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 Козлова И.В.

Группа ИУ7-62Б

Оценка (баллы)

Преподаватель Толпинская Н.Б.

Преподаватель Строганов Ю.В.

Москва — 2022 г.

Практические вопросы

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

```
1  ; одноуровневые список
2  (defun minus-d (lst)
3      (mapcar #'(lambda (x) (- x 10))
4          lst))
5
6  (defun f (lst res)
7      (cond ((null lst) (reverse res))
8          (T (f (cdr lst) (cons (- (car lst) 10) res))) ) )
9
10 (defun my-minus (lst)
11     (f lst ()))
12
13 ; структурированный список
14 (defun minus-d-all (lst)
15     (mapcar #'(lambda (x)
16         (cond ((numberp x)(- x 10))
17             ((listp x)(minus-d-all x))
18             (t x)))
19         lst))
20
21 (defun f (lst num)
22     (cond ((null lst) ())
23         ((symbolp (car lst)) (cons (car lst) (f (cdr lst) num)))
24         ((listp (car lst)) (cons (f (car lst) num) (f (cdr lst) num)))
25         (T (cons (- (car lst) 10) (f (cdr lst) num))) ) )
```

2. 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  ; только числа
2  (defun mult (lst n)
3      (mapcar #'(lambda (x) (* x n)) lst))
4
5  ; одноур. список
6  (defun mult-els (lst num)
7      (mapcar #'(lambda (arg)
8          (cond ((numberp arg) (* arg num))
9              (t arg))) lst))
10
11 ; рекурсивно
12 (defun mult-els-rec (lst num res)
13     (cond ((null lst) (reverse res))
14         ((numberp (car lst)) (mult-els-rec (cdr lst) num (cons (* (car lst)
               num) res)))
15         (t (mult-els-rec (cdr lst) num (cons (car lst) res)))))
16         (defun f (lst num)
17         (mult-els-rec lst num ()))
18
19 ; б) элементы списка -- любые объекты
20 (defun mult-all (lst n)
21     (mapcar #'(lambda (x)
22         (cond ((numberp x)(* x n))
23             ((listp x)(mult-all x n))
24             (t x))) lst))
25
26 ; рекурсия
27 (defun mult-els-rec-deep (lst num)
28     (cond ((null lst) nil)
29         ((numberp (car lst)) (cons (* (car lst) num) (mult-els-rec-deep (cdr
               lst) num)))
30         ((listp (car lst)) (cons (mult-els-rec-deep (car lst) num)
               (mult-els-rec-deep (cdr lst) num)))
31         (t (cons (car lst) (mult-els-rec-deep (cdr lst) num)))))
```

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

```
(defun palindrome-p (list)
    (let* ((length (length list))
        (half-length (ceiling length 2))
        (tail (nthcdr half-length list))
        (reversed-head (nreverse (butlast list half-length))))
        (equal tail reversed-head)))

(defun is-palindrome-2 (lst)
    (equalp lst (reverse lst)))
```

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

```
; функционал
(defun my-subsetp (set1 set2)
    (reduce
        #'(lambda (acc1 set1-el)
            (and acc1 (reduce
                #'(lambda (acc2 set2-el)
                    (or acc2 (= set2-el set1-el))) set2 :initial-value Nil)))
        set1 :initial-value T))

(defun set-equal (set1 set2)
    (if (= (length set1) (length set2))
        (and (my-subsetp set1 set2) (my-subsetp set2 set1))
        Nil))

; рекурсия
(defun find-elem-in-set (set1 elem)
    (cond ((null set1) Nil)
        ((= (car set1) elem) T)
        (T (find-elem-in-set (cdr set1) elem)) ) )

(defun set-equal-rec (set1 set2)
    (cond ((null set1))
        ((find-elem-in-set set2 (car set1)) (set-equal-rec (cdr set1) set2))
        (T Nil)) )

