Princípios de Programação

Trabalho Quarto

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2020/2021

O dia de eleições é um dos momentos mais importantes numa sociedade democrática, pois é nesta altura que todos os cidadãos de um país têm voz para, em conjunto, decidirem qual será o seu próximo líder. Devido à sua importância, é necessário que o procedimento eleitoral ocorra da forma mais transparente e eficiente possível. Neste trabalho vamos construir um módulo, chamado `Eleicoes`, que disponibiliza tipos de dados e funções para simular um sistema eleitoral e decidir o vencedor de uma eleição.
No sistema de eleições indiretas, uma nação encontra-se dividida em vários estados (ou distritos). Os cidadãos de cada estados votam, não no líder da nação, mas sim para os representantes do seu estado. É neste sistema que se enquadra, por examplo, a eleição para presidente nos Estados Unidos da América.
Para simplificar, neste trabalho vamos considerar que:

- a eleição realiza-se entre dois candidatos, aos quais chamamos A e B.
- cada estado tem um peso, que corresponde ao seu número de representantes. O candidato que obtiver mais votos num estado ganha esse estado e recebe *todos* os representantes.
- o vencedor da eleição é o que tiver alcançado mais representantes em todos os estados.

1. Comece por criar os seguintes três tipos de dados:

   - `Candidato`, tipo que representa um candidato. Este tipo deverá poder assumir dois valores possíves: `A` ou `B`, e deverá ser instância das classes **`Eq`** e **`Show`** da forma óbvia. Utilize a seguinte instrução:

     ```
     data Candidato = A | B deriving (Eq, Show)
     ```

- `Estado`, tipo que representa um estado. Deve utilizar a sintaxe de registos para definir este tipo, através de um construtor chamado `Estado`. Os campos são os seguintes: nome (tipo **`String`**), peso (tipo **`Int`**), votos do candidato A (tipo **`Int`**) e votos do candidato B (tipo **`Int`**). Os campos deverão poder ser obtidos através das funções
  - `nome :: Estado -> String`,
  - `peso :: Estado -> Int`,
  - `votosA :: Estado -> Int`,
  - `votosB :: Estado -> Int`
- `Nacao`, tipo que representa uma nação. Deve representar uma nação a partir de uma lista de estados, usando o sinónimo de tipos

  ```
  type Nacao = [Estado]
  ```

2. Escreva uma função `criaNacao` que, recebendo uma lista de pares (nome, peso) para cada estado, cria uma nação com esses estados, no momento inicial de uma eleição. Cada estado começa com zero votos para cada candidato. Exemplo de utilização:

```
> nacao0 = criaNacao [("Norte",4), ("Centro",6), ("Sul",3)]
```

3. Escreva uma função `obterEstado` que, recebendo uma nação e o nome de um estado, devolva o estado com esse nome nessa nação. Pode assumir na sua implementação que existe um e um só estado com esse nome. Exemplo de utilização:

```
> norte0 = obterEstado nacao0 "Norte"
> peso norte0
4
> votosA norte0
0
> votosB norte0
0
```

4. Escreva uma função `adicionaVotosEstado` que, recebendo um estado, um número de votos para o candidato A e um número de votos para o candidato B, adicione esses votos aos respetivos candidatos, e devolva o estado com o número de votos atualizado. Exemplo de utilização:

```
> norte1 = adicionaVotosEstado norte0 15 17
> votosA norte1
15
> votosB norte1
17
```

```
> norte2 = adicionaVotosEstado norte1 8 3
> votosA norte2
23
> votosB norte2
20
```

