



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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 9

Дисциплина	Функциональное и логическое программирование.
Тема	Использование функционалов и рекурсии
Студент	Степанов А. О.
Группа	ИУ7-63Б
Оценка (баллы)	
Преподаватель	Толпинская Н.Б.

Москва, 2020 г.

ЗАДАНИЕ 1

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 set-equal (list1 list2)
2   (let
3     (
4       (l1 (sort list1 #'<))
5       (l2 (sort list2 #'<))
6     )
7     (equal l1 l2)
8   )
9 )
10
11 (set-equal '(1 2 3 4) '(4 3 2 1)) ;;; T
12 (set-equal '(1 2 3) '(3 2 4)) ;;; Nil
```

ЗАДАНИЕ 2

Напишите необходимые функции, которые обрабатывают таблицу из точечных пар: (страна. столица), и возвращают по стране – столицу, а по столице – страну.

```
1 (setq countries '(Russia USA GB Belarus))
2 (setq cities    '(Moscow Washington London Minsk))
3
4 (defun cons_merge (countries cities)
5   (mapcar #'(lambda (ctr cty) (cons ctr cty)) countries cities)
6 )
7
8 (setq cons_cc (cons_merge countries cities))
9
10 (defun cons_city (cons_cc country)
11   (reduce #'(lambda (a b) (or a b))
12     (mapcar #'(lambda (el)
13       (and (equal (car el) country) (cdr el)))
14     ) cons_cc
15   )
16 )
17 )
18
19 (cons_city cons_cc 'GB) ;;; London
20 (cons_city cons_cc 'NotExist) ;;; Nil
21 (cons_city cons_cc 'Russia) ;;; Moscow
```

```

22
23 (defun cons_country (cons_cc city)
24     (reduce #'(lambda (a b) (or a b))
25         (mapcar #'(lambda (el)
26             (and (equal (cdr el) city) (car el))
27             ) cons_cc
28         )
29     )
30 )
31
32 (cons_country cons_cc 'London) ;;; GB
33 (cons_country cons_cc 'NotExist) ;;; Nil
34 (cons_country cons_cc 'Moscow) ;;; Russia

1 (setq countries '(Russia USA GB Belarus))
2 (setq cities '(Moscow Washington London Minsk))
3
4 (defun cons_merge_cc (countries cities)
5     (cond
6         ((null countries) Nil)
7         ((null cities) Nil)
8         (T (cons
9             (cons (car countries) (car cities))
10            (cons_merge_cc (cdr countries) (cdr cities))
11        ))
12    )
13 )
14
15 (setq cons_merged_cc (cons_merge_cc countries cities))
16
17 (defun cons_find_country (list_cc city)
18     (cond
19         ((null list_cc) Nil)
20         ((equal (cadr list_cc) city) (caar list_cc))
21         (T (cons_find_country (cdr list_cc) city))
22     )
23 )
24
25 (cons_find_country cons_merged_cc 'Moscow) ;;; Russia
26 (cons_find_country cons_merged_cc 'Minsk) ;;; Belarus
27 (cons_find_country cons_merged_cc 'Sidney) ;;; Nil
28
29 (defun cons_find_city (list_cc country)

```

```

30      (cond
31        ((null list_cc) Nil)
32        ((equal (caar list_cc) country) (cdar list_cc))
33        (T (cons_find_city (cdr list_cc) country))
34      )
35 )
36
37 (cons_find_city cons_merged_cc 'Russia) ;;; Moscow
38 (cons_find_city cons_merged_cc 'Belarus) ;;; Minsk
39 (cons_find_city cons_merged_cc 'Australia) ;;; Nil

```

ЗАДАНИЕ 3

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

а) все элементы списка – числа,

б) элементы списка – любые объекты.

```

1 (defun number_multiplication_car (lst n)
2   (mapcar #'(lambda (el) (* el n)) lst)
3 )
4
5 (number_multiplication_car '(1 2 3 4) '5) ;;; (5 10 15 20)
6
7 (defun number_multiplication_rec (lst n)
8   (cond
9     ((null lst) Nil)
10    ('T (cons (* n (car lst)) (multiplication_rec (cdr lst) n)))
11  )
12 )
13
14 (number_multiplication_rec '(1 2 3 4) '5) ;;; (5 10 15 20)
15
16 (defun multiplication_car (lst n)
17   (mapcar #'(lambda (el) (if (numberp el) (* el n) el)) lst)
18 )
19
20 (multiplication_car '(1 2 3 4) '5) ;;; (5 10 15 20)
21 (multiplication_car '(a 2 b 4) '5) ;;; (A 10 B 20)
22

```

