1. Пусть дан тип `data List a = Nil | Cons a (List a)` для представления списков:
    1. `[1 point]` Сделайте его представителем класса типов `Functor`
    2. `[0.5 points]` Реализуйте рекурсивную функцию `append :: List a -> List a -> List a` для конкатенации двух списков
    3. `[2 points]` Сделайте список представителем классов типов `Applicative` и `Monad`
    4. `[1 point]` Реализуйте **рекурсивную** функцию `replace :: List a -> Int -> (List Int, Int)`, которая заменяет все значения в списке уникальными целыми значениями (не повторяющимися). Функция принимает неиспользованное уникальное значение и возвращает, кроме самого списка, следующее уникальное значение (которое не содержится в этом списке)
    5. `[2 points]` Реализуйте `replace'` с использованием монады `State`: `replace' :: List a -> Int -> State Int (List Int)`
2. Пусть дан тип для представления деревьев: `data Tree a = Empty | Node a (Tree a) (Tree a)`
    1. `[0.5 points]` Реализуйте функцию `leaves :: Tree a -> Int`, которая вычисляет количество листьев дерева
    2. `[0.5 points]` Реализуйте функцию `fullNodes :: Tree a -> Int`, которая вычисляет количество вершин дерева, имеющих ровно двух потомков
    3. `[4 points]` С помощью библиотеки `QuickCheck` проверьте, что для любого дерева `t` выполняется свойство: `fullNodes t + 1 == leaves t`
3. `[1.5 points]` Добавьте следующие правила в функцию `simplify'`
    1. $\ln 1 = 0$
    2. $\ln x^n = n \cdot \ln x$, где x - произвольное выражение
    3. $\ln \frac{x}{y} = \ln x - \ln y$
4. `[3 points]` Напишите функцию, вычисляющую производную: `derivative :: Expr -> Expr`
5. `[4 points]` "Крутой" последовательностью, назовем такую последовательность чисел, где каждое число не меньше суммы предыдущих чисел. Напишите, функцию, которая найдет длину наибольшей "крутой" подпоследовательности в заданном списке. Пользоваться модулем `Data.List` запрещается.