Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência Modelo 1
Duração: 01h30m

ATENÇÃO

Esta é uma frequência modelo que é disponibilizada para ajudar os alunos a prepararem-se para a frequência. A frequência real será diferente e poderá ter mais ou menos questões e também questões sobre tópicos que não são cobertos nesta frequência modelo. As pontuações atribuídas a cada questão podem também variar.

INSTRUÇÕES

Todas as questões devem ser respondidas usando o ficheiro Freq1Modelo.hs que está disponível no Moodle para download. A estrutura do ficheiro Freq1Modelo.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 Freq1Modelo.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

Todas as funções têm de ser acompanhadas do seu tipo. Para obter nota máxima nas questões 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 12 PERGUNTAS.

Universidade da Beira Interior
Departamento de Informática **Frequência Modelo 1**
Paradigmas de Programação Duração: 01h30m

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

   (a) Qual das seguintes igualdades é verdade para todas as listas xs? **(0.5 pontos)**
   i) reverse (map f xs) = map f (reverse xs)
   ii) map f (map g xs) = map g (map f xs)
   iii) reverse (reverse xs) = reverse xs
   iv) map f (map f xs) = map f xs
   v) reverse xs = xs

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

   (c) A expressão *take 5* tem o tipo: **(0.5 pontos)**
   i) Int
   ii) $Int \rightarrow [a]$
   iii) $Int \rightarrow [a] \rightarrow [a]$
   iv) $[a] \rightarrow [a]$
   v) $[a]$

   (d) Uma função com tipo $Int \rightarrow Int \rightarrow Int$ é uma função: **(0.5 pontos)**
   i) *curried*
   ii) que recebe três argumentos
   iii) polimórfica
   iv) *overloaded*
   v) que recebe uma função como argumento

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência Modelo 1
Duração: 01h30m

2. Descreva as estratégias de redução *Innermost* e *Outermost*. **(0.5 pontos)**

3. Escreva a avaliação da expressão: **(0.5 pontos)**

   ```
   (\x -> (\y -> snd(y,x))) 2 (3+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)**

   ```
   length (xs ++ ys) = length xs + length ys
   ```

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

   ```
   divisores :: Int -> [Int]
   ```

   que dado um número inteiro positivo devolve uma lista com todos os seus divisores (use a função *mod* na sua implementação).

6. Usando recursão defina a função: **(1.5 pontos)**

   ```
   intercala :: [a] -> [a] -> [a]
   ```

   que recebe duas listas com o mesmo tamanho e intercala os seus elementos. Por exemplo:

   ```
   > intercala [1,2,3] [4,5,6]
   [1,4,2,5,3,6]
   ```

   Mostre passo a passo como é que a sua definição avalia *intercala [1,2] [3,4]*.

7. Usando a função sobre listas mais apropriada, defina uma função que filtra todas as letras minúsculas de uma lista.
   Por exemplo, dada a lista $['a','A','b','B']$ deve retornar $['A','B']$. **(1 ponto)**

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência Modelo 1
Duração: 01h30m

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

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

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

9. Usando o *foldr* defina uma função *alterna* que dadas duas funções e uma lista, aplica alternadamente as funções aos elementos da lista. **(2 pontos)**
   Por exemplo:

   ```
   > alterna succ pred [1,2,3,4]
   [2,1,4,3]
   ```

10. Considere o seguinte tipo de árvores binárias: **(2.5 pontos)**

    ```
    data BTree a = Empty | Node a (BTree a) (BTree a) deriving (Show)
    ```

    Declare o tipo *BTree* como instância das classes *Eq* e *Functor*.

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

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

    (a) Descreva por palavras suas este tipo de dados e apresente um valor deste tipo que contenha pelo menos 5 ocorrências de *RTLeaf*. **(0.5 pontos)**

    (b) Implemente uma função que recebe uma RoseTree de inteiros como argumento e adiciona 1 a todos os inteiros contidos na árvore (nas folhas e nodos). **(2 pontos)**

Universidade da Beira Interior
Departamento de Informática
Paradigmas de Programação

Frequência Modelo 1
Duração: 01h30m

12. 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.

```
type Nome = String
type Numero = Int
type NT = Maybe Float
type NP = Maybe Float
type Aluno = (Numero, Nome, NT, NP)
type Turma = [Aluno]
```

(a) Defina a função: **(1 ponto)**

```
filtraAlunos :: Turma -> Turma
```

que remove da turma todos os alunos que têm algum componente da avaliação em falta. Use recursão e *pattern-matching* sobre listas e sobre tuplos.

(b) Usando um *map* defina a função: **(2 pontos)**

```
bonus :: [Numero] -> Turma -> Turma
```

que recebe como argumento uma lista de números de alunos n e uma turma, e aplica a todos os alunos que constam na lista n um bónus de 1 valor na nota teórica, devolvendo uma lista do tipo *Turma* com todos os bónus aplicados. O bónus é aplicado apenas aos alunos que já têm nota na componente teórica. Não necessita de verificar se a nota com bónus excede os 20 valores.
Dica: Pode usar as funções *isJust* e *fromJust* da biblioteca *Data.Maybe* e também a função *elem*.