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

Exame da 1ª Chamada – Duração: 2 horas

15 de Janeiro de 2007

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 uma função que recebe uma lista constrói o par com o primeiro e o último elemento da lista, sem recorrer a funções pré-definidas.
2. Defina a função `unzip :: [(a,b)] -> ([a],[b])` que recebe uma lista de pares e produz um par com as listas constituídas pelas 1ª e 2ª partes dos pares da lista de entrada.
3. Considere a seguinte definição de um tipo de dados polimórfico para árvores binárias:

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

   Defina a função `altura :: BTree a -> Int` que calcula a altura de uma árvore.
4. Considere a seguinte definição de um tipo para representar fracções

   ```
   data Fraccao = Frac Integer Integer   -- numerador denominador
   ```

   Defina a função `prodFracs :: [Fraccao] -> Fraccao` que calcula o produto de uma lista de fracções.
5. Considere as seguintes definições:

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

   precoItens :: ListaCompras -> [(Nome,Float)]
   precoItens [] = []
   precoItens (x:xs) = let a = fst (fst x)
                           b = (snd (fst x)) * (snd x)
                       in (a,b):precoItens xs

   produtosCaros :: ListaCompras -> PrecoKg -> ListaCompras
   produtosCaros lista limite = filter (eCaro limite) lista
   ```

   (a) Explique o papel da função `eCaro` na definição da função `produtosCaros` e apresente uma definição para essa função, incluindo o seu tipo.

   (b) Reescreva a função `precoItens` utilizando uma função de ordem superior, e sem utilizar as funções `fst` e `snd`.

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

Exame da 1ª Chamada

15 de Janeiro de 2007 – Duração: 2 horas

Parte II

Considere as seguintes definições para representar calendários de exames.

```
type Calendario = [Exame]
data Exame = Ex Nome Data Tipo
type Nome = String
data Data = D Ano Mes Dia Periodo deriving Eq
type Ano = Int
data Mes = Jan | Fev | Mar | Abr | Mai | Jun
         | Jul | Ago | Set | Out | Nov | Dez
         deriving (Eq,Enum,Ord,Show)
type Dia = Int
data Periodo = Manha | Tarde
data Tipo = Primeira | Segunda | Recurso

diasMes :: [(Mes,Int)]
diasMes = [ (Jan,31) , (Fev,28) , (Mar,31) , (Abr,30) , (Mai,31) , (Jun,30) ,
            (Jul,31) , (Ago,31) , (Set,30) , (Out,31) , (Nov,30) , (Dez,31) ]
```

1. Defina `Data` como instância da classe `Ord` (note que apenas tem de definir a função `<=` ou a função `compare`).
2. Defina uma função que, dado o nome de uma disciplina e um calendário de exames, calcule a lista das datas associadas aos exames dessa disciplina.
3. Defina uma função que, dadas duas datas de exames, determina o número de dias de intervalo (excluindo os próprios dias dos exames) entre elas. Assuma que o número de dias de cada mês é o indicado na constante `diasMes`.
4. Pretende-se testar a consistência de um calendário. Nomeadamente assegurar que:
   - existem 3 exames para cada disciplina
   - a 2ª chamada ocorre depois da 1ª e antes do recurso
   - o exame de recurso ocorre pelo menos 1 semana depois do da 2ª chamada.

   Defina uma função que, dada a lista de disciplinas e um calendário, dá como resultado as disciplinas que não satisfazem alguma das restrições acima.