

Задачи по Haskell

При решении задач можно использовать сопоставление по шаблону и свои имена переменных у функций. При решении задач можно использовать любые функции из модуля Prelude, а использование любых других модулей запрещено, если в условии задачи не сказано иное.

Функции, которые могут пригодиться

```
head :: [a] -> a
tail :: [a] -> [a]
sum :: [a] -> [a]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith3 :: (a -> b -> c -> d) -> [a] -> [b] -> [c] -> [d]
transpose :: [[a]] -> [[a]]
takeWhile :: (a -> Bool) -> [a] -> [a]
dropWhile :: (a -> Bool) -> [a] -> [a]
not :: Bool -> Bool
length :: [a] -> Int
map :: (a -> b) -> a -> b
concat :: [[a]] -> [a]
replicate :: Int -> a -> [a]
```

Задачи на списки

1. `mul :: Num a => [[a]] -> [[a]] -> [[a]]`
`mul a b = ?`

Перемножение двух матриц. Предполагается, что у матриц подходящие размеры. Можно использовать функцию `transpose` из модуля `Data.List`, подключать модуль не нужно.

```
mul [[1,2],[3,4]] [[0],[1]] = [[2],[4]]
```

2. `divide :: (a -> Bool) -> [a] -> [[a]]`
`divide p xs = ?`

Написать функцию `divide`, которая принимает предикат `p` и список `xs` и разбивает `xs` на список списков, начиная новый список каждый раз, когда изменяется значение предиката.

```
divide even [] = []
divide even [1,2,3,4] = [[1],[2],[3],[4]]
divide even [1,3,5,4,2] = [[1,3,5],[4,2]]
divide even [3,5,7] = [[3,5,7]]
```

3. `compress :: [a] -> [a]`
`compress xs = ?`

Если в списке идут подряд одинаковые элементы, то нужно оставить из них только одно значение.

```
compress "aaaabccaadeeee" = "abcade"
```

4. `encode :: [a] -> [(Int, a)]`
`encode xs = ?`

Результирующий список состоит из пар (число вхождений подряд элемента, элемент).

`encode "aaabccaadeee" = [(3, 'a'), (1, 'b'), (2, 'c'), (2, 'a'), (1, 'd'), (3, 'e')]`

5. `repli :: [a] -> Int -> [a]`
`repli xs n = ?`

Каждый элемент в списке `xs` повторить `n` раз.

`repli "abc" 3 = "aaabbbccc"`

6. `change :: a -> [[a]]`
`change n = ?`

Пусть есть список положительных достоинств монет `coins`, отсортированный по возрастанию. Напишите функцию `change`, которая разбивает переданную ей положительную сумму денег на монеты достоинств из списка `coins` всеми возможными способами. Порядок монет в каждом разбиении имеет значение, то есть наборы `[2,2,3]` и `[2,3,2]` — различаются. Список `coins` определять не надо. Например, если `coins = [2, 3, 7]`

`change 7 = [[2, 2, 3], [2, 3, 2], [3, 2, 2], [7]]`

7. `seqA :: Int -> Integer`
`seqA n = ?`

Функция `seqA` должна находить элементы следующей рекуррентной последовательности:

$$a_0 = 1; a_1 = 2; a_2 = 3; a_{k+3} = a_{k+2} + a_{k+1} - 2a_k$$

`seqA 0 = 1`
`seqA 4 = 2`
`seqA 5 = -1`