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 №7 по курсу "Функциональное и логическое программирование"

Тема Рекурсивные функции

Студент Ковалец К. Э.

Группа ИУ7-63Б

Преподаватели Толпинская Н. Б., Строганов Ю. В.

Москва — 2022 г.

1 Практические задания

Используя рекурсию:

1.1 Задание 1

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

Листинг 1.1 – Решение задания 1

```
(defun my-reverse-rec (rev-lst lst)
    (cond
        ((null lst)
            rev-lst)
        (T
            (my-reverse-rec (cons (car lst) rev-lst) (cdr lst)))))

(defun my-reverse (lst)
    (my-reverse-rec NIL lst))

;; (MY-REVERSE '(1 2 3 4 5)) -> (5 4 3 2 1)
```

1.2 Задание 2

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

Листинг 1.2 – Решение задания 2

```
(defun get-first-elem (lst)
    (cond
        ((null lst)
            NIL)
        ((and (listp (car lst)) (not (null (car lst))))
            (caar lst))
        (T
            (get-first-elem (cdr lst)))))

;; (GET-FIRST-ELEM '(1 () (2 3) (4 5 6))) -> 2
```

1.3 Задание 3

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

Листинг 1.3 – Решение задания 3

```
(defun find-right-numbers (res_lst lst b1 b2)
    (let* ((cur_elem (car lst)))
        (cond
            ((null lst)
                res_lst)
            ((and (numberp cur_elem)
                    (> cur_elem b1)
                    (< cur_elem b2))
                (find-right-numbers
                    (cons cur_elem res_lst)
                    (cdr lst) b1 b2))
            (T
                (find-right-numbers res_lst (cdr lst) b1 b2)))))

(defun select-between (lst)
    (my-reverse (find-right-numbers NIL lst 1 10)))

;; (SELECT-BETWEEN '(1 5 4 12 10 2 3)) -> (5 4 2 3)
;; (SELECT-BETWEEN '(0 1 10 11)) -> NIL
```

1.4 Задание 4

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

- все элементы списка – числа;
- элементы списка – любые объекты.

Листинг 1.4 – Решение задания 4 (a)

```
;; a) все элементы списка -- числа
(defun multiply-all-numbers-rec (res_lst lst x)
    (cond
        ((null lst)
            (my-reverse res_lst))
        (T
            (multiply-all-numbers-rec (cons (* (car lst) x) res_lst) (cdr
               lst) x))))

(defun multiply-all-numbers (lst x)
    (multiply-all-numbers-rec NIL lst x))

;; (MULTIPLY-ALL-NUMBERS '(1 2 3 4 5) 10) -> (10 20 30 40 50)

;; b) элементы списка -- любые объекты
(defun multiply-all-numbers-rec (res_lst lst x)
    (cond
        ((null lst)
            (my-reverse res_lst))
        ((numberp (car lst))
            (multiply-all-numbers-rec (cons (* (car lst) x) res_lst) (cdr
               lst) x))
        ((symbolp (car lst))
            (multiply-all-numbers-rec (cons (car lst) res_lst) (cdr lst) x))
        (T
            (multiply-all-numbers-rec
                (cons (multiply-all-numbers-rec NIL (car lst) x) res_lst)
                (cdr lst) x))))

(defun multiply-all-numbers (lst x)
    (multiply-all-numbers-rec NIL lst x))

;; (MULTIPLY-ALL-NUMBERS '(a (2 c) d ((5) a)) 10) -> (A (20 C) D ((50) A))
```

1.5 Задание 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)).

Листинг 1.5 – my-sort

```
(defun my-append (lst1 lst2)
    (cond
        ((null lst1)
            lst2)
        (T
            (cons (car lst1) (my-append (cdr lst1) lst2)))))

(defun find-min-elem-rec (lst cur_min)
    (cond
        ((null lst)
            cur_min)
        (T
            (find-min-elem-rec (cdr lst)
                (cond
                    ((< (car lst) cur_min)
                        (setf cur_min (car lst)))
                    (T
                        cur_min))))))

(defun find-min-elem (lst)
    (find-min-elem-rec lst (car lst)))

(defun insert (lst elem elem_instead)
    (cond
        ((null lst)
            elem_instead)
        ((eql (car lst) elem_instead)
            (setf (car lst) elem))
        (T
            (insert (cdr lst) elem elem_instead))))

(defun my-sort-rec (res_lst lst)
    (let* ((min_elem (find-min-elem lst)))
        (cond
            ((null lst)
                res_lst)
            (T
                (insert lst (car lst) min_elem)
                (my-sort-rec
                    (my-append res_lst (list min_elem))
                    (cdr lst))))))

(defun my-sort (lst)
    (my-sort-rec NIL lst))

;; (MY-SORT '(4 7 2 2 8 1 3)) -> (1 2 2 3 4 7 8)
```

