Funções sobre listas

- Formas de listas
- Notação Cons
- Funções sobre listas
- Maneiras de descrever listas

Funções sobre listas

- Em vez de elaborar funções utilizando as funções pré-definidas sobre listas (head, tail, inverse, etc.)
- Vamos construir nossas próprias funções sobre listas, sem usar as funções pré-definidas
- Precisaremos usar recursividade para percorrermos os elementos da lista

Formas de Lista

- Estruturalmente uma lista pode ser de duas formas:
- lista vazia
  - não contém nenhum elemento
  - é denotada pelo construtor constante [ ]
- lista não vazia
  - contém elementos

Formas de Lista

- lista não vazia
  - contém pelo menos um elemento
    - [1,2,3,4,5]
  - é formada por uma cabeça (o primeiro elemento da sequência) e por uma cauda (uma lista dos demais elementos da sequência)
  - cabeça : cauda
  - 1:[2,3,4,5]

Notação cons

- lista não vazia pode ser construída pelo construtor **:** (dois pontos)
- **:** é um operador binário infixo.
- Essa notação é chamada de cons.
  - o operando da esquerda é a cabeça da lista
  - o operando da direita é a cauda da lista
  - 1:[2,3,4,5]

Notação cons

- Por exemplo, a lista formada pela sequência dos valores 1, 8, e 6 pode ser escrita das seguintes maneiras. Observe as notações cons

lista
`[1, 8, 6]`
`1 : [8, 6]`
`1 : (8 : [6])`
`1 : (8 : (6 : []))`
`1 : 8 : 6 : []`

Notação cons

- O operador binário infixo **:** tem associatividade à direita e precedência 5
  - imediatamente inferior à precedência dos operadores aditivos (+) e (-)
- Observe que os parênteses no exemplo anterior são desnecessários,
- já que o operador **:** associa-se à direita.

Notação cons

- Os exemplos anteriores (da tabela da aula07) podem ser reescritos com estes construtores de lista

notação estendida	notação básica
`['O', 'B', 'A']`	`'O' : 'B' : 'A' : []`
`['B', 'A', 'N', 'A', 'N', 'A']`	`'B' : 'A' : 'N' : 'A' : 'N' : 'A' : []`
`[False, True, True]`	`False : True : True : []`
`[ [False], [], [True, False, True] ]`	`(False : []) : [] : (True : False : True : []) : []`
`[1., 8., 6., 10.48, -5.]`	`1. : 8. : 6. : 10.48 : -5. : []`

Notação cons

- Observe que podemos construir uma lista aos pedaços usando o operador **:**
- Prelude> let numeros = [1,2,3,4]
- Prelude> 0:numeros

[0,1,2,3,4]

- Prelude> 1:0:numeros

[1,0,1,2,3,4]

Funções sobre listas

- Na maioria das definições sobre listas usamos **recursividade** para percorrer todos os elementos.
- Para isso precisamos definir os casos base e recursivo
- É adequado que notação da lista seja cons, principalmente para o caso recursivo
  - head:tail

Funções sobre listas

- Uma função simples seria a função para somar todos os elementos de uma lista de números inteiros:
- Nomeando e definindo o tipo da função:
- somaLista :: [Int] -> Int
- Qual a função pré-definida para fazer isso?

Funções sobre listas

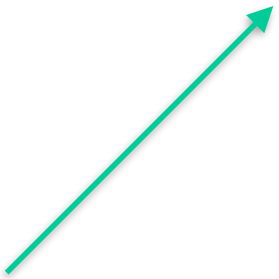
- Vamos usar recursividade.
- Para esta função existem dois casos:
- Caso Base: Somar os elementos de uma lista vazia [ ] que irá resultar em 0
- somaLista [ ] = 0

