sumList [2,3,4,5]
= 2 + sumList [3,4,5]
= 2 + (3 + sumList [4,5])
= 2 + (3 + (4 + sumList [5]))
= 2 + (3 + (4 + (5 + sumList [])))
= 2 + (3 + (4 + (5 + 0)))
= 14

Programação Funcional

Listas

Matheus Carvalho Viana
matheuscviana@ufsj.edu.br

Dcomp
Departamento de Ciência da Computação

UFSJ
Universidade Federal de São João del-Rei

Introdução

Uma lista é uma estrutura que agrupa valores do mesmo tipo.

Um dos pontos fortes da programação funcional é a facilidade para trabalhar com listas.

As listas do Haskell fazem o papel dos arrays do C e do Java, agrupando um conjunto de valores do mesmo tipo, mas são mais flexíveis:

- Não é necessário definir um tamanho prévio;
- Pode aumentar e diminuir de tamanho com facilidade;
- É possível criar listas a partir de equações;
- É possível criar listas infinitas;
- As funções podem receber e retornar listas.

Listas

No Haskell, uma lista é escrita com valores dentro de um par de colchetes.

`[1,2,3]` é uma lista com três inteiros;

`[]` é uma lista vazia.

`['a'..'z']` é uma lista todas as letras minúsculas;

`[1..]` é uma lista infinita com todos os inteiros positivos;

A notação entre colchetes é, na verdade, um açúcar sintático. O Haskell não mantém as listas na memória, mas as constrói quando necessário.

A lista `[1,2,3,4]` é na verdade `(1:(2:(3:(4:[]))))`

É esse mecanismo que permite às listas serem bastante flexíveis, podendo até ter uma quantidade infinita de valores.

Tipo Lista

No cabeçalho das funções, o tipo lista é declarado escrevendo o tipo dos valores dela entre colchetes.

Exemplos:

`[Char]` é uma lista de caracteres;

`[Int]` é uma lista de inteiros;

`[Float]` é uma lista de valores Float;

`[[Int]]` é uma lista de lista de inteiros.

Concatenação

A concatenação é uma operação extremamente importante para a manipulação de listas nas funções recursivas do Haskell.

Dois operadores realizam a concatenação:

++ concatena duas listas.

[1,2,3] ++ [4,5] resulta em [1,2,3,4,5]

: insere um elemento no início da lista.

1 : [2,3] resulta em [1,2,3]

(1:(2:(3:(4:[ ])))) resulta em [1,2,3,4]

Casamento de Padrões com Lista

Em funções que recebem uma lista como entrada, o casamento de padrões pode ser feito da seguinte forma:

`[]` – lista vazia;

[x] – uma lista com um único elemento representado por x;

`(h:t)` – em que `h` representa o primeiro elemento e `t` representa uma lista com todos os elementos restantes (na verdade, é possível separar qualquer quantidade de elementos iniciais, mas raramente usa-se mais de um);

uma variável – que representa a toda a lista (vazia ou não).

Função com Lista

A função `dobraValores` recebe uma lista de inteiros e retorna a lista com cada valor multiplicado por dois.

```
dobraValores :: [Int] -> [Int]
dobraValores [] = []
dobraValores (a:b) = (a*2) : dobraValores b
```

Exemplo de execução:

```
dobraValores [1,2,3]
= (1*2) : dobraValores [2,3]
= (1*2) : (2*2) : dobraValores [3]
= (1*2) : (2*2) : (3*2) : dobraValores []
= (1*2) : (2*2) : (3*2) : []
= (1*2) : (2*2) : [6]
= (1*2) : [4,6]
= [2,4,6]
```

Função com Lista

```
remove :: Int -> [Int] -> [Int]
remove _ [] = []
remove x (h:t)
  | x == h    = remove x t
  | otherwise = h : remove x t


soma :: [Int] -> Int
soma [] = 0
soma [a] = a
soma (a:b:c) = (a+b) + soma c
```

Tipo String

Em Haskell, uma String é na verdade uma lista de caracteres.

Ou seja, `String` é sinônimo de `[Char]`.

Portanto, todas as operações válidas para as listas são válidas para String.

```
Prelude> "abc" ++ "de"
"abcde"
Prelude> 'a' : "bc"
"abc"
Prelude> "abc" == ['a','b','c']
True
```

Operador !!

