



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»

Отчет по лабораторной работе №7
по дисциплине «Функциональное и логическое
программирование»

Тема Рекурсивные функции

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) _____

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

Москва — 2022 г.

Практические задания

Используя рекурсию:

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

```
1 (defun move-to (lst result)
2   (cond
3     ((null lst) result)
4     (t (move-to (cdr lst) (cons (car lst) result))))
5   )
6 )
7
8 (defun my-reverse (lst)
9   (move-to lst ()))
10 )
11
12 (my-reverse '(1 a ((2 . 3) 5 6) 7 8)) => (8 7 ((2 . 3) 5 6) A 1)
```

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

1. Хвостовая рекурсия

```
1 ; дополнительная проверка (listp (cdr x)) нужна из-за следующего результата:
2 ; (listp '(1 . 2)) => T
3
4 (defun f (lst)
5   (cond
6     ((null lst) Nil)
7     (
8       ((lambda (x) (and (listp x) (listp (cdr x)) (not (null x))))
9        (car lst))
10      (car lst)
11     )
12     (t (f (cdr lst)))
13   )
14 )
15 (f '(0 () (1 . 2) (2 3) (3) a)) => (2 3)
16 (f '(1)) => Nil
```

2. С помощью функционала find-if

```
1 (defun f (lst)
2   (find-if
3     #'(lambda (x)
4       (and (listp x) (listp (cdr x)) (not (null x))))
5     )
6   lst
7   )
8 )
9 (f '(0 () (1 . 2) (2 3) (3) a)) => (2 3)
10 (f '(1)) => Nil
```

3. Написать функцию, которая выбирает из заданного списка только числа между двумя заданными границами.

1. Хвостовая рекурсия

```
1 ; а) один cond, но рекурсивный вызов встречается в теле дважды
2 (defun move_to (lst res a b)
3   (cond
4     ((null lst) (reverse res))
5     ((and (numberp (car lst)) (< a (car lst) b))
6      (move_to (cdr lst) (cons (car lst) res) a b))
7     ((move_to (cdr lst) res a b))
8   )
9 )
10 ; б) два cond, но рекурсивный вызов встречается в теле единожды
11 (defun move_to (lst res a b)
12   (cond
13     ((null lst) (reverse res))
14     (t (move_to (cdr lst)
15                (cond
16                  ((and (numberp (car lst)) (< a (car lst) b))
17                   (cons (car lst) res))
18                  (res)
19                ) a b)
20     )
21   )
22 )
23 (defun select_between (lst a b)
24   (cond
25     ((< a b) (move_to lst () a b))
26     (t (move_to lst () b a))
27   )
28 )
```

2. Рекурсия, которая собирает результат на выходе.

```
1 (defun select_between (lst a b)
2   (cond
3     ((null lst) Nil)
4     ((and (numberp (car lst)) (< a (car lst) b)) (cons (car lst) (
5       select_between (cdr lst) a b)))
6     (t (select_between (cdr lst) a b))
7   )
8 )
9 (select_between '(0 3 7 a 5 (4 3) 1 6) 1 6) => (3 5)
```

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

1. Все элементы списка – числа.

```
1 ; а) с помощью рекурсии, собирающей результат на выходе
2 (defun mult_all (lst num)
3   (cond
4     ((null lst) Nil)
5     (t (cons (* (car lst) num) (mult_all (cdr lst) num)))
6   )
7 )
8
9 ; б) через хвостовую рекурсию
10 (defun move_to (lst res num)
11   (cond
12     ((null lst) (reverse res))
13     (t (move_to (cdr lst) (cons (* (car lst) num) res) num))
14   )
15 )
16
17 (defun mult_all (lst num)
18   (move_to lst () num)
19 )
20
21 (mult_all '(0 10 -10 5.5 2/3) 2) => (0 20 -20 11.0 4/3)
```

2. Элементы списка – любые объекты.

С помощью рекурсии, собирающей результат на выходе.

