a função **`ordenaTripla`** que recebe uma tripla de inteiros e ordena a mesma.

Exercícios

- 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

$x_1 == x_2$

Exercícios

- Se uma reta é dada por $\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$ 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.

  – o que acontece caso a reta seja vertical?