Princípios de Programação

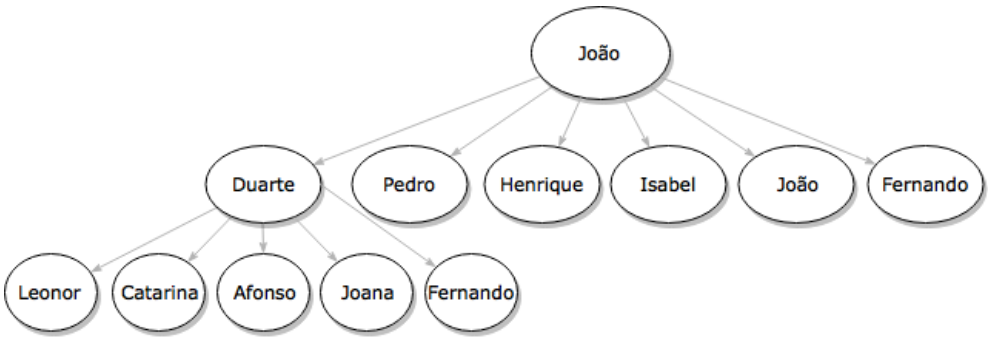
Trabalho para casa 4

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

2017/2018

Neste trabalho pretende-se desenvolver um módulo Haskell que permita representar e comparar descendências familiares.

Para tal deverá desenvolver um tipo de dados `Familia` que represente toda a descendência de uma pessoa. Cada pessoa é identificada por um nome e pode ter zero ou mais descendentes. Por uma questão de simplicidade representamos apenas um dos progenitores de uma dada família.



1. De forma a poder iniciar a construção de uma família deverá definir a função `raiz :: String -> Familia`. Esta função cria uma família com apenas uma pessoa.

2. Para registar a descendência de alguém deverá definir a função `juntar :: String -> String -> Familia -> Familia` que recebe o nome do progenitor, do descendente e a família a que ambos pertencem.

   De notar que o progenitor não precisa de ser o membro mais ancestral da família. Em caso de existirem homónimos na família (situação bastante comum) deverá ser considerado o familiar mais antigo. A operação não tem efeito se o progenitor não for membro da família. Exemplo:

   ```
   Descendencia> let inclitaGeracao = foldr (juntar "
      Joao") (raiz "Joao") ["Duarte", "Pedro", "
      Henrique", "Isabel", "Joao", "Fernando"]
   ```

3. O tipo `Familia` deverá ser convertível em `String` de forma a que cada descendente apareça *dois* espaços mais à direita do que o seu ascendente mais próximo. De notar que não existe nenhuma linha vazia no final da lista. Eis uns exemplos:

   ```
   Descendencia> foldr (juntar "Duarte")
      inclitaGeracao ["Leonor", "Catarina", "Afonso",
       "Joana", "Fernando"]
   Joao
     Duarte
       Leonor
       Catarina
       Afonso
       Joana
       Fernando
     Pedro
     Henrique
     Isabel
     Joao
     Fernando
   Descendencia>
   ```

4. Defina o tipo `Familia` como pertencente à classe **`Eq`**, de modo a que duas famílias sejam iguais apenas quando têm o mesmo número de gerações e o mesmo número de elementos.

5. Defina também o tipo `Familia` como pertencente à classe **`Ord`**, de modo a que uma família seja maior do que outra quando tem um maior número de gerações ou, em caso de empate, seja composta por mais elementos.

```
Descendencia> let a = juntar "1" "2" $ juntar "3"
   "4" $ juntar "1" "3" (raiz "1")
Descendencia> let b = juntar "A" "B" $ raiz "A"
Descendencia> a == b
False
Descendencia> a >= b
True
Descendencia> let c = juntar "b" "e" $ juntar "a"
   "b" $ juntar "a" "c" (raiz "a")
Descendencia> a == c
True
Descendencia> a > c
False
```

Notas

1. Os trabalhos serão avaliados semi-automaticamente. Respeite o nome do módulo: `Descendencia` e as funções que o mesmo exporta: `raiz` e `juntar`.

2. Deverá definir o seu tipo de dados, justificando a decisão. Poderá usar sinónimos de tipos para tornar o seu programa mais legível.

3. Não se esqueça de juntar o tipo de cada função top-level que escrever.

Entrega Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 22 de novembro de 2017.

Ética Os trabalhos de todos os alunos serão comparados por uma aplicação computacional. Lembre-se: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".