Programação Funcional/Paradigmas da Programação I
1° Ano – LEI/LCC/LESI

Exame da 2ª Chamada

30 de Janeiro de 2007 – Duração: 2 horas

Parte I

Esta parte do exame 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 exame.

1. Assuma que a informação sobre os resultados dos jogos de uma jornada de um campeonato de futebol está guardada na seguinte estrutura de dados:

```
type Jornada = [Jogo]
type Jogo = ((Equipa,Golos),(Equipa,Golos))
type Equipa = String
type Golos = Int
```

   (a) Defina a função `pontos :: Jornada -> [(Equipa,Int)]` que calcula os pontos que cada equipa obteve na jornada (venceu - 3 pontos; perdeu - 0 pontos; empatou - 1 ponto)

   (b) Considere a seguinte função:

```
golosMarcados :: Jornada -> Int
golosMarcados j = sum (map soma) j
```

   Apresente uma Definição para a função `soma` de forma a que a função `golosMarcados` calcule o núnero total de golos marcados numa jornada, e reescreva a função `golosMarcados` sem utilizar as funções `sum` e `map`, e utilizando recursividade primitiva.

2. Defina uma função `filtragem :: (a->Bool) -> [a] -> ([a],[a])` que recebe um predicado e uma lista, e devolve um par que tem na 1ª componente os elementos da lista que satisfazem o predicado e na 2ª componente os elementos da lista que não satisfazem o predicado.

3. Considere a seguinte definição de um tipo de dados polimórfico para árvores binárias:

```
data ArvBin a = Vazia | Nodo a (ArvBin a) (ArvBin a)
```

   Defina a função `folhas :: ArvBin a -> Int` que conta quantas folhas uma árvore. (Chamam-se folhas aos nódos que têm as duas sub-árvores vazias.)

4. Defina a função `ocorr :: String -> [String] -> [Int]` que, dada uma palavra e um texto (representado como uma lista de strings/palavras), devolve a lista com as posições em que essa palavra ocorre no texto. Por exemplo:

```
Prelude> ocorr "isto" ["isto","serve","para","ver","como","isto","funciona"]
[1,6]
```

5. Considere as seguintes definições:

```
type ListaCompras = [(Produto,Quantidade)]
type Produto = (Nome,PrecoKg)
type Nome = String
type PrecoKg = Float
type Quantidade = Float
```

   Escreva uma função `verificaSock :: ListaCompras -> ListaCompras -> Bool` que, dada uma lista de compras de um cliente e um valor do mesmo tipo representando as quantidades em *stock*, retorne um valor que indique se o pedido pode ou não ser satisfeito. Não assuma quaisquer pressupostos sobre o conteúdo das listas ou sobre a ordenação dos seus elementos.