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

29 de Janeiro de 2008 – 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 `parte :: Int -> [Int] -> ([Int],[Int],[Int])` que dados um inteiro n e uma lista de inteiros, devolve o tuplo com três listas contendo os elementos da lista de entrada que são, respectivamente, menores, iguais e maiores do que n.

2. Defina a função `merge :: [Float] -> [Float] -> [Float]` que recebe duas listas ordenadas por ordem crescente e junta as duas listas de forma a que a lista produzida fique ordenada por ordem crescente.

3. Considere a seguinte definição para árvores binárias:

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

   Defina a função `semMin :: ABin Int -> (Int, ABin Int)` que dada uma árvore binária de procura **não vazia**, devolve um par com o valor mínimo da árvore e a nova árvore sem esse valor (mínimo).

4. Considere as seguintes definições de tipos de dados para representar filmes:

   ```
   type Filme = (Titulo,Realizador,[Actor],Genero,Ano)
   type Titulo = String
   type Realizador = String
   type Actor = String
   type Ano = Int

   data Genero = Comedia | Drama | Ficcao | Accao | Animacao | Documentario
      deriving Eq

   type Filmes = [Filme]
   ```

   (a) Defina a função `doRealizador :: Filmes -> Realizador -> [Titulo]` que lista o nome dos filmes realizados por um dado realizador.

   (b) Defina a função `doActor :: Filmes -> Actor -> [Titulo]` que lista o nome dos filmes em que um dado actor participa.

   (c) Analise a seguinte definição

   ```
   consulta :: Filmes -> Genero -> Realizador -> [(Ano, Titulo)]
   consulta bd gen rea = map aux (filter (teste gen rea) bd)
     where teste ::  Genero -> Realizador -> Filme -> Bool
           teste g r (_,x,_,y,_) = g==y && r==x
   ```

   e apresente a definição de `aux`, incluindo o seu tipo.

Parte II

Recorde a definição de tipos usada para a representação de filmes (apresentada na Parte I). Pretende-se agora enriquecer a base de dados de filmes, associando a cada filme uma lista de avaliações com classificações do filme. Para isso definiram-se os tipos de dados `Avaliacao` e `FilmesAval`.

```
type Filme = (Titulo,Realizador,[Actor],Genero,Ano)
type Titulo = String
type Realizador = String
type Actor = String
type Ano = Int

data Genero = Comedia | Drama | Ficcao | Accao | Animacao | Documentario
   deriving Eq

type Filmes = [Filme]

data Avaliacao = NaoVi
               | Pontos Int

type FilmesAval = [(Filme,[Avaliacao])]
```

1. Declare `Avaliacao` como instância da classe `Ord`

2. Declare a função `grafico :: Titulo -> FilmesAval -> String` que faz a representação das avaliações de um dado filme num gráfico de barras. Assuma que a pontuação de uma avaliação só pode ir de 1 a 5. Cada barra do gráfico (uma por linha) deve ser uma sequência de * de comprimento igual ao número de avaliações com uma dada pontuação. Por exemplo:

   ```
   1 ***
   2
   3 ****
   4 ***
   5 **
   ```

3. Defina a função `listaPorGeneros :: FilmesAval -> [(Genero,[(Titulo,Avaliacao)])]` que calcula a avaliação de cada filme, e apresenta o resultado organizado por género. A avaliação deve ser a média das classificações, e `NaoVi` não deve contar para a média.

4. Defina a função `avalia :: FilmesAval -> IO FilmesAval` que lê o título de um filme e a sua avaliação e acrescenta essa avaliação à base de dados de filmes. A escala de classificação deve ser de 1 a 5.