Programação Funcional

2012/13

Mini-testes 4

1. (a) Considere o seguinte tipo para representar árvores binárias de procura:

   ```
   data BTree a = Vazia | Nodo a (BTree a) (BTree a)
   ```

   Apresente uma definição da função `filtra :: (a->Bool) -> (BTree a) -> [a]` que coloca numa lista todos os elementos da árvore que satisfazem uma dada condição.

   (b) Para armazenar uma agenda de contactos telefónicos e de correio electrónico definiram-se os seguintes tipos de dados. Não existem nomes repetidos na agenda e para cada nome existe uma lista de contactos.

   ```
   data Contacto = Casa Integer
                 | Trab Integer
                 | Tlm Integer
                 | Email String

   type Nome = String
   type Agenda = [(Nome, [Contacto])]
   ```

   i. Defina a função `acrescEmail :: Nome -> String -> Agenda -> Agenda` que, dado um nome, um email e uma agenda, acrescenta essa informação à agenda.

   ii. Defina a função `verEmails :: Nome -> Agenda -> Maybe [String]` que, dado um nome e uma agenda, retorna a lista dos emails associados a esse nome. Se esse nome não existir na agenda a função deve retornar `Nothing`.

2. Considere o seguinte tipo:

   ```
   data LTree a = Leaf a | Fork (LTree a) (LTree a)
   ```

   (a) Defina uma instância da class `Show` para este tipo por forma a obter o seguinte comportamento.

   ```
   > Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2)
   ((0/\1)/\2)
   ```

   (b) Defina a função `mktree :: Int -> a -> LTree a` que constroi uma árvore balanceada com um dado número de folhas todas iguais ao segundo argumento. Por exemplo,

   ```
   > mktree 5 0
   ((0/\0)/\(0/\(0/\0)))
   ```

3. Considere que para representar ainformação dos concorrentes do Dakar 2013 se utilizou o seguinte tipo de dados:

```
type Dakar = [Piloto]

data Piloto = Carro  Numero Nome Categoria
            | Mota   Numero Nome Categoria
            | Camiao Numero Nome
type Numero = Int
type Nome   = String
data Categoria = Competicao | Maratona
```

   (a) Escreva uma função que verifica se existem, ou não, dois pilotos com o mesmo número.

   (b) Assuma que a lista está ordenada pelo número do piloto, escreva a função `inserePil :: Piloto -> Dakar -> Dakar` que insere ordenadamente um novo piloto na lista.

   (c) Considere que para melhorar a procura de um piloto se utilizou uma árvore binária de procura, ordenada pelo número.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
type Dakar = BTree Piloto
```

   Escreva uma função `menor :: Dakar -> Piloto` que calcula o piloto **com número menor**.

4. (a) Defina uma função `minimo :: (Ord a) => BTree a -> a` que calcula o menor elemento de uma árvore binária de procura **não vazia**.

   (b) Defina uma função `semMinimo :: (Ord a) => BTree a -> BTree a` que remove o menor elemento de uma árvore binária de procura **não vazia**.

   (c) Defina uma função `minSmin :: (Ord a) => BTree a -> (a,BTree a)` que calcula, com uma única consulta da árvore o resultado das duas funções anteriores.

5. (a) Considere o seguinte tipo para representar árvores binárias de procura:

```
data ArvBin a = Vazia | Nodo a (ArvBin a) (ArvBin a)
```

   Defina uma função `minAB :: ArvBin Int -> Maybe Int` que, dada uma árvore de procura, calcule o menor elemento dessa árvore, caso exista.

   (b) Resolveu-se criar uma base de dados de vídeos e, para isso, classificaram-se os vídeos em: filmes, episódios de séries, shows, e outros. Para esse fim, definiram-se os seguintes tipos de dados.

```
type BD = [Video]
data Video = Filme String Int       -- título, ano
           | Serie String Int Int   -- título, temporada, episódio
           | Show  String Int       -- título, ano
           | Outro String
```

   i. Defina a função `espectaculos :: BD -> [(String,Int)]` que indica o título e ano de todos os espetáculos da base de dados.

   ii. Defina a função `filmesAno :: Int -> BD -> [String]` que indica os títulos dos filmes de um dado ano que existem na base de dados.

6. Considere o seguinte tipo:

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)
```

   (a) Defina uma instância da class `Eq` para este tipo idêntica à que seria obtida com a directiva `deriving Eq`.

   (b) Defina a função `mapLT :: (a -> b) -> LTree a -> LTree b` que aplica uma função a todas as folhas de uma árvore. Por exemplo,

```
> mapLT succ (Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2))
Fork (Fork (Leaf 1) (Leaf 2)) (Leaf 3)
```

7. Considere que para organizar os seus livros foi definido o seguinte tipo de dados:

```
type Biblio = [Livro]

data Livro = Romance Titulo Autor Ano Lido
           | Ficcao  Titulo Autor Ano Lido
type Titulo = String
type Autor  = String
type Ano    = Int
data Lido   = Sim
            | Nao
```

   (a) Escreva uma função que verifica se existem, ou não, livros repetidos.

   (b) Defina a função `lido :: Biblio -> Titulo -> Biblio` que marca um dado livro como lido na biblioteca.

