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 №6 по курсу "Функциональное и логическое программирование"

Тема Использование функционалов

Студент Цветков И.А.

Группа ИУ7-63Б

Оценка (баллы)

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

Москва — 2022 г.

1 Практические задания

1.1 Задание 1

Условие: напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции.

```
(defun minus10 (lst)
    (mapcar #'(lambda (elem) (cond (((listp elem) (minus10 elem))
                                    ((numberp el) (- elem 10))
                                    (T elem))
                               )) lst)
)
```

1.2 Задание 2

Условие: напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

- все элементы списка — числа;
- элементы списка – любые объекты.

```
; a)
(defun mul-num (lst num)
    (mapcar #'(lambda (elem) (* elem num)) lst)
)


; б)
(defun mul-num (lst num)
    (mapcar #'(lambda (elem) (cond ((listp elem) (mul-num elem num))
                                   ((numberp elem) (* elem num))
                                   (T elem)
                               )
                    ) lst
    )
```

```
15 )
```

1.3 Задание 3

Условие: написать функцию, которая по своему списку-аргументу `lst` определяет, является ли он палиндромом (то есть равны ли `lst` и `(reverse lst)`).

```
1 (defun palindrom-func (lst rev_lst)
2     (mapcar #'(lambda (elem1 elem2) (equal elem1 elem2)) lst
          rev_lst)
3 )
4
5
6 (defun palindrom (lst)
7     (reduce #'(lambda (elem1 elem2) (and elem1 elem2))
8                               (palindrom-func lst (reverse lst)))
9 )
```

1.4 Задание 4

Условие: написать предикат `set-equal`, который возвращает `t`, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
1 (defun my-subsetp (set1 set2)
2     (reduce #'(lambda (prev_res1 elem1)
3                       (and prev_res1 (reduce #'(lambda (prev_res2
                              elem2)
4                         (or prev_res2 (equal elem1 elem2))
5                         ) set2 :initial-value Nil))
6               ) set1 :initial-value T)
7 )
8
9 (defun set-equal (lst1 lst2)
10    (and (my-subsetp lst1 lst2) (my-subsetp lst2 lst1))
11 )
```

1.5 Задание 5

Условие: написать функцию, которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
(defun quad (lst)
    (mapcar #'(lambda (elem) (* elem elem)) lst)
)
```

1.6 Задание 6

Условие: напишите функцию, `select-between`, которая из списка-аргумента из 5 чисел выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел (+ 2 балла)).

```
(defun select-between (lst a b)
    (remove-if #'(lambda (elem) (not (or (> a elem b) (< a elem
       b)))) lst)
)
```

1.7 Задание 7

Условие: написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

```
(defun decart (lst1 lst2)
    (mapcan #'(lambda (elem1) (
        mapcar #'(lambda (elem2) (list elem1 elem2)) lst2
    )) lst1)
)
```

1.8 Задание 8

Условие: почему так реализовано `reduce`, в чем причина?

Все дело в параметре `initial-value`, который указывается, а затем применяется функция. По умолчанию, для операции сложения этот параметр равен 0, а для опрации умножения – 1. Поэтому:

```
(reduce #'+ ()) ; -> 0
```

Случай ниже выдаст ошибку:

```
(reduce #'+ 0)   ; -> Error
```

Ошибка: `The value 0 is not of type SEQUENCE`

1.9 Задание 9

Условие: пусть `list-of-list` список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов `list-of-list`, т.е. например для аргумента `((1 2) (3 4)) -> 4`.

```
(defun deep-length (lst)
    (reduce '+ (mapcar #'(lambda (elem)
                           (cond ((listp elem) (deep-length elem))
                                 (T 1)
    )) lst))
)
```