

Московский государственный технический университет  
имени Н. Э. Баумана



Факультет: Информатика и системы управления

Кафедра: Программное обеспечение ЭВМ и информационные технологии

Дисциплина: Функциональное и логическое программирование

## Лабораторная работа №6

Выполнили: Никичкин А. С., Фокеев А. И.  
Группа: ИУ7–61

Москва, 2015 г.

## 1 Что будет результатом?

```
(mapcar 'вектор '(570-40-8)) => Undefined function
```

## 2 Функция, которая уменьшает на 10 все числа из списка-аргумента этой функции

### 2.1 с помощью функционала

```
1 (defun 10- (lst)
2   (mapcar #'(lambda (x) (- x 10))
3     lst))
```

### 2.2 с помощью рекурсии с возможностью глубокой обработки

```
1 (defun smart-deep-sub (lst coef)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)
5           ((numberp item)
6            (cons (- item coef)
7                  (smart-deep-sub tail coef)))
8           ((listp item)
9            (cons (smart-deep-sub item coef)
10                  (smart-deep-sub tail coef)))
11           (t
12            (cons item
13                  (smart-deep-sub tail coef))))))
```

## 3 Функция, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком

```
1 (defun first-not-null-list (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)
5           ((atom item)
6            (first-not-null-list tail))
7           (t item))))
```

## 4 Функция, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10

Функция `between` фильтрует список элементов любой вложенности. Возвращает одноуровневый список состоящий из чисел находящихся между границей `left` и `right` игнорируя атомы, которые не являются числом.

```

1 (defun between (lst left right)
2   "Return new list with numbers between left and right"
3   (let ((item (first lst))
4         (tail (rest lst)))
5     (cond ((null lst) nil)
6           ((listp item)
7            (nconc (between item left right)
                    (between tail left right)))
9     ((and (numberp item) (< left item right))
10      (cons item
11            (between tail left right)))
12     (t (between tail left right))))
13
14 (defun 1<x<10 (lst)
15   (between lst 1 10))

```

## 5 Функция, вычисляющая декартово произведение двух своих списков-аргументов

```

1 (defun cross-set (x y)
2   (mapcan #'(lambda (_x)
3              (mapcar #'(lambda (_y)
4                          (list _x _y))
5                        y))
6         x))

```

## 6 Почему так реализовано reduce, в чем причина?

```

(reduce #'+ ()) => 0
(reduce #'* ()) => 1

```

Из дискретной математики известно, что для операции '+' нейтральным элементом является 0, а для операции '\*' — является 1.

## 7 Функция, которая вычисляет сумму длин всех элементов

```

1 (defun deep-length (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) 0)
5           ((atom item)
6            (1+ (deep-length tail)))
7     (t
8      (+ (deep-length item)
9         (deep-length tail)))))

```

## 8 Рекурсивную версия вычисления суммы чисел заданного списка

```
1 (defun rec-add (lst)
2   (let ((item (first lst))
3         (tail (rest lst))))
4     (cond ((null lst) 0)
5           ((listp item)
6            (+ (rec-add item)
7              (rec-add tail)))
8           ((numberp item)
9            (+ item
10              (rec-add tail)))
11          (t (format t "Error: <~a> is not a number" item)))))
```

## 9 Рекурсивная версия функции nth

```
1 (defun rec-nth (n lst)
2   (let ((item (first lst))
3         (tail (rest lst))))
4     (cond ((null lst) nil)
5           ((eql n 0) item)
6           (t (rec-nth (1- n) tail)))))
```

## 10 Рекурсивную функцию alloddp, которая возвращает Т, когда все элементы списка нечётные

```
1 (defun alloddp (lst)
2   (let ((item (first lst))
3         (tail (rest lst))))
4     (cond ((null lst) nil)
5           ((and (consp item) tail)
6            (and (alloddp item)
7                  (alloddp tail)))
8           ((and (consp item))
9            (and (alloddp item)))
10          ((and (oddp item) tail)
11           (and t (alloddp tail)))
12          ((oddp item)
13           t)
14          (t nil)))))
```

## 11 Рекурсивная функция, относящаяся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента

```
1 (defun custom-last (lst)
2   (let ((item (first lst))
3         (tail (rest lst))))
```

```

4      (cond ((null tail) item)
5              (t (custom-last tail))))))

