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 №7 по курсу «Функциональное и логическое программирование»

Тема Рекурсивные функции

Студент Козлова И.В.

Группа ИУ7-62Б

Оценка (баллы)

Преподаватель Толпинская Н.Б.

Преподаватель Строганов Ю.В.

Москва — 2022 г.

Практические вопросы

1. Написать хвостовую рекурсивную функцию my-reverse, которая развернет верхний уровень своего списка-аргумента lst.

```
(defun move-to (lst res)
    (cond ((null lst) res)
    (T (move-to (cdr lst) (cons (car lst) res)))))

(defun my-reverse (lst)
    (move-to lst ()))
```

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

```
; рекурсия
(defun ret-first-lst (lst)
    (cond ((null lst) lst)
    ((and (listp (car lst))
        (not (null (car lst)))) (car lst))
    (t (ret-first-lst (cdr lst)))))

; функционалы
(defun ret-first-lst-fun (lst)
    (find-if #'(lambda (x) (and (listp x) (not (null x)))) lst))
```

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

```
1  ; рекурсивно для од. списка
2  (defun select-rec-one-lvl (lst a b res)
3      (cond ((null lst) res)
4          ((and
5              (numberp (car lst))
6              (<= (car lst) b)
7              (>= (car lst) a))
8                  (select-rec-one-lvl (cdr lst) a b (cons (car lst) res)))
9          (t (select-rec-one-lvl (cdr lst) a b res))))
10
11 ; Рекурсивно. Для смешанного структурированного списка.
12 (defun select-rec (lst a b res)
13     (cond ((null lst) res)
14         ((listp (car lst)) (cons (select-rec (car lst) a b res)
15         (select-rec (cdr lst) a b res)))
16         ((and
17             (numberp (car lst))
18             (<= (car lst) b)
19             (>= (car lst) a))
20                 (select-rec (cdr lst) a b (cons (car lst) res)))
21         (t (select-rec (cdr lst) a b res))))
22
23 ; С использованием функционала. Для смешанного списка.
24 (defun select-fun-one-lvl (lst a b)
25     (remove-if-not #'(lambda (el) (and (numberp el) (<= el b) (>= el a)))
           lst))
26
27 ; С использованием функционала. Для смешанного структурированного списка.
28 (defun select-fun (lst a b)
29     (mapcan #'(lambda (el)
30         (cond ((listp el) (select-fun el a b))
31             ((and (numberp el) (<= el b) (>= el a) (cons el nil))))) lst))
32
33 ; оберточная функция для каждой из предоставленной выше функции
34 (defun select-between (lst)
35     (select-rec lst 1 10 ()))
```

4. Напишите рекурсивную 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  ; одноуровневыq список
2  ; только числа
3  (defun f (lst num res)
4      (cond ((null lst) (reverse res))
5          (T (f (cdr lst) (cons (* (car lst) num) res))) ) )
6
7  ; смешанный список
8  (defun f (lst num res)
9      (cond ((null lst) (reverse res))
```

```
10          ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) num) res)))
11          (T (f (cdr lst) (cons (car lst) res))) ) )
12
13  (defun my-mult (lst num)
14      (f lst num ()))
15
16  ; структурированный список
17  ; вспомогательная функция
18  (defun dop-fun (lst num)
19      (cond ((and (numberp (car lst)) (numberp (cdr lst)))
20          (cons (* (car lst) num) (* (cdr lst) num)))
21      ((and (symbolp (car lst)) (numberp (cdr lst)))
22          (cons (car lst) (* (cdr lst) num)))
23      ((and (numberp (car lst)) (symbolp (cdr lst)))
24          (cons (* (car lst) num) (cdr lst)))
25      ((and (atom (cdar lst)) (numberp (cdr lst)))
26          (cons (dop-fun (car lst)) (* (cdr lst) num)))
27      ((and (atom (cdar lst)) (symbolp (cdr lst)))
28          (cons (dop-fun (car lst)) (cdr lst)))
29      (T lst)))
30
31  ; рекурсия
32  (defun f (lst)
33      (cond ((null lst) ())
34          ((symbolp (car lst)) (cons (car lst) (f (cdr lst))))
35          ((numberp (car lst)) (cons (* (car lst) num) (f (cdr lst))))
36          ((atom (car lst)) (cons (car lst) (f (cdr lst) )))
37          ((atom (cdr lst)) (cons (dop-fun (car lst)) (f (cdr lst) )))
38          (T (cons (f (car lst)) (f (cdr lst))))) )
39
40
41
42  ; "хвостовая" рекурсия
43  (defun f (lst res)
44      (cond ((null lst) (reverse res))
45          ((symbolp (car lst)) (f (cdr lst) (cons (car lst) res)))
46          ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) num) res)))
47          ((atom (car lst)) (f (cdr lst) (cons (car lst) res)))
48          ((atom (cdr lst)) (f (cdr lst) (cons (dop-fun (car lst)) res)))
49          (T (f (cdr lst) (cons (f (csr lst) ()) res))) ) )
```

5. 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  ; Рекурсивно. Для смешанного списка.
2  (defun select-rec-one-lvl (lst a b res)
3      (cond ((null lst) res)
4          ((and (numberp (car lst))
5              (<= (car lst) b)
6              (>= (car lst) a))
7              (select-rec-one-lvl (cdr lst) a b (cons (car lst) res)))
8          (t (select-rec-one-lvl (cdr lst) a b res))))
9
10 ; С использованием функционала. Для смешанного списка.
11 (defun select-fun-one-lvl (lst a b)
12     (remove-if-not #'(lambda (el) (and (numberp el) (<= el b) (>= el a)))
           lst))
13
14 ; обёрточная функция для каждой из предоставленной выше функции
15 (defun select-between (lst fNum sNum)
16     (let ((a (cond ((< fNum sNum) fNum) (t sNum)))
17         (b (cond ((< fNum sNum) sNum) (t fNum))))
18             (select-rec lst a b ())))
```

6. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка:
а) одноуровнего смешанного,
б) структурированного.

```
1  ; без работы со структурированными смешанными списками
2  (defun rec-add-inner (lst acc)
3      (cond ((null (cdr lst)) (+ acc (car lst)))
4          (t (rec-add-inner (cdr lst) (+ acc (car lst))))))
5
6  (defun rec-add (lst)
7      (rec-add-inner lst 0))
8
9  ; С использованием дополняемой рекурсии
10 (defun rec-add (lst)
11     (cond ((null (cdr lst)) (car lst))
12         (t (+ (car lst) (rec-add (cdr lst))))))
13
14 ; с обработкой смешанных структурированных списков
15 (defun rec-add-inner (lst acc)
16     (cond ((null lst) acc)
17         ((listp (car lst)) (rec-add-inner (cdr lst)
18                     (rec-add-inner (car lst) acc )))
19         ((numberp (car lst)) (rec-add-inner (cdr lst) (+ acc (car lst))))
20         (t (rec-add-inner (cdr lst) acc))))
21
22 (defun rec-add (lst)
23     (rec-add-inner lst 0))
24
25 ; С использованием дополняемой рекурсии
26 (defun rec-add (lst)
27     (cond ((null lst) 0)
28         ((symbolp (car lst)) (rec-add (cdr lst)))
29         ((listp (car lst)) (+ (rec-add (car lst)) (rec-add (cdr lst))))
30         ((numberp (car lst)) (+ (car lst) (rec-add (cdr lst))))))
```

7. Написать рекурсивную версию с именем recnth функции nth

```
(defun rec-nth (index lst)
    (cond ((or (< n 0) (null lst)) nil)
        ((zerop index) (car lst))
        (t (rec-nth (- index 1) (cdr lst)))))
```

8. Написать рекурсивную функцию allodd, которая возвращает t когда все элементы списка нечетные.

```
; без работы с структурированными смешанными списками
(defun allodr-rec (lst cur-bool)
    (cond ((null cur-bool) nil)
        ((null lst))
        (t (allodr-rec (cdr lst) (oddp (car lst))))))

(defun allodr (lst)
    (cond ((null lst) Nil)
    (T (allodr-rec lst t))))

; для работы с структурированными смешанными списками
(defun allodr-rec (lst cur-bool)
    (cond ((null cur-bool) nil)
        ((null lst))
        ((listp (car lst)) (and (allodr-rec (car lst) t)
                    (allodr-rec (cdr lst) cur-bool)))
        ((numberp (car lst)) (allodr-rec (cdr lst) (oddp (car lst))))
        (t (allodr-rec (cdr lst) cur-bool))))

(defun allodr (lst)
    (cond ((null lst) Nil)
        (T (allodr-rec lst t))))
```

9. Написать рекурсивную функцию, которая возвращает первое нечетное число из списка (структурированного), возможно создавая некоторые вспомогательные функции.

```
1 (defun is-odd(num)
2     (cond ((eql num 0) Nil)
3           ((eql num 1) t)
4           ((>= num 2) (is-odd (- num 2)))))
5
6 (defun my-odd-rec (lst)
7     (cond ((null lst) Nil)
8           ((oddp (car lst)) (car lst))
9           (T (my-odd-rec (cdr lst))) ))
```

10. Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
1  ; одноуровневый список -- только числа
2  (defun get-sqr-list (lst)
3      (cond ((null lst) nil)
4          (t (cons (* (car lst) (car lst)) (get-sqr-list (cdr lst))))))
5
6  ; одноуровневый список
7  (defun get-sqr-list (lst)
8      (cond ((null lst) nil)
9          ((symbolp (car lst)) (cons (car lst) (get-sqr-list (cdr lst))))
10         ((numberp (car lst)) (cons (* (car lst) (car lst)) (get-sqr-list (cdr
               lst))))
11         (t (get-sqr-list (cdr lst)))))
12
13 ; рекурсия без накопления cons, но с reverse
14 (defun get-sqr-list (lst res)
15     (cond ((null lst) (reverse res))
16         ((symbolp (car lst)) (get-sqr-list (cdr lst) (cons (car lst) res)))
17         ((numberp (car lst)) (get-sqr-list (cdr lst) (cons (* (car lst) (car
               lst)) res)))))
18
19 ; Рекурсивно для смешанного структурированного списка
20 (defun get-sqr-list (lst)
21     (cond ((null lst) nil)
22         ((symbolp (car lst)) (cons (car lst) (get-sqr-list (cdr lst))))
23         ((listp (car lst)) (cons (get-sqr-list (car lst)) (get-sqr-list (cdr
               lst))))
24         ((numberp (car lst)) (cons (* (car lst) (car lst)) (get-sqr-list (cdr
               lst))))
25         (t (get-sqr-list (cdr lst)))))
```