

Programação Funcional

Uma introdução com Haskell

Lucas Albertins de Lima

Departamento de Computação - UFRPE

Programação baseada em definições

```
answer :: Int
```

```
answer = 42
```

```
greater :: Bool
```

```
greater = (answer > 71)
```

```
yes :: Bool
```

```
yes = True
```

Definição de funções

```
square :: Int -> Int
```

```
square x = x * x
```

```
allEqual :: Int -> Int -> Int -> Bool
```

```
allEqual n m p = (n == m) && (m == p)
```

```
maxi :: Int -> Int -> Int
```

```
maxi n m | n >= m = n
```

```
          | otherwise = m
```

Aplicação de Funções

`square 5` — = 25

`square(5)` — = 25

`allEqual 1 2 3` — = *False*

`allEqual(1,2,3)` — *ERRO!!!*

`allEqual(1) (2) (3)` — = *False*

`maxi 24 645` — = 645

Exemplo

$$\begin{aligned}\text{addD } a \ b &= 2 * (a+b) \\ &= 2 * (b+a) = \text{addD } b \ a\end{aligned}$$

Válida para quaisquer argumentos a e b

Não seria válida em linguagens imperativas, com variáveis globais ...

Em uma linguagem imperativa

```
int b = 1;  
...  
int f (int x) {  
    b = x;  
    return (5)  
}
```

Então ...

addD (f 3) b == addD b (f 3) ?

Exemplo

Em um sistema de controle de vendas

- as vendas semanais são dadas pela função `vendas :: Int → Int`
- total de vendas da semana 0 à semana n ?
 $\text{vendas } 0 + \text{vendas } 1 + \dots + \text{vendas } (n-1) + \text{vendas } n$

Definição de `totalVendas`

`totalVendas :: Int → Int`

`totalVendas n`

| `n == 0` = `vendas 0`

| **`otherwise`** = `totalVendas (n-1) + vendas n`

- Definir **caso base**, isto é, valor para **fun** o
- Definir valor para **fun n**, usando o valor de **fun** ($n-1$). Este é o **caso recursivo**

```
maxVendas :: Int -> Int
```

```
maxVendas n
```

```
  | n == 0      = vendas o
```

```
  | otherwise = maxi (maxVendas (n-1))  
                    (vendas n)
```


Casamento de padrões

Permite usar padrões no lugar de variáveis, na definição de funções

```
maxVendas :: Int -> Int
```

```
maxVendas 0 = vendas 0
```

```
maxVendas n = maxi (maxVendas (n-1)) (vendas n)
```

```
totalVendas :: Int -> Int
```

```
totalVendas 0 = vendas 0
```

```
totalVendas n = totalVendas (n-1) + vendas n
```

Casamento de Padrões

myNot :: **Bool** → **Bool**

myNot **True** = **False**

myNot **False** = **True**

myOr :: **Bool** → **Bool** → **Bool**

myOr **True** **x** = **True**

myOr **False** **x** = **x**

myAnd :: **Bool** → **Bool** → **Bool**

myAnd **False** **x** = **False**

myAnd **True** **x** = **x**

Regras para padrões

- Todos os padrões (esquerda) devem ter **tipos compatíveis**
 - Não necessariamente iguais
- Os casos devem ser exaustivos
 - não é obrigatório, mas pode levar a funções parciais
- Não deve haver ambiguidade
 - ordem dos padrões usada para resolver conflitos

$$\frac{f(n+1)}{f(n+1)} = (f(n) + 1)$$

$$\begin{matrix} 2 & + & 3 \\ (+) & 2 & 3 \end{matrix}$$

$$\begin{matrix} \text{maxi} & 2 & 4 \\ 2 & \text{'maxi'} & 4 \end{matrix}$$

Erros comuns

```
square x =          x
*x
parse error on input  '**'
```

```
answer = 42; newline = '\n'  —OK
```

```
funny x = x +
1
parse error (possibly incorrect indentation)
```

```
Funny x = x+1
Not in scope: data constructor 'Funny'
```

Defina as seguintes funções

- `fat :: Int -> Int` que calcula o fatorial de um inteiro dado como argumento
- `all4Equal :: Int -> Int -> Int -> Int -> Bool` que compara se quatro valores inteiros são iguais
- `equalCount :: Int -> Int -> Int -> Int` que retorna quantos argumentos são iguais

Definições locais

```
sumSquares :: Int -> Int -> Int
```

```
sumSquares x y = sqX + sqY  
  where sqX = x * x  
        sqY = y * y
```

```
sumSquares x y = sq x + sq y  
  where sq z = z * z
```

```
sumSquares x y = let sqX = x * x  
                  sqY = y * y  
                  in sqX + sqY
```

Definições locais (cont.)

```
maxThreeOccurs :: Int -> Int -> Int -> (Int, Int)
maxThreeOccurs m n p = (mx, eqCount)
  where mx = maxiThree m n p
        eqCount = equalCount mx m n p
```

- `let <definicoes> in <expressao>`
- `<definicoes> where <definicoes>`

Exercícios

1. Defina uma função que, dado um valor inteiro s e um número de semanas n , retorna quantas semanas de 0 a n tiveram vendas iguais a s .
2. Crie uma função que retorna a média de três inteiros.
3. Usando a função anterior, defina uma função que verifica quantos dos números estão acima da média.

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