Faculdade de Computação
Programação Funcional (BCC/BSI) - 1° Período
Aula Prática: Processamento de Listas em Haskell

Lista

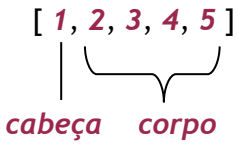
Uma lista é uma estrutura de dados que representa uma coleção de objetos homogêneos em sequência. Um elemento pode ser recuperado a partir dos elementos anteriores a ele.

Exemplos:

```
[]                      ->    lista vazia
[10, (-7), 11, 19]      ->    lista de inteiros [Int]
[0.6, 0.75, 0.9]        ->    lista de reais [Float]
['a', 'b', 'c', 'd']    ->    lista de caracteres [Char]
[True, False, False]    ->    lista de booleanos [Bool]
["maio", "abril"]       ->    lista de palavras [String] ou [[Char]]
[(12,'a'),(13,'b')]     ->    lista de tuplas [(Int, Char)]
```

Representação Cabeça:Corpo

Uma lista pode estar vazia ou conter elementos. Caso não esteja vazia, podemos representá-la por 2 segmentos: cabeça (*head*) e corpo (*tail*). A cabeça da lista é o primeiro elemento, e o corpo a sub-lista restante:



Exemplos:

```
5:[] == [5]
'a':['n','a'] == ['a','n','a']  == "ana"
1:[2,3] ++ [4,5,6] == [1,2,3,4,5,6]
1:2:3:[] == [1,2,3]
```

Função: contar elementos

Uma função para contar elementos (valores inteiros) pode ser definida recursivamente. Primeiramente definimos o caso base, em que o comprimento de uma lista vazia é zero. Em seguida definimos o passo da indução: ao encontrar um elemento, contamos 1 e somamos ao comprimento da sub-lista, até que ela esteja vazia.

```
conta :: [Int] -> Int
conta [] = 0
conta (c : r) = 1 + conta r
```

Exercícios:

1) Teste as listas em Haskell no interpretador WinHugs e anote os resultados:

```
> [1,2,3]
> [0,2,'a']
> [1,2,3] == [2,1,3]
> [1 .. 60]
> [1,4 .. 60]
> [0.1,0.2 .. 0.9]
> [0.1 .. 9]
> ["apartamento",['c','a','s','a']]
> [('a',3),('b',4),('c',5)]
> ['a','b',['c','d']] /= ['a','b','c','d']
> sum [1..10]
```

2) Escreva a função ***conta*** definida anteriormente no bloco de notas, salve o arquivo (.hs) e abra-o no WinHugs. Em seguida faça os seguintes testes:

```
> conta [1,2,3]
> conta [1..50]
> conta [0.01, 0.02 .. 0.10]
> conta [(0,'a'),(1,'b'),(2,'c')]
```

Quais os resultados? Como modificar a função para que possamos obter o comprimento de uma lista com elementos de qualquer tipo? Por exemplo, lista de números reais, palavras, tuplas? Faça esta modificação e teste novamente a função.

3) Escreva uma função recursiva que verifica se um dado elemento ***pertence*** à uma lista. A função deve ter como entrada: uma lista e o elemento a ser procurado, e deve retornar como resultado um valor booleano (True/False) como no exemplo abaixo.

```
> pertence 'a' ['b','c','a','d']
True
```

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.

```
> media [4,5,2,7]
4.5
```

5) Teste a função abaixo para retornar o ***maior*** elemento de uma lista:

```
maior [a] = a
maior (a : t) = if (a > (maior t)) then  a
                                   else  (maior t)
```

(a) Reescreva a função eliminando o comando *if*.
(b) Baseado na função maior, faça uma nova função que devolva o menor valor de uma lista.
(c) Usando as funções maior e menor, crie uma nova função que dada uma lista, retorne o maior e menor elemento numa tupla-2.

6) No módulo Char encontramos a função *toUpper* que converte uma letra minúscula na sua correspondente maiúscula. Para que possamos converter todas as letras de uma palavra em maiúsculas podemos escrever:

```
import Char
maius::[Char]->[Char]
maius [] = []
maius (a:as) = toUpper a: maius as
```

(a) Teste a função para a palavra “aBC1cde”.

(b) Usando a função *isAlpha* exemplificada abaixo, refaça a função ***maius*** para descartar símbolos ou números.

```
> isAlpha 'b'
True
> isAlpha '1'
False
> isAlpha '&'
False
```

(b) Faça uma nova função que recebe uma palavra e retorne numa tupla-2 a palavra original e a sua correspondente escrita em maiúsculas.