EXAME - 2ª Chamada
04·Fevereiro·2005
Duração: 2:00 horas

Programação Funcional
Paradigmas de Programação I
LMCC

Nº: _____ **NOME:** _____

I

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 Votacao     = [Candidato]
type Escrutinio  = [(Candidato,Int)]
```

representando, respectivamente, os candidatos (identificados por uma *string*), as votações como listas de candidatos (cada ocorrência de um candidato na lista representa um voto) e os resultados de uma contagem como uma lista de associações `(candidato,#votos)`.

1. Construa uma função

   ```
   votos :: Candidato -> Votacao -> Int
   ```

   que determina o número de votos de um candidato numa dada votação.

2. Construa uma função

   ```
   contagem :: Votacao -> Escrutinio
   ```

   que, dada uma votação, apura o escrutínio final associando a cada candidato o seu número de votos.

3. Construa uma função que combine os resultados de dois escrutínios (provenientes, por exemplo, de duas freguesias) num só resultado global.

   ```
   comb :: Escrutinio -> Escrutinio -> Escrutinio
   ```

Nº: _____ NOME: _____

II

Para descrever documentos HTML pode-se utilizar o seguinte tipo:

```
type DocHtml = [Elemento]
data Elemento = S String | T String DocHtml
                deriving (Eq, Show)
```

Por exemplo, o texto HTML

```
<html>
<head>
  <title> Hello </title>
</head>
<body>
  Hello World
</body>
</html>
```

será representado pelo termo

```
hello = T "html" [T "head" [T "title" [S "Hello"]],
                  T "body" [S "Hello World"]      ]
```

1. Escreva uma função

   ```
   strings :: DocHtml -> [String]
   ```

   que determina a lista das *strings* presentes (como argumentos do constructor `S`) num documento HTML. Por exemplo para a expressão acima obter-se-ia a lista

   ```
   ["Hello", "Hello World"]
   ```

2. Defina uma função

   ```
   showHtml :: DocHtml -> String
   ```

   que produz uma *string* a partir de uma expressão do tipo `DocHtml` (por exemplo, a partir de `hello` poder-se-ia obter um texto como o inicialmente apresentado).

3. A representação acima corresponde a uma árvore em que o número de descendentes de cada nodo é variável. Escreva uma função

   ```
   altura :: DocHtml -> Int
   ```

   que calcula a distância entre a raíz da árvore e a folha (constructor `S`) mais profunda da árvore.

Nº: _____ NOME: _____

III

Considere as declarações da classe `FigFechada` e da função `fun` a seguir apresentadas

```
class FigFechada a where
  area :: a -> Float
  perimetro :: a -> Float

fun figs = filter (\fig -> (area fig) > 100) figs
```

1. Indique, justificado, qual é o tipo inferido pelo interpretador Haskell para a função `fun`.

2. No plano cartesiano um rectângulo acom os lados paralelos aos eixos podem ser univocamente determinado pelas coordenadas do vértice inferior esquerdo e pelos comprimentos dos lados, ou por uma diagonal dada por dois pontos. Assim, para representar esta figura geométrica, definiu-se em Haskell o seguinte tipo de dados:

   ```
   type Ponto = (Float,Float)
   type Lado = Float
   data Rectangulo = PP Ponto Ponto
                   | PLL Ponto Lado Lado
   ```

   Declare `Rectangulo` como instância da classe `FigFechada`.

3. Defina a função

   ```
   somaAreas :: [Rectangulo] -> Float
   ```

   que calcula o somatório de uma lista de rectângulos. (De preferência, utilize funções de ordem superior.)