5. Utilizando pelo menos uma função de ordem superior, escreva uma função `adicionaVotosNacao` que, recebendo uma nação e uma *lista* de triplos (nome, votos A, votos B), adicione esses votos aos estados respetivos, e devolva a nação com o número de votos atualizado. Pode assumir que para cada nome na lista de triplos existe um único estado na nação com esse nome. Exemplo de utilização:

```
> nacao1 = adicionaVotosNacao nacao0 [("Norte",23,20),
    ("Sul",16,14), ("Centro",30,41)]
> centro1 = obterEstado nacao1 "Centro"
> votosA centro1
30
> votosB centro1
41
> sul1 = obterEstado nacao1 "Sul"
> votosA sul1
16
> votosB sul1
14
```

6. Escreva uma função `vencedorEstado` que, recebendo um estado, devolva **`Just`** `vencedor` com o vencedor desse estado (aquele que tiver mais votos). Em caso de empate, a função deverá devolver o valor **`Nothing`**. Utilize o construtor de tipos **`Maybe`**. Exemplo de utilização:

```
> vencedorEstado centro1
Just B
> vencedorEstado sul1
Just A
> sul2 = adicionaVotosEstado sul1 0 2
> vencedorEstado sul2
Nothing
```

7. Escreva uma função `vencedorEleicao` que, recebendo uma nação, devolva **`Just`** `vencedor` com o vencedor da eleição, ou seja, o candidato que alcançou mais representantes em todos os estados. Assuma que, se um estado estiver empatado, nenhum dos candidatos recebe os representantes desse estado. Em caso de empate (a nível nacional), a função deverá devolver o valor **`Nothing`**. Utilize o construtor de tipos **`Maybe`**.
Observe que, na eleição correspondente a `nacao1`, o candidato A é o vencedor, pois venceu nos estados `"Norte"` e `"Sul"`, conseguindo 7

representantes, enquanto B venceu apenas no estado `"Centro"`, que lhe valeu 6 representantes. Exemplo de utilização:

```
> vencedorEleicao nacao0
Nothing
> vencedorEleicao nacao1
Just A
```

8. Torne o tipo de dados `Estado` instância da classe **`Eq`**. Para este trabalho, dois estados deverão ser considerados iguais se tiverem o mesmo peso, e se o vencedor nesse estado for o mesmo. Exemplo de utilização:

```
> centro1 == sul1
False
> ilhas = Estado {nome = "Ilhas", peso = 3, votosA = 8,
    votosB = 3}
> sul1 == ilhas
True
```

9. Torne o tipo de dados `Estado` instância da classe **`Show`**. A representação textual de um estado é a seguinte: nome do estado, seguido do peso do estado, seguido do número de votos de cada candidato, separados por `"␣"`. Atenção aos espaços: vamos testar os vossos trabalhos de um modo automático. Exemplo de utilização:

```
> centro1
Centro 6 30 41
> sul1
Sul 3 16 14
```

Notas

1. O seu trabalho é constituído pelo módulo `Eleicoes`. Deverá submeter um ficheiro com o nome `t4_fcXXXXX.hs`, onde `XXXXX` é o seu número de aluno. O seu ficheiro deve começar por

   ```
   module Eleicoes ( ... ) where
   ```

   onde no lugar de `...` deverão estar os vários tipos, construtores e funções pedidos neste enunciado.

2. Os trabalhos serão avaliados automaticamente. Respeite os nomes e os tipos de todas as funções e tipos de dados enunciados acima.

3. Cada função (ou expressão) que escrever deverá vir sempre acompanhada de uma assinatura. Isto é válido para as funções enunciadas acima bem como para outras funções ajudantes que decidir implementar.

4. Para resolver este exercício pode utilizar toda a matéria do livro de texto até ao capítulo sobre criação de tipos (*Making Our Own Types and Typeclasses*).
5. Pode usar qualquer função constante no `Prelude`.
6. Lembre-se que as boas práticas de programação Haskell apontam para a utilização de várias funções simples em lugar de uma função única mas complicada.

Entrega. Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 30 de novembro de 2020.

Plágio. Os trabalhos de todos os alunos serão comparados por uma aplicação computacional. Relembramos aqui um excerto da sinopse: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".