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  ; только числа
3  (defun minus-d (lst)
4      (mapcar #'(lambda (x) (- x 10)) lst))
5
6  ; смешанный список
7  (defun minus-d (lst)
8      (mapcar #'(lambda (x) (cond ((numberp x) (- x 10)) (x))) lst))
9
10 ; рекурсия
11 ; только числа
12 (defun f (lst res)
13     (cond ((null lst) (reverse res))
14         (T (f (cdr lst) (cons (- (car lst) 10) res))) ) )
15
16 ; смешанный список
17 (defun f (lst res)
18     (cond ((null lst) (reverse res))
19         ((numberp (car lst)) (f (cdr lst) (cons (- (car lst) 10) res)))
20         (T (f (cdr lst) (cons (car lst) res))) ) )
21
22 (defun my-minus (lst)
23     (f lst ()))
24
25 ; структурированный список
26 ; вспомогательная функция
27 (defun dop-fun (lst)
28     (cond ((and (numberp (car lst)) (numberp (cdr lst)))
29             (cons (- (car lst) 10) (- (cdr lst) 10)))
30         ((and (symbolp (car lst)) (numberp (cdr lst)))
31             (cons (car lst) (- (cdr lst) 10)))
32         ((and (numberp (car lst)) (symbolp (cdr lst)))
33             (cons (- (car lst) 10) (cdr lst)))
34         ((and (atom (cdar lst)) (numberp (cdr lst)))
35             (cons (dop-fun (car lst)) (- (cdr lst) 10)))
36         ((and (atom (cdar lst)) (symbolp (cdr lst)))
37             (cons (dop-fun (car lst)) (cdr lst)))
38         (T lst)))
39
40 ; функционал
41 (defun minus-d-all (lst)
42     (mapcar #'(lambda (x)
43         (cond ((numberp x) (- x 10))
44             ((atom x) x)
45             ((atom (cdr x)) (dop-fun x))
46             ((listp x) (minus-d-all x)))) lst))
47
48
49
50
```

