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"

Группа ИУ7-63Б (ИУ7и-67б)

Студент Тэмуужин Я.

Преподаватель Толпинская Н.Б.

Преподаватель Строганов Ю. В.

Москва — 2022 г.

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

```
(defun my-reverse-ins (lst tmp)
  (cond ((null (cdr lst)) (cons (car lst) tmp))
        (T (my-reverse-ins (cdr lst) (cons (car lst) tmp)))))

(defun my-reverse (lst)
  (my-reverse-ins lst ()))
```

3. Написать функцию, которая возвращает первый элемент списка-аргумента, который сам является непустым списком. (1 2 (3 4) 6 (7 8 9)) -> (3 4)

```
(defun first-lst-el (lst)
  (cond ((listp (car lst)) (car lst))
        ((null (cdr lst)) nil)
        (T (first-lst-el (cdr lst)))
   ))
```

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

8. 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)). (3 2 9 10 4 8 6) 2 8 -> (3 4 6 8)

```
;; from lecture -- not a tail recursion
(defun add-to-sorted (x lst)
  (cond ((null lst) (cons x lst))
        ((<= x (car lst)) (cons x lst))
        ((null (cdr lst)) (cons x nil))
        (T (cons (car lst) (add-to-sorted x (cdr lst)))) ))

(defun is-between (x b1 b2)
  (or (and (>= x b1) (<= x b2))
      (and (<= x b1) (>= x b2))))

;; res = '(nil)
(defun select-between-ins (lst b1 b2 res)
  (cond ((null lst) res)
        ((is-between (car lst) b1 b2)
         (cons (car lst)
               (select-between-ins (cdr lst) b1 b2 res)))
        (T (select-between-ins (cdr lst) b1 b2 res))))
```

7. 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.

(a 3 (1 j 2) db 4 5 sdf)

(a 30 (10 j 20) db 40 50 sdf)

```
(defun mul-lst (lst x)
  (cond ((null lst) nil)
        ((listp (car lst))
         (mul-lst (car lst) x)
         (mul-lst (cdr lst) x))
        ((numberp (car lst))
         (setf (car lst) (* (car lst) 10))
         (mul-lst (cdr lst) x))
        (T (mul-lst (cdr lst) x)) )))
```

8. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка:

а) одноуровнего смешанного, (a 3 db 4 5 sdf) -> 12

б) структурированного. ((7 3) 1 2 (3 4)) -> 20

```
(defun rec-add-a (lst res)
  (cond ((null lst) res)
        ((symbolp (car lst)) (rec-add-a (cdr lst) res))
        (T (rec-add-a (cdr lst) (+ res (car lst)))) ))

(defun rec-add (lst res)
  (cond ((null lst) res)
        ((listp (car lst))
         (+ (rec-add (car lst) 0)
            (rec-add (cdr lst) res)))
        ((symbolp (car lst)) (rec-add (cdr lst) res))
        (T (rec-add (cdr lst) (+ res (car lst)))) ))
```

9. Написать рекурсивную версию с именем recnth функции nth.

```
(defun recnth (n lst)
  (cond ((null lst) Nil)
        ((eq 0 n) (car lst))
        (T  (recnth (- n 1) (cdr lst)))))
```

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

```
(defun allodd (lst)
  (cond ((null lst) T)
        ((evenp (car lst)) 'not_all_odd)
        (T (allodd (cdr lst)))  ))
```

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

'((9 8) 3 7 4 (3 6))) -> 8

'((9 3) 3 7 4 (3 6))) -> 4

'((9 3) 3 7 (3 6))) -> 6

```
(defun first-even-el (lst)
  (cond ((null lst) nil)
        ((listp (car lst))
         (or (first-even-el (car lst))
             (first-even-el (cdr lst))))
        ((evenp (car lst)) (car lst))
        (T (first-even-el (cdr lst))) )))
```

12. 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 2 3) -> (1 4 9)

```
(defun square (lst)
  (cond ((null (cdr lst)) (cons (* (car lst) (car lst)) nil))
        (T (cons (* (car lst) (car lst))(square (cdr lst))))))
```