



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический
университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 7

Дисциплина	Функциональное и логическое программирование
Студент	Сиденко А.Г.
Группа	ИУ7-63Б
Преподаватель	Толпинская Н.Б., Строганов Ю.В.

Москва, 2020 г.

1. Написать предикат, который принимает два числа-аргумента и возвращает Т, если первое число не меньше второго.

```
1 (defun eqb (a b)
2   (if (> b a) Nil T) )
```

2. Какой из следующих двух вариантов предиката ошибочен и почему?

```
1 (defun pred1 (x)
2   (and (numberp x) (plusp x)))
3
4 (defun pred2 (x)
5   (and (plusp x) (numberp x)))
```

Ошибочен второй, так как перед выполнением операции, не делается проверка на число.

3. Переписать функцию how-alike, приведенную в лекции и не использующую COND, используя конструкция IF, AND/OR.

```
1 (defun how_alike (x y)
2   (cond ((or (= x y) (equal x y)) 'the_same)
3         ((and (oddp x) (oddp y)) 'both_odd)
4         ((and (evenp x) (evenp y)) 'both_even)
5         (T 'difference)) )
6
7 (defun how_alike_if (x y)
8   (if (or (= x y) (equal x y)) 'the_same
9       (if (and (oddp x) (oddp y)) 'both_odd
10          (if (and (evenp x) (evenp y)) 'both_even
11              'difference)))) )
```

Результат:

(how_alike 11 30) → DIFFERENCE

(how_alike_if 11 30) → DIFFERENCE

(how_alike 3 3) → THE_SAME

(how_alike_if 3 3) → THE_SAME

(how_alike 1 3) → BOTH_ODD

(how_alike_if 1 3) → BOTH_ODD

(how_alike -2 30) → BOTH_EVEN

(how_alike_if -2 30) → BOTH_EVEN

4. **Чем принципиально отличаются функции cons, list, append?**

$(\text{cons } \text{lst1 } \text{lst2}) \rightarrow ((1\ 2\ 3)\ 4\ 5)$

$(\text{list } \text{lst1 } \text{lst2}) \rightarrow ((1\ 2\ 3)\ (4\ 5))$

$(\text{append } \text{lst1 } \text{lst2}) \rightarrow (1\ 2\ 3\ 4\ 5)$

CONS – позволяет создавать списки (возвращает бинарную ячейку (точечная пара, список), расставляя указатели, обязательно 2 аргумента).

APPEND – функция двух аргументов x и y, сцепляющая два списка в один.

LIST – создает столько списковых ячеек, сколько аргументов (всегда возвращает список).

5. **Каковы результаты вычисления следующих выражений?**

$(\text{reverse } ()) \rightarrow \text{Nil}$

$(\text{last } ()) \rightarrow \text{Nil}$

$(\text{reverse } '(a)) \rightarrow (a)$

$(\text{last } '(a)) \rightarrow (a)$

$(\text{reverse } '((a\ b\ c))) \rightarrow ((a\ b\ c))$

$(\text{last } '((a\ b\ c))) \rightarrow ((a\ b\ c))$

6. **Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.**

Рекурсивно – пока второй элемент не Nil идем дальше, иначе возвращаем голову:

```
1 (defun last_elem (lst)
2   (if (NULL (cadr lst)) (car lst) (last_elem (cdr lst))))
```

С использованием функционала – с использованием функционала reduce, возвращая последний полученный результат, а возвращаем каждый раз второй аргумент:

```
1 (defun last_elem (lst)
2   (reduce #'(lambda (a x) x) lst))
```

7. **Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.**

С использованием функционала – пользуясь тем, что маркар будет работать до хвоста в самом коротком списке (cdr lst всегда на один короче lst), возвращаем значение из lst, таким образом получится список без последнего элемента:

```

1 (defun centr (lst)
2   (mapcar (lambda (x y) y) (cdr lst) lst))

```

Рекурсивно – пока хвост не Nil, соединяем с помощью cons голову и возвращаем значение функции от хвоста:

```

1 (defun centr (lst)
2   (if (NULL (cadr lst)) (cons (car lst) (centr (cdr lst)))

```

8. Написать простой вариант игры в кости, в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 – выигрыш, если выпало (1,1) или (6,6) — игрок право снова бросить кости, во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то выигрывает тот игрок, у которого больше очков. Результат игры и значения выпавших костей выводить на экран с помощью функции print.

Функция game запускает игру, в которой анализируются 2 рандомных числа (от 1 до 6) для 2 игроков. Далее сравниваются их суммы, в зависимости от этого возвращается результат.

Функция analyze принимает на вход 2 числа, и в зависимости от них, либо перекидывает кости (функция вызывается еще раз с новыми параметрами) либо возвращает число (сумму очков).

Функция printS выводит очки и их сумму на экран.

```

1 (defun printS (x y s)
2   (print (list x '+ y '= s)))
3
4 (defun analyze (x y)
5   (cond ((and (= 1 x) (= 1 y)) (printS x y (+ x y))
6     (analyze (+ 1 (random 6)) (+ 1 (random 6))))
7     ((and (= 6 x) (= 6 y)) (printS x y (+ x y))
8     (analyze (+ 1 (random 6)) (+ 1 (random 6))))
9     (T (printS x y (+ x y)) (+ x y))) )
10
11 (defun game ()
12   (let ((s1 (analyze (+ 1 (random 6)) (+ 1 (random 6))))
13         (s2 (analyze (+ 1 (random 6)) (+ 1 (random 6)))))
14     (cond ((or (= s1 7) (= s1 11) (> s1 s2)) 'you-win)
15           ((or (= s1 7) (= s2 11) (< s1 s2)) 'he-wins)
16           (T 'draw))) )

```

Примеры работы:

(1 + 5 = 6)
(6 + 6 = 12)
(4 + 6 = 10) HE-WINS%

(3 + 2 = 5)
(3 + 2 = 5) DRAW%

(2 + 2 = 4)
(5 + 6 = 11) HE-WINS%

(3 + 3 = 6)
(5 + 4 = 9) HE-WINS%

(4 + 4 = 8)
(5 + 1 = 6) YOU-WIN%