Haskell:
Tuplas e Funções Locais

UFRN, 2018

Tuplas

- Haskell permite a definição de tipos compostos, chamados de tuplas.
- Tuplas são construídas a partir de tipos mais simples.
- O tipo (t1,t2,...tn) consiste de tuplas com valores (v1,v2,...vn) onde v1::t1, v2::t2, vn::tn.

Tuplas

- Definições de novos tipos podem ser introduzidas pelo comando type.
- Exemplo: informações sobre uma pessoa, representadas pelo nome, telefone e idade.

```
1 type Pessoa = (String, String, Int)
2 maria :: Pessoa
3 maria = ("Maria", "32162724", 56)
```

Tuplas

- Tuplas são usadas quando é necessário agrupar dados.
  - Exemplo: uma função que deve retornar mais de um valor como resposta.
- A função abaixo deve retornar o mínimo e também o máximo valor de 3 inteiros fornecidos:

```
min3Max :: Int -> Int -> Int -> (Int,Int)
```

Tuplas

- Padrões podem ser utilizados na definição de funções sobre tuplas.
- Em vez de usar uma variável x para um argumento de tipo (Int, Int), p. e., pode-se usar um padrão como (x,y).

```
1  addPair :: (Int,Int) -> Int
2  addPair (x,y) = x + y
```

```
Main> addPair (34,32)
```

Tuplas

- Importante: as duas definições seguintes são diferentes!

```
1 addPair :: (Int,Int) -> Int
2 addPair (x,y) = x + y
3
4 addTwo :: Int -> Int -> Int
5 addTwo a b = a + b
```

```
Main> addPair (34,32) + addTwo 34 32
```

Tuplas

- Padrões podem ser aninhados...

```
1 shift :: ((Int,Int),Int) -> (Int,(Int,Int))
2 shift ((a,b),c) = (a,(b,c))
```

```
Main> shift ((1,2),3)
(1,(2,3))
```

Tuplas

- Funções que extraem partes de uma tupla podem ser especificadas usando casamento de padrões.

```
1 type Pessoa = (String, String, Int)
2 nome :: Pessoa -> String
3 fone :: Pessoa -> String
4 idade :: Pessoa -> Int
5
6 nome (n,f,i) = n
7 fone (n,f,i) = f
8 idade (n,f,i) = i
```

```
1   type Pessoa = (String, String, Int)
2   nome :: Pessoa -> String
3   fone :: Pessoa -> String
4   idade :: Pessoa -> Int
5   nome (n,f,i) = n
6   fone (n,f,i) = f
7   idade (n,f,i) = i
8   maria :: Pessoa
9   maria = ("Maria", "32162724", 56)
```

```
Main> nome maria
"Maria"
Main> idade maria
56
```

Tuplas

- Haskell possui funções de extração pré-definidas para tuplas de 2 elementos:

```
1 fst (x,y) = x
2 snd (x,y) = y
```

```
Main> fst (1,2)
1
Main> snd (1,2)
2
```

Definições Locais

- Cada equação pode ser seguida por uma lista de definições locais, escritas após a palavra-chave `where`.

```
1 sumSquares :: Int -> Int -> Int
2 sumSquares m n
3    = sqM + sqN
4      where
5      sqM = m * m
6      sqN = n * n
```

Definições Locais

- Formato geral:

```
fun p1 p2 ... pn
  | g1            = e1
  ...
  | otherwise = er
   where
   v1 = r1
   v2 a1...ak = r2
   ...
```

Indentação continua sendo importante!

Definições locais podem incluir funções!

Definições Locais

```
1 sumSquares :: Int -> Int -> Int
2 sumSquares m n
3   = sqM + sqN
4     where
5     sqM = m * m
6     sqN = n * n
```

```
1 sumSquares :: Int -> Int -> Int
2 sumSquares m n
3   = sq m + sq n
4     where
5     sq x = x * x
```

Definições Locais

- É possível fazer também definições locais a expressões, usando a palavra-chave let.

```
Main> let x = 5 in x^2 + 2*x - 4
31
```

- Duas ou mais definições, devem ser separadas por ";" .

```
Main> let x = 5; y = 4 in x^2 + 2*x - y
31
```