а) Работа только по верхнему уровню.

```
1 (defun mult_all (lst num)
2   (cond
3     ((null lst) Nil)
4     ((numberp (car lst)) (cons (* (car lst) num) (mult_all (cdr lst)
5       num))))
6     (t (cons (car lst) (mult_all (cdr lst) num))))
7   )
8 (mult_all '(0 a "abc" (1 k) 2/3 ((1 . 2) . 3)) 2) => (0 A "abc" (1 K)
4/3 ((1 . 2) . 3))
```

б) Работа по всем уровням структурированного списка.

```
1 ; Для определения того, является ли x точечной парой, используется проверка (atom (cdr x))
2 ; вместо (consp x) из-за следующего результата:
3 ; (listp '(k . 10))=(consp '(k . 10)) => T
4
5 ; вспомогательная функция, для точечных пар
6 (defun mult_all_cons (cns num)
7   (cond
8     ((and (numberp (car cns)) (numberp (cdr cns))) (cons (* (car cns)
9       num) (* (cdr cns) num)))
10    ((numberp (car cns)) (cons (* (car cns) num) (cdr cns)))
11    ((and (numberp (cdr cns)) (atom (car cns))) (cons (car cns) (* (cdr
12      cns) num)))
13    ((and (numberp (cdr cns)) (consp (car cns))) (cons (mult_all_cons (
14      car cns) num) (* (cdr cns) num)))
15    ((consp (car cns)) (cons (mult_all_cons (car cns) num) (cdr cns)))
16    (t (cons (car cns) (cdr cns))))
17   )
18 )
19 (defun mult_all (lst num)
20   (cond
21     ((null lst) Nil)
22     ((numberp (car lst)) (cons (* (car lst) num) (mult_all (cdr lst)
23       num))))
24     ((atom (car lst)) (cons (car lst) (mult_all (cdr lst) num)))
25     ((atom (cdr (car lst))) (cons (mult_all_cons (car lst) num) (
26       mult_all (cdr lst) num)))
27     (t (cons (mult_all (car lst) num) (mult_all (cdr lst) num))))
28   )
29 (mult_all '(0 a "abc" (1 k) 2/3 ((1 . 2) . 3)) 2) => (0 A "abc" (2 K)
4/3 ((2 . 4) . 6))
```

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)).

(Границы не включительно)

Сортировка по невозрастанию:

```
1 ; блочная (карманная, корзинная) сортировка
2 (defun my_sort (lst)
3   (cond ((null lst) Nil)
4         (t (nconc
5             (my_sort (remove-if-not (lambda (x) (< (car lst) x)) (cdr lst)))
6             (remove-if-not (lambda (x) (= (car lst) x)) lst)
7             (my_sort (remove-if-not (lambda (x) (> (car lst) x)) (cdr lst)))
8             )
9         )
10  )
11 )
12 ; сортировка выбором
13
14 ; в эту функцию не должен попадать пустой список
15 ; (лямбда-функция принимает строго 2 аргумента)
16 (defun list_min (lst)
17   (reduce
18     #'(lambda (a b) (cond ((< a b) a) (b)))
19     lst
20   )
21 )
22 (defun my_sort_inner (lst res)
23   (cond
24     ((null lst) res)
25     (t (let*
26         (
27           (cur_min (list_min lst))
28           (lst_rest (remove cur_min lst :count 1))
29         )
30       (my_sort_inner lst_rest (cons cur_min res))
31     )
32   )
33 )
34 )
35
36 (defun my_sort (lst)
37   (my_sort_inner lst ()))
```

Сначала исходный список сортируется по невозрастанию. Затем рекурсивно отсекается его голова, пока голова не станет меньше верхней границы. Наконец, отсекается его конец, начиная с элемента, который меньше или равен нижней границе, в процессе чего список инвертируется и становится отсортированным по неубыванию. Такой подход позволяет избежать рекурсии, в которой результат собирается на выходе.

```

1 ; в функциях cut_upper и cut_lower предполагается, что переданный список
2 ; отсортирован по невозрастанию
3 (defun cut_upper (lst b)
4   (cond
5     ((null lst) Nil)
6     ((< (car lst) b) lst)
7     (t (cut_upper (cdr lst) b)))
8   )
9 )
10 (defun cut_lower (lst a res)
11   (cond
12     ((null lst) res)
13     ((> a (car lst)) res)
14     (t (cut_lower (cdr lst) a (cons (car lst) res))))
15   )
16 )
17 (defun select-between-sorted (lst a b)
18   (cut_lower (cut_upper (my_sort lst) b) a ())
19 )
20
21 (select-between-sorted '(0 3 7 6 5 4 1) 1 6) => (3 4 5)

