



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 №5

По курсу: "Функциональное и Логическое программирование"

Тема      Использование управляющих структур, работа со списками.

Группа      ИУ7-63Б

Студент      Сукочева А.

Преподаватель      Толпинская Н.Б.

Преподаватель      Строганов Ю. В.

## Практическая часть

**Задание 1.** Написать функцию, которая принимает целое число и возвращает первое чётное число, не меньшее аргумента.

```
(defun f1 (a) (cond
  ((= (rem a 2) 0) a)
  ((+ a 1)) ))
```

Результат работы:

```
(f1 11) ;; 12
(f1 10) ;; 10
```

**Задание 2.** Написать функцию, которая принимает число и возвращает число того же знака, но с модулем на 1 больше модуля аргумента.

```
(defun f2 (a) (cond
  ((< a 0) (- a 1))
  (t (+ 1 a)) ))
```

Результат работы:

```
(f2 3) ;; 4
(f2 -3) ;; -4
```

**Задание 3.** Написать функцию, которая принимает два числа и возвращает список из этих чисел, расположенный по возрастанию

```
(defun f3 (a b) (cond
  ((> a b) (list b a))
  (T (list a b)) ))
```

Результат работы:

```
(f3 1 2) ;; (1 2)
(f3 2 1) ;; (1 2)
```

**Задание 4.** Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим.

```
(defun f4 (a b c) (cond
  ((and (> a b) (< a c)) T)) )
```

Результат работы:

```
(f4 2 1 3) ;; T
(f4 1 2 3) ;; Nil
```

**Задание 5.** Каков результат вычисления следующих выражений?

При вычислении приведенного ниже выражения последовательно слева направо вычисляются аргументы функции до тех пор, пока не встретится значение аргумента, равное NIL. В этом случае вычисление прерывается и значение функции равно NIL. Если же были вычислены все значения аргументов и они все отличны от NIL, то результирующим значением функции and будет значение последнего аргумента.

```
(and 'free 'fir 'foe) ;; foe
```

При вычислении приведенного ниже выражения последовательно слева направо вычисляются аргументы до тех пор, пока не встретится значение аргумента, отличное от NIL. В этом случае вычисление прерывается и значение функции равно значению этого аргумента. Если же вычисленные значения всех аргументов равны NIL, то результирующее значение функции равно NIL.

```
(or 'fee 'fie 'foe) ;; fee
```

Сначала вычисляется (equal 'abc 'abc) и так как вычисленное значение равно T продолжается вычисление аргументов. 'yes - вычисляется как T и является последним аргументом, поэтому он является результатом выполнения функции.

```
(and (equal 'abc 'abc) 'yes) ;; yes
```

Возвращает первый аргумент, отличный от nil, в нашем случае это второй аргумент.

```
(or nil 'fie 'foe) ;; fie
```

Возвращает nil, т.к. один из аргументов (в нашем случае первый) вычисляется как nil,

```
(and nil 'fie 'foe) ;; nil
```

Возвращает T, т.к. or - возвращает значение первого аргумента, отличного от нуля. В нашем случае это первый аргумент - его значение T.

```
(or (equal 'abc 'abc) 'yes) ;; T
```

**Задание 6.** Написать предикат, который принимает два числа-аргумента и возвращает T, если первое число не меньше второго.

```
(defun f6 (a b) (cond ((>= a b) T)))
```

Результат работы:

```
(f6 2 1)      ;; T
(f6 5 5)      ;; T
(f6 1 2)      ;; Nil
```

**Задание 7.** Какой из следующих двух вариантов предикатов ошибочен и почему?

Предикат `numberp` возвращает T, если значение его аргумента - числовой атом, и NIL в противном случае. `plusp` возвращает T, если аргумент больше нуля, и NIL в противном случае.

Приведенный ниже предикат возвращает T, если аргументом является числовой атом и он больше нуля.

```
(defun pred1 (x)
  (and (numberp x) (plusp x)) )
```

Результат работы:

```
(pred1 1)      ;; T
(pred1 0)      ;; Nil
(pred1 -1)     ;; Nil
(pred1 'a)     ;; Nil
```

Приведенный ниже предикат ошибочен, потому что при вызове функции `and` аргументы вычисляются слева направо. При попытке проверить, является ли аргумент больше нуля (с помощью предиката `plusp`) возникает ошибка, если его аргументом не является числовой атом. Поэтому изначально следует проверить, что аргумент является числовым атомом и только после этого вызывать функцию `plusp`.

```
(defun pred2 (x)
  (and (plusp x) (numberp x)) )
```

