Функциональное программирование
Задание V

Используйте GHC версии не ниже 8.4. Решение присылайте в виде tar.gz-,zip-архива с исходниками или ссылку на github-репозиторий.

FP1 Реализуйте функцию, циклически сдвигающую элементы списка влево:

```
circShiftL :: Int -> [a] -> [a]
-- circShiftL 2 [1,2,3,4] == [3,4,1,2]
-- circShiftL (-1) [1,2,3,4] == [4,1,2,3]
```

FP2 Реализуйте следующие функции. В данном задании запрещено использовать рекурсию в явном виде. Стандартные функции и генераторы использовать можно.

```
-- По списку возвращает список пар -- (индекс, элемент)
indices :: [a] -> [(Integer, a)]
-- "Обнуляет" элементы данного списка, неудовлетворяющие заданному условию
zeroBy :: Monoid a => [a] -> (a -> Bool) -> [a]
{- Формирует список, каждый элемент которого - сумма соответствующих
 элементов исходных списков. Длина результата ограничена длиной самого
 короткого списка -}
triplewiseSum :: [Integer] -> [Integer] -> [Integer] -> [Integer]
```

FP3 Используя $unfoldr$, реализуйте функцию, которая возвращает в обратном алфавитном порядке список символов, попадающих в заданный парой диапазон. Попадание символа x в диапазон пары (a,b) означает, что $a \leq x$ и $x \leq b$.

```
revRange :: (Char,Char) -> [Char]
revRange = unfoldr fun
fun = undefined
```

FP4 С помощью приема "tying the knot"("завязывание в узел") реализуйте функцию, генерирующую бесконечный список вида $\left[1, \frac{1}{k}, \frac{1}{k^2}, \frac{1}{k^3}, \ldots\right]$

```
seriesK :: Int -> [Rational]
-- take 3 $ seriesK 2 == [1 % 1,1 % 2,1 % 4]
-- take 4 $ seriesK 3 == [1 % 1,1 % 3,1 % 9,1 % 27]
```

FP5 Сделайте сортированный список представителем класса Monoid (операция — слияние списков)

```
newtype SortedList a = SortedList { getSorted :: [a] } deriving (Eq, Show)
```

FP6 Реализуйте функцию, осуществляющую сортировку слиянием, существенно используя моноидальность отсортированных списков.

```
msort :: Ord a => [a] -> SortedList a
```

FP7 Сделайте двоичное дерево

```
data Tree a = Nil | Node (Tree a) a (Tree a) deriving (Eq, Show)
```

представителем класса типов $Foldable$, реализовав симметричную стратегию (in-order traversal). Реализуйте также три другие стандартные стратегии (pre-order traversal, post-order traversal и level-order traversal), упаковав дерево в типы-обертки

```
newtype Preorder a = PreO (Tree a) deriving (Eq, Show)
newtype Postorder a = PostO (Tree a) deriving (Eq, Show)
newtype Levelorder a = LevelO (Tree a) deriving (Eq, Show)
```

и сделав эти обертки представителями класса $Foldable$.