

Haskell: Tipos Básicos e Programas Simples

UFRN, 2018

Números Inteiros

- Na linguagem Haskell, o tipo de dados `Int` representa os números inteiros.
- O valor máximo de `Int` é 2.147.483.647.
- O tipo `Integer` também representa números inteiros, sem restrição de tamanho.
- A vantagem de usar `Int` é um possível ganho em eficiência (depende da implementação).

Números Inteiros

- Operações aritméticas: +, -, *, /, div, mod, abs.
- Comparação: >, <, >=, <=, ==, /=.

```
Main> 5 + 3
```

```
8
```

```
Main> 10 <= 20
```

```
True
```

```
Main> 10 / 3
```

```
3.33333
```

Números Inteiros

- Funções com 2 parâmetros podem ser usada na forma infixa, usando ``.
- Operadores entre parênteses são tratados como funções.

```
Main> div 22 5
```

```
4
```

```
Main> mod 12 5
```

```
2
```

```
Main> 12 `mod` 5
```

```
2
```

```
Main> (+) 5 3
```

```
8
```

Programas com Inteiros

- Seja a função `vendas :: Int -> Int`.
- A função `vendas` retorna o número de unidades vendidas, de um determinado produto, se for fornecido o número do dia desejado.
- Os dias são numerados como 0,1,2 ...

Programas com Inteiros

```
Main> vendas 3
25
Main> vendas 0
12
```

```
1 vendas :: Int -> Int
2 vendas x
3   | x == 0      = 12
4   | x == 1      = 20
5   | x == 2      = 18
6   | x == 3      = 25
7   | otherwise   = 0
```



**Uma definição
para vendas.**

Programas com Inteiros

- Problema: definir $\text{totalvendas} :: \text{Int} \rightarrow \text{Int}$, função que retorna o total de vendas até um determinado dia.

$\text{totalvendas } 2 = \text{vendas } 0 + \text{vendas } 1 + \text{vendas } 2$

$\text{totalvendas } n = \text{vendas } 0 + \dots + \text{vendas } (n-1) + \text{vendas } n$

- Para $n=0$: $\text{totalvendas } n = \text{vendas } 0$
- Para $n>0$: $\text{totalvendas } n = \text{totalvendas } (n-1) + \text{vendas } n$

Programas com Inteiros

```
8 totalvendas :: Int -> Int
1 totalvendas n
2   | n == 0      = vendas 0
3   | otherwise   = vendas n +
4                   totalvendas (n-1)
```

```
1 vendas :: Int -> Int
2 vendas x
3   | x == 0      = 12
4   | x == 1      = 20
5   | x == 2      = 18
6   | x == 3      = 25
7   | otherwise   = 0
```

```
Main>
totalvendas 0
12
Main>
totalvendas 2
50
```


Programas com Inteiros

```
8 totalvendas :: Int -> Int
1 totalvendas n
2   | n == 0      = vendas 0
3   | otherwise   = vendas n +
4                   totalvendas (n-1)
```

```
totalvendas 2
= vendas 2 + totalvendas 1
= vendas 2 + (vendas 1 + totalvendas 0)
= vendas 2 + (vendas 1 + vendas 0)
```

Programas com Inteiros

- Problema: definir `maxvendas :: Int -> Int`, função que retorna o valor máximo de unidades vendidas em um dia, em determinado período.

`maxvendas n = máximo em vendas 0, ..., vendas n`

- Para `n=0`: `maxvendas n = vendas 0`
- Para `n>0`: `maxvendas n = máximo entre
 maxvendas(n-1) e vendas n`

