Listas em Haskell

Profa. Juliana Kaizer Vizzotto

juvizzotto@gmail.com

Disciplina de Programação Funcional - Juliana Kaizer Vizzotto

Listas

- Para qualquer tipo `t`, existe um tipo lista de itens `t`, chamada de `[t]`. Por exemplo:

```
['a','b','c'] :: [Char]
"abc"         :: [Char]
[1,2,3,4]     :: [Int]
```

- Uma `String` é somente um sinônimo para listas de caracteres

```
type Stting = [Char]
"abc" :: String
```

- Listas podem conter qualquer tipo de elementos, incluindo, tuplas, funções e outras listas:

```
[tabela,tabela] :: [Int -> String]
[(1,2),(3,4),(5,4)]  :: [(Int, Int)]
[[1,2,3,4],[], [4,5], [6,7]] :: [[Int]]
```

Listas

- A lista vazia representada por [] é um tipo especial de lista que contém nenhum elemento e representa a lista vazia para qualquer tipo.

```
[] :: Int
[] :: Float
[] :: [Int -> Int]
```

- Existe uma outra maneira de se descrever listas: `[n .. m]` é a lista `[n, n+1, ..., m]`

```
> [2 .. 6]
[2,3,4,5,6]
> [3.1 .. 7]
[3.1, 4.1, 5.1,6.1,7.1]
```

Listas

- A lista `[n, p .. m]` é a lista de números de `n` até `p`, sendo que cada elemento aumenta em passos `p - n`

```
> [1,2 .. 5]
[1,2,3,4,5]
> [6,5 .. 3]
[6,5,4,3]
```

Construindo Listas

- Uma lista em Haskell, se parece com uma lista encadeada. Por exemplo a lista `[1,2,3,4]`, é construída usando o operador `cons(:)`, ou **construtor** de listas:

  ```
  1 : 2 : 3 : 4 : []
  ```

- O operador `:` é uma função que serve para construir listas de qualquer tipo:

  ```
  [True, False, False] == True : False : False : []
  [1]                  == 1 : []
  ```

- Toda vez que o **cons** é utilizado ele assume o tipo da lista que estamos construindo:

  ```
  (:) :: Int -> [Int] -> [Int]
  (:) :: Bool -> [Bool] -> [Bool]
  ```

Construindo Listas

- Por isso pode-se dizer que o **cons** tem um tipo **polimórfico**

  ```
  (:) :: t -> [t] -> [t]
  ```

  onde `t` é uma variável que pode assumir qualquer tipo.
- Outro operador de lista é o operador de concatenação `++`

  ```
  > [1,2,3] ++ [4,5,6]
  ```

Funções que Trabalham com Listas

- Supomos que queremos definir a função:

  ```
  somaLista :: [Int] -> Int
  ```

  que soma os elementos de uma lista. Essa função deve trabalhar com listas de qualquer tamanho. Para implementar `somaLista` devemos usar recursão. A recursão sobre listas geralmente possui dois casos:
  - O caso da lista vazia `[]`
  - O caso da lista não-vazia. Toda a lista não-vazia possui um elemento cabeça (**head**) e o resto da lista (**tail**). Ex: a lista **[1,2,3]** possui cabeça 1 e tail `[2,3]`. Uma lista com cabeça `c` e tail `x`, é representada pelo padrão `(c:x)`.

Construindo Listas

- A função `somaLista` pode ser definida então:
  - A soma da lista vazia `[]` é 0
  - A soma de uma lista não-vazia `(a:x)` é conseguida a somada a soma dos elementos de `x`. Então:

  ```
  somaLista :: [Int] -> Int
  somaLista []     = 0
  somaLista (a:x) = a + somaLista x
  ```

- Exemplo:

  ```
  > somaLista [1,2,3,4,5]
  15
  ```

- Exemplifique passo a passo a construção desse cálculo.

Ordenando uma Lista

- Para se ordenar uma lista

  `[7,3,9,2]`

  usando um algoritmo de inserção, podemos retirar o primeiro elemento e ordenar o tail da lista: Temos então a lista

  `[2,3,9]`

  Para terminar a ordenação da lista, basta colocar o valor 7 em sua posição correta dentro da lista. Temos então:

  `[2,3,7,9]`
- Em Haskell, o programa de ordenação utilizando esse algoritmo seria:

```
iSort :: [Int] -> [Int]
iSort []     = []
iSort (a:x) = ins a (iSort x)
```

Ordenando uma Lista

- O programa foi implementado utilizando-se uma abordagem top-down. Difininos a função `iSort` assumindo que sabemos definir a função `ins`.
- Dividir o problema em várias partes e depois resolver cada uma delas separadamente é bem mais fácil do que resolver um problema como um todo.
- A função `ins` pode ser definida da seguinte maneira:

```
ins :: Int -> [Int] -> [Int]
ins a []     = []
ins a (b:x)
  | a < b     = a : (b:x)
  | otherwise = b : ins a x
```

Exercícios

- Você consegue utilizar a função `iSort` para achar o maior e o menor elemento dentro de uma lista? Implementa a função:

  `maiorEmenor :: [Int] -> (Int, Int)`
- Mudando a definição de `ins`, podemos mudar o comportamento da função `iSort`. Defina duas novas versões de `ins`, uma que retire números repetidos da lista e outra que faça com que a lista seja ordenada de forma decrescente.

Listas de Tuplas

- Existem duas funções pré-definidas para se trabalhar com tuplas:`fst` que devolve o primeiro elemento da tupla, e `snd` que retorna o segundo elemento da tupla:

```
> fst (1,2)
1
> snd (3,4)
4
```