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»

Лабораторная работа №6

По предмету: «Функциональное и логическое программирование»

Студент: Гасанзаде М.А.,
Группа: ИУ7-66Б

Москва, 2020 г.

1. **Что будет результатом (mapcar 'вектор '(570-40-8))**

```
Ошибка. Функции "вектор" не существует.
```

2. **Напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции.**

```
(defun all_minus_10 (lst)
    (mapcar #'(lambda (x)
                (cond   ((numberp x) (- x 10))
                    ((listp x) (all_minus_10 x))
                    (t      x)
                )
            )
            lst
    )
)
```

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

```
(defun get_first_element (lst)
    (if (and    (listp (car lst))
            (not (null (car lst)))
        )
        (car lst)
        (get_first_element (cdr lst))
    )
)
```

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

```
(defun select_between_inner (lst left right result)
    (mapcar #'(lambda (x)
                (cond   ((listp x) (select_between_inner x left right result))
((and       (numberp x) (> x left) (< x right))
                        (nconc result (cons x nil))
                    )
                )
            )
            lst
    )
    (cdr result)
)
(defun select_between (lst left right);
    (select_between_inner lst left right (cons nil nil))
)
```

5. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. A x B - это множество всевозможных пар (a b), где a принадлежит A, b принадлежит B.

```
(defun decart (a b)
    (mapcan #'(lambda (x) (mapcar   #'(lambda (y) (list x y))
                                    b
                          )
            )
            a
    )
)
```

6. Почему так реализовано `reduce`, в чем причина?

```
(reduce #'+ ()) ->  0

(reduce #'* ()) -> 1
```

Сначала функция проверяет список-аргумент. Если он пуст, возвращается значение функции при отсутствии аргументов.
Также reduce использует аргумент **:initial-value.** Этот аргумент определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Если список-аргумент пуст, то будет возвращено значение initial-value.

7. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е. например для аргумента ((1 2) (3 4)) -> 4.

```
(defun sum_lengths (list-of-lists);
    (reduce #'+
            (mapcar (lambda (x)
                        (if (listp x) (sum_lengths x) 1)
                    )
                    list-of-lists
            )
    )
)
```

8. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка. Например: (rec-add (2 4 6)) -> 12

```
(defun rec_add_inner (lst sum)
    (let (  (head (car lst))
            (tail (cdr lst))
        )
        (cond   ((null lst) sum)
            ((listp head) (rec_add_inner tail (rec_add_inner head sum)) )
            ((numberp head) (rec_add_inner  tail (+ sum head)) )
            (t (rec_add_inner tail sum))
        )
    ))
(defun rec_add (lst)
    (if (eq lst nil)
        nil
        (rec_add_inner lst 0)
    )
)
```

9. Написать рекурсивную версию с именем rec-nth функции nth.

```
(defun rec_nth_inner (elem curr target)
    (cond   ((= curr target) (car elem))
        ((eq elem nil) nil)
        (t (rec_nth_inner (cdr elem) (+ curr 1) target))
    )
)
```

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

```
(defun alloddr (lst);
    (let    ((head (car lst))
         (tail (cdr lst))
        )
        (cond   ((null lst) t)
            ((listp head)
                (and (alloddr head) (alloddr tail))
            )
            ((not (numberp head)) nil)
            ((evenp head) nil)
            (t (alloddr tail))
        )
    )
)
```

11. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента.

```
(defun mylast (curr)
    (if (eq (cdr curr) nil)
        (car curr)
        (mylast (cdr curr))
    )
)
(defun mylast (curr)
    (if (eq (cdr curr) nil)
        (car curr)
        (mylast (cdr curr))
    )
)
```

12. Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-аргумента функции.

Варианты: 1) от n-аргумента функции до последнего >= 0,
2) от n-аргумента функции до m-аргумента с шагом d.

```
(defun get_n_sum (curr n)
    (if (or (eq curr nil) (= n 0))
        0
        (+ (car curr) (get_n_sum (cdr curr) (- n 1)))
    )
)
```

13. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

```
(defun get_last_odd_inner (curr value)
    (cond   ((eq curr nil) value)
        ((oddp (car curr)) (get_last_odd_inner (cdr curr) (car curr)))
        (t (get_last_odd_inner (cdr curr) value))
    )
)
(defun get_last_odd (lst)
    (get_last_odd_inner lst nil)
)
```

14. 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 square_all (lst);
    (mapcar #'(lambda (x)
                (cond   ((numberp x) (* x x))
                    ((listp x) (square_all x))
                    (t      x)
                )
            )
            lst
    )
)
```

15. Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа.

Вариант: select-even;

Вариант: вычисляет сумму только всех нечетных чисел (sum-all-odd) или сумму всех четных чисел (sum-all-odd) из заданного списка.)

```
(defun select_odd_inner (lst result)
    (mapcar #'(lambda (x)
                (cond   ((listp x) (select_odd_inner x result))
                    ((and   (numberp x) (oddp x))
                        (nconc result (cons x nil))
                    )
                )
            )
            lst
    )
    (cdr result)
)
(defun select_odd (lst);
    (select_odd_inner lst (cons nil nil))
)
```