Programas com Inteiros

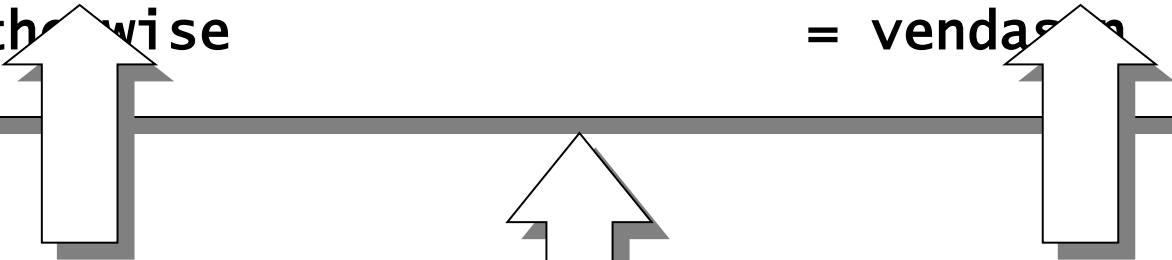
```
12 maxvendas :: Int -> Int
13 maxvendas n
14   | n == 0           = vendas 0
15   | maxvendas(n-1) >= vendas n = maxvendas(n-1)
16   | otherwise        = vendas n
```

```
1 vendas :: Int -> Int
2 vendas x
3   | x == 0      = 12
4   | x == 1      = 20
5   | x == 2      = 18
6   | x == 3      = 25
7   | otherwise   = 0
```

```
maxvendas 2
?? maxvendas 1 >= vendas 2
?? maxvendas 0 >= vendas 1
?? vendas 0 >= vendas 1
?? 12 >= 20
= 20
?? 20 >= 18
= 20
```

Programas com Inteiros

```
12 maxvendas :: Int -> Int
13 maxvendas n
14   | n == 0                = vendas 0
15   | maxvendas(n-1) >= vendas n = maxvendas(n-1)
16   | otherwise             = vendas n
```



Ineficiência:
maxvendas(n-1) é
calculada 2 vezes!

Programas com Inteiros

- Solução alternativa:

```
12 maxvendas :: Int -> Int
13 maxvendas n
14   | n == 0      = vendas 0
15   | otherwise = maxi (maxvendas(n-1))
                        (vendas n)
```

```
16 maxi :: Int -> Int -> Int
17 maxi m n
18   | m >= n      = m
19   | otherwise    = n
```

Sintaxe

- **Definições e layout:**
 - Em Haskell, o layout de um programa é utilizado para determinar quando termina a definição de uma função e quando começa a próxima.
 - Uma definição termina com o primeiro elemento no mesmo nível de indentação (ou à esquerda deste).

Sintaxe

**f x = x + x
+ x**

+2

g y z = ...

Sintaxe

- Definições e layout:
 - Outra forma para determinar o fim de uma definição é utilizar um símbolo terminador: ";".
 - Exemplo:

```
f x = x + x;   g x y = x * y
```


Sintaxe

- Erro comum:

```
f x = x +  
1
```

- Mensagem de erro:

```
ERROR ... Syntax error in expression (unexpected  
";", possibly due to bad layout)
```

Sintaxe

- Layout recomendado:

```
f v1 v2 ... vn
  | g1          = e1
  | g2          = e2
  ...
  | otherwise   = er   (ou | gr = er)
```

Sintaxe

- Quando expressões ou guardas são muito longas:

```
f v1 v2 ... vn
  | uma guarda muito longa que
    pode ocupar mais de uma linha
    = uma expressão muito longa
  | g2      = e2
  ...
```

Sintaxe

- Nomes em Haskell

- Começam com uma letra, seguida (opcionalmente) por letras, dígitos, "_" e "'";
- Começando com minúscula: funções e variáveis;
- Começando com maiúscula: nomes de módulos, nomes de tipos e construtores de tipo (ex: True e False).

- Palavras reservadas:

```
case class data default deriving else hiding if  
import in infix infixl infixr instance interface let  
module of renaming then to type where
```

Sintaxe

- **Comentários de linha:**
 - Símbolo usado: `--`
 - O texto à direita é considerado comentário.
- **Comentários aninhados:**
 - Símbolos usados: `{- -}`
 - Podem se estender por várias linhas.
 - Agem como parênteses, podendo existir comentários aninhados.