Ficha 5

Programação Funcional

2015/16

1. Uma forma de representar polinómios de uma variável é usar listas de monómios representados por pares *(coeficiente, expoente)*

```
type Polinomio = [Monomio]
type Monomio = (Float,Int)
```

   Por exemplo, `[(2,3), (3,4), (5,3), (4,5)]` representa o polinómio $2\,x^3 + 3\,x^4 + 5\,x^3 + 4\,x^5$.

   (a) Defina uma função `conta :: Int -> Polinomio -> Int` de forma a que `(conta n p)` indica quantos monómios de grau `n` existem em `p`.

   (b) Defina a função `grau :: Polinomio -> Int` que indica o grau de um polinómio.

   (c) Defina `selgrau :: Int -> Polinomio -> Polinomio` que selecciona os monómios com um dado grau de um polinómio. Use uma função de ordem superior.

   (d) Complete a definição da função `deriv` de forma a que esta calcule a derivada de um polinómio.

```
deriv :: Polinomio -> Polinomio
deriv p = map ........ p
```

   (e) Defina a função `calcula :: Float -> Polinomio -> Float` que calcula o valor de um polinómio para uma dado valor de x.

   (f) Defina a função `simp :: Polinomio -> Polinomio` que retira de um polinómio os monómios de coeficiente zero. De preferência, use funções de ordem superior.

   (g) Complete a definição da função `mult` de forma a que esta calcule o resultado da multiplicação de um monómio por um polinómio.

```
mult :: Monomio -> Polinomio -> Polinomio
mult (c,e) p = map ........ p
```

   (h) Defina `normaliza :: Polinomio -> Polinomio` que que dado um polinómio constrói um polinómio equivalente em que não podem aparecer varios monómios com o mesmo grau.

   (i) Defina a função `soma :: Polinomio -> Polinomio -> Polinomio` que faz a soma de dois polinómios de forma que se os polinómios que recebe estiverem normalizados produz também um polinómio normalizado.

   (j) Defina a função `produto :: Polinomio -> Polinomio -> Polinomio` que calcula o produto de dois polinómios

   (k) Defina a função `ordena :: Polinomio -> Polinomio` que ordena um polonómio por ordem crescente dos graus dos seus monómios.

   (l) Defina a função `equiv :: Polinomio -> Polinomio -> Bool` que testa se dois polinómios são equivalentes.

2. Defina uma função `nzp :: [Int] -> (Int,Int,Int)` que, dada uma lista de inteiros, conta o número de valores nagativos, o número de zeros e o número de valores positivos, devolvendo um triplo com essa informação. Certifique-se que a função que definiu percorre a lista apenas uma vez.

3. Defina a função `digitAlpha :: String -> (String,String)`, que dada uma string, devolve um par de strings: uma apenas com as letras presentes nessa string, e a outra apenas com os números presentes na string. Implemente a função de modo a fazer uma única travessia da string. (Relembre que as funções `isDigit, isAlpha::Char->Bool` estão já definidas no módulo `Data.Char`).

4. Para cada uma das expressões seguintes, exprima por enumeração a lista correspondente. Tente ainda, para cada caso, descobrir uma outra forma de obter o mesmo resultado.

   (a) `[x | x <- [1..20], mod x 2 == 0, mod x 3 == 0]`

   (b) `[x | x <- [y | y <- [1..20], mod y 2 == 0], mod x 3 == 0]`

   (c) `[(x,y) | x <- [0..20], y <- [0..20], x+y == 30]`

   (d) `[sum [y | y <- [1..x], odd y] | x <- [1..10]]`

5. Defina cada uma das listas seguintes por compreensão.

   (a) `[1,2,4,8,16,32,64,128,256,512,1024]`

   (b) `[(1,5),(2,4),(3,3),(4,2),(5,1)]`

   (c) `[[1],[1,2],[1,2,3],[1,2,3,4],[1,2,3,4,5]]`

   (d) `[[1],[1,1],[1,1,1],[1,1,1,1],[1,1,1,1,1]]`

   (e) `[1,2,6,24,120,720]`

6. Apresente definições das seguintes funções de ordem superior, já pré-definidas no Prelude:

   (a) `zipWith :: (a->b->c) -> [a] -> [b] -> [c]` que combina os elementos de duas listas usando uma função específica; por exemplo:
   `zipWith (+) [1,2,3,4,5] [10,20,30,40] = [11,22,33,44]`.

   (b) `takeWhile :: (a->Bool) -> [a] -> [a]` que determina os primeiros elementos da lista que satisfazem um dado predicado; por exemplo:
   `takeWhile odd [1,3,4,5,6,6] = [1,3]`.

   (c) `dropWhile :: (a->Bool) -> [a] -> [a]` que elimina os primeiros elementos da lista que satisfazem um dado predicado; por exemplo:
   `dropWhile odd [1,3,4,5,6,6] = [4,5,6,6]`.

   (d) `span :: (a-> Bool) -> [a] -> ([a],[a])`, que calcula simultaneamente os dois resultados anteriores. Note que apesar de poder ser definida à custa das outras duas, usando a definição

   ```
   span p l = (takeWhile p l, dropWhile p l)
   ```

   nessa definição há trabalho redundante que pode ser evitado. Apresente uma definição alternativa onde não haja duplicação de trabalho.