```
51 ; рекурсия
52 (defun f (lst)
53     (cond ((null lst) ())
54         ((symbolp (car lst)) (cons (car lst) (f (cdr lst))))
55         ((numberp (car lst)) (cons (- (car lst) 10) (f (cdr lst))))
56         ((atom (car lst)) (cons (car lst) (f (cdr lst) )))
57         ((atom (cdr lst)) (cons (dop-fun (car lst)) (f (cdr lst) )))
58         (T (cons (f (car lst)) (f (cdr lst))))) )
59
60 ; "хвостовая" рекурсия
61 (defun f (lst res)
62     (cond ((null lst) (reverse res))
63         ((symbolp (car lst)) (f (cdr lst) (cons (car lst) res)))
64         ((numberp (car lst)) (f (cdr lst) (cons (- (car lst) 10) res)))
65         ((atom (car lst)) (f (cdr lst) (cons (car lst) res)))
66         ((atom (cdr lst)) (f (cdr lst) (cons (dop-fun (car lst)) res)))
67         (T (f (cdr lst) (cons (f (csr lst) ()) res))) ) )
```

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  ; одноуровневые список
2  ; только числа
3  (defun mult-f (lst num)
4      (mapcar #'(lambda (x) (* x num)) lst))
5
6  ; смешанный список
7  (defun mult-f (lst num)
8      (mapcar #'(lambda (x) (cond ((numberp x) (* x num)) (x))) lst))
9
10 ; рекурсия
11 ; только числа
12 (defun f (lst num res)
13     (cond ((null lst) (reverse res))
14         (T (f (cdr lst) (cons (* (car lst) num) res))) ) )
15
16 ; смешанный список
17 (defun f (lst num res)
18     (cond ((null lst) (reverse res))
19         ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) num) res)))
20         (T (f (cdr lst) (cons (car lst) res))) ) )
21
22 (defun my-mult (lst num)
23     (f lst num ()))
24
25 ; структурированный список
26 ; вспомогательная функция
27 (defun dop-fun (lst num)
28     (cond ((and (numberp (car lst)) (numberp (cdr lst)))
29         (cons (* (car lst) num) (* (cdr lst) num)))
30     ((and (symbolp (car lst)) (numberp (cdr lst)))
31         (cons (car lst) (* (cdr lst) num)))
32     ((and (numberp (car lst)) (symbolp (cdr lst)))
33         (cons (* (car lst) num) (cdr lst)))
34     ((and (atom (cdar lst)) (numberp (cdr lst)))
35         (cons (dop-fun (car lst)) (* (cdr lst) num)))
36     ((and (atom (cdar lst)) (symbolp (cdr lst)))
37         (cons (dop-fun (car lst)) (cdr lst)))
38     (T lst)))
39
40 ; функционал
41 (defun mult-d-all (lst num)
42     (mapcar #'(lambda (x)
43         (cond ((numberp x) (* x num))
44             ((atom x) x)
45             ((atom (cdr x)) (dop-fun x))
46             ((listp x) (minus-d-all x)))) lst))
47
48
49
50
51
52
53
54
```

```
55 ; рекурсия
56 (defun f (lst)
57     (cond ((null lst) ())
58         ((symbolp (car lst)) (cons (car lst) (f (cdr lst))))
59         ((numberp (car lst)) (cons (* (car lst) num) (f (cdr lst))))
60         ((atom (car lst)) (cons (car lst) (f (cdr lst) )))
61         ((atom (cdr lst)) (cons (dop-fun (car lst)) (f (cdr lst) )))
62         (T (cons (f (car lst)) (f (cdr lst))))) )
63
64 ; "хвостовая" рекурсия
65 (defun f (lst res)
66     (cond ((null lst) (reverse res))
67         ((symbolp (car lst)) (f (cdr lst) (cons (car lst) res)))
68         ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) num) res)))
69         ((atom (car lst)) (f (cdr lst) (cons (car lst) res)))
70         ((atom (cdr lst)) (f (cdr lst) (cons (dop-fun (car lst)) res)))
71         (T (f (cdr lst) (cons (f (csr lst) ()) res))) ) )
```

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  ; одноуровневые список
2  ; только числа
3  (defun mult-f (lst)
4      (mapcar #'(lambda (x) (* x x)) lst))
5
6  ; смешанный список
7  (defun mult-f (lst)
8      (mapcar #'(lambda (x) (cond ((numberp x) (* x x)) (x))) lst))
9
10 ; рекурсия
11 ; только числа
12 (defun f (lst res)
13     (cond ((null lst) (reverse res))
14         (T (f (cdr lst) (cons (* (car lst) (car lst)) res))) ) )
15
16 ; смешанный список
17 (defun f (lst res)
18     (cond ((null lst) (reverse res))
19         ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) (car lst)) res)))
20         (T (f (cdr lst) (cons (car lst) res))) ) )
21
22 (defun my-mult (lst num)
23     (f lst num ()))
24
25 ; структурированный список
26 ; вспомогательная функция
27 (defun dop-fun (lst)
28     (cond ((and (numberp (car lst)) (numberp (cdr lst)))
29         (cons (* (car lst) (car lst)) (* (cdr lst) (car lst))))
30     ((and (symbolp (car lst)) (numberp (cdr lst)))
31         (cons (car lst) (* (cdr lst) (car lst))))
32     ((and (numberp (car lst)) (symbolp (cdr lst)))
33         (cons (* (car lst) (car lst)) (cdr lst)))
34     ((and (atom (cdar lst)) (numberp (cdr lst)))
35         (cons (dop-fun (car lst)) (* (cdr lst) (car lst))))
36     ((and (atom (cdar lst)) (symbolp (cdr lst)))
37         (cons (dop-fun (car lst)) (cdr lst)))
38     (T lst)))
39
40 ; функционал
41 (defun mult-d-all (lst)
42     (mapcar #'(lambda (x)
43         (cond ((numberp x) (* x x))
44             ((atom x) x)
45             ((atom (cdr x)) (dop-fun x))
46             ((listp x) (minus-d-all x)))) lst))
47
48
49 ; рекурсия
50 (defun f (lst)
51     (cond ((null lst) ())
52         ((symbolp (car lst)) (cons (car lst) (f (cdr lst))))
53         ((numberp (car lst)) (cons (* (car lst) (car lst)) (f (cdr lst))))
54         ((atom (car lst)) (cons (car lst) (f (cdr lst) )))
55         ((atom (cdr lst)) (cons (dop-fun (car lst)) (f (cdr lst) )))
56         (T (cons (f (car lst)) (f (cdr lst))))) )
57
```

