Exame de Recurso

Programação Funcional – 1° Ano, LEI / LCC / MIEF
9 de Fevereiro de 2015

Duração: 2 horas

1. Considere o seguinte tipo para representar os alunos de uma turma.

```
type TurmaL = [(Numero,Aluno)]
type Aluno = (Nome,Nota)
type Numero = Int
type Nome = String
type Nota = Float
```

Admita que não há alunos com números repetidos e que a lista se encontra ordenada por ordem crescente do número de aluno.

(a) Defina a função `taxaAp :: TurmaL -> Float` que calcula a taxa (número entre 0 e 1) de alunos aprovados (i.e., com uma nota superior ou igual a 9.5).

(b) Defina a função `top :: Int -> TurmaL -> [String]` que dado um inteiro `n` e uma turma, determina o nome dos `n` melhores alunos da turma, i.e., a função calcula uma lista com no máximo `n` elementos e a nota desses alunos é maior ou igual à dos outros alunos. (Sugestão: Comece por ordenar a lista dos alunos).

(c) Defina usando funções de ordem superior a função `lNomeMax :: TurmaL -> Int` que, dada uma turma calcula o comprimento do nome mais longo.

(d) Defina a função `listaT :: TurmaL -> IO ()` que imprime no ecran a informação sobre uma dada turma com o seguinte formato:

- Em cada linha deve aparecer o número, nome e classificação de um aluno;
- Os números, nomes e classificações devem aparecer alinhados verticalmente;
- Na classificação deve aparecer um número inteiro caso o aluno tenha uma classificação superior ou igual a 9.5 (use a função `round`) ou `R` caso contrário.

2. Uma alternatina ao uso de listas (ordenadas) para guardar a informação dos alunos de uma turma consiste em usar uma árvore cujas folhas estão ordenadas e têm essa informação. Para facilitar as procuras guardam-se ainda nos pontos de bifurcação os limites dos números dos alunos armazenados nessa árvore.

```
data TurmaA = Al (Numero,Aluno)
            | Fork (Numero,Numero) TurmaA TurmaA
```

Por exemplo, as constantes `tL` e `tA` definidas abaixo representam as mesmas turmas nestas duas representações alternativas.

```
tL :: TurmaL
tL = [(1,("Joao", 12.3)),
      (3, ("Maria",5.4)),
      (6, ("Joana",9.5)),
      (12, ("Anastacia",18.8))]
```

```
tA :: TurmaA
tA = Fork (1,12) -- folhas entre 1 e 12
          (Fork (1,3) -- folhas entre 1 e 3
                (Al (1, ("Joao"     ,12.3)))
                (Al (3, ("Maria"    , 5.4))))
          (Fork (6,12) -- folhas entre 6 e 12
                (Al (6, ("Joana"    , 9.5)))
                (Al (12,("Anastacia",18.8))))
```

(a) Defina as funções `toList :: TurmaA -> TurmaL` e `fromList :: TurmaL -> TurmaA` de conversão entre as duas representações.

(b) Defina a função `lookupA :: TurmaA -> Numero -> Maybe Aluno` que procura um dado aluno numa turma. Tal como a função `lookup`, a função deverá retornar `Nothing` se o aluno não existir.

(c) Defina a função `remove :: TurmaA -> Numero -> Maybe TurmaA` que remove da árvore o aluno com esse número. A função deverá retornar `Nothing` se a árvore resultante for vazia. Se esse aluno não existir, a função deverá retornar o árvore recebida.

(d) Defina a função `acrescenta :: TurmaA -> (Numero,Aluno) -> Maybe TurmaA` que acrescenta a informação de um aluno a uma turma. No caso do número já existir na turma a função deve retornar `Nothing`