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

17 de Setembro de 2008 – Duração: 2 horas

Época Especial

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 `ultimos3 :: [a] -> (a,a,a)` que recebe uma lista e devolve os três últimos elementos dessa lista.

2. Considere as seguintes definições:

```
unzip3 :: [(a,b,c)] -> ([a],[b],[c])
unzip3 l = (map fst l, map snd l, map trd l)

trd :: (a,b,c) -> c
trd (_,_,z) = z
```

   Apresente uma definição alternativa que não percorra três vezes a lista argumento.

3. Considere a seguinte definição de árvores binárias:

```
data Tree a = Empty | Node a (Tree a) (Tree a)
```

   Defina a função `contaF :: Tree a -> Int` que conta o número de folhas (i.e. ocorrências do construtor `Empty`) numa árvore binária.

4. Para organizar a informação referente ao registo de temperaturas (mínima e máxima de cada dia) e de precipitação, definiram-se os seguintes tipos

```
type Data = (Int,Int,Int)
type TempMin = Int
type TempMax = Int
type Precipitacao = Float
type Registo = (Data , TempMin, TempMax, Precipitacao)
```

   (a) Defina uma função `maxTemp :: [Registo] -> TempMax` que calcula a temperatura mais alta registada.

   (b) Defina uma função `chuva :: [Registo] -> [(Data,Precipitacao)]` que indica as datas em que chuveu e a respectiva precipitação.

   (c) Defina uma função `temps :: [Registo] -> Data -> Maybe (TempMin,TempMax)` que permita saber as temperaturas de um determinado dia.

Parte II

1. Uma forma de representar polinómios de uma variável é usar listas de pares *(coeficiente, expoente)*

   ```
   type Pol = [(Float,Integer)]
   ```

   Note que o polinómio pode não estar simplificado. Por exemplo,

   ```
   [(2.4,3), (3.0,4), (5.2,3), (4.1,5)]  :: Pol
   ```

   representa o polinómio $2.4\,x^3 + 3.0\,x^4 + 5.2\,x^3 + 4.1\,x^5$.

   (a) Defina uma função `simp :: Pol -> Pol` que faça a simplificação de um polinómio, isto é, que dado um polinómio construa um polinómio equivalente ao primeiro em que não podem aparecer varios monómios com o mesmo grau.

   (b) Defina uma função `equiv :: Pol -> Pol -> Bool` que teste se dois polinómios são equivalentes.

2. Considere as declarações dos tipos `Raio`, `Ponto`, `Cor` e `Figura`, e da classe `FigColorida` a seguir apresentadas

   ```
   type Raio = Float
   type Ponto = (Float,Float)

   data Cor = Azul | Verde | Vermelho | Amarelo deriving Eq

   data Figura = Rect Ponto Ponto Cor
               | Circ Ponto Raio Cor

   class FigColorida a where
      area :: a -> Float
      cor :: a -> Cor
   ```

   (a) Declare `Figura` como instância da classe `FigColorida`. (Assuma que a representação do rectângulo é feita por uma diagonal dada por dois pontos.)

   (b) Defina a função `areaPorCor :: [Figura] -> [(Cor,Float)]` que dada uma lista de figuras, calcula a área total (i.e., o somatório) das figuras de cada cor.