МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №5**

по дисциплине: **«Функциональное программирование»**

на тему: «**Функции высшего порядка**»

ВЫПОЛНИЛ студент группы 16 ИТ-3

Яблонский А.С.

ПРОВЕРИЛ преподаватель

Попкова Д.В.

Полоцк, 2018 г.

**Задание №1**

Определите следующие функции с использованием функций выс-

шего порядка:

1. Функция вычисления арифметического среднего элементов списка вещественных чисел с использованием функции foldr. Функция должна осуществлять только один проход по списку.  
   **Реализация:**

average :: [Double] -> Double

average x = (foldr (\a b -> a+b) 0.0 x) / (fromIntegral (length x))

**Использование:**

average [4.22, 1.22, 1.23, 1.0]

1. Функция, вычисляющая скалярное произведение двух списков (используйте функции foldr и zipWith).  
   **Реализация:**

scalProd :: [Double] -> [Double] -> Double

scalProd x y = foldr (\a b -> a+b) 0.0 (zipWith (\a b-> a\*b) x y)

**Использование:**

scalProd [1.22, 1.23, 1.0] [1.22, 1.23, 1.0]

1. Функция countEven, возвращающая количество четных элементов в списке.  
   **Реализация:**

countEven :: [Int] -> Int

countEven l = length (filter even l)

**Использование:**

countEven[1,2,3,4,5,6]

1. Функция quicksort, осуществляющая быструю сортировку списка по следующему рекурсивному алгоритму. Для того, чтобы отсортировать список xs, из него выбирается первый элемент (обозначим его x). Остальной список делится на две части: список, состоящий из элементов xs, меньших x и список элементов, больших x.

Эти списки сортируются (здесь проявляется рекурсия, поскольку они сортируются этим же алгоритмов), а затем из них составляется результирующий список вида as ++ [x] ++ bs, где as и bs — отсортированные списки меньших и больших элементов соответственно.  
**Реализация:**

quickSort :: Ord a => [a] -> [a]

quickSort [] = []

quickSort (x:xs) = quickSort (filter (\a -> a<x) xs) ++ [x] ++ quickSort (filter (\a -> a>=x) xs)

**Использование:**

quickSort [1,221,31,41]

1. Определенная в предыдущем пункте функция quicksort сортирует список в порядке возрастания. Обобщите ее: пусть она принимает еще один аргумент — функцию сравнения типа a -> a -> Bool и сортирует список в соответствие с нею.  
   **Реализация:**

quickSort2 :: (a -> a -> Bool) -> [a] -> [a]

quickSort2 fun [] = []

quickSort2 fun (x:xs) = quickSort2 fun (filter (fun x) xs) ++ [x] ++ (quickSort2 fun (filter (\a -> not(fun x a)) xs))

**Использование:**

quickSort2 (\a b -> a > b) [1,23,4234,10]

**Задание №2**

Вернитесь к заданиям из лабораторной работы No3 и реализуйте их с помощью функций высшего порядка. Постарайтесь полностью исключить из определений функций явный проход по списку.

**Реализация:**

-- ЗАДАНИЕ 1

getByName :: String -> [Record]

getByName name = searchByName2 name myList

searchByName2 :: String -> [Record] -> [Record]

searchByName2 name records = filter (\a -> (getName a) == name) records

**Использование:**

getByName "Vano"

**Реализация:**

-- ЗАДАНИЕ 2

getByLetter :: Char -> [Record]

getByLetter letter = searchByLetter2 letter myList

searchByLetter2 :: Char -> [Record] -> [Record]

searchByLetter2 letter records = filter (\a -> toUpper (head (getName a)) == toUpper letter) records

**Использование:**

getByLetter 'V'

**Реализация:**

-- ЗАДАНИЕ 3

getAssignment :: Date -> [Record]

getAssignment date = searchByDate2 date myList

searchByDate2 :: Date -> [Record] -> [Record]

searchByDate2 date records = filter (\a -> (not (isContact a) && (getDate a)==date)) records

**Использование:**

getAssignment Dm {d = 3, m = 4}

getAssignment Dm {d = 18, m = 9}

**Вывод**

Оказывается, функции высшего порядка могут быть очень полезны при разработке. Они значительно сокращают количество кода и время, затраченое на их написание. Также весьма полезными оказались лямбда-функции, их использование также сокращает количество кода.