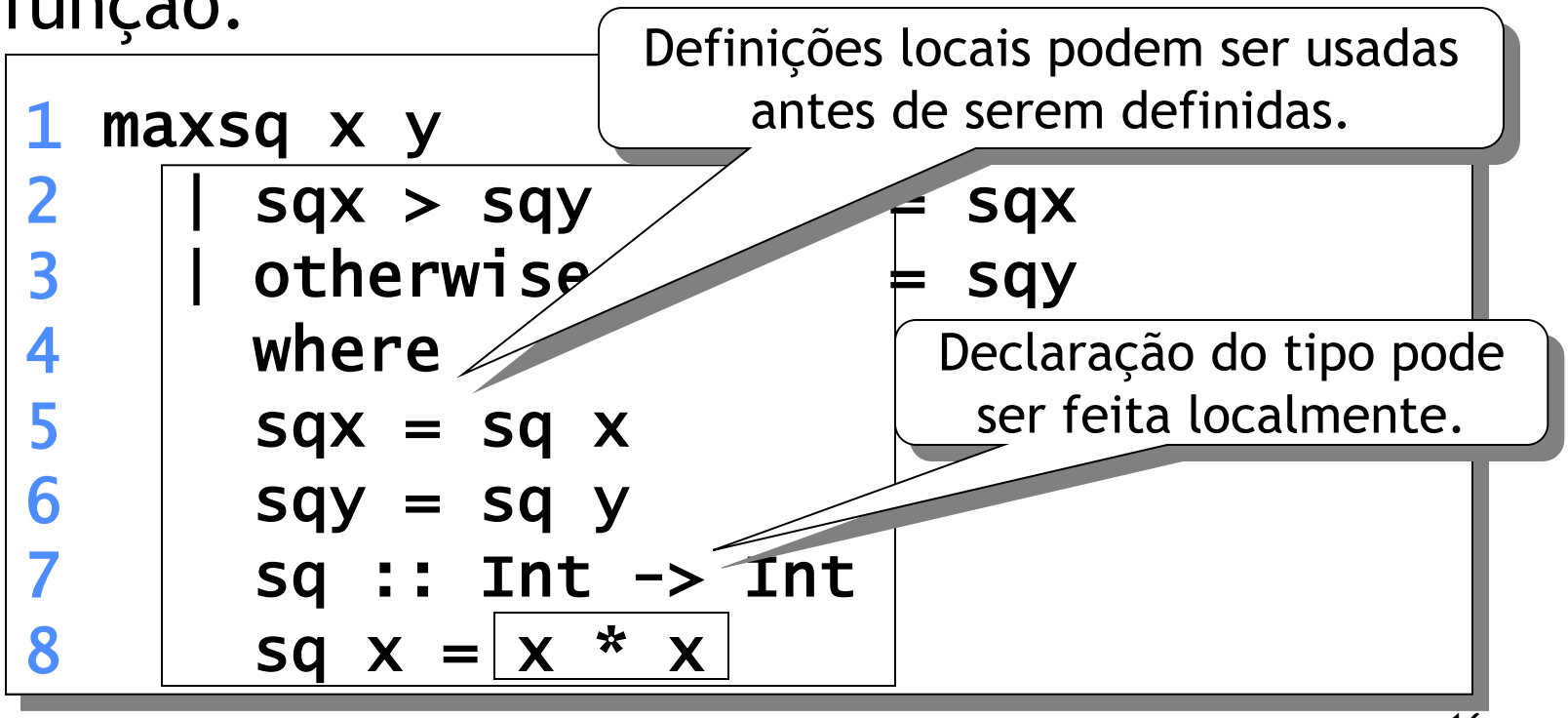
Regras de Escopo

- As definições no primeiro nível de indentação de um script haskell são visíveis em todo o programa.
- A ordem de definição não importa.

```
1 isOdd, isEven :: Int -> Bool
2
3 isOdd 0 = False
4 isOdd n = isEven (n-1)
5 isEven 0 = True
6 isEven n = isOdd (n-1)
```

Regras de Escopo

- Cláusulas where têm escopo mais reduzido, assim como as variáveis na definição de uma função.



Exemplos

- Problema: construir uma função

  max3oc :: Int -> Int -> Int -> (Int,Int)

  que retorna o máximo de 3 inteiros, e também o número de vezes que esse valor ocorre entre os 3.
- Exemplo de execução:

```
Main> max3oc 10 30 20
(30,1)
Main> max3oc 15 12 15
(15,2)
```

Exemplos

- Solução: achar o máximo e contar quantas vezes ocorre entre os valores.

```
1 max3oc :: Int -> Int -> Int ->
2 (Int,Int)
3
4 max3oc x y z
5   = (max, igCont)
6     where
7     max    = maxi3 x y z
8     igCont = iguais3 max x y z
```

Exemplos

- Para achar o máximo entre 3 valores, pode-se usar a função que encontra o máximo entre 2 valores.

```
1 maxi :: Int -> Int -> Int
2 maxi x y
3    | x >= y       = x
4    | otherwise    = y
5
6 maxi3 :: Int -> Int -> Int -> Int
7 maxi3 x y z = maxi x (maxi y z)
```

Exemplos

```
1 iguais3 v a b c
2   = va + vb + vc
3     where
4     va = if a==v then 1 else 0
5     vb = if b==v then 1 else 0
6     vc = if c==v then 1 else 0
```

```
1 iguais3 v a b c
2   = igv a + igv b + igv c
3     where
4     igv :: Int -> Int
5     igv x = if x==v then 1 else 0
```