sumList [2,3,4,5]
= 2 + sumList [3,4,5]
= 2 + (3 + sumList [4,5])
= 2 + (3 + (4 + sumList [5]))
= 2 + (3 + (4 + (5 + sumList [])))
= 2 + (3 + (4 + (5 + 0)))
= 14

Programação Funcional

Tuplas e Definições de Tipos

Matheus Carvalho Viana
matheuscviana@ufsj.edu.br

Dcomp
Departamento de Ciência da Computação
UFSJ
Universidade Federal de São João del-Rei

Introdução

Todas as linguagens de programação permitem aglomerar diversos valores em uma única estrutura.

As listas do Haskell, assim como os vetores das linguagens imperativas, são uma forma de aglomerar valores do mesmo tipo.

Também como as linguagens imperativas, Haskell permite ao programador agrupar valores de tipos diferentes em uma única estrutura.

Essa estrutura, parecido com o *struct* do C/C++, é chamada de **tupla** no Haskell.

Tupla

Representa um tipo composto de um conjunto finito de tipos já existentes.

Sua sintaxe é no cabeçalho das funções é:

```
( Tipo1, Tipo2, ..., TipoN )
```

Exemplos:

- `(Int, String)` é uma tupla com um número inteiro e uma string;
- `([Char], Float, Int)` é uma tupla com uma string, um número float e um número inteiro;
- `(String, (Int, Int, Int))` é uma tupla com uma String e uma tupla com três inteiros.

Tupla

Um valor de um tipo tupla é escrito de forma semelhante à sua declaração, porém substituem-se os nomes dos tipos internos por valores desses tipos.

Exemplos:

- **`(5, "JOAO")`** é um valor válido para uma tupla do tipo `(Int, String)`;
- **`("ONIX 1.4", 56.000, 2017)`** é um valor válido para uma tupla do tipo `([Char], Float, Int)`;
- **`("Validade", (05, 04, 2017))`** é um valor válido para uma tupla do tipo `(String, (Int, Int, Int))`.

Casamento de Padrões com Tupla

Em funções que recebem uma tupla como entrada, o casamento de padrões pode ser feito da seguinte forma:

- $(v_1, v_2, ..., v_n)$ – em que cada v é uma variável que representa um valor da tupla. A quantidade de variável deve ser o mesmo de tipos dentro da tupla;

- uma variável – que representa a toda a tupla.

Função com Tupla

A função `distanciaPontos` recebe duas tuplas que representam pontos e retorna a distância entre eles.

```
distanciaPontos :: (Float, Float) -> (Float, Float) -> Float
distanciaPontos (xa, ya) (xb, yb) =
  sqrt ( (xb-xa)**2 + (yb-ya)**2 )
```

A função `shift` inverte a posição da tupla interna.

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

Funções podem ser criadas para extrair dados das tuplas.

```
nome :: (String, String, Int) -> String
nome (n, _, _) = n
idade :: (String, String, Int) -> Int
idade (_, _, i) = i
```

Função com Tupla

Deve-se ter algum cuidado com os tipos de dados que, em algumas situações, podem conduzir a erros.

Por exemplo, as funções abaixo são diferentes:

```
somaPar :: (Int, Int) -> Int
somaPar (a, b) = a + b

somaDois :: Int -> Int -> Int
somaDois a b = a + b
```

Apesar dos resultados das aplicações das funções **somaPar** e **somaDois** serem os mesmos, elas são distintas.

A função **somaPar** requer apenas um argumento, neste caso uma tupla, enquanto a função **somaDois** requer dois argumentos do tipo inteiro.

Funções para Manipulação de Tuplas

O Prelude fornece duas funções bem simples para manipulação de tuplas com dois elementos.

fst: retorna o primeiro elemento de uma tupla com dois elementos.

```
Prelude> fst (2, 5)
2
Prelude> fst (True, "boo")
True
```

snd: retorna o segundo elemento de uma tupla com dois elementos.

```
Prelude> snd (5, "Hello")
"Hello"
```

Tuplas e Listas

Elementos de uma tupla podem ser listas.

```
somaNotas :: (String, [Float]) -> Float
somaNotas (_, []) = 0
somaNotas (n, (a:b)) = a + somaNotas (n,b)
```

```
mediaNotas :: (String, [Float]) -> String
mediaNotas (nome, notas) =
  "A media das notas de " ++ nome ++ " eh " ++
  show ((sum notas) / fromIntegral (length notas))
```

Tuplas e Listas

Assim como pode haver listas de tuplas.

```
pontoMaisAlto :: [(Float,Float)] -> (Float,Float)
pontoMaisAlto [(x,y)] = (x,y)
pontoMaisAlto ((x,y):r)
  | max y (cordY pm) == y = (x,y)
  | otherwise = pm
  where
    pm = pontoMaisAlto r

cordY :: (Float,Float) -> Float
cordY (_,y) = y
```

Definições de Tipos

Quando possuem muitos elementos de vários tipos, as tuplas são complicadas de escrever e ler. Por exemplo:

```
getEndereco :: (String, (String, Int, String, String), Int)
  -> (String, Int, String, String)
getEndereco (_, e, _) = e
```

Isso pode induzir o programador a erros, principalmente no cabeçalho e no casamento de padrões das funções.

A solução para isso é o uso de **Definições de Tipo.**

Definições de Tipos

Por meio dela é possível definir um nome para uma tupla, uma lista, ou até mesmo sinônimos para tipos existentes.

Isso é feito por meio da palavra reservada **`type`**.

O sinônimo pode aparecer em qualquer lugar que o tipo original aparece.

```
type Ponto = (Float, Float)
type Endereco = (String, Int, String, String)
type Nome = String
type Idade = Int
type Pessoa = (Nome, Endereco, Idade)
type Pessoas = [Pessoa]
```

Definições de Tipos

Ao invés de escrever:

```
getEndereco :: (String, (String, Int, String, String), Int)
  -> (String, Int, String, String)
getEndereco (_, e, _) = e
```

É mais fácil, mais rápido e mais legível escrever:

```
getEndereco :: (Nome, Endereco, Idade) -> Endereco
getEndereco (_, e, _) = e
```

E pode ficar melhor:

```
getEndereco :: Pessoa -> Endereco
getEndereco (_, e, _) = e
```

Exercícios

1. Use uma combinação de fst e snd para extrair o 4 da tupla `(("Hello", (4, True)), 15.5)`.
2. Crie a tupla Trapézio, com teto, piso e altura. Crie uma função que calcula a sua área com a fórmula (teto + piso) * altura / 2.
3. Crie uma função que recebe uma lista do tipo Ponto e retorna um ponto, cuja coordenada x é a soma de todas as coordenadas x dos pontos da lista e o mesmo acontece com a coordenada y.
4. Crie uma função que retorna uma o primeiro e o último elemento de uma lista.
5. Crie as tuplas Tabuleiro (Char,Int) e Jogada (Jogador,Tabuleiro), sendo Jogador um sinônimo para String. Crie uma função que recebe uma lista de jogadas e um jogador e retorne a quantidade de jogadas do jogador informado.
6. Crie uma função que recebe uma lista do tipo Triângulo (Float, Float, Float) e retorna uma lista do mesmo tipo contendo somente triângulos equilateros.