EXAME - 1ª Chamada
21·Janeiro·2005
Duração: 2:00 horas

Programação Funcional
Paradigmas de Programação I
LMCC

Nº: _____ **NOME:** _____

I

Multiconjuntos ("bags" em inglês) são colecções finitas de objectos, análogos a *conjuntos* mas que admitem a possibilidade de repetição de elementos; distinguem-se de *sequências* pelo facto de neles ser irrelevante a ordem entre os elementos.

Por exemplo: $\{1,2,2,1\}$, $\{2,1,2,1\}$ e $\{2,1,1,2\}$ denotam o mesmo multiconjunto.

A seguinte definição de classe pretende representar as operações básicas sobre multiconjuntos: `nada` é o multiconjunto vazio, `adic` adiciona um elemento a um multiconjunto e `iter` é o iterador genérico sobre os elementos de um multiconjunto.

```
class Bag b a  where
     nada :: b a
     adic :: a -> b a -> b a
     iter :: (a -> x -> x) -> x -> b a -> x
```

1. Defina, a partir destas funções básicas, duas funções derivadas com as assinaturas seguintes

   ```
   card  :: (Eq a, Bag b a) => a -> b a -> Int
   uniao :: Bag b a => b a -> b a -> b a
   ```

   que implementam, respectivamente, a "cardinalidade" de um elemento dentro de um multiconjunto (isto é, o número de cópias desse elemento que ocorrem no multiconjunto), e a "união" de dois multiconjuntos.

2. Porque é indispensável na função `card` que o tipo `a` seja uma instância de `Eq` sem que essa restrição seja também imposta na função `uniao`?

3. Crie uma função que, dado um multiconjunto, construa a lista de todos os seus elementos distintos.

Nº: _____ NOME: _____

II

Considere os seguintes tipos de dados:

```
data ArvBin a = Vazia | Nodo a (ArvBin a) (ArvBin a)
type TList a = [(a, ArvBin a)]
```

1. Escreva funções recursivas

   ```
   contaF :: ArvBin a -> Int
   contaFolhas :: TList a -> Int
   ```

   em que `contaF` conta o número de folhas (i.e. ocorrências do construtor `Vazia`) numa árvore binária e `contaFolhas` conta o número total de folhas de árvore presentes numa `TList`.

2. Escreva agora uma nova versão `contaFolhas'` da função anterior recorrendo a um acumulador.

   ```
   contaFolhas' :: TList a -> Int -> Int
   ```

3. Com base no seguinte exemplo, defina uma função de conversão do tipo `TList` em árvores binárias.

   ```
   converte :: TList a -> ArvBin a
   ```

   Quando aplicada à lista:

   $$[(1, (Nodo\ 2\ (Nodo\ 4\ Vazia\ Vazia)\ (Nodo\ 5\ Vazia\ Vazia))),\\ (3, (Nodo\ 6\ Vazia\ Vazia)),\\ (7, Vazia)]$$

   resulta na seguinte árvore:

   $$(Nodo\ 1\ (Nodo\ 2\ (Nodo\ 4\ Vazia\ Vazia)\ (Nodo\ 5\ Vazia\ Vazia))\\ (Nodo\ 3\ (Nodo\ 6\ Vazia\ Vazia)\ (Nodo\ 7\ Vazia\ Vazia)))$$

   (Sugestão: desenhe a árvore e verifique que a lista inicial é a sua "espinha" da direita).

Nº: _____ NOME: _____

III

Para implementar uma agenda telefónica definiram-se os seguintes tipos

```
type Nome = String
type Telefone = Integer
type Agenda = [(Nome,[Telefone])]
```

A agenda deve estar organizada por ordem alfabética. Um exemplo de uma agenda seria

```
agenda = [("Ana",[912345678, 253123123]), ("Andre",[931222333]),
          ("Pedro", [933111777, 912999888]),
          ("Rui",[226111888, 965553331, 212333555])]
```

1. Defina a função `insere :: Agenda -> Nome -> Telefone -> Agenda`
   que dada uma agenda, um nome e um número de telefone, acrescenta esta informação à agenda.

2. Defina a função `acrescenta :: Agenda -> IO Agenda`
   que, dada uma agenda, lê do teclado um nome e um número de telefone, e acrescenta esta informação à agenda.

3. Use uma função de ordem superior para definir a função

   ```
   consulta :: Agenda -> Char -> Agenda
   ```

   que dada uma agenda e um caracter devolve a (sub)agenda correspondente aos nomes iniciados por esse caracter.

4. Defina a função `apresenta :: Agenda -> IO ()`
   que dada uma agenda, produz uma listagem com o seguinte aspecto e a apresente no écran.

   ```
   Ana     912345678 -- 253123123
   Andre   931222333
   Pedro   933111777 -- 912999888
   Rui     226111888 -- 965553331 -- 212333555
   ```