Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência 1
Duração: 01h40m
Tolerância: 15 minutos

INSTRUÇÕES

Todas as questões devem ser respondidas usando o ficheiro Freq1.hs que está disponível no Moodle para download. A estrutura do ficheiro Freq1.hs deve ser seguida, caso contrário poderá perder pontos. Para que a frequência seja considerada terá de estar presente na reunião Zoom com a câmara ligada até receber confirmação da regente de que a sua submissão foi recebida. Terá também de participar numa entrevista pelo Zoom (ver detalhes no Moodle).

O ficheiro Freq1.hs tem que ser submetido no Moodle até ao final da frequência para ser avaliado e de seguida deve ser também enviado para o email amendes@di.ubi.pt
A frequência tem a duração de 1h40m com 15 minutos adicionais de tolerância para a sua submissão no Moodle.

Todas as funções têm de ser acompanhadas do seu tipo. Para obter nota máxima terá de usar uma abordagem funcional e será valorizada a elegância e concisão das soluções apresentadas.

Caso alguma função esteja a dar erro e não o consiga resolver, comente a função mas deixe-a no ficheiro indicando que dá erro. Assim, essa função poderá ser também considerada.

Chama-se a atenção para os documentos do **código de integridade** e **regulamento disciplinar dos Estudantes da Universidade da Beira Interior** (links para estes documentos encontram-se disponíveis no Moodle).

ESTE TESTE TEM 5 PÁGINAS E 11 GRUPOS DE QUESTÕES.

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência 1
Duração: 01h40m
Tolerância: 15 minutos

1. Escolha uma resposta para cada uma das alíneas seguintes:

(a) A expressão Node 1 (Leaf 2) (Leaf 3) é um valor do tipo: **(0.5 pontos)**
i) data Tree = Node | Leaf | Int
ii) data Tree = Leaf Int | Node Int Int Int
iii) data Tree = Leaf Tree | Node Int Int Int
iv) data Tree = Leaf Int | Node Int Tree Tree
v) data Tree = Leaf Tree | Node Int Tree Tree

(b) Qual das seguintes igualdades é verdade para todas as listas finitas xs, ys, e zs? **(0.5 pontos)**
i) reverse (xs ++ ys) = reverse xs ++ reverse ys
ii) xs ++ (ys ++ zs) = (xs ++ ys) ++ zs
iii) reverse xs = xs
iv) reverse xs = reverse (reverse xs)
v) reverse [x] = x

(c) Assumindo o tipo Int para x e y , a função

```
soma x y = x + y
```

tem o tipo: **(0.5 pontos)**
i) Int
ii) $(Int, Int) \rightarrow Int$
iii) $(Int \rightarrow Int) \rightarrow Int$
iv) $Int \rightarrow Int \rightarrow Int$
v) $Int \rightarrow Int$

(d) A função

```
aplica f x = f x
```

tem o tipo: **(0.5 pontos)**
i) $a \rightarrow b \rightarrow c$
ii) $(a \rightarrow b) \rightarrow a \rightarrow b$
iii) $a \rightarrow (b \rightarrow a) \rightarrow b$
iv) $a \rightarrow b \rightarrow (a \rightarrow b)$
v) $(a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow c$

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência 1
Duração: 01h40m
Tolerância: 15 minutos

2. Descreva em que consiste a estratégia de redução *Lazy*. **(1 pontos)**

3. Escreva a avaliação da expressão: **(1 pontos)**

```
fst(10,1+4)
```

usando as estratégias *Innermost* e *Outermost* e indique qual é preferível e porquê.

4. Usando **indução** prove a seguinte propriedade para todas as listas finitas xs e ys: **(2 pontos)**

```
map f (xs ++ ys) = map f xs ++ map f ys
```

5. Usando **listas por compreensão** defina uma função: **(1.5 pontos)**

```
paresSemRep :: [Char] -> [Char] -> [(Char,Char)]
```

que dadas duas listas de *Char*, devolve uma lista de pares de Char que combina todos os elementos de ambas as listas mas filtrando todos os pares com dois elementos iguais.
Por exemplo:

```
> paresSemRep ['a','b','c'] ['a','b','c']
[('a','b'),('a','c'),('b','a'),('b','c'),('c','a'),('c','b')]
```

A ordem dos elementos da lista final não é relevante.

