# Exame de Recurso

Programação Funcional – 1º Ano, LEI / LCC / MIEF
14 de Fevereiro de 2014

Duração: 2 horas

1. Pretende-se contabilizar a pontação do campeonato de Fórmula 1. Para isso definiram-se os seguintes tipos de dados:

```
type TabClass = [(Piloto,Equipa,Pontos)]
type Piloto = String
type Pontos = Int
type Equipa = String
```

   (a) No campeonato de Fórmula 1 a pontuação de cada equipa é obtida pela soma dos pontos dos seus pilotos. Defina uma função `pontosEquipa :: Equipa -> TabClass -> Pontos` que calcula os pontos de uma dada equipa.

   (b) Numa prova de Fórmula 1 os dez primeiros pilotos a cortar a linha de meta recebem, respectivamente, 25, 18, 15, 12, 10, 8, 6, 4, 2 e 1 pontos.
   Defina a função `junta :: [Piloto] -> TabClass -> TabClass` que recebe a lista de pilotos (pela ordem de chegada à meta numa prova) e a tabela de classificação geral, e calcula a nova classificação geral após a prova.

   (c) Defina a função `ordena :: TabClass -> TabClass` que recebe uma tabela de classificação geral e a devolve ordenada por ordem decrescente de pontos.

2. Considere o seguinte tipo 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]]`

   (a) Defina uma 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.

   (b) Defina uma função `triSup :: Mat a -> Bool` que testa se uma matriz quadrada é triangular superior (i.e., todos os elementos abaixo da diagonal são nulos).

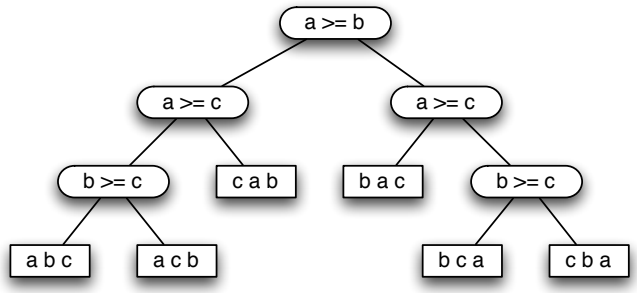
   (c) Defina uma função `rotateLeft :: Mat a -> Mat a` que roda uma matriz $90^o$ para a esquerda.
   Por exemplo, para a matriz acima o resultado de a rodar deve corresponder à matriz $\begin{bmatrix} 3 & 5 & 6 \\ 2 & 4 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.

3. Considere o seguinte tipo para representar um questionário com respostas binárias.

```
data Questionario = Solucao String
                  | Questao String Questionario Questionario
```

Para uma dada questão, uma resposta negativa (falso) correspode ao sub-questionário da esquerda e uma resposta positiva ao da direita. Um exemplo de um questionário (para ordenação de três valores) é dado por

```
q = Questao "a >= b"
      (Questao "a >= c"
         (Questao "b >= c"
             (Solucao "a b c")
             (Solucao "a c b"))
         (Solucao "c a b"))
      (Questao "a >= c"
         (Solucao "b a c")
         (Questao "b >= c"
            (Solucao "b c a")
            (Solucao "c b a")))
```




(a) Defina uma função `respostas :: Questionario -> [String]` que calcula todas as respostas possíveis desse questionário.

(b) Defina uma função `seqResp :: Questionario -> String -> Maybe [Bool]` que, dado um questionário e uma string correspondente a uma resposta, calcula a sequência de respostas necessárias para obter essa resposta. No caso da resposta não existir, a função deverá retornar `Nothing`.

(c) Defina `Questionário` como instância da classe `Show`, de forma a que, para cada resposta possível, seja impressa uma linha com as questões (e repostas) necessárias para obter essa resposta.

Por exemplo, para o questionário acima, `show q` deve dar como resultado

```
*Main> q
a >= b? Nao a >= c? Nao b >= c? Nao SOLUCAO: a b c
a >= b? Nao a >= c? Nao b >= c? Sim SOLUCAO: a c b
a >= b? Nao a >= c? Sim SOLUCAO: c a b
a >= b? Sim a >= c? Nao SOLUCAO: b a c
a >= b? Sim a >= c? Sim b >= c? Nao SOLUCAO: b c a
a >= b? Sim a >= c? Sim b >= c? Sim SOLUCAO: c b a
```

(d) Considere a seguinte função sobre questionários.

```
toString :: Questionario -> [String]
toString (Solucao s) = ['S':s]
toString (Questao q nao sim) = (toString nao) ++ (toString sim) ++ ['Q':q]
```

Defina uma função `fromString :: [String] -> Questionario` inversa da anterior, no sentido em que, para todo o questionário `q` se verifica que `fromString (toString q) == q`.