**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
31             ((listp el) (select-fun el a b))
32             ((and (numberp el) (<= el b) (>= el a) (cons el nil))))) lst))
33
34 ; обёрточная функция для каждой из предоставленной выше функции
35 (defun select-between (lst)
36     (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  ; С использованием функционала для одномерного смешанного списка
2  (defun mult-els (lst num)
3      (mapcar #'(lambda (arg)
4          (cond
5              ((numberp arg) (* arg num))
6              (t arg))) lst))
7
8  ; С использованием функционала для структурированного смешанного списка
9  (defun mult-els-deep (lst num)
10     (mapcar #'(lambda (arg)
11         (cond
12             ((listp arg) (mult-els-deep arg num))
13             ((numberp arg) (* arg num))
14             (t arg))) lst))
15
16 ; Рекурсивно для одномерного смешанного списка
17 (defun mult-els-rec (lst num res)
18     (cond
19         ((null lst) (reverse res)
20         ((numberp (car lst)) (mult-els-rec (cdr lst) num (cons (* (car lst)
               num) res )))
21         (t (mult-els-rec (cdr lst) num (cons (car lst) res)))))
22
23 (defun mult-els (lst num)
24     (mult-els-rec (lst num ())))
25
26 ; Рекурсивно для структурированного смешанного  списка
27 (defun mult-els-rec-deep (lst num)
28     (cond
29         ((null lst) nil)
30         ((listp (car lst)) (cons (mult-els-rec-deep (car lst) num)
               (mult-els-rec-deep (cdr lst) num)))
31         ((numberp (car lst)) (cons (* (car lst) num) (mult-els-rec-deep (cdr
               lst) num)))
32         (t (cons (car lst) (mult-els-rec-deep (cdr lst) num)))))
```

**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 ; Рекурсивно. Для смешанного структурированного списка.
11 (defun select-rec (lst a b res)
12     (cond ((null lst) res)
13         ((listp (car lst)) (cons (select-rec (car lst) a b res)
14                                  (select-rec (cdr lst) a b res)))
15         ((and (numberp (car lst))
16             (<= (car lst) b)
17             (>= (car lst) a))
18                 (select-rec (cdr lst) a b (cons (car lst) res)))
19         (t (select-rec (cdr lst) a b res))))
20
21 ; С использованием функционала. Для смешанного списка.
22 (defun select-fun-one-lvl (lst a b)
23     (remove-if-not #'(lambda (el) (and (numberp el) (<= el b) (>= el a)))
        lst))
24
25 ; С использованием функционала. Для смешанного структурированного списка.
26 (defun select-fun (lst a b)
27     (mapcan #'(lambda (el)
28     (cond ((listp el) (select-fun el a b))
29         ((and (numberp el) (<= el b) (>= el a) (cons el nil)))))) lst))
30
31 ; обёрточная функция для каждой из предоставленной выше функции
32 (defun select-between (lst fNum sNum)
33     (let ((a (cond ((< fNum sNum) fNum) (t sNum)))
34         (b (cond ((< fNum sNum) sNum) (t fNum))))
35             (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
4          ((null (cdr lst)) (+ acc (car lst)))
5          (t (rec-add-inner (cdr lst) (+ acc (car lst))))))
6
7  (defun rec-add (lst)
8      (rec-add-inner lst 0))
9
10 ; С использованием дополняемой рекурсии
11 (defun rec-add (lst)
12     (cond
13         ((null (cdr lst)) (car lst))
14         (t (+ (car lst) (rec-add (cdr lst))))))
15
16 ; с обработкой смешанных структурированных списков
17 (defun rec-add-inner (lst acc)
18     (cond
19     ((null lst) acc)
20     ((listp (car lst)) (rec-add-inner (cdr lst) (rec-add-inner (car lst) acc
           )))
21     ((numberp (car lst)) (rec-add-inner (cdr lst) (+ acc (car lst))))
22     (t (rec-add-inner (cdr lst) acc))))
23
24 (defun rec-add (lst)
25     (rec-add-inner lst 0))
26
27 ; С использованием дополняемой рекурсии
28 (defun rec-add (lst)
29     (cond
30     ((null lst) 0)
31     ((symbolp (car lst)) (rec-add (cdr lst)))
32     ((listp (car lst)) (+ (rec-add (car lst)) (rec-add (cdr lst))))
33     ((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)))))
```