Funções sobre listas

- Caso Recursivo: Somar os elementos de uma lista não vazia.
- Em uma lista não vazia existe sempre o elemento *head* (o primeiro elemento),
- e o *tail* da lista, que é a lista que sobra sem o elemento head.
- Por exemplo, a lista [1, 2, 3] tem head 1 e tail [2,3].
- Esta lista pode ser escrita desta maneira (notação cons):
- (1:[2,3])

Funções sobre listas

- Uma lista com head *a* e tail *x* é escrita (*a*:*x*).
- Então a soma dos elementos de uma lista não vazia (*a*:*x*) é dada somando *a* à soma dos elementos de *x*.
- somaLista (a:x) = a + somaLista x



Funções sobre listas

- A definição completa da função seria:
- somaLista :: [Int] -> Int
- somaLista [ ] = 0
- somaLista (a:x) = a + somaLista x

Funções sobre listas

- Exemplo de uso:
- Prelude> somaLista [ ]

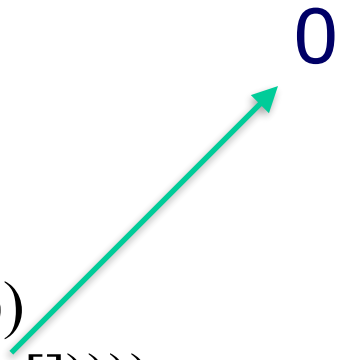
0

- Prelude> somaLista [1, 2, 3 ,4 ,5]

15

Funções sobre listas

- O comando é avaliado da seguinte maneira:
- somaLista [1, 2, 3, 4, 5]



= 1 + somaLista [2, 3, 4, 5]
= 1 + (2 + somaLista [3, 4, 5])
= 1 + (2 + (3 + somaLista [4, 5]))
= 1 + (2 + (3 + (4 + somaLista [5])))
= 1 + (2 + (3 + (4 + (5 + somaLista []))))

= 1 + (2 + (3 + (4 + (5 + 0))))
= 15

Funções sobre listas

- Uma outra função simples seria a função para determinar a lista cujos elementos são o dobro dos elementos de uma lista:
- Nomeando e definindo o tipo da função:
- dobraLista :: [Int] -> [Int]
- Qual a função pré-definida para fazer isso?

Funções sobre listas

- Como temos que percorrer a lista inteira, vamos precisar usar recursividade.
- Quais são os dois casos desta função?
- Caso base ?
- Caso recursivo?

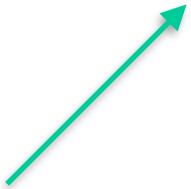
Funções sobre listas

- O caso base é determinar a lista cujos elementos são o dobro dos elementos de uma lista vazia.
- A resposta seria [ ].
- dobraLista [ ] = [ ]

- Até agora temos o seguinte
- dobraLista :: [Int] -> [Int]
- dobraLista [ ] = [ ]

Funções sobre listas

- O passo recursivo consiste em considerar uma lista não vazia.
- Como se faz isso?
- Calcula-se o dobro do head e coloca-se este elemento como o primeiro da lista cujos elementos são o dobro dos elementos do tail.
- A resposta seria 2*a : dobraLista x
- dobraLista (a:x) = 2*a : dobraLista x



Funções sobre listas

- Finalmente, a função completa:
- dobraLista :: [Int] -> [Int]
- dobraLista [ ] = [ ]
- dobraLista (a:x) = 2*a : dobraLista x

Funções sobre listas

- Exemplo de uso:
- Prelude> dobraLista [ ]

[ ]

- Prelude> dobraLista [1, 2, 3 ,4 ,5]

[2,4,6,8,10]

Funções sobre listas

- Mais uma função simples seria a função para determinar o número de elementos de uma lista.
- Agora pode ser uma lista de qualquer tipo mas vamos considerar uma lista de caracteres.
- Nomeando e definindo o tipo da função:
- comprimentoLista :: [Char] -> Int
- Qual a função pré-definida para fazer isso?

