A Biblioteca Prelude

Programação Funcional em Haskell

Prof. Rodrigo Ribeiro

Prelude — (I)

- Todo módulo em Haskell importa automaticamente a biblioteca Prelude.
- O que contém essa biblioteca?
  - Definições de tipos dados "frequentemente" usados
  - Definições de funções úteis.

```
module Main where
```

Prelude — (II)

- Já vimos diversas funções definidas na Prelude:
  - map, foldr, +, *...
- Objetivos:
  - Apresentar outras funções da biblioteca Prelude.

Prelude — (III)

- Função filter

```
filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (x : xs)
        | p x = x : filter p xs
        | otherwise = filter xs

*Main> filter (> 3) [1..10]
[4,5,6,7,8,9,10]
```

Prelude — (IV)

- Função filter

```
filter :: (a -> Bool) -> [a] -> [a]
filter p xs = [x | x <- xs , p x]
```

Prelude — (V)

- Função filter

```
filter :: (a -> Bool) -> [a] -> [a]
filter p = foldr (\ x ac -> if p x then x : ac else ac) []
```

Prelude — (VI)

- Função zip:

```
zip :: [a] -> [b] -> [(a,b)]
zip [] _ = []
zip _ [] = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys

*Main> zip [1,2,3] [4..]
[(1,4),(2,5),(3,6)]
```

Prelude — (VII)

- Função unzip:

```
unzip :: [(a,b)] -> ([a],[b])
unzip [] = ([],[])
unzip ((x,y):xs) = (x:xs',y:ys')
                   where
                     (xs',ys') = unzip xs

*Main> unzip [(1,4),(2,5),(3,6)]
([1,2,3],[4,5,6])
```

Prelude — (VIII)

- Função zipWith

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f [] _ = []
zipWith f _ [] = []
zipWith f (x : xs) (y : ys) = f x y : zipWith xs ys

*Main> zipWith (+) [1..10] [1..5]
[2,4,6,8,10]
```

Prelude — (IX)

- Função concat

```
concat :: [[a]] -> [a]
concat = foldr (++) []

*Main> concat [[1,2,3],[4,5],[1,2],[]]
[1,2,3,4,5,1,2]
```

Prelude — (X)

- Função take

```
take :: Int -> [a] -> [a]
take 0 _ = []
take n (x:xs) = x : take (n - 1) xs
take _ _ = Prelude.error "error..."

*Main> take 10 [1..]
[1,2,3,4,5,6,7,8,9,10]
```

Prelude — (XI)

- Função drop

```
drop :: Int -> [a] -> [a]
drop 0 xs = xs
drop n (x:xs) = drop (n - 1) xs
drop _ [] = []

*Main> drop 10 [1..20]
[11,12,13,14,15,16,17,18,19,20]
```

Prelude — (XII)

- Função takeWhile

```
takeWhile :: (a -> Bool) -> [a] -> [a]
takeWhile p [] = []
takeWhile p (x:xs)
      = if p x then x : takeWhile p xs else []

*Main> takeWhile (< 5) [1..10]
[1,2,3,4]
```

Prelude — (XIII)

- Função dropWhile

```
dropWhile :: (a -> Bool) -> [a] -> [a]
dropWhile _ []          =  []
dropWhile p xs@(x:xs')
            | p x       =  dropWhile p xs'
            | otherwise =  xs

*Main> dropWhile (< 5) [1..10]
[5,6,7,8,9,10]
```

Prelude — (XIV)

- Função scanl

```
scanl :: (a -> b -> a) -> a -> [b] -> [a]
scanl f q ls =  q : (case ls of
                       []   -> []
                       x:xs -> scanl f (f q x) xs)
```

```
*Main> scanl (+) 0 [3,5,2,1]
[0,3,8,10,11]
```

Prelude — (XV)

- Função scanr

```
scanr :: (a -> b -> b) -> b -> [a] -> [b]
scanr _ q0 [] =  [q0]
scanr f q0 (x:xs) =  f x q : qs
                     where qs@(q:_) = scanr f q0 xs

*Main> scanr (+) 0 [3,5,2,1]
[11,8,3,1,0]
```

Prelude — (XVI)

- Hoogle
  - Máquina de busca para a bibliotecas em Haskell
  - Permite a busca por nomes ou por tipos.
  - Muito útil para evitar “reinventar a roda”.
- http://www.haskell.org/hoogle/
- É possível instalar uma cópia local com: cabal install hoogle

Prelude — (XVII)

- Regra de Horner
  - Algoritmo eficiente para avaliação de polinômios.
  - Representação de polinômios como uma lista de coeficientes em ordem crescente de expoentes.
- Exemplo
  - O polinômio $x^3 + 4x^2 + 3$ pode ser representado pela seguinte lista:

```
p :: [Int]
p = [1,4,0,3]
```

Prelude — (XVIII)

- O algoritmo de Horner para avaliar polinômios.
  - Dado um polinômio $p(x) = \sum_{i=1}^{n} a_i x^i$, e um valor x_0, temos que:
    $p(x_0) = a_0 + x_0 \times (a_1 + x_0 \times (a_2 + ... + x_0 \times (a_{n-1} + x_0 \times a_n)...))$
- Desenvolva a função eval, que a partir de um inteiro e uma lista que representa um polinômio, retorna o valor deste polinômio para o inteiro em questão.

Prelude — (XIX)

- Solução simples

```
eval' :: Int -> [Int] -> Int
eval' x = fst . worker (0,1) x

worker :: (Int,Int) -> Int -> [Int] -> (Int,Int)
worker p x [] = p
worker (r, vn) x (t:ts) = (t * vn + r, vn * x)
```

Prelude — (XX)

- Exemplo

```
p :: [Int]
p = [3,-2,0,1]

test :: Bool
test = eval' 1 p == 3
```

Prelude — (XXII)

- Como melhorar a solução apresentada?
  - Primeiro, precisamos identificar o padrão de recursão utilizado e tentar utilizar alguma função da biblioteca Prelude que representa esse padrão.
- Qual seria esse padrão de recursão?

Prelude — (XXIII)

- Solução elegante

```
eval :: Int -> [Int] -> Int
eval x = fst . foldr step  (0,1)
         where
           step t (r,vn) = (t * vn + r, vn * x)
```

Prelude — (XXIV)

- Segunda lista de exercícios
  - Já disponível no repositório no github! (Junto com o material desse encontro).
- Recomendo a leitura dos 4 primeiros capítulos do livro Real World Haskell para resolver essa lista.
- Livro Real World Haskell pode ser lido gratuitamente em :
  - http://book.realworldhaskell.org/read/
  - Existem exemplares desse livro na biblioteca.