Haskell:
Listas e Casamento de Padrões

UFRN, 2018

Padrões em Haskell

- Constantes e variáveis.
- Tuplas de padrões (p1,p2,...) onde p1,p2,... também são padrões.
- Listas de padrões (p1:p2) onde p1 e p2 também são padrões.
- Padrão “_” .

Padrões em Haskell

- Quando um argumento *a* casa com um padrão *p*?
  - se *p* é uma constante e *p*=*a*;
  - se *p* é uma variável *x*, sendo que *a* será associado a *x*;
  - se *p* é uma tupla (*p1*,*p2*,...), a=(*a1*,*a2*,...) e cada componente de *a* casa com o respectivo componente de *p*.
  - se *p* é uma lista (*p1*:*p2*), *a* é uma lista não vazia e a cabeça de *a* casa com *p1* e sua cauda casa com *p2*.
  - se o padrão p for "_".

Padrões em Haskell

- Restrição importante: não é permitido usar um mesmo nome mais de uma vez dentro do mesmo padrão!

```
1 func :: Int -> Int -> Bool
2 func x x = True
3 func x y = False
```

ERRO - variável repetida em padrão!!!

Construtor case

```
1  numToString :: Int -> String
2  numToString x =
3    case x of
4    0 -> "zero"
5    1 -> "um"
6    2 -> "dois"
7    3 -> "três"
8    4 -> "quatro"
9    5 -> "cinco"
10   otherwise -> ""
```

Diferentes Definições para Funções

- Definir uma função

  sumPairs :: [(Int,Int)] -> Int

  que calcula a soma total dos pares de inteiros de uma lista.

```
Main> sumPairs [(2,3),(5,-1),(3,3)]
15
```

Caso Base

- Função

  sumPairs :: [(Int,Int)] -> Int

  que calcula a soma total dos pares de inteiros de uma lista.

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 ... ???
```

Possíveis Soluções

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = fst a +snd a +sumPairs x
```

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = c + d + sumPairs x
4                  where
5                  c = fst a
6                  d = snd a
```

Possíveis Soluções

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = c + d + sumPairs x
4                  where
5                  c = fst a
6                  d = snd a
```

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = c + d + sumPairs x
4                  where (c,d) = a
```

Possíveis Soluções

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = c + d + sumPairs x
4                  where (c,d) = a
```

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs ((c,d):x) = c + d + sumPairs x
```

Possíveis Soluções

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs ((c,d):x) = c + d + sumPairs x
```

```
1 sumPairs :: [(Int,Int)] -> Int
2 sumPairs [] = 0
3 sumPairs (a:x) = sumPair a + sumPairs x
4 sumPair  (c,d) = c + d
```

Diferentes Definições para Funções

- Definir uma função

  mzip:: [Int] -> [Int] -> [(Int,Int)]

  que agrupa duas listas de inteiros em uma única lista de pares.

```
Main> mzip [1,2,3] [4,6,1]
[(1,4), (2,6), (3,1)]
Main> mzip [1,3] [1,5,8]
[(1,1), (3,5)]
Main> mzip [1,3] [2]
[(1,2)]
```

Possíveis Soluções

```
1 mzip :: [Int] -> [Int] -> [(Int,Int)]
2 mzip (a:x) (b:y) = (a,b) : mzip x y
3 mzip x [] = []
4 mzip [] y = []
```

```
1 mzip :: [Int] -> [Int] -> [(Int,Int)]
2 mzip (a:x) (b:y) = (a,b) : mzip x y
3 mzip _ _ = []
```

List Comprehensions

- Maneira alternativa para construir e manipular listas.
- Útil para construir listas baseadas em outras listas.
- Exemplo: se a lista x é [1,3,8] então [2*a | a <- x] será [2,6,16]

List Comprehensions

- O exemplo acima pode ser lido como: construa uma lista onde cada elemento tem a forma `2*a`, sendo que `a` vem da lista `x`.
- Uma list comprehension `a <- x` é chamada gerador.

List Comprehensions

- Do lado esquerdo de "<- " sempre se coloca um padrão (ex:variável), e do lado direito, uma lista.
- Geradores podem ser combinados com testes:

```
[ 2*a | a <- x, isEven a, a>3 ]
```

se x é [2,6,3,8], produz como resposta:

[12,16]

(supor isEven retorna True se argumento for par).

Exemplos

- Exemplo:

```
1 addPairs :: [(Int,Int)] -> [Int]
2 addPairs pL = [ a+b | (a,b) <- pL ]
```

```
Main>addPairs [(2,3), (4,3), (1,7)]
[5,7,8]
```

Exemplos

- Exemplo com teste:

```
1 newAddPairs :: [(Int,Int)] -> [Int]
2 newAddPairs pL = [ a+b | (a,b) <- pL, a<b ]
```

```
Main> newAddPairs [(2,3), (4,3), (1,7)]
[5,8]
```

Exemplos

- Redefinindo exemplos anteriores:

```
1 double :: [Int] -> [Int]
2 double [] = []
3 double (a:x) = (2*a) : double x
```

```
1 double :: [Int] -> [Int]
2 double l = [ 2*a | a <- l ]
```

Exemplos

- Redefinindo exemplos anteriores:

```
1 digits :: String -> String
2 digits [] = []
3 digits (a:x)
4   | isDigit a  = a : digits x
5   | otherwise  = digits x
```

```
1 digits :: String -> String
2 digits st = [ ch | ch <- st,
  isDigit ch ]
```