```

## 12 Написать рекурсивную функцию, которая

### 12.1 вычисляет сумму всех чисел

```

1 (defun deep-sum (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) 0)
5            ((listp item)
6             (+ (deep-sum item)
7                (deep-sum tail)))
8            (t (+ item (deep-sum tail))))))

```

### 12.2 вычисляет сумму всех чисел от 0 до n-аргумента функции

```

1 (defun sum-args (lst n)
2   (let ((item (first lst))
3         (tail (rest lst))
4         (i (1- n)))
5     (cond ((or (null lst) (eql -1 n)) 0)
6            ((listp item)
7             (+ (deep-sum item)
8                (sum-args tail i)))
9            (t (+ item (sum-args tail i))))))

```

### 12.3 от n-аргумента функции до последнего $\geq 0$

```

1 (defun sum-with>=0 (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((or (null lst) (< item 0)) 0)
5            (t (+ item
6                  (sum-with>=0 tail))))))
7
8 (defun sum-from (lst n)
9   (declare (fixnum n))
10  (let ((tail (rest lst)))
11    (cond ((null lst) 0)
12           (> n 0)
13           (sum-from tail (1- n)))
14    (t (sum-with>=0 lst))))

```

## 12.4 от n-аргумента функции до m-аргумента с шагом d

```
1 (defun sum-to-with-step (lst to-arg step)
2   (declare (fixnum to-arg step))
3   (let ((item (first lst))
4         (tail (nthcdr step lst)))
5     (cond ((or (null lst) (< to-arg 0)) 0)
6           (t (+ item
7                 (sum-to-with-step tail
8                                   (- to-arg step)
9                                   step))))))
10
11 (defun sum-from-to-with-step (lst from-arg to-arg step)
12   (declare (fixnum from-arg to-arg step))
13   (let ((tail (rest lst)))
14     (cond ((null lst) 0)
15           ((> from-arg 0)
16            (sum-from-to-with-step tail
19                                   (1- from-arg)
20                                   (1- to-arg)
21                                   step))
22           (t (sum-to-with-step lst
23                                 to-arg
24                                 step))))))
```

## 13 Рекурсивная функция, которая возвращает последнее нечётное число из числового списка

```
1 (defun first-odd (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)
5           ((oddp item) item)
6           (t (first-odd tail))))
7
8 (defun last-odd (lst)
9   (first-odd (reverse lst)))
```

## 14 Функция которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке

```
1 (defun square (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)
5           ((consp item)
6            (cons (square item)
7                  (square tail)))
```

```

8      (t
9      (cons (* item item)
10            (square tail))))))

```

## 15 Написать функцию, которая

### 15.1 из списка выбирает все нечётные числа

```

1 (defun select-odd (lst)
2   (remove-if-not #'oddp
3                 (remove-if-not #'numberp lst)))
4
5 (defun sum-all-odd (lst)
6   (reduce #'+ (select-odd lst)))

```

### 15.2 из списка выбирает все нечётные числа (рекурсия)

```

1 (defun select-odd-rec (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)
5           ((consp item)
6            (nconc (select-odd-rec item)
7                  (select-odd-rec tail)))
8           ((and (numberp item) (oddp item))
9            (cons item
10                  (select-odd-rec tail)))
11          (t (select-odd-rec tail))))
12
13 (defun sum-all-odd-rec (lst)
14   (reduce #'+ (select-odd-rec lst)))

```

### 15.3 из списка выбирает все чётные числа

```

1 (defun select-even (lst)
2   (remove-if-not #'evenp
3                 (remove-if-not #'numberp lst)))
4
5 (defun sum-all-even (lst)
6   (reduce #'+ (select-even lst)))

```

### 15.4 из списка выбирает все чётные числа (рекурсия)

```

1 (defun select-even-rec (lst)
2   (let ((item (first lst))
3         (tail (rest lst)))
4     (cond ((null lst) nil)

```

```

5      ((cons item)
6      (nconc (select-even-rec item)
7              (select-even-rec tail)))
8      ((and (numberp item) (evenp item))
9      (cons item
10         (select-even-rec tail)))
11      (t (select-even-rec tail))))
12
13 (defun sum-all-even-rec (lst)
14   (reduce #'+ (select-even-rec lst)))

```