PROGRAMAÇÃO FUNCIONAL [BCC222]

Segunda prova — Parte 2 (2018–1)
13 de junho de 2018

Matrícula: _____

Nome: _____

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias

Instruções

- Não se esqueça de preencher o nome e o número de matrícula na folha de respostas.
- A avaliação é individual.
- A interpretação das questões faz parte da avaliação.
- Faça as observações que achar necessárias por escrito.
- **Não é permitido o uso de computadores ou aparelho celular**. Os aparelhos celulares deverão permanecer desligados e guardados fora do alcance de suas mãos.
- Qualquer tentativa de consulta não autorizada será punida com rigor, de acordo com as regras da instituição.

Questão 1. Em uma aplicação que simula um caixa automático a carteira de contas é representada por uma lista de contas, onde cada conta tem um número e um histórico de movimentos (uma lista). Cada movimento tem uma data (uma string), uma descrição (abertura, saque, ou depósito), o valor movimentado, e o saldo final.

```haskell
data DescMov = Abertura | Saque | Depósito
               deriving (Eq, Ord, Show)
type Data = String
type Movimento = (Data, DescMov, Double, Double)
type Histórico = [ Movimento ]
type Conta = (Integer, Histórico)
type Carteira = [ Conta ]
```

Exemplo de carteira de contas:

```haskell
exemplo :: Carteira
exemplo = [ (10, [ ("2018/06/06", Depósito,  20,  90)
                 , ("2018/05/06", Saque   ,  80,  70)
                 , ("2018/04/03", Depósito,  50, 150)
                 , ("2018/02/01", Abertura, 100, 100)
                 ] )
          , (20, [ ("2018/06/15", Depósito,  35,  50)
                 , ("2018/02/09", Saque,      5,  15)
                 , ("2018/02/04", Depósito,   2,  20)
                 , ("2018/02/04", Depósito,  18,  18)
                 , ("2018/02/03", Abertura,   0,   0)
                 ] )
          , (30, [ ("2018/02/09", Saque,      8,   2)
                 , ("2018/05/03", Abertura,  10,  10)
                 ] )
          ]
```

1. Escreva uma função `ultimaData :: Carteira -> String` que recebe uma carteira de contas e resulta na data mais recente em que houve movimento em alguma conta. Considere que há pelo menos uma conta na carteira, e que o histórico das contas não é vazio. Use funções de ordem superior.

   Exemplo:

   ```haskell
   ultimaData exemplo ⇝ "2018/06/15"
   ```

   Dicas:

   - use variáveis locais para nomear valores intermediários
   - use funções da biblioteca padrão, como `map`, `filter`, `concat`, `snd`, `maximum`, etc.

- expressões lambda junto com `map`, `filter` e outras funções de superior podem ser úteis

Questão 2. Considere a seguinte definição de tipo para representar uma expressão algébrica:

```haskell
data Exp = Cte Double       - contante
         | Var String       - variável
         | Som Exp Exp      - soma
         | Sub Exp Exp      - subtração
         | Mul Exp Exp      - multiplicação
         | Div Exp Exp      - divisão
         | Neg Exp          - simétrico
  deriving (Show)
```

Defina uma função `variáveis :: Exp -> [String]` que recebe uma expressão algébrica e resulta na lista de todas as variáveis que aparecem na expressão.

Por exemplo:

```haskell
variáveis (Cte 5.6)                                  ⇝ []
variáveis (Som (Cte 5.6) (Var "x"))                  ⇝ ["x"]
variáveis (Div (Som (Var "a") (Var "b")) (Cte 2)     ⇝ ["a", "b"]
```

Questão 3. Considere a seguinte definição de tipo polimórfico para árvores binárias:

```haskell
data ArvBin t = Vazia | No t (ArvBin t) (ArvBin t)
  deriving (Show)
```

Basicamente uma árvore binária pode ser de duas formas possíveis: vazia (representada pelo construtor constante `Vazia`), ou pode ser um nó (representada pelo construtor `No`) contendo uma informção e duas subárvores do mesmo tipo do nó. Exemplo:

```haskell
arvore1 = (No 5
              (No 9 Vazia Vazia)
              (No (-5)
                  (No 8
                      Vazia
                      (No 3 Vazia Vazia))
                  Vazia))
```

1. Escreva uma função `elemento` que recebe um um valor e uma árvore e verifica se o valor é um elemento da árvore, resultando em um valor lógico.

   Por exemplo:

   ```haskell
   elemento "abacaxi" Vazia
   ⇝ False

   pesquisa 8 (No 8 Vazia Vazia)
   ⇝ True

   pesquisa "maria" (No "paulo" Vazia (No "roberto" Vazia Vazia))
   ⇝ False

   pesquisa (-5) arvore1
   ⇝ True
   ```