**Teste Final de Programação Funcional** – 1º Ano, MIEI / LCC / MIEF
20 de Janeiro de 2021 (Duração: 2 horas)

1. Apresente uma definição recursiva da função `(\\) :: Eq a => [a] -> [a] -> [a]` que retorna a lista resultante de remover (as primeiras ocorrências) dos elementos da segunda lista da primeira. Por exemplo, `(\\) [1,2,3,4,5,1,2] [2,3,4,1,2] == [5,1]`.

2. Considere o tipo `MSet a` para representar multi-conjuntos de elementos de `a`: `type MSet a = [(a,Int)]` Considere ainda que nestas listas não há pares cuja primeira componente coincida, nem cuja segunda componente seja menor ou igual a zero.

   (a) Defina a função `removeMSet :: Eq a => a -> [(a,Int)] -> [(a,Int)]` que remove um elemento a um multi-conjunto. Se o elemento não existir, deve ser retornado o multi-conjunto recebido. Por exemplo, `removeMSet 'c' [('b',2), ('a',4), ('c',1)] == [('b',2), ('a',4)]`.

   (b) Usando uma função de ordem superior, defina a função `calcula :: MSet a -> ([a],Int)` que, numa única travessia do multi-conjunto, calcula simulanemente a lista (sem repetidos) de elementos do multi-conjunto e o número total de elementos. Por exemplo, `calcula [('b',2), ('a',4), ('c',1)] == (['b','a','c'],7)`.

3. Defina a função `partes :: String -> Char -> [String]`, que parte uma string pelos pontos onde um dado caracter ocorre. Por exemplo, `partes "um;bom;exemplo;" ';' == ["um","bom","exemplo"]` e `partes "um;exemplo;qualquer" ';' == ["um","exemplo","qualquer"]`.

4. Considere o seguinte tipo para representar árvores binárias de procura.

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

a1 = Node 5 (Node 3 Empty Empty)
            (Node 7 Empty (Node 9 Empty Empty))
```

   (a) Defina a função `remove :: Ord a => a -> BTree a -> BTree a`, que remove um elemento de uma árvore binária de procura.

   (b) Defina `BTree a` como uma instância da classe `Show` de forma a que `show a1` produza a string

```
"((* <-3-> *) <-5-> (* <-7-> (* <-9-> *)))"
```

5. Apresente uma definição da função `sortOn :: Ord b => (a -> b) -> [a] -> [a]` que ordena uma lista comparando os resultados de aplicar uma função de extracção de uma chave a cada elemento de uma lista. Por exemplo: `sortOn snd [(3,1),(2,5),(1,2)] == [(3,1),(1,2),(2,5)]`.

6. Considere o seguinte tipo para representar um sistema hierárquico de ficheiros

```
data FileSystem = File Nome | Dir Nome [FileSystem]
type Nome = String

fs1 = Dir "usr" [Dir "xxx" [File "abc.txt", File "readme", Dir "PF" [File "exemplo.hs"]],
                 Dir "yyy" [],  Dir "zzz" [Dir "tmp" [], File "teste.c"] ]
```

   (a) Defina a função `fichs :: FileSystem -> [Nome]`, que lista o nome de todos os ficheiros de um *file system*.

   (b) Defina a função `dirFiles :: FileSystem -> [Nome] -> Maybe [Nome]` que lista o nome dos ficheiros de um *file system* que estão numa determinada *path*. Se a *path* não for válida, a função deve devolver `Nothing`. Por exemplo, `dirFiles fs1 ["usr","xxx"] == Just ["abc.txt","readme"]`

   (c) Defina a função `listaFich :: FileSystem -> IO ()` que lê uma *path* do teclado e imprime no ecran os nomes dos ficheiros que estão na diretoria indicada pela *path*. A *path* deve ser lida como uma string com o formato usual (por exemplo: *"usr/xxx/PF"*. Se a *path* não for válida, deve ser escrita a mensagem *"Não é uma directoria."*