Programação Funcional – 1º Ano, LEI / LCC / LEF – 9 de Janeiro de 2023 **Número:** _____ **Nome:** _____ **Curso:** _____

1. Apresente uma definição recursiva da função `unlines :: [String] -> String` que junta todas as strings da lista numa só, separando-as pelo caracter `'\n'`.

   Por exemplo, `unlines ["Prog", "Func"] == "Prog\nFunc"`.

2. O formato **csv** (**c**omma **s**eparated **v**alues) serve para descrever tabelas de uma forma textual: cada linha da tabela corresponde a uma linha do texto, enquanto que os elementos de cada linha se encontram separados por vírgulas.

   Por exemplo, a *string* `"2,3,6,4\n12,3,12,4\n3,-4,5,7"` pode ser usada para descrever a matriz

   $$\begin{bmatrix} 2 & 3 & 6 & 4 \\ 12 & 3 & 12 & 4 \\ 3 & -4 & 5 & 7 \end{bmatrix}$$

   (a) Considere o tipo `type Mat = [[Int]]` para representar matrizes e a seguinte definição da função `stringToMat` que converte strings desse formato em matrizes:

   ```
   stringToMat :: String -> Mat
   stringToMat s = map stringToVector (lines s)
   ```

   Apresente a definição da função `stringToVector` e indique explicitamente o seu tipo.

   (b) Defina a função `transposta :: String -> String` que recebe a tabela em formato textual e devolve a tabela transposta também em formato textual.

3. Considere o seguinte tipo de dados para representar uma lista em que os elementos podem ser acrescentados à esquerda (`Esq`) ou à direita (`Dir`) da lista. `Nula` representa a lista vazia.

```
data Lista a =  Esq a (Lista a) | Dir (Lista a) a | Nula
```

   (a) Defina a função `semUltimo :: Lista a -> Lista a` que recebe uma Lista não vazia e devolve a Lista sem o seu elemento mais à direita.

   (b) Defina `Lista` como instância da classe `Show` de forma a que a lista `Esq 1 (Dir (Dir (Esq 9 Nula) 3) 4)` seja apresentada como `[1, 9, 3, 4]`.

4. Relembre a definição do tipo das árvores binárias e da função que faz uma travessia *inorder* de uma árvore.

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

inorder :: BTree a -> [a]
inorder Empty = []
inorder (Node r e d) = (inorder e) ++ (r:inorder d)
```

   (a) Defina uma função `numera :: BTree a -> BTree (a,Int)` que coloca em cada nodo da árvore argumento o número de ordem desse nodo numa travessia *inorder*. A função deve percorrer a árvore uma única vez.

   Por exemplo, `numera (Node 'a' (Node 'b' Empty Empty) (Node 'c' Empty Empty))` deve dar como resultado `(Node ('a',2) (Node ('b',1) Empty Empty) (Node ('c',3) Empty Empty))`

   Sugestão: Comece por definir a função `numeraAux :: Int -> BTree a -> (Int,BTree (a,Int))` que recebe um inteiro (o primeiro número a ser usado) e retorna a árvore numerada bem como o número de elementos dessa árvore.

   (b) A função `inorder` não é injectiva: há muitas árvores diferentes que dão origem à mesma travessia: por exemplo, as árvores `Node 1 Empty (Node 2 Empty Empty)` e `Node 2 (Node 1 Empty Empty) Empty` têm como travessia a lista `[1,2]`

   Defina a função `unInorder :: [a] -> [BTree a]` que, dada uma lista, calcula (a lista de) todas as árvores cuja travessia *inorder* corresponde a essa lista.