1. Практические задания

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

```
(defun my-reverse (lst &optional (buf-lst Nil))
        (cond ((null lst) buf-lst)
                (t (my-reverse (cdr lst) (cons (car lst) buf-lst)))))
```

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

```
(defun not-null-lst (lst)
     (cond ((null lst) Nil)
           ((and (listp (car lst)) (not (null (caar lst)))) (car lst))
           (t (not-null-lst (cdr lst)))))
```

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

```
(defun append-elem (lst elem &optional (before-lst Nil))
        (cond ((and (null lst) (null before-lst)) (cons elem Nil))
                ((null lst) (my-reverse (cons elem before-lst)))
                (t (append-elem (cdr lst) elem (cons (car lst) before-lst)))))

(defun append-lst (lst1 lst2 &optional (before-lst Nil))
        (cond ((and (null lst1) (null lst2)) before-lst)
                ((not (null lst1)) (append-lst (cdr lst1) lst2 (append-elem
                    before-lst (car lst1))))
                (t (append-lst lst1 (cdr lst2) (append-elem before-lst (car
                    lst2))))))

(defun get-between (num1 num2 lst &optional (res-lst Nil))
        (cond ((null lst) res-lst)
                ((and (numberp (car lst)) (< num1 (car lst) num2)) (
                    get-between num1 num2 (cdr lst)
                (append-elem res-lst (car lst))))
        (t (get-between num1 num2 (cdr lst) res-lst))))
```

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.*

```
(defun all-num-mult-a (num lst &optional (res-lst Nil))
    (cond ((null lst) res-lst)
          (t (all-num-mult-a num (cdr lst) (append-elem res-lst (* num (car
              lst)))))))
(print (all-num-mult-a 2 '(1 5.9 6 -12)))
(defun all-num-mult-b (num lst &optional (res-lst Nil))
    (cond ((null lst) res-lst)
          ((numberp (car lst)) (all-num-mult-b num (cdr lst) (append-elem
              res-lst (* num (car lst)))))
          ((listp (car lst)) (all-num-mult-b num (cdr lst)
            (append-elem res-lst (all-num-mult-b num (car lst)))))
          (t (all-num-mult-b num (cdr lst) (append-elem res-lst (car lst))))))
```

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)).*

```
(defun get-between (num1 num2 lst &optional (res-lst Nil))
    (cond ((null lst) res-lst)
          ((< num1 (car lst) num2) (get-between num1 num2 (cdr lst)
            (append-elem res-lst (car lst))))
          (t (get-between num1 num2 (cdr lst) res-lst))))
(defun insert-elem (lst elem &optional (before-lst Nil))
    (cond ((and (null lst) (not (null before-lst))) (append-elem before-lst
        elem))
          ((null lst) (cons elem Nil))
          ((< elem (car lst)) (append-lst (append-elem before-lst elem) lst))
          (t (insert-elem (cdr lst) elem (append-elem before-lst (car lst)))))
             )
(defun sort-elem (lst &optional (res-lst Nil))
    (cond ((null lst) res-lst)
          (t (sort-elem (cdr lst) (insert-elem res-lst (car lst))))))
(defun select-between (num1 num2 lst)
    (cond ((null lst) Nil)
          ((> num1 num2) (sort-elem (get-between num2 num1 lst)))
          (t (sort-elem (get-between num1 num2 lst)))))
```

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

```
(defun rec-add-a (lst &optional (sum 0))
    (cond ((null lst) sum)
          ((numberp (car lst)) (rec-add-a (cdr lst) (+ sum (car lst))))
```

```
4           (t (rec-add-a (cdr lst) sum))))
5 (print (rec-add-a '(1 v 4 0)))
6 (defun rec-add-b (lst &optional (sum 0))
7     (cond ((null lst) sum)
8           ((numberp (car lst)) (rec-add-b (cdr lst) (+ sum (car lst))))
9           ((listp (car lst)) (rec-add-b (cdr lst) (+ sum (rec-add-b (car lst))
                ) ))
10          (t (rec-add-b (cdr lst) sum))))
11 (print (rec-add-b '(1 (b 19 (0 a)) (1.24) l)))
```

7. *Написать рекурсивную версию с именем recnth функции nth.*

```
1 (defun recnth (ind lst &optional (cnt 0))
2     (cond ((null lst) Nil)
3           ((= cnt ind) (car lst))
4           (t (recnth ind (cdr lst) (+ cnt 1)))))
```

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

```
1 (defun all-odd (lst)
2     (cond ((null lst) t)
3           ((and (numberp (car lst)) (evenp (car lst))) Nil)
4           ((listp (car lst)) (all-odd (car lst)))
5           (t (all-odd (cdr lst)))))
```

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

```
1 (defun first-num (lst)
2     (cond ((numberp (car lst)) (car lst))
3           (t (first-num (cdr lst)))))
4 (defun first-odd-num (lst)
5     (cond ((null lst) Nil)
6           ((oddp (first-num lst)) (first-num lst))
7           (t (first-odd-num (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 (defun all-square (lst)
2     (cond ((null lst) Nil)
3           (t (cons (* (car lst) (car lst)) (all-square (cdr lst))))))
```