

Paradigmas de Programação

Fabício Olivetti de França

25 de Julho de 2018

Lista de exercícios 05

1. De modo semelhante à função add

```
add :: Nat -> Nat -> Nat
add Zero    n = n
add (Succ m) n = Succ (add m n)
```

defina uma função recursiva de multiplicação

```
mult :: Nat -> Nat -> Nat
```

Para o tipo de números naturais:

```
data Nat = Zero | Succ Nat
```

Dica: utilize a função add para definir a multiplicação.

2. O prelude padrão define

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

com a função

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

que decide se um valor do tipo *ordering* é menor que (LT), igual a (EQ) ou maior que (GT) outro valor. Usando essa função, redefina

```
occurs :: Ord a => a -> Tree a -> Bool
```

para árvores de busca do tipo

```
data Tree a = Leaf a | Node (Tree a) a (Tree a)
              deriving (Show)
```

3. Defina uma função

```
flatten :: Tree a -> [a]
```

que retorna uma lista ordenada com os elementos da árvore percorrida em ordem.

4. Considere o tipo de árvores binárias

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

Uma árvore é dita balanceada se o número de folhas das sub-árvores à esquerda e à direita em cada nó difere em, no máximo, um. As folhas estão trivialmente balanceadas. Defina a função

```
balanced :: Tree a -> Bool
```

que verifica se uma árvore binária está balanceada ou não. Dica: primeiro defina uma função que retorna o número de folhas em uma árvore.

5. Defina a função

```
balance :: [a] -> Tree a
```

que converte uma lista não vazia em uma árvore balanceada. Dica: primeiro, defina uma função que divide uma lista em duas partes, cujo tamanho difere em no máximo um.

6. Dada a declaração de tipo

```
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 Val em uma expressão pelo retorno da função f, e cada Add pelo retorno de g.

7. Usando folde, defina a função

```
eval :: Expr -> Int
```

que avalia uma expressão para um valor inteiro, e a função

```
size :: Expr -> Int
```

que calcula o número de valores em uma expressão.