```

23 (defun multiplication_rec (lst n)
24   (cond
25     ((null lst) Nil)
26     ('T
27       (let*
28         (
29           (el (car lst))
30           (final_el (if (numberp el) (* el n) el))
31           (next_iteration (multiplication_rec (cdr lst) n))
32         )
33       (cons final_el next_iteration)
34     )
35   )
36 )
37 )
38
39 (multiplication_rec '(1 2 3 4) '5) ;;; (5 10 15 20)
40 (multiplication_rec '(a 2 b 4) '5) ;;; (A 10 B 20)

```

ЗАДАНИЕ 4

Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

```

1 (defun minus_ten_car (lst)
2   (mapcar #'(lambda (el) (- el 10)) lst)
3 )
4
5 (minus_ten_car '(1 2 3 4 5)) ;;; (-9 -8 -7 -6 -5)
6
7 (defun minus_ten_rec (lst)
8   (cond
9     ((null lst) Nil)
10    ('T (cons (- (car lst) 10) (minus_ten_rec (cdr lst))))
11  )
12 )
13
14 (minus_ten_rec '(1 2 3 4 5)) ;;; (-9 -8 -7 -6 -5)

```

ЗАДАНИЕ 5

Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

```

1 (defun first_list_map (lst)
2   (reduce #'
3     (lambda (el1 el2)
4       (or
5         (and (not (atom el1)) el1)
6         (and (not (atom el2)) el2)
7       )
8     ) lst
9   )
10 )
11
12 (first_list_map '(1 (2 3) 4 5 (6 7))) ;;; (2 3)
13 (first_list_map '(1 2 3 4)) ;;; Nil
14
15 (defun first_list_rec (lst)
16   (cond
17     ((null lst) Nil)
18     ((not (atom (car lst))) (car lst))
19     ('T (first_list_rec (cdr lst)))
20   )
21 )
22
23 (first_list_rec '(1 (2 3) 4 5 (6 7))) ;;; (2 3)
24 (first_list_rec '(1 2 3 4)) ;;; Nil

```

ЗАДАНИЕ 6

Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10. (Вариант: между двумя заданными границами.)

```

1 (defun take_between_map (a b lst)
2   (reduce #'
3     (lambda (lst el)
4       (append
5         (if (numberp lst)
6           (if (and (<= a lst) (>= b lst))
7             (list lst)
8           )
9         lst
10      )
11      (if (and (<= a el) (>= b el))
12        (list el)
13      )

```

```

14         )
15     ) lst
16 )
17 )
18
19 (take_between_map '1 '10 '(-10 -5 0 5 10 15)) ;;; (5 10)
20 (take_between_map '-10 '5 '(-10 -5 0 5 10 15)) ;;; (-10 -5 0 5)
21
22 (defun take_between_rec (a b lst)
23     (cond
24         ((null lst) Nil)
25         ((and (<= a (car lst)) (>= b (car lst))) (cons (car lst) (take_betw
26             ('T (take_between_rec a b (cdr lst))))
27     )
28 )
29
30 (take_between_rec '1 '10 '(-10 -5 0 5 10 15)) ;;; (5 10)
31 (take_between_rec '-10 '5 '(-10 -5 0 5 10 15)) ;;; (-10 -5 0 5)

```

ЗАДАНИЕ 7

Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. (Напомним, что $A \times B$ это множество всевозможных пар (a, b) , где a принадлежит A , принадлежит B .)

```

1 (defun decart_map (lst1 lst2)
2     (mapcan #'
3         (lambda (x)
4             (mapcar #'(lambda (y) (cons x y)) lst2))
5         ) lst1
6     )
7 )
8
9 (decart_map '(1 2 3 4 5) '(a b c d e))
10 (decart_map '(1 2) '(a b)) ;;; ((1 . A) (1 . B) (2 . A) (2 . B))
11
12 (defun decart_one_element (el lst)
13     (cond
14         ((null lst) Nil)
15         ('T (cons (cons el (car lst)) (decart_one_element el (cdr lst))))
16     )
17 )
18

```

```

19 (defun decart_rec (lst1 lst2)
20   (cond
21     ((null lst1) Nil)
22     ('T (append (decart_one_element (car lst1) lst2) (decart_rec (cdr lst1) lst2)))
23   )
24 )
25
26 (decart_rec '(1 2 3 4 5) '(a b c d e))
27 (decart_rec '(1 2) '(a b)) ;;; ((1 . A) (1 . B) (2 . A) (2 . B))

```

ЗАДАНИЕ 8

Почему так реализовано `reduce`, в чем причина?

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

Поскольку список, к которому применяется функция **reduce**, является пустым, то результатом остается начальное значение, которое по умолчанию равно 0.