INE 5416/5636 - Paradigmas de programação

Turmas 04208/08238

Prof. Dr. João Dovicchi – dovicchi@inf.ufsc.br

http://www.inf.ufsc.br/~dovicchi

Listas em Haskell

Dados de mesmo tipo - diferentemente de C: não é possível saber sua localização na memória.

Listas em Haskell

Dados de mesmo tipo - diferentemente de C: não é possível saber sua localização na memória.

Costrutores:

[ ] : . .

Mais listas: Haskell

Exemplos:

- Uma lista vazia:

```
[]
```

- Uma lista de inteiros:

```
2:4:6:8:[] = [2,4,6,8]
```

- Uma lista de caracteres (string):

```
'i':'n':'f':'o':[] = ['i','n','f','o']="info"
```

- Uma lista de n-uplas (tuplas):

```
(1,'a'):(2,'b'):[] = [(1,'a'),(2,'b')]
```

Listas de listas

```
[[1,2,3],[3,2,1],[2,1,3]]
```

```
["bar","carro","cinema"]
```

List comprehension

Listas podem ser definidas matematicamente:

```
[x|x<-[1,2,3]]
```

List comprehension

Listas podem ser definidas matematicamente:

```
[x|x<-[1,2,3]]
```

Ou por uma expressão lambda:

```
(\x->x)  [1,2,3,4]
```

Para toda ocorrência de x em lambda x, x recebe uma lista contendo [1,2,3,4]

Listas implícitas

Nas linguagens funcionais as listas podem ser declaradas de forma explícita:

```
[1,2,3,4,5,6,7,8,9,10]
```

ou de forma implícita:

```
[1..10]
```

Exemplos

```
[2,4..20] -- representa [2,4,6,8,10,12,14,16,18,20]
          -- ou uma PA de 2 a 20 de razão 2

[1,4..15] -- representa [1,4,7,10,13] ou uma PA de
          -- 1 a 15 de razão 3.
```

Listas finitas e infinitas

As listas implícitas podem ser representadas de forma finita:

```
[1..100], [10,20..100]
```

ou de forma infinita:

```
[1..], [10,20..]
```

Listas e argumentos

Existe uma forma genérica de representação de listas para argumentos de funções:

```
(x:xs) -- lê-se 'xis' e 'xises'
```

onde 'x' representa o primeiro elemento da lista, chamado de "cabeça" e 'xs' representa os demais elementos, chamado de "cauda".

Listas e argumentos

Tipagem de funções - argumentos listas:

função :: [a]−> (retorna?)
função [] = (que fazer caso lista vazia?)
função (x:xs) = (expressão)

Exemplo:

```
-- 'head' retorna primeiro elemento da lista
head :: [a] -> a
head [] = []
head (x:xs) = x
```

Funções de funções

Em Haskell existem várias funções podem aplicar outras funções sobre listas de elementos

```
funcao :: (a->b)->[a]->[b]
```

Funções de funções

Em Haskell existem várias funções podem aplicar outras funções sobre listas de elementos

```
funcao :: (a->b)->[a]->[b]
```

Ex:

```
Prelude> : map
map :: (a -> b) -> [a] -> [b]
Prelude> map abs [-1,2,-5,3,-8]
[1,2,5,3,8]
Prelude> map odd [1..5]
[True,False,True,False,True]
Prelude>
```

Mais listas: Haskell

+ exemplo

Mapeando funções de alta ordem:

```
Prelude> let f = recip.negate
Prelude> map f [1,-2,4,-5]
[-1.0,0.5,-0.25,0.2]
Prelude>
```

O operador '.' em `recip.negate` junta duas funções em uma função de alta ordem.

Filter

A função `filter` testa uma condição em uma lista de argumentos e retorna os elementos que casam com a condição. Por exemplo:

```
Prelude> filter odd [1,7,4,5,3,8]
[1,7,5,3]
Prelude> filter (>5) [1..8]
[6,7,8]
Prelude> let n = ["aaaa","bbbbb","cc","dddddd"]
Prelude> filter (\x -> length x > 4) n
["bbbbb","dddddd"]
Prelude>
```

Operadores

(++): concatena duas listas

```
Prelude> [1,3,6,8]++[2,4]
[1,3,6,8,2,4]
```

Mais listas: Haskell

Operadores

(++): concatena duas listas

```
Prelude> [1,3,6,8]++[2,4]
[1,3,6,8,2,4]
```

(!!): retorna um elemento na posição do segundo argumento:

```
Prelude> [1,2,3,4,5]!!3
4
```

Operadores

(++): concatena duas listas

```
Prelude> [1,3,6,8]++[2,4]
[1,3,6,8,2,4]
```

(!!): retorna um elemento na posição do segundo argumento:

```
Prelude> [1,2,3,4,5]!!3
4
```

(\\): subtrai duas listas (não associativa)

```
Prelude> [1,2,2,3,4,5]\\[2,3,4]
[1,2,5]
```

Prática

Roteiro Prática 10