6. Defina a função: **(1.5 pontos)**

```
replica :: Int -> [a] -> [a]
```

que dado um inteiro n e uma lista l de elementos de qualquer tipo, produz uma lista com cada um dos elementos da lista l repetido n vezes. Por exemplo:

```
> replica 4 [4,5]
[4,4,4,4,5,5,5,5]
```

Dica: A função *replicate* pode ser útil.

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência 1
Duração: 01h40m
Tolerância: 15 minutos

7. Usando a função sobre listas mais apropriada, defina uma função que dada uma lista de inteiros filtra essa lista de forma a que contenha apenas números que são simultaneamente ímpares e divisíveis por 3. Por exemplo, dada a lista $[1, 3, 6, 9, 12]$ a função deve retornar $[3, 9]$.
Use as funções *odd* e *mod* na sua implementação. **(1.5 pontos)**

8. Sem usar ou consultar a definição do Standard Prelude, implemente a função de ordem superior *myUncurry* com o tipo: **(1 ponto)**

```
myUncurry :: (a -> b -> c) -> (a, b) -> c
```

que converte uma função *curried* numa função sobre pares.

9. Considere a função *separa* que divide um lista em duas de comprimento idêntico: **(2 pontos)**

```
separa [] = ([],[])
separa (h:t) = (h:r,l)
    where (l,r) = separa t
```

Note que para listas ímpares esta função devolve uma das listas com mais um elemento que a outra. Por exemplo:

```
> separa [1,2,3,4,5]
([1,3,5],[2,4])
```

Redefina esta função usando **foldr**.

10. Considere o seguinte tipo de *Rose Trees*:

```
data RoseTree a = RTLeaf a | RTNode a [RoseTree a] deriving (Show)
```

(a) Apresente um valor deste tipo que contenha pelo menos 2 ocorrências de RTNode e 4 ocorrências de *RTLeaf*. **(0.5 pontos)**

(b) Declare o tipo *RoseTree* como instância das classes *Eq* e *Functor*. **(2.5 pontos)**

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência 1
Duração: 01h40m
Tolerância: 15 minutos

11. Considere os seguintes tipos de dados que representam a informação relativa à avaliação dos alunos inscritos numa turma. Note-se o uso do tipo *Maybe* para representar se cada estudante tem ou não uma nota teórica ou prática definida. Note-se também o uso do tipo *Estatuto* que indica se um aluno é ou não trabalhador estudante (TE).

```
type Nome = String
type Numero = Int
type NT = Maybe Float
type NP = Maybe Float
data Estatuto = Normal | TE deriving (Show, Eq)
type Aluno = (Numero, Nome, NT, NP, Estatuto)
type Turma = [Aluno]
```

(a) Defina a função: **(1.5 pontos)**

```
admitidos :: Turma -> Turma
```

que recebe uma turma e devolve uma lista com todos os alunos cuja nota prática seja igual ou superior a 7 (i.e. a lista de todos os alunos admitidos a exame). Use **recursão e *pattern-matching*** sobre listas e sobre tuplos.
Dica: Pode usar as funções *isJust* e *fromJust* da biblioteca *Data.Maybe*.

(b) Defina a função: **(2 pontos)**

```
alteraTipo :: Turma -> [(Numero, Estatuto)] -> Turma
```

que recebe como argumento uma turma e uma lista l de pares compostos por um *Numero* de aluno e um *Estatuto* do aluno. A função devolve uma lista com a mesma turma recebida como argumento com a diferença de que o registo de cada aluno cujo número consta na lista l está actualizado com o estatuto que lhe está associado no par – os registos dos restantes alunos mantêm-se inalterados. Por exemplo, considerando-se a turma:

```
turma = [(1, "Alexandra", Nothing, Nothing, Normal),
         (2, "Joao", Nothing, Just 15, Normal),
         (3, "Luis", Just 14, Just 15, TE)]
```

o comportamento esperado é o seguinte:

```
> mudaTipo turma [(1,TE),(3,Normal)]
[(1, "Alexandra", Nothing, Nothing, TE),
 (2, "Joao", Nothing, Just 15, Normal),
 (3, "Luis", Just 14, Just 15, Normal)]
```