Задание 1

Условие: Написать хвостовую рекурсивную функцию my-reverse, которая развернёт верхний уровень своего списка-аргумента lst.

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

(defun reverse-helper (lst res)
    (if (null lst)
        res
        (reverse-helper (cdr lst) (cons (car lst) res))))
```

Задание 2

Условие: Написать функцию, которая возвращает первый элемент списка-аргумента, который сам является не пустым списком.

```
(defun true-listp (lst)
    (or
        (null lst)
        (and (consp lst) (true-listp (cdr lst)))))

(defun get-non-list-element (lst)
    (if (not (null lst))
        (if (and (true-listp (car lst)) (not (null (car lst))))
            (car lst)
            (get-non-list-element (cdr lst)))))
```

Задание 3

Условие: Напишите рекурсивную функцию которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

1) все элементы списка – числа;
2) элементы списка – любые объекты.

```
(defun mult-list-numbers (lst multiplier)
    (if (not (null lst))
        (cons
            (* (car lst) multiplier)
            (mult-list-numbers (cdr lst) multiplier))))
```

```
(defun mult-list-any (lst multiplier)
(if (not (null lst))
        (cons
            (if (numberp (car lst))
                (* (car lst) multiplier)
                (car lst))
            (mult-list-any (cdr lst) multiplier))))
```

Задание 4

Условие: 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).

```
(defun betweenp (elem lborder rborder)
    (and (<= elem rborder) (>= elem lborder)))

(defun filter-by-range (lst lborder rborder)
    (if (not (null lst))
        (if (betweenp (car lst) lborder rborder)
            (cons
                (car lst)
                (filter-by-range (cdr lst) lborder rborder))
            (filter-by-range (cdr lst) lborder rborder)
        )))

(defun sort-range (lst border1 border2)
    (sort (if (< border1 border2)
        (filter-by-range lst border1 border2)
        (filter-by-range lst border2 border1)
    ) #'<))
```

Задание 5

Условие: Написать рекурсивную версию (с именем rec-add) вычисления суммы заданного списка

1) одноуровнего смешанного;
2) структурированного.

```
(defun rec-add-1level (lst)
    (cond ((null lst) 0)
```

```
        (
          (numberp (car lst))
          (+ (car lst) (rec-add-1level (cdr lst))))
        (t (rec-add-1level (cdr lst)))))

(defun rec-add-multilevel (lst)
    (cond ((null lst) 0)
        (
          (numberp (car lst))
          (+ (car lst) (rec-add-multilevel (cdr lst))))
        (
          (listp (car lst))
          (+
              (rec-add-multilevel (car lst))
              (rec-add-multilevel (cdr lst))))
        (t (rec-add-multilevel (cdr lst)))))
```

Задание 6

Условие: Написать рекурсивную версию с именем recnth функции nth.

```
(defun recnth (n lst)
    (if (>= n 0)
        (recnth-helper n lst)))

(defun recnth-helper (n lst)
    (if (> n 0)
        (recnth (- n 1) (cdr lst))
        (car lst)))
```

Задание 7

Условие: Написать рекурсивную функцию allodd, которая возвращает t, когда все элементы списка нечётные.

```
(defun allodd (lst)
    (or
        (null lst)
        (and (oddp (car lst)) (allodd(cdr lst)))))
```

Задание 8

Условие: Написать рекурсивную функция, которая возвращает первое нечётное число из структурированного списка, возможно создавая некоторые вспомогательные функции.

```
(defun odd-numberp (num)
    (if (and (numberp num) (oddp num))
        num))

(defun first-odd (lst)
    (if (not (null lst))
        (or
            (odd-numberp (car lst))
            (and (listp (car lst)) (first-odd (car lst)))
            (first-odd (cdr lst)))))
```

Задание 9

Условие: 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.

```
(defun recsquare (lst)
    (if (not (null lst))
        (cons (* (car lst) (car lst)) (recsquare (cdr lst)))))
```

Задание 10

Условие: Преобразовать структурированный список в одноуровневый.

```
(defun into-1level (lst)
    (into-1level-helper lst ())
)

(defun into-1level-helper (lst res)
    (if (not (null lst))
        (if (listp (car lst))
            (into-1level-helper
                (car lst)
                (into-1level-helper (cdr lst) res))
            (cons (car lst) (into-1level-helper (cdr lst) res)))
        res))
```