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»

Лабораторная работа № 6

Дисциплина Функциональное и логическое программирование

Тема Рекурсивные функции

Студент Ильясов И. М.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Толпинская Н. Б.

Москва, 2020 г.

Задание. Есть два списка с фамилиями работников и их зарплатами соответственно. Позиции фамилий работников из первого списка соответствуют позициям их зарплат из второго списка (Ivanov => 50000, Stepanov => 100000).

Необходимо получить результирующий список из двух списков в формате «фамилия-зарплата»:

- при помощи двухэлементных списков;
- при помощи точечных пар;

Два списка с фамилиями работников и их зарплатами:

```
(defvar workers)
(defvar salary)

(setq workers (Stepanov Ivanov Gorbunov))
(setq salary (100000 50000 80000))
```

Способ №1: с использованием двухэлементных списков.

Функция создания результирующего списка, состоящего из двухэлементных списков:

```
(defun merge_list (workers salary)
        (mapcar #'list workers salary)
)
```

Способ №2: с использованием точечных пар.

Функция создания результирующего списка, состоящего из точечных пар:

```
(defun merge_cons (workers salary)
        (mapcar #'cons workers salary)
)
```

Изменение зарплат всех работников с использованием функционалов:

```
(defun multiply (el)
        (list (car el) (* (cadr el) 2))
)


(defun task (lst)
        (mapcar #'func lst)
)
```

Изменение зарплат работников, у которых зарплата меньше 60000 с использованием функционалов:

```
(defun task_if (lst)
        (if (< (cadr lst) 60000)
                (func lst)
                lst
        )
)

(print (mapcar #'task_if lst1))
```

Задание.

- ```
  (defun how_alike (x y)
          (cond
                  ((or (= x y) (equal x y)) ‘same)
                  ((and (oddp x) (oddp y)) ‘odd)
                  ((and (evenp x) (evenp y) ‘even)
                  (T ‘difference)
          )

  )
  ```

- ```
  (defun how_alike_if (x y)
          (if (or (= x y) (equal x y))
                  ‘same
                  (if (and (oddp x) (oddp y))
                          ‘odd
                          (if (and (evenp x) (evenp y))
                                  ‘even
                                  ‘difference
                          )
                  )
          )

  )
  ```

- (defun how_alike_or (x y)
    (or
        (and (or (= x y) (equal x y)) ‘same)
        (and (and (oddp x) (oddp y)) ‘odd)
        (and (and (evenp x) (evenp y)) ‘even)
)

)