PROGRAMAÇÃO FUNCIONAL [BCC222]

Segunda prova (2017–2)
20 de novembro de 2017
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.
- É permitida a consulta somente à folha individual de anotações, a qual deverá ser anexada à prova.
- 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. [4 PONTOS]
Determine o tipo mais geral das seguintes expressões em Haskell.

a) `return (even 8)`

b) `do {x <- getLine; return (length x)}`

c) `[return 18.1, readLn, do {x <- readLn; return (2*x :: Float)}]`

d) `(odd 5, return (toUpper 'x'), [])`

Questão 2. [1 PONTO]
A seguinte expressão apresenta um erro de tipo.

```
do entrada <- getLine
   length entrada
   putStrLn (map toUpper entrada)
```

Explique qual é o erro.

Questão 3. [2 PONTOS]
Considere a seguinte definição de tipo para representar uma expressão aritmética:

```
data Exp = Cte Double
         | Som Exp Exp
         | Sub Exp Exp
         | Mul Exp Exp
         | Div Exp Exp
         | Neg Exp
  deriving (Show)
```

Defina uma função `avalia :: Exp -> Double` que recebe uma expressão aritmética e resulta no valor da expressão dada.

Por exemplo:

```
avalia (Cte 5.6)                              ⇝ 5.6
avalia (Som (Cte 5.6) (Cte 1.4))              ⇝ 7.0
avalia (Div (Som (Cte 5.6) (Cte 1.4)) (Cte 2) ⇝ 3.5
```

Questão 4. [3 PONTOS]
Podemos declarar um tipo algébrico polimórfico em Haskell através de variáveis de tipo escritas logo após o construtor de tipo. Estas variáveis representam tipos quaisquer da linguagem e serão conhecidas quando o tipo de um valor for calculado.

Por exemplo, a declaração

```
data Maybe a = Nothing | Just a
```

define o tipo `Maybe a` da biblioteca padrão que já usamos para indicar um valor opcional do tipo `a`. O construtor constante `Nothing` representa uma estrutura de dados vazia (valor ausente). Já o construtor de dados `Just` é usado para representar uma estrutura de dados que contém um valor do tipo `a`.

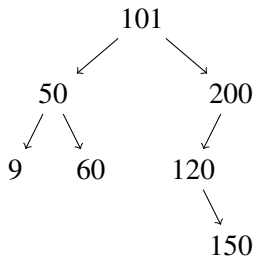
Assim o valor `Just True` é do tipo `Maybe Bool`, `Just "rosa"` é do tipo `Maybe String`, `Just 8` é do tipo `(Num a) => Maybe a`, e `Nothing` é do tipo `Maybe a`.

Considere a seguinte definição de tipo polimórfico para árvores binárias:

```haskell
data ArvBin t = Vazia | Nó 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 `Nó`) contendo uma informção e duas subárvores do mesmo tipo do nó.

1. Defina uma variável `arvore1` cujo valor é a representação da árvore da figura a seguir.



2. Escreva uma função `maxArvBin` que recebe uma árvore binária e resulta possivelmente no maior elemento da árvore, caso exista. Use o construtor de tipo `Maybe` no resultado da função. Anote o tipo mais geral da sua função.

   Por exemplo:

```haskell
maxArvBin Vazia                                    ⇝ Nothing
maxArvBin (Nó Vazia 8 Vazia)                       ⇝ Just 8
maxArvBin (Nó Vazia "paulo" (Nó Vazia "roberto" Vazia)) ⇝ Just "roberto"
maxArvBin arvore1                                  ⇝ Just 200
```

Questão 5. [2 PONTOS]
Defina uma ação de E/S `multiplicar :: IO ()` que, quando executada, lê da entrada padrão uma sequência de números inteiros (um número por linha), e exibe o seu produto, como ilustrado a seguir:

```
Quantos números?
5

Digite os números:
1
3
5
7
2

O produto é 210
```

Se necessário faça definições auxiliares.