Funções sobre listas

- Como temos que percorrer a lista inteira, vamos precisar usar recursividade.
- Quais são os dois casos desta função?
- Caso base ?
- Caso recursivo?

Funções sobre listas

- O caso base é a lista vazia. A lista vazia tem tamanho 0.
- comprimentoLista [ ] = 0

- Até agora temos o seguinte
- comprimentoLista :: [Char] -> Int
- comprimentoLista [ ] = 0

Funções sobre listas

- O passo recursivo consiste em considerar uma lista não vazia.
- Como se faz isso?
- A lista não vazia, possui sempre um elemento a mais que o seu tail.
- A resposta seria 1 + comprimentoLista x
- comprimentoLista (a:x) = 1 + comprimentoLista x

Funções sobre listas

- Finalmente, a função completa:
- comprimentoLista :: [Char] -> Int
- comprimentoLista [ ] = 0
- comprimentoLista (a:x) = 1 + comprimentoLista x

Funções sobre listas

- Exemplo de uso:
- Prelude> comprimentoLista [ ]

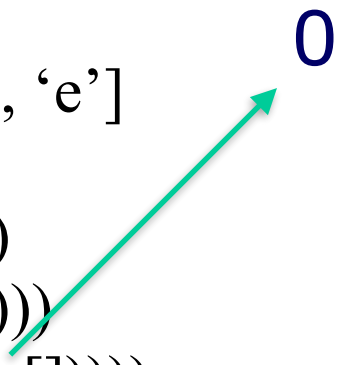
0

- Prelude> comprimentoLista [‘a’,’b’,’c’,’d’,’e’]

5

Funções sobre listas

- O comando é avaliado da seguinte maneira:
- comprimentoLista [‘a’,’b’,’c’,’d’,’e’]



= 1 + comprimentoLista [‘b’, ‘c’, ‘d’, ‘e’]
= 1 + (1 + somaLista [‘c’, ‘d’, ‘e’])
= 1 + (1 + (1 + somaLista [‘d’, ‘e’]))
= 1 + (1 + (1 + (1 + somaLista [‘e']))
= 1 + (1 + (1 + (1 + (1 + somaLista []))))

= 1 + (1 + (1 + (1 + (1 + 0))))
= 5

Funções sobre listas - Resumo

- Passos para implementar uma função sobre lista usando recursividade:
- 1) Definir o nome e o tipo da função
- 2) Definir o caso base
- 3) Definir o caso recursivo
- 4) Testar a função

Maneiras de descrever listas

- Existem outras maneiras de descrever listas:
- [a..b] é a lista [a,a+1,…,b]
- Exemplos:
- Prelude > [1 .. 6]

[1,2,3,4,5,6]

- Prelude > [4..2]

[ ]

Maneiras de descrever listas

- Outra maneira seria:
- [a,b..c] é a lista de elementos de a até c passo b–a
- Exemplos:
- Prelude > [2,4 .. 10]

[2, 4, 6, 8, 10]

- Prelude > [1,3 .. 10]

[1, 3, 5, 7, 9]

- O último elemento da lista é o maior da seqüência e deve ser menor ou igual a c.

ATS 1

- Quais dessas são listas válidas em Haskell e quais não são? Reescreva em notação cons.
- [1,2,3,[ ]]
- [1, [2,3], 4]
- [[1,2,3],[ ]]

ATS

- Qual o resultado do código abaixo:
- Prelude> let nums = [1,2,3,4,5,6]
- Prelude> 6:5:4:3:2:1:0:nums
- ?

- Por que Lista de listas podem ser útil? Cite um exemplo de seu uso.

APS 7

- Crie o programa fonte aula08.hs (com todas as funções e variáveis) como codificado nesta aula. Teste todas as funções do programa fonte, carregando no GHCi.
- Observe a necessidade de definir os tipos de dados das variáveis e também das funções.
- Essa APS não precisa ser enviada para o professor, mas deve ser realizada.