

Ficha 6

Programação Funcional

2015/16

1. Defina uma função `toDigits :: Int -> [Int]` que, dado um número (na base 10), calcula a lista dos seus dígitos (por ordem inversa). Por exemplo, `toDigits 1234` deve corresponder a `[4,3,2,1]`. Note que

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

2. Pretende-se agora que defina a função inversa da anterior `fromDigits :: [Int] -> Int`. Por exemplo, `fromDigits [4,3,2,1]` deve corresponder a 1234.

- (a) Defina a função com auxílio da função `zipWith`.
- (b) Defina a função com recursividade explícita. Note que

$$\begin{aligned} \text{fromDigits } [4,3,2,1] &= 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 \\ &= 4 + 10 \times (3 + 10 \times (2 + 10 \times (1 + 10 \times 0))) \end{aligned}$$

- (c) Defina agora a função usando um `foldr`.

3. Usando as funções anteriores e as funções `intToDigit :: Int -> Char` e `digitToInt :: Char -> Int` do módulo `Data.Char`,

- (a) Defina a função `intStr :: Int -> String` que converte um inteiro numa string. Por exemplo, `intStr 1234` deve corresponder à string `"1234"`.
- (b) Defina a função `strInt :: String -> Int` que converte a representação de um inteiro (em base 10) nesse inteiro. Por exemplo, `strInt "12345"` deve corresponder ao número 12345.

4. Defina a função `agrupa :: String -> [(Char,Int)]` que dada uma string, junta num par `(x,n)` as `n` ocorrências consecutivas de um carácter `x`. Por exemplo, `agrupa "aaakkkkkwaa"` deve dar como resultado a lista `[('a',3), ('k',4), ('w',1), ('a',2)]`. Sugestão: use a função `span`.

5. Defina a função `subLists :: [a] -> [[a]]` que calcula todas as sublistas de uma lista; por exemplo,
`subLists [1,2,3] = [[1,2,3], [1,2], [1,3], [1], [2,3], [2], [3], []]`.

6. Considere a seguinte definição para representar matrizes:

```
type Mat a = [[a]]
```

Por exemplo, a matriz (triangular superior) $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$ seria representada por

```
[[1,2,3], [0,4,5], [0,0,6]]
```

Defina as seguintes funções sobre matrizes (use, sempre que achar apropriado, funções de ordem superior).

- (a) `dimOK :: Mat a -> Bool` que testa se uma matriz está bem construída (i.e., se todas as linhas têm a mesma dimensão).
- (b) `dimMat :: Mat a -> (Int,Int)` que calcula a dimensão de uma matriz.
- (c) `addMat :: Num a => Mat a -> Mat a -> Mat a` que adiciona duas matrizes.
- (d) `transpose :: Mat a -> Mat a` que calcula a transposta de uma matriz.
- (e) `multMat :: Num a => Mat a -> Mat a -> Mat a` que calcula o produto de duas matrizes.
- (f) Defina a função `zipWMat :: (a -> b -> c) -> Mat a -> Mat b -> Mat c` que, à semelhança do que acontece com a função `zipWith`, combina duas matrizes. Use essa função para definir uma função que adiciona duas matrizes.
- (g) Defina a função `triSup :: Mat a -> Bool` que testa se uma matriz quadrada é triangular superior (i.e., todos os elementos abaixo da diagonal são nulos).
- (h) Defina a função `rotateLeft :: Mat a -> Mat a` que roda uma matriz 90° para a esquerda. Por exemplo, o resultado de rodar a matriz acima apresentada deve corresponder à matriz $\begin{bmatrix} 3 & 5 & 6 \\ 2 & 4 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.