Programação Funcional
1º Ano – LEI/LCC

9 de Fevereiro de 2009 – Duração: 2 horas

Exame

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. Defina a função `posImpares :: [a] -> [a]` que recebe uma lista e devolve a lista contendo apenas os elementos que estão nas posições ímpares da lista de entrada.
   Por exemplo, `posImpares ‘‘exemplo’’ = ‘‘eepo’’`.

2. Defina uma função que testa se uma lista tem elementos repetidos. Indique claramente o tipo da função (note que os elementos da lista podem ser de um tipo qualquer).

3. Considere a seguinte definição para árvores binárias:

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

   Defina a função `somaNum :: Int -> Tree Int -> Tree Int`, que soma uma dado valor inteiro a todos os elementos de uma árvore de inteiros.

4. Defina a função `multip :: Int -> [(Int,Int,Int)] -> [(Int,Int,Int)]` que recebe um inteiro n e uma lista de triplos, e seleciona da lista apenas os triplos cujo resultado da soma das 3 partes é múltiplo de n. Por exemplo, `multip 5 [(2,4,6),(3,2,10),(4,2,4),(1,4,8)] = [(3,2,10),(4,2,4)]`

5. Considere as seguintes definições de tipos de dados para representar uma tabela de abreviaturas

   ```
   type TabAbrev = [(Abreviatura,Palavra)]
   type Abreviatura = String
   type Palavra = String
   ```

   (a) Defina a função `daPal :: TabAbrev -> Abreviatura -> Maybe Palavra`, que dadas uma tabela de abreviaturas e uma abreviatura, devolve a palavra correspondente.

   (b) Analise a seguinte função que pretende transformar um texto, substituindo as abreviaturas que lá ocorrem pelas palavras correspondentes.

   ```
   transforma :: TabAbrev -> String -> String
   transforma t s = unwords (trata t (words s))
   ```

   Apresente uma definição adquada para a função `trata` e indique o seu tipo.

Parte II

1. Considere que a particiapação dos alunos nas aulas TP está registada num ficheiro de texto com o seguinte formato: um número de aluno por linha, e qualquer linha de comentário começa por '-' . Por exemplo:

```
------Aula 9------
47086
47094
51831
--
51763
49349
49403
------Aula 10------
--
51763
51834
49403
```

   Considera ainda que já tem o conteúdo desse ficheiro de texto numa string. Defina a função `participou :: String -> [(String, Int)]`, que indica o número de participações de cada aluno.

2. Considere uma *árvore genealógica ascendente* (que relaciona uma pessoa com os seus pais) implementada no seguinte tipo de de dados:

```
data AG = Pessoa Nome Pai Mae
        | Desconhecida
  deriving Show

type Pai = AG
type Mae = AG
type Nome = String
```

   (a) Defina a função `nomesMF :: AG -> ([Nome],[Nome])` que para a árvore genealógica de uma pessoa calcula as listas com os nomes masculinos e os nomes femininos dos seus ascendentes.

   (b) Defina a função `avos :: Nome -> AG -> [Nome]` que recebe o nome de uma pessoa e uma árvore genealógica e dá a lista com os nomes dos avós dessa pessoa. Note que o nome da pessoa pode não estar na raiz da árvore. Assuma que não existem nomes repetidos na árvore.

   (c) Defina a função `grau :: AG -> Nome -> Maybe Int` que dada uma árvore genealógica e o nome de uma pessoa, calcula o grau de parentesco entre essa pessoa e a pessoa que está na raiz da árvore. Assuma o grau de parentesco de pais é 1, de avós é 2, de bisavós é 3, etc ... Assuma ainda que não existem nomes repetidos na árvore.