```

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

а) одноуровневого смешанного

```

1 (defun rec-add-inner (lst sum)
2   (cond
3     ((null lst) sum)
4     ((numberp (car lst)) (rec-add-inner (cdr lst) (+ sum (car lst))))
5     (t (rec-add-inner (cdr lst) sum)))
6   )
7 )
8 (defun rec-add (lst)
9   (rec-add-inner lst 0)
10 )
11 (rec-add '(1 a 2 -1/2 "abc")) => 5/2

```

б) структурированного (то есть элементами могут быть списки)

```
1 (defun rec-add-inner (lst sum)
2   (cond
3     ((null lst) sum)
4     ((numberp (car lst)) (rec-add-inner (cdr lst) (+ sum (car lst))))
5     ((atom (car lst)) (rec-add-inner (cdr lst) sum))
6     (t (rec-add-inner (cdr lst) (+ sum (rec-add (car lst))))))
7   )
8 )
9
10 (defun rec-add (lst)
11   (rec-add-inner lst 0)
12 )
13
14 (rec-add '(1 (1 (1 1)) a 1 ((1 "abc") 1))) => 7
```

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

```
1 (defun recnth (n lst)
2   (cond
3     ((null lst) Nil)
4     ((= n 0) (car lst))
5     (t (recnth (- n 1) (cdr lst)))
6   )
7 )
8
9 (recnth 0 '()) => Nil
10 (recnth 2 '(0 1)) => Nil
11 (recnth 0 '(0 1 2)) => 0
12 (recnth 2 '(0 (1 1) (2))) => (2)
```


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

```
1 ; если гарантируется, что все элементы списка – целые числа
2 (defun allodd (lst)
3   (cond
4     ((null lst) t)
5     ((oddp (car lst)) (allodd (cdr lst)))
6   )
7 )
8
9 (allodd '(1 3 5)) => T
10 (allodd '()) => T
11 (allodd '(2 3)) => Nil
12 (allodd '(1 3 4)) => Nil
13
14 ; если такой гарантии нет (то есть если встречается элемент списка,
15 ; который не является целым нечетным числом, возвращается Nil)
16 (defun allodd (lst)
17   (cond
18     ((null lst) t)
19     ((and (integerp (car lst)) (oddp (car lst))) (allodd (cdr lst)))
20   )
21 )
22
23 (allodd '(1 3 5)) => T
24 (allodd '(1 3.0)) => Nil
25 (allodd '(1 a)) => Nil
```

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

```
1 ; x может быть как атомом, так и структурой
2 (defun first_odd (x)
3   (cond
4     ((null x) Nil)
5     ((and (integerp x) (oddp x)) x)
6     ((atom x) Nil)
7     ((or (first_odd (car x)) (first_odd (cdr x))))
8   )
9 )
10
11
```

```

12 (first_odd '()) => NIL
13 (first_odd '(() 2 1)) => 1
14 (first_odd '(1.0 1)) => 1
15 (first_odd '(a 1)) => 1
16 (first_odd '("abc" 1)) => 1
17 (first_odd '((2 4) 1)) => 1
18 (first_odd '((2 . 4) 1)) => 1
19 (first_odd '(((2 . 1) 4) 3)) => 1
20 (first_odd '((2 4 (2 1)) 5)) => 1
21 (first_odd '((2 4 (2)) 6 (1 (3 5)))) => 1

```

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 ; а) с помощью рекурсии, собирающей результат на выходе
2 (defun squares (lst)
3   (cond
4     ((null lst) Nil)
5     (t (cons (* (car lst) (car lst)) (squares (cdr lst)))))
6   )
7 )
8
9 ; б) через "хвостовую рекурсию"
10 (defun squares_inner (lst result)
11   (cond
12     ((null lst) (reverse result))
13     (t (squares_inner (cdr lst) (cons (* (car lst) (car lst)) result))))
14   )
15 )
16
17 (defun squares (lst)
18   (squares_inner lst ()))
19 )
20
21 (squares '(0 10 -10 5.5 2/3)) => (0 100 100 30.25 4/9)

```