Programação Funcional
1° Ano – LEI/LCC

26 de Janeiro de 2009 – Duração: 2 horas

Teste

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 `paresImpares :: [Int] -> ([Int],[Int])` que recebe uma lista de inteiros e devolve um par de listas: uma contendo apenas os números pares e a outra os números impares da lista de entrada.

2. Defina a função `sufixo :: (Eq a) => [a] -> [a] -> Bool` que testa se uma lista é sufixo da outra. Por exemplo, `sufixo ``comodo ``incomodo'' = True` e `sufixo ``com'' ``incomodo'' = False`.

3. Considere a seguinte definição para árvores binárias:

   ```
   data ABin a = Vazia | No a (ABin a) (ABin a)
   ```

   Defina a função `altura :: ABin a -> Int`, que calcula a altura de uma árvore.

4. Considere as seguintes definições de tipos de dados para representar uma tabela de preços em euros e uma tabela de cotações de diversas moedas face ao euro.

   ```
   type TabPrecos = [(Produto,Preco)]  -- preço em euros
   type Produto = String
   type Preco = Float

   type Cotacoes = [(Moeda,Valor)]     -- das diversas moedas face ao euro
   type Moeda = String
   type Valor = Float
   ```

   (a) Defina a função `maiorValor :: Cotacoes -> (Moeda,Valor)`, que indica a moeda de maior valor na tabela de cotações.

   (b) Defina a função `convPrecos :: TabPrecos -> Cotacoes -> Moeda -> TabPrecos` converte a tabela de preços para uma dada moeda.

   (c) Analise a seguinte função que pretende gerar uma tabela de preços em várias moedas:

   ```
   precosMultiMoeda :: TabPrecos -> [Moeda] -> Cotacoes -> [(Produto,[(Moeda,Preco)])]
   precosMultiMoeda t m c = map fun t
     where fun (n,p) = (n, aux p m c)
   ```

   e apresente uma definição adquada para a função `aux` e indique o seu tipo.

Parte II

Considere as seguintes definições de tipos de dados para representar um horário semanal.

```
type Horario = [(Dia,[(Hora,Hora,Actividade)])]

data Dia = Segunda | Terca | Quarta | Quinta | Sexta | Sabado | Domingo
  deriving (Eq, Ord)

type Hora = (Int,Int)    -- horas, minutos

data Actividade = Aula Disciplina Tipo Sala
                | Outra String
  deriving Eq

type Disciplina = String
data Tipo = T | TP | PL   deriving Eq
type Sala = String
```

Para a definição das funções que a seguir são pedidas, assuma que o horário em causa está bem construido, isto é:

– todos os dias da semana estão presentes no horário e não estão repetidos;
– os dias da semana aparecem ordenadamente na lista;
– as horas que estão presentes no horário são horas válidas;
– a hora de inicio de uma actividade é sempre inferior à hora do seu fim;
– em cada dia, as actividades aparecem por ordem crescente do seu início.

1. Defina a função `consulta :: Dia -> Hora -> Horario -> Maybe Actividade`, que indica qual é a actividade num dado dia e hora.

2. Defina a função `aulasPorDia :: Horario -> [(Dia,Hora)]`, que calcula o tempo total de aulas em cada dia da semana.

3. Defina a função `livre :: Dia -> (Hora,Hora) -> Horario -> Bool`, que testa se entre um determinado periodo de tempo, num dado dia, o horário está livre.

4. Defina a função `alerta :: Horario -> [(Dia,Sobreposicao)]`, que assinala todas as sobreposições de actividades de um dado horário. Note que uma actividade pode estar em sobreposição a mais do que uma actividade. Considere a seguinte declaração de tipo.

   ```
   type Sobreposicao = ((Hora,Hora,Actividade),(Hora,Hora,Actividade))
   ```