   (c) Considere que para melhorar a procura de um livro se utilizou uma árvore binária de procura, ordenada pelo título.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
type Biblio = BTree Livro
```

   Escreva uma função `LivroAutor :: Biblio -> Autor -> [Livro]` que devolve a lista de livros do autor dado.

8. (a) Considere o seguinte tipo para representar árvores binárias de procura:

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
```

   Apresente uma definição da função `mapBT :: (a->b) -> (BTree a) -> (BTree b)` que aplica uma função a todos os elementos de uma árvore binária.

   (b) Para armazenar uma agenda de contactos telefónicos e de correio electrónico definiram-se os seguintes tipos de dados. Não existem nomes repetidos na agenda e para cada nome existe uma lista de contactos.

```
data Contacto = Casa Integer
              | Trab Integer
              | Tlm Integer
              | Email String

type Nome = String
type Agenda = [(Nome, [Contacto])]
```

i. Defina a função `consTelefs :: [Contacto] -> [Integer]` que, dada uma lista de contactos, retorna a lista de todos os números de telefone dessa lista (tanto telefones fixos como telemóveis).

ii. Defina a função `casa :: Nome -> Agenda -> Maybe Integer` que, dado um nome e uma agenda, retorna o número de telefone de casa (caso exista).

9. Considere o seguinte tipo:

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)
```

(a) Defina uma instância da class `Eq` para este tipo que considere iguais quaisquer duas árvores com a mesma forma.

(b) Defina a função `build :: [a] -> LTree a` que constroi uma árvore com uma folha por cada elemento da lista argumento (que deve ser não vazia).

10. (a) Considere o seguinte tipo para representar árvores binárias de procura:

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

Defina uma função `maxBT :: BTree Float -> Maybe Float` que, dada uma árvore de procura, calcule o maior elemento dessa árvore, caso exista.

(b) Resolveu-se criar uma base de dados de vídeos e, para isso, classificaram-se os vídeos em: filmes, episódios de séries, shows, e outros. Para esse fim, definiram-se os seguintes tipos de dados.

```
type BD = [Video]
data Video = Filme String Int       -- título, ano
           | Serie String Int Int   -- título, temporada, episódio
           | Show  String Int       -- título, ano
           | Outro String
```

i. Defina a função `outros :: BD -> BD` que seleciona da base de dados todos os vídeos que <u>não</u> são filmes, séries ou shows.

ii. Defina a função `totalEp :: String -> BD -> Int` que indica quantos episódios de uma dada série existem na base de dados.

11. Considere o seguinte tipo:

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)
```

(a) Defina uma instância da class `Show` para este tipo que apresente uma folha por cada linha, precedida de tantos pontos quanta a sua profundidade na árvore.

```
> Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2)
..0
..1
.2
```

Sugere-se que use uma função auxilar na definição da função `show`.

(b) Defina a função `cresce :: LTree a -> LTree a` que cresce uma árvore duplicando todas as suas folhas. Por exemplo,

```
> cresce (Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2))
...0
...0
...1
...1
..2
..2
```

12. Considere que para representar ainformação dos concorrentes do Dakar 2013 se utilizou o seguinte tipo de dados:

```
type Dakar = [Piloto]

data Piloto = Carro  Numero Nome Categoria
            | Mota   Numero Nome Categoria
            | Camiao Numero Nome
type Numero = Int
type Nome   = String
data Categoria = Competicao | Maratona
```

    (a) Assuma que a lista está ordenada pelo nome do piloto, escreva a função `inserePil :: Piloto -> Dakar -> Dakar` que insere ordenadamente um novo piloto na lista.

    (b) Escreva uma função que verifica se a lista está ou não ordenada.

    (c) Considere que para melhorar a procura de um piloto se utilizou uma árvore binária de procura, **ordenada pelo número**.

    ```
    data BTree a = Vazia | Nodo a (BTree a) (BTree a)
    type Dakar = BTree Piloto
    ```

    Escreva uma função `maior :: Dakar -> Piloto` que calcula o piloto **com número maior**.

13. (a) Defina uma função `maximo :: (Ord a) => BTree a -> a` que calcula o maior elemento de uma árvore binária de procura **não vazia**.

    (b) Defina uma função `semMaximo :: (Ord a) => BTree a -> BTree a` que remove o maior elemento de uma árvore binária de procura **não vazia**.

    (c) Defina uma função `maxSmax :: (Ord a) => BTree a -> (a,BTree a)` que calcula, com uma única consulta da árvore o resultado das duas funções anteriores.

14. Considere que para organizar os seus livros foi definido o seguinte tipo de dados:

```
type Biblio = [Livro]

data Livro = Romance Titulo Autor Ano Lido
           | Ficcao  Titulo Autor Ano Lido
type Titulo = String
type Autor  = String
type Ano    = Int
data Lido   = Sim
            | Nao
```

    (a) Escreva uma função que calcula o número de livros lidos.

(b) Considere que a lista de livros está ordenada por ordem alfabética consoante o seu autor. Escreva uma função `compra :: Titulo -> Autor -> Ano -> Biblio -> Biblio` que insere ordenadamente o livro comprado (e não lido) na biblioteca.

(c) Considere que para melhorar a procura de um livro se utilizou uma árvore binária de procura, **ordenada pelo título**.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
type Biblio = BTree Livro
```

Escreva uma função `naoLidos :: Biblio -> [Livros]` que devolve a lista de livros não lidos (considere a possibilidade desta lista estar ordenada pelo título do livro).