Linguagens de Programação

Aula 14

Celso Olivete Júnior

olivete@fct.unesp.br

unesp

Na aula passada

- Linguagem *Haskell*
  - Funções
  - Tipos básicos
  - Expressões

unesp

Na aula de hoje

- Linguagem *Haskell*
  - Listas

unesp

Listas e Tuplas

- A linguagem *Haskell* nos fornece dois mecanismos para a construção de dados compostos: **listas** e **tuplas**
- A **lista** possibilita a união de vários elementos - todos do mesmo tipo - numa única estrutura.
- Numa **tupla** podemos combinar os componentes de um dado numa única estrutura, e os componentes podem ter tipos e propriedades distintas.

unesp

Listas
Fundamentos

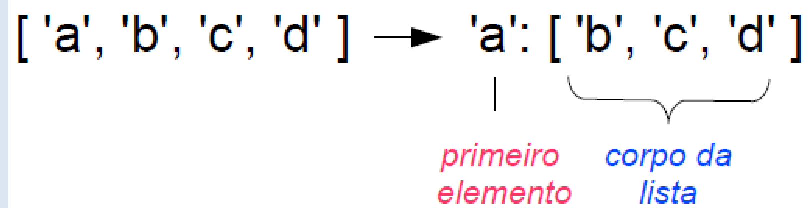
- Uma lista é uma estrutura de dados que representa uma coleção de objetos homogêneos em sequência.
- Para alcançar qualquer elemento, todos os anteriores a ele devem ser recuperados.
- Em programação, uma lista vazia (representada por [ ] em Haskell) é a estrutura base da existência de uma lista.

Listas

Fundamentos

- Uma lista é composta sempre de dois segmentos: cabeça (*head*) e corpo (*tail*). A cabeça da lista é sempre o primeiro elemento.

```
> ['a','b','c','d']
"abcd"
> 'a':['b','c','d']
"abcd“
```



Listas

Operador (:)

- O símbolo (:) é o operador de construção de listas. Toda lista é construída através deste operador. Exemplos:

```
Hugs> 'a':['b','c','d']
"abcd"
Hugs> 2:[4,6,8]
[2,4,6,8]
```

Listas

Hugs> 'a':['b','c','d']

"abcd"

Hugs> 1:[2,3]

[1,2,3]

Hugs> ['a','c','f'] == 'a':['c','f']

True

Hugs> [1,2,3] == 1:2:3:[]

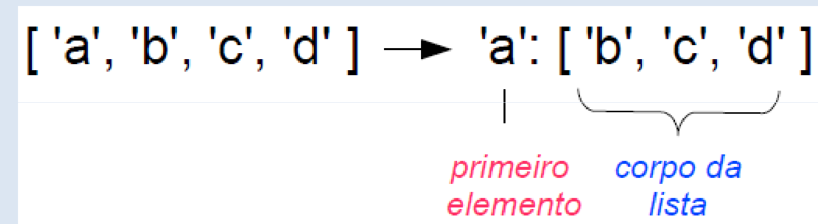
True

Hugs> 1:[2,3] == 1:2:[3]

True

Hugs> "papel" == 'p':['a','p','e','l']

True



Escrevendo Listas

- Pode-se definir uma lista indicando os limites inferior e superior de um conjunto conhecido, onde existe uma relação de ordem entre os elementos, no seguinte formato:

[<limite-inferior> .. <limite-superior>]

```
> [1..4]
[1,2,3,4]
> ['m'..'n']
"mn"
> [1,3..6]
[1,3,5]
> ['a','d'..'p']
"adgjmp"
> [3.1..7]
[3.1,4.1,5.1,6.1,7.1]
```

Escrevendo Listas

- Podemos definir qualquer progressão aritmética em uma lista utilizando a seguinte notação:

[<1o. termo>, <2o. termo> .. <limite-superior>]

```
> [7,6..3]
[7,6,5,4,3]
> [6,5..0]
[6,5,4,3,2,1,0]
> [-5,2..16]
[-5,2,9,16]
> [5,6..5]
[5]
> [1,1.1 .. 2]
[1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0]
```

Listas por compreensão

- A descrição de uma lista pode ser feita em termos dos elementos de uma outra lista. Por exemplo, temos a lista L1 = [2,4,7]. Uma lista definida por compreensão pode ser escrita:

```
> [ 2 * n | n <- L1 ]
[4,8,14]
```

- A lista resultante contém todos os elementos da lista L1, multiplicados por 2. Assim, podemos ler: Obtenha todos os 2*n dos elementos n contidos em L1 = [2,4,7].

