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.)

```
; рекурсивно для од. 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))))

; Рекурсивно. Для смешанного структурированного списка.
(defun select-rec (lst a b res)
    (cond ((null lst) res)
        ((listp (car lst)) (cons (select-rec (car lst) a b res)
        (select-rec (cdr lst) a b res)))
        ((and
            (numberp (car lst))
            (<= (car lst) b)
            (>= (car lst) a))
                (select-rec (cdr lst) a b (cons (car lst) res)))
        (t (select-rec (cdr lst) a b res))))
```

```
21
22 ; С использованием функционала. Для смешанного списка.
23 (defun select-fun-one-lvl (lst a b)
24     (remove-if-not #'(lambda (el) (and (numberp el) (<= el b) (>= el a)))
           lst))
25
26 ; С использованием функционала. Для смешанного структурированного списка.
27 (defun select-fun (lst a b)
28     (mapcan #'(lambda (el)
29         (cond ((listp el) (select-fun el a b))
30             ((and (numberp el) (<= el b) (>= el a) (cons el nil))))) lst))
31
32 ; оберточная функция для каждой из предоставленной выше функции
33 (defun select-between (lst)
34     (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
32
33
34
35
```

```
; рекурсия
(defun f (lst)
    (cond ((null lst) ())
        ((symbolp (car lst)) (cons (car lst) (f (cdr lst))))
        ((numberp (car lst)) (cons (* (car lst) num) (f (cdr lst))))
        ((atom (car lst)) (cons (car lst) (f (cdr lst) )))
        ((atom (cdr lst)) (cons (dop-fun (car lst)) (f (cdr lst) )))
        (T (cons (f (car lst)) (f (cdr lst))))) )



; "хвостовая" рекурсия
(defun f (lst res)
    (cond ((null lst) (reverse res))
        ((symbolp (car lst)) (f (cdr lst) (cons (car lst) res)))
        ((numberp (car lst)) (f (cdr lst) (cons (* (car lst) num) res)))
        ((atom (car lst)) (f (cdr lst) (cons (car lst) res)))
        ((atom (cdr lst)) (f (cdr lst) (cons (dop-fun (car lst)) res)))
        (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)).

```
; 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))))

; С использованием функционала. Для смешанного списка.
(defun select-fun-one-lvl (lst a b)
    (remove-if-not #'(lambda (el) (and (numberp el) (<= el b) (>= el a)))
        lst))

; обёрточная функция для каждой из предоставленной выше функции
(defun select-between (lst fNum sNum)
    (let ((a (cond ((< fNum sNum) fNum) (t sNum)))
        (b (cond ((< fNum sNum) sNum) (t fNum))))
            (select-rec lst a b ())))
```

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

```
; без работы со структурированными смешанными списками
(defun rec-add-inner (lst acc)
    (cond ((null (cdr lst)) (+ acc (car lst)))
        (t (rec-add-inner (cdr lst) (+ acc (car lst))))))

(defun rec-add (lst)
    (rec-add-inner lst 0))

; С использованием дополняемой рекурсии
(defun rec-add (lst)
    (cond ((null (cdr lst)) (car lst))
        (t (+ (car lst) (rec-add (cdr lst))))))

; с обработкой смешанных структурированных списков
(defun rec-add-inner (lst acc)
    (cond
        ((numberp lst) (+ acc lst))
        ((or (null lst) (symbolp lst))acc)
        (t (rec-add-inner (cdr lst) (rec-add-inner (car lst) acc )))))

(defun rec-add (lst)
    (rec-add-inner lst 0))

; С использованием дополняемой рекурсии
(defun rec-add (lst)
    (cond ((null lst) 0)
        ((symbolp (car lst)) (rec-add (cdr lst)))
        ((listp (car lst)) (+ (rec-add (car lst)) (rec-add (cdr lst))))
        ((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)
```

```
10      (cond ((null lst) Nil)
11      (T (allodr-rec lst t))))
12
13  ; для работы с структурированными смешанными списками
14  (defun allodr-rec (lst cur-bool)
15      (cond ((null cur-bool) nil)
16          ((null lst))
17          ((listp (car lst)) (and (allodr-rec (car lst) t)
18                      (allodr-rec (cdr lst) cur-bool)))
19          ((numberp (car lst)) (allodr-rec (cdr lst) (oddp (car lst))))
20          (t (allodr-rec (cdr lst) cur-bool))))
21
22  (defun allodr (lst)
23      (cond ((null lst) Nil)
24          (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
14
15
16
17
18 ; рекурсия без накопления cons, но с reverse
```

```
19 (defun get-sqr-list (lst res)
20     (cond ((null lst) (reverse res))
21         ((symbolp (car lst)) (get-sqr-list (cdr lst) (cons (car lst) res)))
22         ((numberp (car lst)) (get-sqr-list (cdr lst) (cons (* (car lst) (car
               lst)) res)))))
23
24 ; Рекурсивно для смешанного структурированного списка
25 (defun get-sqr-list (lst)
26     (cond ((null lst) nil)
27         ((symbolp (car lst)) (cons (car lst) (get-sqr-list (cdr lst))))
28         ((listp (car lst)) (cons (get-sqr-list (car lst)) (get-sqr-list (cdr
               lst))))
29         ((numberp (car lst)) (cons (* (car lst) (car lst)) (get-sqr-list (cdr
               lst))))
30         (t (get-sqr-list (cdr lst)))))
```