A házi feladatot egy `Homework9` nevű modulként kell beadni. Figyeljetek arra, hogy a függvényeitek a module szóval egy "oszlopba" kerüljenek, azaz ne legyenek beljebb húzva! Minden definiálandó függvényhez adjuk meg a hozzá tartozó típus szignatúrát is! (Ezt most megadtam, a saját modulotokba is másoljátok be a definíciótok elé.)

Hajtogatás

A feladatokban mindegyik függvényt írjuk meg rekurzívan, majd hatogatással! A hajtogatással implementált műveletet suffixáljuk egy ' szimbólummal!

- Definiáljuk a `composeAll` függvényt ami egy listányi függvényt összekomponál! (`composeAll :: [a -> a] -> (a -> a)`)
- Definiáljuk a `minimum` és `maximum` műveleteket, amelyek egy lista legkisebb/legnagyobb elemét adják vissza. Ha a lista üres, akkor a második paraméterül kapott értéket adják vissza! (`minimum2, maximum2 :: Ord a => [a] -> a -> a`)
- Definiáljuk a `reverse` függvényeket, amely egy listát megordít! (`reverse2 :: [a] -> [a]`)
- Definiáljuk a `concatMap` függvényt, amely egy olyan variánsa a `map`-nak, ami listába képez. (`concatMap2 :: (a -> [b]) -> [a] -> [b]`)
- Definiáljuk a `safeIndex` függvényt ami egy lista `k`-adik elemét visszaadja. Ha `k`. indexe nincs a listának, akkor adjunk vissza `Nothing`-ot. A `Data.Maybe` modulban lévő függvények segíthetnek. (`safeIndex :: Int -> [a] -> Maybe a`)

Tesztek:

```
composeAll [(+1), (*2), (+3)] 4 == 15
composeAll [(+3), (*2), (+1)] 2 == 9
composeAll [] 5 == 5
composeAll' [(+1), (*2), (+3)] 4 == 15
composeAll' [(+3), (*2), (+1)] 2 == 9
composeAll' [] 5 == 5
minimum2 [1,2,3,4] 100 == 1
minimum2 [] 100 == 100
minimum2 [1,2,3,0,2] 101 == 0
maximum2 [1,2,3,4] 0 == 4
maximum2 [] 0 == 0
maximum2 [1,2,3,1,2] 0 == 3
minimum2' [1,2,3,4] 100 == 1
minimum2' [] 100 == 100
minimum2' [1,2,3,0,2] 101 == 0
maximum2' [1,2,3,4] 0 == 4
maximum2' [] 0 == 0
maximum2' [1,2,3,1,2] 0 == 3
reverse2 [1,2,3] == [3,2,1]
reverse2 [0,3,0] == [0,3,0]
reverse2 [] == []
reverse2' [1,2,3] == [3,2,1]
reverse2' [0,3,0] == [0,3,0]
reverse2' [] == []
concatMap2 (\x -> [x, x + 1]) [1,2,3] == [1,2,2,3,3,4]
concatMap2 (\x -> replicate x x) [1,2,3] == [1,2,2,3,3,3]
concatMap2 (const []) [1,2,3,4,5] == []
concatMap2' (\x -> [x, x + 1]) [1,2,3] == [1,2,2,3,3,4]
concatMap2' (\x -> replicate x x) [1,2,3] == [1,2,2,3,3,3]
concatMap2' (const []) [1,2,3,4,5] == []
safeIndex 3 [1,2,3,4] == Just 4
safeIndex (-1) [1,2,3,4] == Nothing
safeIndex 4 [1,2,3,4] == Nothing
```

```
safeIndex' 3 [1,2,3,4] == Just 4
safeIndex' (-1) [1,2,3,4] == Nothing
safeIndex' 4 [1,2,3,4] == Nothing
```