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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Название: Рекурсивные функции

Дисциплина: Функциональное и логическое программирование

Студент	ИУ7-66Б		Т. А. Казаева
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Н. Б. Толпинская
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

1. Практические задания

Используя рекурсию:

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

```
(defun my-reverse(lst &optional (acc ()))
  (cond ((null lst) acc)
        ((consp lst) (my-reverse (cdr lst) (cons (car lst) acc)))))
```

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

```
(defun get-fst-list(lst)
  (cond ((null lst) nil)
        ((listp (car lst)) (car lst))
        (t (get-fst-list (cdr lst)))))
```

3. *Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.*

```
(defun find-between(lst l r &optional (dump ()))
  (cond ((null lst) dump)
        (t (cond ((and (> (car lst) l) (< (car lst) r))
                  (find-between (cdr lst) l r (cons (car lst) dump)))
                 (t (find-between (cdr lst) l r dump))))))
```

4. *Напишите рекурсивную функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда*

   a) *все элементы списка – числа,*

   b) *элементы списка – любые объекты.*

```lisp
;; ========== (a) ==========
(defun multiply-numbers(lst m)
  (cond ((null lst) nil)
        (t (cons (* (car lst) m) (multiply-numbers (cdr lst) m)))))
;; ========== (b) ==========
(defun multiply-objcts(lst m)
  (cond ((null lst) nil)
        ((numberp (car lst))
         (cons (* (car lst) m) (multiply-objcts (cdr lst) m)))
        ((listp (car lst))
         (cons (multiply-objcts (car lst) m) (multiply-objcts (cdr lst) m)))
        (t (cons (car lst) (multiply-objcts (cdr lst) m)))))
```

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)*

```lisp
(defun select-between(lst l r &optional (dump ()))
  (sort (cond ((null lst) dump)
              (t (cond ((and (> (car lst) l) (< (car lst) r))
                        (select-between (cdr lst) l r (cons (car lst) dump)))
                       (t (select-between (cdr lst) l r dump))))) #'<))
```

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

   a) *одноуровнего смешанного,*

   b) *структурированного.*

```lisp
;; ========== (a) ==========
(defun rec-add-onelevel (lst &optional (acc 0))
  (cond ((null lst) acc)
        ((numberp (car lst)) (rec-add-onelevel (cdr lst) (+ acc (car lst))))
        (t (rec-add-onelevel (cdr lst) acc))))

;; ========== (b) ==========
(defun rec-add-structured (lst &optional (acc 0))
  (cond ((null lst) acc)
        ((numberp (car lst)) (rec-add-structured (cdr lst) (+ acc (car lst))))
        ((listp (car lst))
                (rec-add-structured (cdr lst)
                                    (rec-add-structured (car lst) acc)))
        (t (rec-add-structured (cdr lst) acc))))
```

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

```
(defun recnth(lst n &optional (acc 0))
  (cond ((< n 0) nil)
        ((null lst) nil)
        ((= acc n) (car lst))
        (t (recnth (cdr lst) n (+ 1 acc)))))
```

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

```
(defun allodd(lst)
  (cond ((null lst) t)
        ((oddp (car lst)) (allodd (cdr lst)))
        (t nil)))
```

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

```
(defun get-fst-odd(lst)
  (cond ((null lst) nil)
        ((listp (car lst)) (let ((fnd (get-fst-odd (car lst))))
                             (if (not fnd)
                                 (get-fst-odd (cdr lst))
                                 fnd)))
        ((oddp (car lst)) (car lst))
        (t (get-fst-odd (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.*

```
(defun get-squares (lst)
  (cond ((null lst) nil)
        (t (cons (* (car lst) (car lst)) (get-squares (cdr lst))))))
```