Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №9 по курсу «Функциональное и логическое программирование»

Тема Использование функционалов и рекурсии

Студент Кононенко С.С.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватели Толпинская Н.Б., Строганов Ю.В.

Москва — 2021 г.

Задание 1

Постановка задачи. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.

Решение.

Листинг 1 – Решение задания 1

```
(defun make-select-in-ten (lst)
    (reduce #'(lambda (acc el) (if (and (> el 1) (< el 10))
                                   (append acc (cons el Nil))
                                   acc))
                                   lst :initial-value ()))
```

Задание 2

Постановка задачи. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

Решение.

Листинг 2 – Решение задания 2

```
(defun make-cartesian (flst slst)
    (mapcan #'(lambda (fel) (mapcar #'(lambda (sel) (cons fel sel)) slst)) flst))
```

Задание 3

Постановка задачи. Почему так реализовано `reduce` и в чем причина?

Решение. `(reduce #'+ ()) -> 0`

Поведение в данном примере обусловлено работой функции `+`. Эта функция – функционал, который при 0 количестве аргументов возвращает значение 0. Если подать на вход `reduce` функцию, которая не может обработать 0 аргументов, то вызов `reduce` с пустым списком в качестве второго

аргумента вернет ошибку. При этом, если подано более одного аргумента, то `reduce` выполняет следующие действия:

1. Сохраняет первый элемент списка в область памяти (`acc`);
2. Для всех остальных элементов списка выполняет переданную в качестве первого аргумента функцию, подавая на вход 2 аргумента (`acc` и очередной элемент списка) и сохраняя результат в `acc`.

Для умножения ситуация аналогичная.

Задание 4

Постановка задачи. Пусть `list-of-list` список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов `list-of-list`.

Решение.

Листинг 3 – Решение задания 4

```
(defun make-lsts-len (lst)
    (reduce #'(lambda (acc el) (+ acc (length el))) lst :initial-value 0))
```

Задание 5

Постановка задачи. Используя рекурсию, написать функцию, которая по исходному списку строит список квадратов чисел смешанного структурированного списка.

Решение.

Листинг 4 – Решение задания 5

```
(defun make-square-lsts (lst)
    (cond ((null lst) Nil)
    ((symbolp (car lst)) (make-square-lsts (cdr lst)))
    ((numberp (car lst)) (cons (* (car lst) (car lst)) (make-square-lsts (cdr lst))))
    (T (nconc (make-square-lsts (car lst)) (make-square-lsts (cdr lst))))))
```

Ответы на контрольные вопросы

Вопрос 1. Классификация рекурсивных функций.
Ответ. Классификация рекурсивных функций:

- Простая (рекурсивный вызов — единственный);
- Второго порядка (несколько рекурсивных вызовов);
- Взаимная рекурсия (используются несколько рекурсивных функций, которые могут друг друга вызывать).
- Хвостовая рекурсия (при очередном вызове рекурсивной функции все действия до входа выполнены, а при выходе ничего более делать не приходится);
- Дополняемая рекурсия (результат рекурсии используется, как аргумент некоторой другой функции (которую называют *дополняемой функцией*); частный случай — `cons`-дополняемая рекурсия).