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

12 de Fevereiro de 2008 – Duração: 2 horas

Exame

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. Considere a seguinte definição de árvores binárias:

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

   Defina a função `menoresMaiores :: Float -> Tree Float -> ([Float],[Float])` que dados um número n e uma árvore binária, devolve um par de listas com os elementos da árvore que são, respectivamente, menores e maiores do que n.

2. Defina a função `ePrefixo :: String -> String -> Bool` que dadas duas strings testa se a primeira string é um prefixo da segunda. Por exemplo `ePrefixo ‘‘mal’’ ‘‘maldade’’ = True` e `ePrefixo ‘‘normal’’ ‘‘anormal’’ = False`.

3. Considere que para representar os movimentos de uma conta bancária se definiram os seguintes tipos

   ```
   type Extracto = [Movimento]
   type Movimento = (Descricao, Tipo, Data, Montante)
   type Descricao = String
   data Tipo = Credito | Debito   deriving Eq
   type Ano = Int
   type Mes = Int
   type Dia = Int

   data Data = D Ano Mes Dia
     deriving (Eq,Ord)

   type Montante = Float    -- valor positivo
   ```

   (a) Definia a função `procura :: Descricao -> Extracto -> [(Tipo,Data,Montante)]` que colecciona do extracto informação relativa aos movimentos que correspondem uma dada descrição.

   (b) Defina a função `saldo :: Extracto -> Float` que dado extracto determina o saldo final da conta.

   (c) Defina a função `porOrdem :: Extracto -> Bool` que teste se no extracto os movimentos estão organizados por ordem crescente de data.

   (d) A função `dmaxDebito`, a seguir apresentada, calcula a data e o montante do maior débito de um extracto.

   ```
   dmaxDebito :: Extracto -> Maybe (Data,Montante)
   dmaxDebito ((_,Debito,d,m):t) = maxdeb (d,m) (dmaxDebito t)
   dmaxDebito (h:t) = dmaxDebito t
   dmaxDebito [] = Nothing
   ```

   Apresente a definição de `maxdeb` e indique claramente o seu tipo.

Parte II

1. Recorde a definição de tipos usada, na Parte I, para representar os movimentos de uma conta bancária.

```
type Extracto = [Movimento]
type Movimento = (Descricao, Tipo, Data, Montante)
type Descricao = String
data Tipo = Credito | Debito   deriving Eq
type Ano = Int
type Mes = Int
type Dia = Int
data Data = D Ano Mes Dia   deriving (Eq,Ord)
type Montante = Float
```

   (a) Defina a função `relatorio :: Extracto -> String` que faz conversão de um extracto de conta numa string, de tal forma que o relatório do extracto tenha o seguinte aspecto, quando impresso

```
Data       Descricao   Credito   Debito
---------------------------------------
2007/4/5   DEPOSITO    2000
2007/8/10  COMPRA                37,5
2007/9/1   LEV                   60
2008/1/7   JUROS       100
2008/1/22  ANUIDADE              8
--------------
Saldo: 1994,5
```

   (b) Assumindo que os vários movimentos de um extracto aparecem por ordem cronológica, defina a função `aDescoberto :: Extracto -> [(Data,Montante)]` que determina todas as datas (e os respectivos montantes) em que a conta esteve com saldo negativo.

2. A sequência 1, 11, 21, 1211, 111221, 312211, ... pode ser construída a partir do primeiro número – 1– seguido da descrição do número anterior:

   - 11 pode ser lido como *um 1*
   - 21 pode ser lido como *dois 1's*
   - 1211 pode ser lido como *um 2, um 1*
   - 111221 pode ser lido como *um 1, um 2's, dois 1*
   - 312211 pode ser lido como *três 1's, dois 2's, um 1*

   Assim, na sequência apresentada, o próximo elemento seria 13112221, correspondendo à descrição *um 3, um 1, dois 2's, dois 1's.*

   (a) Escreva uma função `proximo :: Integer -> Integer` que dado um número da sequência, calcula o próximo (segundo o método descrito).

   (b) Defina agora a função inversa `anterior :: Integer -> Integer`, que dado um elemento da sequência (com excepção do primeiro) dá o anterior. (Note que, à excepção do primeiro elemento da sequência, todos os elementos têm um número par de algarismos).