O operador !! recebe uma lista e uma posição (inteiro) e retorna o elemento da lista que está na posição informada.

```
Prelude> [10,20..100] !! 3
40
Prelude> [10,20..100] !! 0
10
Prelude> [10,20..100] !! 9
100
Prelude> "ABCDEFGHIJ" !! 5
'F'
```

Atenção

[10,20..100] é a lista
[10,20,30,40,50,60,70,80,90,100]

Funções para Manipulação de Listas

head: recebe uma lista e retorna o primeiro elemento dela.

```
> head [5,6,7,8,9]
5
```

last: recebe uma lista e retorna o último elemento dela.

```
> last [5,6,7,8,9]
9
```

tail: recebe uma lista e a retorna sem o primeiro elemento.

```
> tail [5,6,7,8,9]
[6,7,8,9]
```

init: recebe uma lista e a retorna sem o último elemento.

```
> init [5,6,7,8,9]
[5,6,7,8]
```

length: recebe uma lista e retorna o tamanho dela.

```
> length [5,6,7,8,9]
5
```

Funções para Manipulação de Listas

maximum: recebe uma lista e retorna o maior elemento dela.

```
> maximum [5,1,7,8,3]
8
```

minimum: recebe uma lista e retorna o menor elemento dela.

```
> minimum [5,1,7,8,3]
1
```

sum: recebe uma lista de números e a retorna a soma dos elementos. Retorna 0 se a lista for vazia.

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

product: recebe uma lista de números e a retorna o produto dos elementos. Retorna 1 se a lista for vazia.

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

null: retorna True se a lista for vazia e False se não for.

```
> null [5,6,7,8,9]
False
```

Funções para Manipulação de Listas

take: recebe um inteiro n e uma lista e retorna uma lista com os n primeiros elementos da lista passada.

```
> take 2 [5,6,7,8,9]
[5,6]
```

drop: recebe um inteiro n e uma lista e retorna uma lista sem os n primeiros elementos da lista passada.

```
> drop 2 [5,6,7,8,9]
[7,8,9]
```

elem: recebe um elemento e uma lista retorna True se a lista contém o elemento ou False, caso o elemento não exista na lista.

```
> elem 'a' ['A'..'Z']
False
```

concat: concatena uma lista de listas em uma única lista.

```
> concat ["foo","bar","car"]
"foobarcar"
```

Funções para Manipulação de Listas

reverse: recebe uma lista e a retorna invertida.

```
> reverse [5,6,7,8,9]
[9,8,7,6,5]
```

intersperse: recebe um elemento e uma lista e retorna uma lista com o elemento passado intercalado entre os da lista original.

```
> intersperse ',' "abcde"
"a,b,c,d,e"
```

intercalate: recebe uma lista x e uma lista de listas xx, intercala x entre as listas de xx e concatena tudo. Equivale a `concat (intersperse x xx)`.

```
> intercalate ", " ["letra a", "letra b", "letra c"]
"letra a, letra b, letra c"
```

transpose: recebe uma lista de listas e com bina os primeiros elementos de cada uma, os segundos e assim por diante.

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

As três últimas pertencem a biblioteca **Data.List.**

Funções para Manipulação de String

lines: divide uma String em uma lista de Strings com base no caractere '\n'.

```
> lines "Bom dia!\nTudo bem?\nSim"
["Bom dia!", "Tudo bem?", "Sim"]
```

words: divide uma String em uma lista de Strings com base nos espaços.

```
> words "Programa Haskell"
["Programa","Haskell"]
```

unlines: o inverso da função lines.

```
> unlines ["letra a", "letra b", "letra c"]
"letra a\nletra b\nletra c\n"
```

unwords: o inverso da função words.

```
> unwords ["Programa","Haskell"]
"Programa Haskell"
```

Compreensão de Lista

Mecanismo de construção de listas baseadas em listas existentes.

Esse mecanismo é baseado na teoria de conjuntos* da matemática e permite criar listas a partir de uma equação.

Sua sintaxe é:

```
[<expr_de_saída> | <variável> <- <lista>, <expr_de_filtragem>]
```

Exemplo:

```
[ x | x <- [0..], x^2>3 ]
```

significa uma lista de elementos x, tal que x pertence à lista [0..] e x^2 > 3.

* Na matemática, conjuntos não possuem elementos repetidos. No haskell, as listas podem ter.

Compreensão de Lista

```
dobraValores :: [Int] -> [Int]
dobraValores lista = [x*2 | x <- lista]

remove :: Int -> [Int] -> [Int]
remove x lst = [ y | y <- lst, y /= x ]

quicksort :: [Int] -> [Int]
quicksort [] = []
quicksort (p:r) =
  quicksort [ x | x <- r, x < p] ++
  [p] ++
  quicksort [ x | x <- r, x >= p]
```

Exercícios

Obs: não defina o cabeçalho das funções deste exercício para que funcionem com qualquer tipo.

1. Crie uma função que faz a mesma coisa que a função length.
2. Crie uma função que recebe um caractere c e um inteiro n e retorna uma String em que c ocorre n vezes.
3. Crie uma função que recebe uma lista e retorna uma lista que possui somente os elementos maiores que a média deles.
4. Crie uma função que faz a mesma coisa que a função elem.
5. Crie uma função que faz a mesma coisa que função reverse.
6. Crie uma função que recebe uma lista e um inteiro e retorna a lista organizada em uma lista e listas de tamanho n.
7. Crie uma função que implemente o algoritmo de selectionsort.