Вариант 1

1. Напишите функцию `strReplace :: Eq a => [a] -> [a] -> [a]`, которая принимает три списка и заменяет в третьем списке все непересекающиеся вхождения первого списка на второй список с помощью функций `length`, `(==)`, `take`, `drop` и `splitAt`.

2. Без использования рекурсии напишите функцию `pack :: Eq a => [a] -> [[a]]`, которая собирает одинаковые стоящие рядом элементы в список. Например:

```
> pack "aaaabccaadeeee"
["aaaa","b","cc","aa","d","eeee"]
```

Вариант 2

1. Напишите функцию `elemIndices :: Eq a => a -> [a] -> [Int]`, которая находит, под какими индексами в списке встречается заданный элемент.

2. Без использования рекурсии напишите функцию `uniq :: Eq a => [a] -> [a]`, которая удаляет копии одинаковых соседних элементов из списка. Например:

```
> uniq [1,2,2,3,3,3,1,1]
[1,2,3,1]
```

Вариант 3

1. Напишите функцию `dropEvery :: [a] -> Int -> [a]`, такую что `dropEvery ls n` удаляет из списка `ls` каждый `n`-й элемент, считая с 1. Например:

```
> dropEvery "abcdefghik" 3
"abdeghk"
```

2. Напишите функцию `longSum :: [[Int]] -> [Int]`, которая по списку списков чисел строит список чисел, получающийся при сложении всех списков выписанных один под другим "столбиком"с выравниванием по первому элементу каждого списка. Например, `sumEqMy [[1],[2,2],[3,3,3,3]]` возвращает `[6,5,5,3,3]`.

Вариант 4

1. Напишите функцию `repli :: [a] -> Int -> [a]`, которая повторяет каждый элемент списка заданное количество раз. Например, `repli "abc"3` возвращает `"aaabbbccc"`.

2. Напишите функцию `shift :: [a] -> Int -> [a]`, такую что `shift ls n` возвращает циклический сдвиг списка `ls` на `n` элементов влево, если `0 <= n < length`. Например: `rotate "abcdefgh"3` возвращает `"defghabc"`.
По желанию сделайте, чтобы функция работала при любом целом `n`. При `n < 0` сдвиг должен осуществляться вправо.

Вариант 5

1. Напишите рекурсивную функцию `evenElems :: [a] -> [a]`, которая возвращает список из элементов, стоящих на четных позициях (позицией головы списка считается 0). Например, `evenElems "Haskell is power"` возвращает `"Hseli oe"`.

2. Без использования рекурсии напишите функцию `prime :: Int -> Bool`, определяющую, является ли ее аргумент простым числом. Можно считать, что аргумент неотрицательный.

Вариант 6

1. Дан тип `data NestedList a = Elem a | List [NestedList a]`. Напишите функцию `flatten :: NestedList a -> [a]`, которая преобразует вложенный список в «плоский» список, рекурсивно заменяя каждых подсписок на его элементы. Например, `flatten (Elem 5)` возвращает `[5]`, `flatten (List [Elem 1, List [Elem 2, List [Elem 3, Elem 4], Elem 5]])` возвращает `[1,2,3,4,5]`, а `flatten (List [])` возвращает `[]`.

2. Без использования рекурсии напишите функцию `coprime :: Int -> Int -> Bool`, которая определяет, являются ли ее аргументы взаимно простыми. Можно считать, что аргументы неотрицательные.

Вариант 7

1. Напишите функцию `range :: Int -> Int -> Int -> Int`, такую что `range m n d` возвращает арифметическую последовательность с первым элементом m, разностью d и последним элементом p, таким что $p \leq n < p + d$.

2. Дан следующий тип данных.

```
data Tree a = Node a (Tree a) (Tree a) | Leaf deriving Show
```

Глубиной вершины в дереве называется неотрицательная величина, равная длине пути (т.е. числу ребер) от этой вершины до корня дерева. Напишите функцию `treeLevel :: Tree a -> Int -> [a]` возвращающую список вершин (точнее, содержащихся в них значений), имеющих заданную глубину.

Вариант 8

1. Напишите функцию `unevenHandWriting :: String -> String`, которая берет строку и возвращает ее же, но каждая третья буква должна стать прописной, если была строчной и наоборот.

2. Полином с целочисленными коэффициентами представлен списком коэффициентов, начиная со старшего. Без использования рекурсии напишите функцию `poly :: [Int] -> Int -> Int`, вычисляющую значение полинома на данном аргументе. Например:

```
> poly [5,3,2] 2
28
```

Вариант 9

1. Напишите функцию `sumDigits :: String -> Int`, складывающую все цифры в строке. Например, `sumDigits "IN 47405"` возвращает 20, а `sumDigits "No digits here!"` возвращает 0.

2. Без использования рекурсии напишите функцию `abbrev :: [String] -> [String]`, которая в заданном списке имен людей выполняет сокращение всех имен, кроме фамилии, до инициалов. Фамилией считается последнее слово. Например:

```
> abbrev ["Синицин", "Сергей Есенин", "Игорь Федорович Поддубный",
  "Иоганн Хризостом Вольфганг Амадей Моцарт"]
["Синицин", "С. Есенин", "И.Ф. Поддубный", "И.Х.В.А. Моцарт"]
```

Можно использовать функцию `words`.

Вариант 10

1. Напишите функцию `strPos :: Eq a => [a] -> [a] -> [Int]`, которая находит все вхождения первого списка во второй и возвращает список номеров элементов, с которых эти вхождения начинаются с помощью функций `length`, `(==)`, `take`, `drop` и `splitAt`.

2. Напишите функцию `transpose :: [[a]] -> [[a]]`, которая транспонирует прямоугольную матрицу. Матрица представлена в виде списка строк одинаковой длины. Например:

```
> transpose [[1,2,3],[4,5,6]]
[[1,4],[2,5],[3,6]]
```