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»

Отчёт по лабораторной работе №5 по курсу

«Функциональное и логическое программирование»

Тема Использование управляющих структур, работа со списками.

Студент Сироткина П.Ю.

Группа ИУ7-66Б

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

Москва — 2022 г.

Практические задания

1. Написать функцию, которая по своему списку-аргументу lst определяет, является ли он палиндромом (то есть равны ли lst и reverse(lst)).

```
(defun palindrome(lst) (equal lst (reverse lst)))
```

2. Написать предикат set-equal, который возвращает T, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
(defun set_equal(set1 set2)
  (and (= (length set1) (length set2))
     (every #'(lambda (elem) (member elem set1 :test #'equal)) set2)
     (every #'(lambda (elem) (member elem set2 :test #'equal)) set1)))
```

3. Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна.столица), и возвращает по стране столицу, а по столице - страну.

```
(defun get_country(table capital)
    (cond ((null table) Nil)
          ((equal (cdar table) capital) (caar table))
          (T (get_country (cdr table) capital))))

(defun get_country(table capital)
    (car (rassoc capital table)))

(defun get_capital(table country)
    (cond ((null table) Nil)
          ((equal (caar table) country) (cdar table))
          (T (get_capital (cdr table) country))))

(defun get_capital(table country)
    (cdr (assoc country table)))
```

4. Напишите функцию swap-first-last, которая переставляет в списке-аргументе первый и последний элементы.

```
(defun swap_first_last(lst)
  (let ((first_elem (car lst)))
        (setf (car lst) (car (last lst)))
        (setf (car (last lst)) first_elem)
        lst))
```

5. Напишите функцию swap-two-element, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке.

```
(defun swap_two_elements(lst i j)
    (if (and (< -1 i (length lst)) (< -1 j (length lst)))
        (let ((i_elem (nth i lst)) (j_elem (nth j lst)))
            (setf (nth i lst) j_elem) (setf (nth j lst) i_elem) lst)
        lst))
```

6. Напишите две функции, swap-to-left и swap-to-right, которые производят одну круговую перестановку в списке-аргументе влево и вправо соответственно.

```
(defun swap_to_left(lst)
  (cond ((null lst) Nil)
      (T (append (cdr lst) (cons (car lst) Nil)))))

(defun swap_to_right(lst)
  (cond ((null lst) Nil)
      (T (cons (car (reverse lst)) (reverse (cdr (reverse lst)))))))
```

7. Напишите функцию, которая добавляет к множеству двухэлементных списков новый двухэлементный список, если его там нет.

```
(defun add_pair_to_set(src_set pair)
  (if (not (member pair src_set :test #'equal))
            (append src_set (list pair))
             src_set))
```

8. Напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка-аргумента, когда

- Все элементы списка – числа;
- Элементы списка – любые объекта.

```
(defun mul_num(lst multiplier)
(mapcar #'(lambda(elem) (* elem multiplier)) lst))

(defun mul(lst multiplier)
  (mapcar #'(lambda (elem)
                          (cond ((listp elem) (mul (cdr elem) multiplier))
                                ((numberp elem) (* elem multiplier))
                                (T elem))) lst))
```

9. 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)
    (cond ((null lst) Nil)
          ((< a (car lst) b)(cons (car lst)(select_between (cdr lst) a b)))
          (T (select_between (cdr lst) a b))))
```