Результат работы:

```
(pred2 1)      ;; T
(pred2 0)      ;; Nil
(pred2 -1)     ;; Nil
(pred2 'a)     ;; Error:  The value A is not of type NUMBER
```

**Задание 8.** Решить задачу 4, используя для её решения конструкции IF, COND, AND/OR. AND можно представить как:

```
(cond (x y))
```

OR можно представить как:

```
(cond (x) (y))
```

Используя COND.

```
(defun f4 (a b c) (cond
                  ((> a b) (< a c))) )
```

Используя AND/OR.

```
(defun f4 (a b c) (and (> a b) (< a c)) )
```

Используя IF.

```
(defun f4 (a b c)
  (if (> a b)
      (if (< a c) T Nil) Nil) )
```

**Задание 9.** Переписать функцию how-alike, приведенную в лекции и использующую COND, используя конструкции IF, AND/OR.

```
(defun how-alike (x y)
  (cond ((or (= x y) (equal x y)) 'the_same)
        ((and (oddp x) (oddp y)) 'both_odd)
        ((and (evenp x) (evenp y)) 'both_even)
        (t 'diff))) )
```

Используя IF.

```
(defun how-alike (x y)
  (if (or (= x y) (equal x y)) 'the_same
      (if (and (oddp x) (oddp y)) 'both_odd
          (if (and (evenp x) (evenp y)) 'both_even 'diff)))) )
```

Используя AND/OR.

```
(defun how-alike (x y)
  (or (and (or (= x y) (equal x y)) 'the_same)
      (and (and (oddp x) (oddp y)) 'both_odd)
      (and (and (evenp x) (evenp y)) 'both_even)
      'diff) )
```

Результат работы:

```
(how-alike 1 1)      ;; THE_SAME
(how-alike 2 4)      ;; BOTH_EVEN
(how-alike 7 11)     ;; BOTH_ODD
(how-alike 12 7)     ;; DIFF
```

# Теоретическая часть

## Классификация функций

1. Чистые математические функции - имеет фиксированное количество аргументов и один результат.
2. Специальные функции (формы) - произвольное количество аргументов.
3. Псевдофункции - создают эффект на экране.
4. Рекурсивные функции.
5. Функции с вариантными значениями, которые возвращают одно значение.
6. Функции высших порядков - используются для построения синтаксически управляемых программ.

## Работа функций `and`, `or`, `if`, `cond`

- При работе функции `and` последовательно слева направо вычисляются аргументы функции до тех пор, пока не встретится значение аргумента, равное `NIL`. В этом случае вычисление прерывается и значение функции равно `NIL`. Если же были вычислены все значения аргументов и они все отличны от `NIL`, то результирующим значением функции `and` будет значение последнего аргумента.

```
(and arg1 arg2 ... argn)
```

Пример:

```
(and 'one 'two 'three) ;; three
```

- При работе функции `or` последовательно слева направо вычисляются аргументы до тех пор, пока не встретится значение аргумента, отличное от `NIL`. В этом случае вычисление прерывается и значение функции равно значению этого аргумента. Если же вычисленные значения всех аргументов равны `NIL`, то результирующее значение функции равно `NIL`.

```
(or arg1 arg2 ... argn)
```

Пример:

```
(or 'one 'two 'three) ;; one
```

- `if`. В случае, если условие `test` истинно возвращается `t_body`, иначе, если `test` ложно возвращается `f_body`

```
(if test t_body f_body)
```

- `cond` (сокращение от `condition` - условие) служит средством разветвления вычислений.

```
(cond (test1 value1)  
      (test2 value2)  
      ...  
      (testn valuen) )
```

Функция `cond` последовательно слева направо вычисляет `testi` ( $i=0\dots n$ ) до тех пор, пока что не встретит значение, отличное от `Nil`. Результатом будет вычисленное значение `valuei`.

## Способы определения функции

1. С помощью `lambda`.

Базовое определение лямбда-выражения:

(`lambda` лямбда-список тело\_функции)

Пример:

```
(lambda (x y) (+ x y))
```

Применение лямбда-выражений:

(лямбда-выражение фактические\_параметры)

Пример:

```
((lambda (x y) (+ x y)) 1 2) ;; 3
```

2. С помощью `defun`. Для неоднократного применения функции (а также для построения рекурсивной функции) используется встроенная функция `defun`.

Синтаксис:

(`defun` имя\_функции лямбда-список тело\_функции)

Пример:

```
(defun sum (x y) (+ x y))
```

Пример вызова:

```
(sum 1 2) ;; 3
```