Ciências ULisboa | Informática

Princípios de Programação
Exame Um
10 janeiro 2020

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, os sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* **`Prelude`**.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Funções básicas, recursão e funções de ordem superior. 3 valores]

a) Escreva uma função chamada `primeiroUltimo` que receba uma lista e que devolva o primeiro ou o último elemento da lista, dependendo da lista ter um número par ou impar de elementos respectivamente.

b) A função `ultimo` devolve o último elemento de uma lista não vazia. Implemente a função de forma recursiva.

c) Escreva uma função que calcule o maior e o menor elemento de uma não lista vazia de elementos comparáveis (instâncias da classe de tipos **`Ord`**), *efetuando uma só passagem na lista* e sem usar as funções **`min`** nem **`max`**. Utilize *recursão*. Exemplo:

```
ghci> minmax [1,5,8,5,3,2,7,9,5,3]
(1,9)
```

d) Refaça o exercício **c)** sem usar recursão e sem usar as funções **`min`** nem **`max`**.

Grupo 2. [Tipos de dados abstratos. 3 valores]

Considere as seguintes definições de tipos, parte de um sistema de gestão da frota do metropolitano de sua cidade preferida (a nossa é Lisboa).

```
-- Um Metro (comboio) é caracterizado pelo número de
   carruagens
data Metro = Metro Int deriving Show
-- Uma linha de metro é representada por uma String
type Linha = String
```

```
-- A Alocação descreve que metros estão atribuídos a
   que linhas e que metros estão livres
data Alocacao = Alocacao [(Metro, Linha)] [Metro]
   deriving Show
```

Por exemplo, o valor

```
ghci> let umaAloc =
  Alocacao [(Metro 6, "Amarela")] [Metro 3, Metro 5]
```

descreve uma frota com um metro de 6 carruagens alocado à linha Amarela e dois metros de 3 e de 5 carruagens que não estão alocados a nenhuma linha.

a) Frequentemente necessitamos de saber se existe um metro com um número mínimo de carruagens disponível (por exemplo, para alocar a uma linha necessitada). O predicado

```
existeMetroLivre :: Int -> Alocacao -> Bool
```

faz exatamente isso. Por exemplo:

```
ghci> existeMetroLivre 6 umaAloc
False
ghci> existeMetroLivre 4 umaAloc
True
```

Escreva a função `existeMetroLivre`.

b) Por vezes necessitamos de colocar um metro com *pelo menos* um dado número de carruagens numa dada linha. Para isso retiramos um metro apropriado da lista dos metros livres e alocamo-lo à linha pretendida. A função

```
alocaMetro :: Int -> Linha ->  Alocacao ->  Alocacao
```

calcula a nova alocação. Assumimos que a alocação é sempre possível, isto é, que uma chamada `alocaMetro n linha alocacao` só é efectuada quando `existeMetroLivre n alocacao` for verdade. Por exemplo:

```
ghci> alocaMetro 4 "Verde" $ alocaMetro 2 "Azul" umaAloc
Alocacao [(Metro 5,"Verde"),(Metro 3,"Azul"),
  (Metro 6,"Amarela")] []
```

Escreva a função `alocaMetro`.

c) Dizemos que duas frotas são iguais se tiverem o mesmo número de carruagens. Torne o tipo de dados `Alocacao` uma instancia da classe de tipos **`Eq`**. Por exemplo

```
ghci> umaAloc == Alocacao [] [Metro 5, Metro 2, Metro 7]
True
```

Grupo 3. [Teste. 3 valores]

a) De modo a testar as funções do módulo de gestão de um metropolitano, torne o tipo de dados `Alocacao` instancia da classe de tipos `Arbitrary`.

b) Escreva a seguinte propriedade `QuickCheck`:

> Se existe um metro livre (com um dado número de carruagens) numa dada `Alocacao`, então a lista de metros disponíveis nessa `Alocacao` não é vazia.

c) Escreva a seguinte propriedade `QuickCheck`:

> A alocação de um metro a uma dada linha não altera o número de carruagens na alocação.

Grupo 4. [Raciocínio sobre programas. 3 valores]

A função `intercalar` recebe um elemento separador `x` e uma lista `ys` e devolve uma nova lista onde `x` é colocado entre cada um dos elementos de `ys`. Por exemplo:

```
ghci> intercalar '.' "Janelas do meu quarto"
"J.a.n.e.l.a.s. .d.o. .m.e.u. .q.u.a.r.t.o"
```

Dada a definição

```
intercalar :: a -> [a] -> [a]
intercalar _ [] = []
intercalar _ [y] = [y]
intercalar x (y:ys) = y : x : intercalar x ys
```

mostre que

$$\texttt{length (intercalar x ys)} = \begin{cases} \texttt{2 * length ys - 1} & \text{se } \texttt{length ys > 0} \\ \texttt{0} & \text{caso contrário} \end{cases}$$

para todos os elementos `x` e todas as listas finitas `ys`.

Grupo 5. [Programação funcional em Java. 2 valores]
a) A referência para um método é descrita utilizando o simbolo `::`.

A Falso.

B Verdadeiro.

b) Qual das seguintes opções está correta sobre expressões lambda em Java-8?

A Expressões lambda são usadas principalmente para definir implementações *inline* de uma interface funcional.

B Expressões lambda eliminam a necessidade de classes anónimas e dão ao Java capacidades de uma típica linguagem de programação funcional.

C Todas as anteriores.

D Nenhuma das anteriores.

c) Utilizando streams do Java-8, escreva um método com a seguinte assinatura:

```
<T> Collection<T> foldl(BiFunction<Collection<T>,
                                   T,
                                   Collection<T>> funcao,
                        Collection<T> acumulador,
                        Collection<T> lista)
```

que emula o funcionamento da função Haskell `fold`.
d) Fazendo uso da função `fold`, implementada na alínea anterior, escreva um método `removeDuplicados` que remova os elementos duplicados de uma lista. Por exemplo, o seguinte pedaço de código:

```
List<Integer> numeros = Arrays.asList(1, 2, 2, 3, 3, 3, 1, 1);
System.out.println(removeDuplicados(numeros));
```

deverá retornar uma lista contendo apenas os números `1, 2, 3`.

```
<T> Collection<T> removeDuplicadosWithFoldl(Collection<T> lista)
```