Programação Funcional – 1º Ano, LEI / LCC / LEF – 24 de Janeiro de 2024

Número: _____ **Nome:** _____ **Curso:** _____

1. Defina a função `substs :: Eq a => (a,a) -> [a] -> [a]` que recebe uma par (x,y) e uma lista l e substitui todas as ocorrências de x em l por y.
   Por exemplo, `subst (5,1) [6,5,2,1,5,7] == [6,1,2,1,1,7]`.

2. Defina a função `progressao :: [Int] -> Maybe Int` que verifica se uma dada uma lista contém os primeiros valores de uma progressão aritmética de inteiros (na qual todos os valores consecutivos possuem uma diferença comum), devolvendo essa diferença. Por exemplo, `progressao [-3,0,3,6,9,12] == Just 3`.

3. Defina a função `removeElems :: [Int] -> [a] -> [a]` que recebe uma lista (não necessariamente ordenada) de índices e uma lista e remove os elementos da lista nessas posições. Por exemplo, a invocação `removeElems [7,1,5,3] "Programacao"` deve dar como resultado `"Pormcao"`.

4. Considere as seguintes definições de tipos para representar uma tabela de abreviaturas que associa a cada abreviatura a palavra que ela representa.

```
type TabAbrev = [(Palavra,Abreviatura)]
type Palavra = String
type Abreviatura = String

ex = [("muito","mt"),("que","q"),("maior",">"),("que","k"),("muito","mto")]
```

   Defina a função `associa :: TabAbrev -> [(Palavra,[Abreviatura])]` que recebe uma tabela de abreviaturas, e transforma a tabela de forma a que a cada palavra fica associada a lista de todas as abreviaturas dessa palavra. Por exemplo, a invocação `associa ex` pode dar como resultado `[("muito",["mt","mto"]), ("que",["q","k"]), ("maior",[">"])]`.

5. Apresente uma definição alternativa da função `func`, usando recursividade explícita em vez de funções de ordem superior e fazendo uma única travessia da lista.

```
func :: Int -> [Int] -> [(Int,Int)]
func x l = filter ((>10) . snd)  (map (\y -> (x,y+x)) l)
```

6. Considere o seguinte tipo para representar árvores binárias:

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

arv = Node 20 (Node 3 (Node 9 (Node 7 Empty Empty)
                              (Node 15 Empty Empty)) Empty)
              (Node 4 Empty (Node 3 Empty Empty))
```

   (a) Defina uma função `isTrace :: Eq a => [a] -> BTree a -> Bool` que testa se uma dada lista é um traço de uma árvore (um traço é um caminho desde a raíz até uma das subárvores vazias).
   Por exemplo, `isTrace [20,3,9,15] arv == True`, mas `isTrace [20,3] arv == False`.

   (b) Defina a função `listaValores :: BTree a -> [[a]]` que produz a lista dos valores que estão em cada nível da árvore.
   Por exemplo, `listaValores arv == [[20],[3,4],[9,3],[7,15]]`.