Ficha 5

Programação Funcional

LEI 1º ano

1. A função `divMod :: Int -> Int -> (Int,Int)`, já predefinida no Prelude, poderia ser definida pela seguinte equação:

   ```
   divMod x y = (div x y, mod x y)
   ```

   Apresente uma definição alternativa desta função sem usar `div` e `mod` como funções auxiliares.

2. A função `splitAt :: Int -> [a] -> ([a],[a])`, já predefinida no Prelude, poderia ser definida pela seguinte equação:

   ```
   splitAt n l = (take n l, drop n l)
   ```

   no entanto nessa definição há uma duplicação de trabalho, dado que se fazem duas travessias da lista. Apresente uma versão alternativa para esta função que faça apenas uma travessia da lista.

3. Apresente definições das seguintes funções de ordem superior, já predefinidas no Perlude:

   (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.

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

5. 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

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

6. 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

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

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

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

   (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`.

8. Considere a função seguinte

```
indicativo  :: String -> [String] -> [String]
indicativo ind telefs = filter (concorda ind) telefs
  where concorda :: String -> String -> Bool
        concorda [] _ = True
        concorda (x:xs) (y:ys) = (x==y) && (concorda xs ys)
        concorda (x:xs) [] = False
```

   que recebe uma lista de algarismos com um indicativo, uma lista de listas de algarismos representando números de telefone, e seleciona os números que começam com o indicativo dado. Por exemplo:

   `indicativo "253" ["253116787","213448023","253119905"]`

   devolve `["253116787","253119905"]`.

   Redefina esta função com recursividade explícita, isto é, evitando a utilização de `filter`.