Листинг 1.6 – Решение задания 5

```
(defun find-right-numbers (res_lst lst b1 b2)
    (let* ((cur_elem (car lst)))
        (cond
            ((null lst)
                res_lst)
            ((and (numberp cur_elem)
                    (> cur_elem b1)
                    (< cur_elem b2))
                (find-right-numbers
                    (cons cur_elem res_lst)
                    (cdr lst) b1 b2))
            (T
                (find-right-numbers res_lst (cdr lst) b1 b2)))))

(defun select-between (lst b1 b2)
    (my-sort (find-right-numbers NIL lst b1 b2)))

;; (SELECT-BETWEEN '(1 5 4 2 3) 2 5) -> (3 4)
;; (SELECT-BETWEEN '(1 5 4 2 3) 0 6) -> (1 2 3 4 5)
;; (SELECT-BETWEEN '(1 5 4 2 3) 7 9) -> NIL
```

1.6 Задание 6

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

- одноуровнего смешанного;
- структурированного.

Листинг 1.7 – Решение задания 6

```
;; a) одноуровнего смешанного
(defun rec-add-rec (lst sum)
    (cond
        ((null lst)
            sum)
        ((numberp (car lst))
            (rec-add-rec (cdr lst) (+ sum (car lst))))
        (T
            (rec-add-rec (cdr lst) sum))))


```

```
(defun rec-add (lst)
    (rec-add-rec lst 0))

;; (REC-ADD '(1 2 3 (1 2 3) 4)) -> 10

;; b) структурированного
(defun rec-add-rec (lst sum)
    (cond
        ((null lst)
            sum)
        ((numberp (car lst))
            (rec-add-rec (cdr lst) (+ sum (car lst))))
        ((symbolp (car lst))
            (rec-add-rec (cdr lst) sum))
        (T
            (rec-add-rec (cdr lst) (rec-add-rec (car lst) sum)))))

(defun rec-add (lst)
    (rec-add-rec lst 0))

;; (REC-ADD '(1 (2) (3 (4 5 6)) ((7)))) -> 28
```

1.7 Задание 7

Написать рекурсивную версию с именем recnth функции nth.

Листинг 1.8 – Решение задания 7

```
(defun recnth (numb lst)
    (cond
        ((eql numb 0)
            (car lst))
        (T
            (recnth (- numb 1) (cdr lst)))))

;; (RECNTH 3 '(1 2 3 4 5)) -> 4
;; (NTH 3 '(1 2 3 4 5)) -> 4
```

1.8 Задание 8

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

Листинг 1.9 – Решение задания 8

```
(defun allodd (lst)
    (cond
        ((null lst)
            T)
        ((oddp (car lst))
            (allodd (cdr lst)))
        (T
            NIL)))

;; (ALLODD '(1 3 5)) -> T
;; (ALLODD '(1 3 4)) -> NIL
```

1.9 Задание 9

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

Листинг 1.10 – Решение задания 9

```
(defun get-first-odd-numb (lst)
    (cond
        ((null lst)
            NIL)
        ((and (numberp (car lst)) (oddp (car lst)))
            (car lst))
        ((listp (car lst))
            (or
                (get-first-odd-numb (car lst))
                (get-first-odd-numb (cdr lst))))
        (T
            (get-first-odd-numb (cdr lst)))))

;; (GET-FIRST-ODD-NUMB '(2 a (2 a) (4 ((1))) 3)) -> 1
```

1.10 Задание 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.

Листинг 1.11 – Решение задания 10

```
(defun get-quad-lst (lst)
    (cond
        ((null lst)
            NIL)
        (T
            (cons (* (car lst) (car lst))
                  (get-quad-lst (cdr lst))))))

;; (GET-QUAD-LST '(1 2 3 4)) -> (1 4 9 16)
```