Programação Funcional
1º Ano – LEI/LCC

22 de Janeiro de 2010 – Duração: 2 horas

Teste

Parte I

Esta parte do teste representa 12 valores da cotação total. Cada uma das (sub-)alíneas está cotada em 2 valores.
A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.

1. Defina a função `intercala :: a -> [a] -> [a]` que intercala um dado elemento entre cada dois elementos da lista. Por exemplo, `intercala ‘,‘ "abc" == "a,b,c"`.

2. Considere a seguinte definição da função `(!!) :: [a] -> Int -> a` pré definida em Haskell, de selecção de um elemento da lista.

   ```
   l !! n = head (drop n l)
   ```

   Apresente uma definição alternativa (recursiva) sem usar a função `drop`.

3. Considere o seguinte tipo para representar valores opcionais:

   ```
   data Maybe a = Nothing | Just a
   ```

   Defina a função `catMaybes :: [Maybe a] -> [a]` que extrai todo o conteúdo da lista de entrada.

4. Considere-se o problema de contar o número de votos de cada um dos candidatos em uma eleição. Os tipos de dados de partida são

   ```
   type Candidato  = String
   type Boletim    = [Candidato]   -- lista de nomes que consta nos boletins de voto
   type Votacao    = [Candidato]   -- cada ocorrência de um candidato representa um voto
   type Escrutinio = [(Candidato,Int)]
   ```

   representando, respectivamente, os candidatos, o boletim de voto, as votações e os resultados de uma contagem.

   (a) Construa uma função `votos :: Candidato -> Votacao -> Int`, que determina o número de votos de um candidato numa dada votação.

   (b) Construa uma função `contagem :: Boletim -> Votacao -> Escrutinio`, que, dada uma votação, apura o escrutínio final associando a cada candidato o seu número de votos.

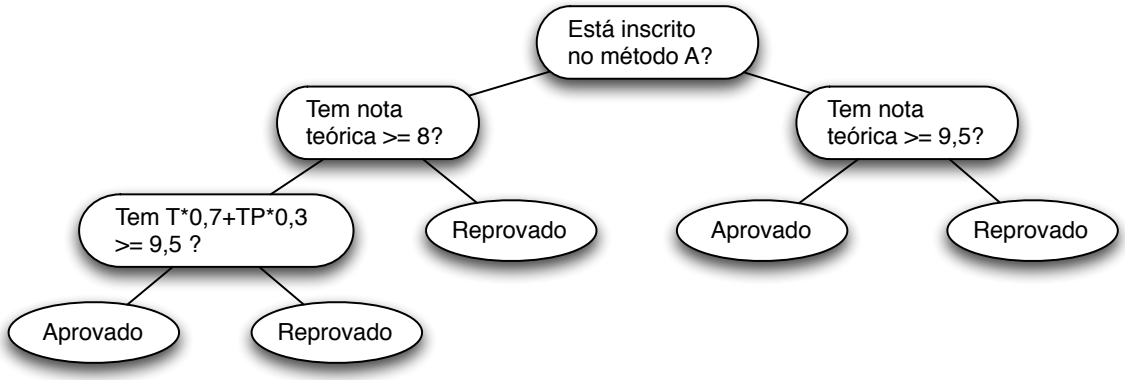
   (c) Construa uma função `estatistica :: Escrutinio -> [(Candidato,Float)]`, que indica a percentagem obtida por cada candidato no final do escrutínio.

Parte II

1. Considere que, para armazenar um questionário cujas respostas às questões são sempre da forma sim/não, se definiu o seguinte tipo de dados:

```
data Questionario a = Resultado a
                    | Questao String (Questionario a) (Questionario a)
```

Assume-se que uma resposta afirmativa à questão implica continuar com o questionário da sub-árvore esquerda, e que uma resposta negativa à questão implica continuar com o questionário da sub-árvore direita. Por exemplo:



   (a) Defina uma função `resp :: [Bool] -> Questionario a -> Maybe a` que dada uma sequência de respostas e um questionário calcula o eventual resultado.

   (b) Defina a função `contaRes` que recebe um questionário e calcula o número de resultados distintos que se pode obter como resposta ao questionário (para o exemplo dado o resultado de `contaRes` deve ser 2). Apresente claramente o tipo desta função Haskell.

   (c) Defina a função `questiona :: Questionario a -> IO a` que dado um questionário vai, de modo interactivo, colocando as questões ao utilizador e, no final, apresenta o resultado encontrado.

2. Considere o seguinte tipo para representar listas.

```
data List a = List Int (Int -> a)
```

Onde por exemplo a lista `['A', 'B', 'C']` pode ser representada por `List 3 (\x->chr(x+65))`. Defina as seguintes funções sobre este tipo:

   (a) `fromList :: (List a) -> [a]` que converte este tipo de listas na notação habitual.

   (b) `toList :: [a] -> List a` que converte a notação habitual nesta notação.

   (c) `rev :: (List a) -> List a` que inverte uma lista