Listas por compreensão

- Exemplos

```
listaQuad = [x^2 | x <- [1..30]]
>listaQuad
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676,729,784,841,900]
listaQuadInf = [ x^2 | x <- [1..] ]
> listaQuadInf
[1,4,9,16,25,36,49,64,81,100,121,144,169,196,225,256,289,324,361,400,441,484,529,576,625,676,729,784,841,900,961,1024,1089,1156,1225,1296,1369,1444,1521,1600 ...
> elem 4 listaQuadInf
True
```

- A função **elem** verifica se um elemento pertence a uma lista. Retorna True/False.

unesp

Gerador e Expressões Booleanas

- Na definição de lista por compreensão, o símbolo <- é chamado de gerador da lista, pois permite obter os dados através dos quais a nova lista será construída.
- Os geradores podem ser combinados com um ou mais testes, que são expressões booleanas.

```
listaQuadPares = [x^2 | x <- [1..20], even x]

Hugs> listaQuadPares

[4,16,36,64,100,144,196,256,324,400]
```

Gerador e Expressões Booleanas

```
listaQuadParesSup = [x^2 | x <- [1..20], even x, x > 6]

Hugs> listaQuadParesSup

[64,100,144,196,256,324,400]
```

Listas com mais de um Gerador

- Adicionalmente, é possível que mais de um gerador seja utilizado na definição de uma lista por compreensão:

```
Hugs> [ x*y | x <- [1,2,3], y <- [3,7,9]]
[3,7,9,6,14,18,9,21,27]
Hugs> [(x,y) | x <- [1,3,5], y <- [2,4,6], x < y]
[(1,2),(1,4),(1,6),(3,4),(3,6),(5,6)]
```

Listas
Funções Pré-definidas

- Funções Pré-definidas

Função	Descrição	Exemplo
(++)	Concatena duas listas	> [1,2,3]++[4,5,6] [1,2,3,4,5,6]
concat	Recebe uma lista de listas e as concatena	> concat [[1,2],[3,4]] [1,2,3,4]
head	Retorna o primeiro elemento da lista	> head "abc" 'a'
tail	Retorna o corpo da lista	> tail "abc" "bc"
last	Retorna o último elemento da lista	> last [4,3,2] 2

Listas
Funções Pré-definidas

❑ Funções Pré-definidas

Função	Descrição	Exemplo
`elem`	Verifica se um elemento pertence à lista	`> elem 5 [1,5,10]` `True`
`null`	Retorna verdadeiro (`True`) se uma lista é vazia	`> null []` `True`
`length`	Retorna o tamanho de uma lista	`> length "abcxyz"` `6`

`(!!)`	Operador de índice da lista, retorna o elemento mantido numa posição	`> [1,3,5,7,9] !!0`
		`1`
		`> (!!)['b','g','r','w'] 3`
		`'w'`

`replicate`	Constrói uma lista pela replicação de um elemento	`> replicate 4 'c'` `"cccc"`
`reverse`	Inverte os elementos de uma lista	`> reverse [4,5,2,2]` `[2,2,5,4]`

Listas
Funções Pré-definidas

- Funções Pré-definidas

Função	Descrição	Exemplo
`take`	Gera uma lista com os *n* primeiros elementos da lista original	`> take 2 ['d','f','g','r']` `"df"`
`drop`	Retira *n* elementos do início da lista	`> drop 3 [3,3,4,4,5,5]` `[4,5,5]`
`takeWhile`	Retorna o maior segmento inicial de uma lista que satisfaz uma condição	`> takeWhile (<10) [1,3,13,4]` `[1,3]`
`dropWhile`	Retira o maior segmento inicial de uma lista que satisfaz uma condição	`> dropWhile (<10) [1,3,13,4]` `[13,4]`
`replicate`	Constrói uma lista pela replicação de um elemento	`> replicate 4 'c'` `"cccc"`
`reverse`	Inverte os elementos de uma lista	`> reverse [4,5,2,2]` `[2,2,5,4]`

