Exame de Recurso de Programação Funcional – 1° Ano, MIEI / LCC / MIEF
30 de Janeiro de 2018 (Duração: 2 horas)

1. Apresente uma definição recursiva da função (pré-definida) `(!!) :: [a] -> Int -> a` que dada uma lista e um inteiro, calcula o elemento da lista que se encontra nessa posição (assume-se que o primeiro elemento se encontra na posição `0`).

   Por exemplo, `(!!) [10,20,30] 1` corresponde a `20`.

   Ignore os casos em que a função não se encontra definida (i.e., em que a posição fornecida não corresponde a nenhuma posição válida da lista).

2. Considere o seguinte tipo para representar movimentos de um robot.

   ```
   data Movimento = Norte | Sul | Este | Oeste deriving Show
   ```

   Defina a função `posicao :: (Int,Int) -> [Movimento] -> (Int,Int)` que, dada uma posição inicial (abcissa e ordenada) e uma lista de movimentos (um movimento para Norte aumenta a ordenada e para Este aumenta a abcissa), calcula a posição final do robot depois de efectuar essa sequência de movimentos.

3. Apresente uma definição recursiva da função `any :: (a -> Bool) -> [a] -> Bool` que testa se um predicado é verdade para algum elemento de uma lista. Por exemplo, `any odd [1..10] == True`.

4. Considere o sequinte tipo `type Mat a = [[a]]` para representar matrizes.

   Defina a função `triSup :: Num a => Mat a -> Bool` que testa se uma matriz quadrada é triangular superior (i.e., todos os elementos abaixo da diagonal são nulos). Esta função deve devolver `True` para a matriz `[[1,2,3], [0,4,5], [0,0,6]]`

5. Defina um programa `movimenta :: IO (Int,Int)` que lê uma sequência de comandos do teclado (`'N'` para `Norte`, `'S'` para `Sul`, `'E'` para `Este`, `'O'` para `Oeste` e qualquer outro caracter para parar) e devolve a posição final do robot (assumindo que a posição inicial é `(0,0)`).

6. Considere o tipo `Imagem` para representar imagens compostas por quadrados (apenas com coordenadas positivas).

   Ao lado apresenta-se um exemplo de uma destas imagens constituída por três quadrados (cujos lados têm dimensão 5, 4 e 2).

   ```
   data Imagem = Quadrado Int
               | Mover (Int,Int) Imagem
               | Juntar [Imagem]
   ```

   ```
   ex :: Imagem
   ex = Mover (5,5)
         (Juntar [Mover (0,1) (Quadrado 5),
                  Quadrado 4,
                  Mover (4,3) (Quadrado 2)])
   ```

   (a) Defina a função `vazia :: Imagem -> Bool` que testa se uma imagem **não tem** nenhum quadrado. A função devolve `False` para o exemplo acima.

   (b) Defina a função `maior :: Imagem -> Maybe Int` que calcula a largura do maior quadrado de uma imagem. No exemplo acima, `maior ex == Just 5`. Note que a imagem pode não ter quadrados.

   (c) Defina `Imagem` como uma instância de `Eq` de forma a que duas imagens são iguais sse forem compostas pelos mesmos quadrados nas mesmas posições. Por exemplo, a imagem `ex` acima deverá ser igual a `Juntar [Mover (5,5) (Quadrado 4), Mover (5,6) (Quadrado 5), Mover (9,8) (Quadrado 2)]`.