(defun set-equal (set1 set2)
    (if (= (length set1) (length set2))
        (set-equal-rec set1 set2) Nil) )
```

5. Написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
1  ; одноуровневый список
2  (defun get-sqr-list (lst)
3      (cond
4          ((null lst) nil)
5          ((symbolp (car lst)) (cons (car lst) (get-sqr-list (cdr lst))))
6          ((numberp (car lst)) (cons (* (car lst) (car lst)) (get-sqr-list (cdr
               lst))))
7          (t (get-sqr-list (cdr lst)))))
8
9  ; рекурсия без накопления cons, но с reverse
10 (defun get-sqr-list (lst res)
11     (cond
12         ((null lst) (reverse res))
13         ((symbolp (car lst)) (get-sqr-list (cdr lst) (cons (car lst) res)))
14         ((numberp (car lst)) (get-sqr-list (cdr lst) (cons (* (car lst) (car
               lst)) res)))))
15
16 (defun get-sqr (lst)
17     (get-sqr-list (lst ())))
18
19 ; с функционалами
20 (defun get-sqr-helper (el)
21     (cond
22         ((numberp el) (cons (* el el) nil))
23         ((symbolp el) (cons el nil))
24         (t nil)))
25
26 (defun get-sqr-list-fun (lst)
27     (mapcan #'get-sqr-helper lst))
28
29 ; Рекурсивно для смешанного структурированного списка
30 (defun get-sqr-list (lst)
31     (cond
32         ((null lst) nil)
33         ((symbolp (car lst)) (cons (car lst) (get-sqr-list (cdr lst))))
34         ((listp (car lst)) (cons (get-sqr-list (car lst)) (get-sqr-list (cdr
               lst))))
35         ((numberp (car lst)) (cons (* (car lst) (car lst)) (get-sqr-list (cdr
               lst))))
36         (t (get-sqr-list (cdr lst)))))
37
38 ; С использованием функционала для смешанного структурированного списка
39 (defun get-sqr-helper (el)
40     (cond
41         ((listp el) (cons (get-sqr-list-fun el) nil))
42         ((numberp el) (cons (* el el) nil))
43         ((symbolp el) (cons el nil))
44         (t nil)))
45
46 (defun get-sqr-list-fun (lst)
47     (mapcan #'get-sqr-helper lst))
```

6. Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел (+ 2 балла)).

```
1  (defun bubble_move (lst)
2      (cond ((atom (cdr lst)) lst)
3          ((> (car lst) (cadr lst)) (cons (cadr lst) (bubble_move (cons (car
               lst) (cddr lst)))))
4          (T lst)  ))
5
6  (defun my-sort (lst)
7      (cond ((atom (cdr lst)) lst)
8      (T (bubble_move (cons (car lst) (my-sort (cdr lst)))))))
9
10 (defun find-elements (lst left right)
11     (remove-if #'(lambda (x) (null x))
12         (mapcar #'(lambda (x) (if (< left x right) x)) lst)))
13
14 (defun select-between (lst b1 b2)
15     (cond ((null lst) nil)
16         ((not (and (numberp b1) (numberp b2))) nil)
17         ((= b1 b2)(and (print "ERROR: wrong format of borders (equal)") nil))
18         ((> b1 b2)(my-sort (find-elements lst b2 b1)))
19         ((> b2 b1)(my-sort (find-elements lst b1 b2)))) )
20
21 ; также номер 8 из ЛР 7
```

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

```
1  ; функционалы
2  (defun decart (lstx lsty)
3      (mapcan #'(lambda (x)
4          (mapcar #'(lambda (y)
5          (list x y)) lsty)) lstx))
6
7  ; рекурсия без накапливаний cons
8  (defun decart-rec (el lst2 res)
9      (cond ((null lst2) res)
10     (t (decart-rec el (cdr lst2) (cons (cons el (cons (car lst2) Nil)) res)
           ))))
11
12 (defun decart (lst1 lst2 res)
13     (cond ((null lst1) res)
14     (t (decart (cdr lst1) lst2 (decart-rec (car lst1) lst2 res)))))
15
16 ; просто рекурсия
17 (defun decart-elem (lst elem)
18     (cond ((null lst) ())
19     (T (cons (list elem (car lst)) (decart-elem (cdr lst) elem)))) )
20
21 (defun decart (lst1 lst2)
22     (cond ((null lst1) nil)
23     (T (append (decart-elem lst2 (car lst1)) (decart (cdr lst1) lst2)))) )
```

8. Почему так реализовано reduce, в чем причина?
(reduce #+́ 0) -> 0
(reduce #+́ ()) -> 0

9. 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.

```
1  (defun my-length-rec (lst n)
2      (cond
3          ((null lst) n)
4          (T (my-length-rec (cdr lst) (+ n 1)))) )
5
6  (defun my-length (lst)
7      (my-length-rec lst 0) )
8
9  (defun list-of-list-rec (lst len)
10     (cond ((null lst) len)
11         ((atom (car lst)) (list-of-list-rec (cdr lst) (+ len 1)))
12         ((and (atom (caar lst)) (atom (cdar lst))) (list-of-list-rec (cdr
               lst) (+ len 2)))
13         (T (list-of-list-rec (cdr lst) (+ len (my-length (car lst)) )))))
14
15 (defun list-of-list (lst)
16     (list-of-list-rec lst 0))
```