Engenharia de Software
Campus Garanhuns

Disciplina: Programação 3 - Programação Funcional
Professor: Emanoel Barreiros
Assunto: Criando tipos e classes

Resumo: Utilize a linguagem de programação Haskell para resolver os problemas desta lista. Esta lista aborda o conceito de criação de tipos e classes.

1) Semelhante à função **somar**, defina uma função de multiplicação recursiva para números naturais `mult :: Nat -> Nat -> Nat`. Dica: use a função somar definida no material de aula para os números naturais.

2) O Prelude define o tipo Ordering

   ```
   data Ordering = LT | EQ | GT
   ```

   e a função

   ```
   compare :: Ord a => a -> a -> Ordering
   ```

   que decide se o primeiro valor recebido como argumento é menor (LT), igual (EQ) ou maior (GT) que o segundo argumento. Usando essa função redefina a função existe :: Ord a => a -> Arvore a -> Bool para árvores binárias de busca.

3) Considere o seguinte tipo de árvores binárias:

   ```
   data Arvore a = Folha a | No (Arvore a) (Arvore a)
   ```

   Digamos que a árvore é balanceada se a quantidade de folhas do lado esquerdo e do lado direito de todos os nós são iguais ou sua diferença é no máximo 1, e suas folhas são consideradas balanceadas por definição. Defina uma função `balanceada :: Arvore a -> Bool` que decide se uma árvore é balanceada ou não. Dica: primeiro defina uma função que conta a quantidade de folhas em uma árvore.

4) Defina a função `balancear :: [a] -> Arvore a` que converte uma lista não vazia em uma árvore balanceada. Dica: primeiro defina uma função que divide uma lista em duas metades cujos tamanhos diferem em no máximo 1.

5) Dada a definição

   ```
   data Expr = Val Int | Add Expr Expr
   ```

   defina a função de alta ordem

```
folde :: (Int -> a) -> (a -> a -> a) -> Expr -> a
```

tal que `folde f g` substitui cada construtor `Val` na expressão pela aplicação da função `f`, e cada construtor `Add` pela aplicação da função `g`.

6) Usando a função `folde`, defina a função `eval :: Expr -> Int` que avalia uma expressão para um valor inteiro, e uma função `size :: Expr -> Int` que calcula o número de valores em uma expressão.