Lista 12

Zdefiniowany jest następujący typ dla drzew wielokierunkowych:

```
data MtreeL a = MTL a  [MtreeL a]
                deriving (Eq, Ord, Show, Read)

mt1 = MTL 1 [MTL 2 [], MTL 3 [], MTL 4 []]
mt2 = MTL 5 [MTL 6 [], MTL 7 [MTL 11 [MTL 12 [], MTL 13 [], MTL 14 []]], MTL 8 []]
mt3 = MTL 10 [mt1, mt2]
```

1. Wzorując się na odpowiednich funkcjach dla drzew binarnych, które były zdefiniowane na wykładzie, napisz poniższe funkcje dla drzew wielokierunkowych.
   a) `foldMtL :: Monoid a => MtreeL a -> a`
      `foldMtL` ma zwijać drzewo wielokierunkowe, w którym typ elementów należy do klasy `Monoid`.
   b) `foldMtLMap :: Monoid a => (t -> a) -> MtreeL t -> a`
      `foldMtLMap` zwija dowolne drzewo wielokierunkowe. Jej pierwszym argumentem jest funkcja „podnosząca" wszystkie zwijane elementy do klasy `Monoid`.

   Przeprowadź testy funkcji `foldMtL` oraz `foldMtLMap` podobne do testów z wykładu dla funkcji `foldBT` oraz `foldBTMap`.

2. „Włóż" typ `MtreeL` do klasy typów `Functor`.
   Sprawdź, że `MtreeL` jest funktorem, np. `((+1) <$> mt1) == MTL 2 [MTL 3 [], MTL 4 [], MTL 5 []]`

3. „Włóż" typ `MtreeL` do klasy typów `Foldable`.
   Sprawdź, że `MtreeL` jest instancją klasy `Foldable`, np. `sum mt1 == 10`

Powyższy efekt można osiągnąć, wykorzystując klauzulę `derivable` (jak poniżej), ale w zadaniach 2 i 3 należy zrobić to „ręcznie".

```
data MtreeL' a = MTL a  [MtreeL' a]
                 deriving (Eq, Ord, Show, Read, Functor, Foldable)
```