ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Цель лабораторной работы: научиться определять рекурсивные функции. Получить представление о механизме сопоставления с образцом. Приобрести навыки определения функций для обработки списков.

Ход работы:

1. Определите функцию, принимающую на вход целое число n и возвращающую список, содержащий n элементов, упорядоченных по возрастанию.

1) список натуральных чисел

```
natural n = take n [1..]
```

2) список нечётных натуральных чисел

```
oddIntegers n = [item | item <- natural n, odd item]
```

3) список чётных натуральных чисел

```
evenNumbers n = [item | item <- natural n, even item]
```

4) список квадратов натуральных чисел

```
sqrNumbers n = [item^2 | item <- natural n]
```

5) список факториалов

```
factNumbers n = [ product (natural item) | item <- natural n ]
```

6) список степеней двойки

```
powTwoNumbers n = [2^item | item <- natural n]
```

7) список треугольных чисел

С арифметической точки зрения, n-е треугольное число — это сумма n первых натуральных чисел.

```
triangularNumber n = [ sum (natural item) | item <- natural n ]
```

8) список пирамидальных чисел

Квадратные пирамидальные числа могут быть вычислены по формуле Фаулхабера

```
faulhaber n = ( 2*n^3 + 3*n^2 + n ) `div` 6
pyramidalNumbers n = [ faulhaber item | item <- natural n ]
```

Пример работы:

```
*Main> natural 5
[1,2,3,4,5]
*Main> oddIntegers 5
[1,3,5]
*Main> evenNumbers 5
[2,4]
*Main> sqrNumbers 5
[1,4,9,16,25]
*Main> factNumbers 5
[1,2,6,24,120]
*Main> powTwoNumbers 8
[2,4,8,16,32,64,128,256]
*Main> triangularNumber 15
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120]
*Main> pyramidalNumbers 13
[1,5,14,30,55,91,140,204,285,385,506,650,819]
```

2. Определите следующие функции:

1) функция, принимающая на входе список вещественных чисел и вычисляющую их арифметическое среднее. Постарайтесь, чтобы функция осуществляла только один проход по списку.

```
avg :: [Double] -> Double
avg list = (sum list) / fromIntegral (length list)
```

2) функция вычленения n-го элемента из заданного списка

```
getElement :: Int -> [a] -> a
getElement n list = list !! (n-1)
```

3) функция сложения элементов двух списков. Возвращает список, составленный из сумм элементов списков-параметров. Учесть, что переданные списки могут быть разной длины.

```
addElemList list1 list2 = [ a+b | (a,b) <- zip list1 list2 ]
```

4) функция перестановки местами соседних четных и нечетных элементов в заданном списке

```
swapNeighbours [] = []
swapNeighbours [x] = [x]
swapNeighbours (x:y:xs)
  | (odd y && even x) = y:x:swapNeighbours xs
  | (odd x && even y) = y:x:swapNeighbours xs
  | otherwise = x:y:swapNeighbours xs
```

5) функция twopow n вычисляет 2^n, исходя из следующих соображений. Пусть необходимо возвести 2 в степень n. Если n чётно, т.е. n = 2k, то $2^n = 2*2^{2*k} = (2^k)^2$. Если n нечётно, т.е. n = 2k + 1, то $2^n = 2^{2*k+1} = 2(2^k)^2$. Функция twopow не должна использовать оператор ^ или любую функцию возведения в степень из стандартной библиотеки. Количество рекурсивных вызовов функции должно быть пропорционально *log n*.

```
-- x^0 = 1, a even 0 возвращает True, поэтому проводится сопоставление с образцом
twopow 0 = 1
twopow n
  -- k = n `div` 2  из  n = 2k
  | even n = do
        let res = twopow (n `div` 2)
        res * res
  -- k = (n-1)/2  из   n = 2k + 1   n - 1 = 2k
  | otherwise = do
        let res = twopow ((n-1) `div` 2)
        2 * res * res
```

6) функция removeOdd, которая удаляет из заданного списка целых чисел все нечетные числа.

```
removeOdd :: [Integer] -> [Integer]
removeOdd list = [item | item <- list, even item]
```

7) функция removeEmpty, которая удаляет пустые строки из заданного списка строк. Например: removeEmpty ["", "Hello", "", "", "World!"] возвращает ["Hello", "World"]

```
removeEmpty :: [String] -> [String]
removeEmpty listStr = [str | str <- listStr, str /= ""]
```

8) функция countTrue :: [Bool] -> Int, возвращающая количество элементов списка, равных True.

```
countTrue :: [Bool] -> Int
countTrue listBool = length [item | item <- listBool, item]
```

9) функция makePositive, которая меняет знак всех отрицательных элементов списка чисел, например: makePositive [-1,0,5,-10,-20] дает [1,0,5,10,20]

```
makePositive :: [Integer] -> [Integer]
makePositive numbers = [abs number | number <- numbers]
```

10) функция delete :: Char -> String -> String, которая принимает на вход строку и символ и возвращает строку, в которой удалены все вхождения символа. Пример: delete '1' "Hello world!" должно возвращать "Heo word!"

```
delete :: Char -> String -> String
delete c str = [s | s <- str, s /= c]
```

11) функция substitute :: Char -> Char -> String -> String, которая заменяет в строке указанный символ на заданный. Пример: substitute 'e' 'i' "eigenvalue" возвращает "iiginvalui"

```
substitute :: Char -> Char -> String -> String
substitute c1 c2 str = substitute' c1 c2 str ""
substitute' :: Char -> Char -> String -> String -> String
substitute' c1 c2 ""             acc = reverse acc
substitute' c1 c2 (head:tail) acc
  | head == c1 = substitute' c1 c2 tail (c2:acc)
  | otherwise  = substitute' c1 c2 tail (head:acc)
```

Пример работы:

```
*Main> avg [1.0,2.0,4.5]
2.5
*Main> getElement 2 [5,4,3,7]
4
*Main> addElemList [1,2,3] [1,2,3,4]
[2,4,6]
*Main> swapNeighbours [1,2,3,4,5]
[2,1,4,3,5]
*Main> swapNeighbours [1,3,4,5,6]
[1,3,5,4,6]
*Main> twopow 8
256
*Main> twopow 10
1024
*Main> removeOdd [1,2,3,4,5]
[2,4]
*Main> removeEmpty ["", "World", "", "", "Haskell"]
["World","Haskell"]
*Main> countTrue [True,True,False]
2
*Main> makePositive [-1,0,5,-10,-20]
[1,0,5,10,20]
*Main> delete 'l' "Hello Haskell!"
"Heo Haske!"
*Main> substitute 'g' 'h' "google"
"hoohle"
```

Вывод: научился определять рекурсивные функции. Получил представление о механизме сопоставления с образцом. Приобрел навыки определения функций для обработки списков.