Listas

Funções Pré-definidas

- Funções Pré-definidas

Função	Descrição	Exemplo
`splitAt`	Divide uma lista num par de sub-listas fazendo a divisão numa determinada posição	`> splitAt 2 [3,4,2,1,5]` `([3,4],[2,1,5])`
`zip`	Recebe duas listas como entrada e retorna uma lista de pares	`> zip [1,2] ['a','b']` `[(1,'a'),(2,'b')]`
`sum`	Retorna a soma dos elementos da lista	`> sum [4,5,7,2,1]` `19`
`product`	Retorna o produto dos elementos da lista	`> product [5,3,6,1]` `90`
`maximum`	Retorna o maior elemento de uma lista	`> maximum [4,5,1,2]` `5`
`minimum`	Retorna o menor elemento de uma lista	`> minimum [5.2,0.3,7.2]` `0.3`

Listas

Exemplos sobre funções pré-definidas

```
geraPalindroma n = [1..n] ++ reverse [1..n]

> geraPalindroma 5

[1,2,3,4,5,5,4,3,2,1]

fat n = produto [1..n]

> fat 5

120
```

Listas
Funções recursivas

- Para que possamos contar quantos elementos estão contidos numa lista, podemos escrever uma função recursiva:

```
conta:: [t] -> Int
conta [] = 0
conta (a:x) = 1 + conta x
> conta ['a','b','c']
3
```

- Conta é uma função polimórfica, servindo para listas de qualquer tipo (”t”e uma variável de tipo, e pode ser substituída por qualquer tipo).

Listas

função pertence

```
pertence:: Eq t => t -> [t] -> Bool

pertence a [] = False

pertence a (x:z) = if (a == x) then True

else pertence a z

> pertence 3 [4,5,2,1]

False
```

Listas

encontrar o maior elemento

```
maior [x] = x

maior (x:y:resto) | x > y = maior (x: resto)

| otherwise = maior (y: resto)

> maior [4,5,2,1]

5
```

Exercícios

1. Remover um elemento da lista a partir de sua posição
2. Defina funções para implementar a união e interseção entre duas listas.
3. Analise a função abaixo e descreva sua funcionalidade. Em seguida teste a função no interpretador WinHugs.

```
misterio :: String -> String
misterio p = [c | c<-p, c>='a' && c<='z']
```

4. Escreva uma função para calcular a média dos elementos de uma lista de números. Podem-se usar duas funções, uma para obter a quantidade de elementos e outra para obter a soma dos elementos, e finalmente, calcular a divisão entre a soma e a quantidade.

Exercícios

5. Defina uma função que dada uma lista de inteiros, retorna o número de elementos de valor superior a um número n qualquer.

> retornaSup 4 [3,2,5,6]

6. Defina uma função que dada uma lista de inteiros, retorna outra lista que contém apenas de elementos de valor superior a um número n qualquer.

> retornaListaSup 4 [3,2,5,6]

[5,6]

unesp

Exercícios

7. Escreva uma função que recebe duas listas de inteiros e produz uma lista de listas. Cada uma corresponde à multiplicação de um elemento da primeira lista por todos os elementos da segunda.

```
> mult_listas [1,2] [3,2,5]
[[3,2,5],[6,4,10]]
```

8. Escreva uma função para verificar se os elementos de uma lista são distintos.

9. Seja a função união abaixo, definida através da construção de listas por compreensão. Teste esta função e implemente a interseção utilizando a mesma estratégia.

```
uniao :: Eq t => [t] -> [t] -> [t]
uniao as bs = as ++ [b | b <- bs, not (pertence b as)]
```