```
58 ; "хвостовая" рекурсия
59 (defun f (lst res)
60     (cond ((null lst) (reverse res))
61         ((symbolp (car lst)) (f (cdr lst) (cons (car lst) res)))
62         ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) (car lst)) res)))
63         ((atom (car lst)) (f (cdr lst) (cons (car lst) res)))
64         ((atom (cdr lst)) (f (cdr lst) (cons (dop-fun (car lst)) res)))
65         (T (f (cdr lst) (cons (f (csr lst) ()) res))) ) )
```

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)).

```
(defun bubble_move (lst)
    (cond ((atom (cdr lst)) lst)
        ((> (car lst) (cadr lst))
            (cons (cadr lst) (bubble_move (cons (car lst) (cddr lst)))))
        (T lst) ))

(defun my-sort (lst)
    (cond ((atom (cdr lst)) lst)
        (T (bubble_move (cons (car lst) (my-sort (cdr lst)))))))

;-----------------------
(defun find-elements (lst left right)
    (remove-if #'(lambda (x) (null x))
        (mapcar #'(lambda (x) (if (< left x right) x)) lst)))

(defun select-between (lst b1 b2)
    (cond ((null lst) nil)
        ((not (and (numberp b1) (numberp b2))) nil)
        ((= b1 b2)(and (print "ERROR: wrong format of borders (equal)") nil))
        ((> b1 b2)(my-sort (find-elements lst b2 b1)))
        ((> b2 b1)(my-sort (find-elements lst b1 b2)))))
;-----------------------
; Рекурсивно. Для смешанного списка.
(defun select-rec-one-lvl (lst a b res)
    (cond ((null lst) res)
        ((and
            (numberp (car lst))
            (<= (car lst) b)
            (>= (car lst) a))
            (select-rec-one-lvl (cdr lst) a b (cons (car lst) res)))
        (t (select-rec-one-lvl (cdr lst) a b res))))
(defun select-between (lst)
    (select-rec-one-lvl lst 1 10 ()))

; 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

У reduce есть особый параметр

```
initial-value
```

который помещается перед последовательностью и затем применяется функция. Его можно задать, используя:

```
(reduce #'+ () :initial-value 100)  -> 100
```

По умолчанию, для сложения оно равно 0, для умножения 1.
Случай ниже выдаст ошибку

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

Ошибка:

```
debugger invoked on a TYPE-ERROR in thread
The value
0
is not of type
SEQUENCE
```

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  ; функционал
2  (defun list-of-list (lst)
3      (reduce #'(lambda (acc lst) (+ acc (length lst)))
4          (cons 0 lst)))
5
6  (defun my-length-rec (lst n)
7      (cond
8          ((null lst) n)
9          (T (my-length-rec (cdr lst) (+ n 1)))) )
10
11 (defun my-length (lst)
12     (my-length-rec lst 0) )
13
14 (defun list-of-list-rec (lst len)
15     (cond ((null lst) len)
16         ((atom (car lst)) (list-of-list-rec (cdr lst) (+ len 1)))
17         ((and (atom (caar lst)) (atom (cdar lst))) (list-of-list-rec (cdr
               lst) (+ len 2)))
18         (T (list-of-list-rec (cdr lst) (+ len (my-length (car lst)) )))))
19
20 (defun list-of-list (lst)
21     (list-of-list-rec lst 0))
```