



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»

## Отчёт по лабораторной работе №4 по курсу «Функциональное и логическое программирование»

Тема Использование управляющих структур, работа со списками

Студент Богаченко А. Е.

Группа ИУ7-65Б

Оценка (баллы) \_\_\_\_\_

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

# Задание 1

Каковы результаты следующих выражений?

Листинг 1 – Задание 1

```
1 (setf lst1 '(a b))
2 (setf lst2 '(c d))
3 (cons lst1 lst2) ; ((a b) c d)
4 (list lst1 lst2) ; ((a b) (c d))
5 (append lst1 lst2) ; (a b c d)
```

# Задание 2

Каковы результаты следующих выражений?

Листинг 2 – Задание 2

```
1 (reverse ()) ; nil
2 (last ()) ; nil
3 (reverse '(a)) ; (a)
4 (last '(a)) ; (a)
5 (reverse '((a b c))) ; ((a b c))
```

# Задание 3

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

Листинг 3 – Задание 3

```
1
2 (defun last-recursive (lst)
3   (if (cdr lst)
4       (last-recursive (cdr lst))
5       (car lst)))
6
7 (defun int-last-recursive-2 (lst el)
8   (if (eql nil lst)
9       el
```

```

10     (last-recursive-2 (cdr lst) (car lst))))
11 (defun my-last-recursive-2 (lst)
12   (int-last-recursive-2 lst nil))
13
14 (defun last-reduce (lst)

```

## Задание 4

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

### Листинг 4 – Задание 4

```

1
2 (defun no-last-int (lst acc)
3   (if (cdr lst)
4       (no-last-int (cdr lst) (cons (car lst) acc))
5       (nreverse acc)))
6
7 (defun no-last (lst)
8   (and lst (no-last-int lst nil)))
9
10 (defun no-last-kernel (lst)

```

## Задание 5

в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 — выигрыш, если выпало (1, 1) или (6, 6) — игрок получает 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`.

## Листинг 5 – Задание 5

```

1 (defmacro ->> (initial-form &rest forms)
2   (reduce #'(lambda (acc next)
3     (if (listp next)
4       (append next (list acc))
5       (list next acc)))
6     forms :initial-value initial-form))
7
8 (defconstant +auto-win-scores+ '(7 11))
9 (defconstant +rethrow-combinations+ '((1 1) (6 6)))
10
11 (defun throw-dices ()
12   (let ((first-throw (1+ (random 6)))
13         (second-throw (1+ (random 6)))))
14     (list first-throw second-throw)))
15
16 (defun init-n-throws (n acc)
17   (if (zerop n)
18     acc
19     (init-n-throws (decf n) (cons (throw-dices) acc))))
20
21 (defun score-results (player-id dices sum-acc)
22   (let* ((dices-sum (apply #' + dices))
23         (result-sum (+ sum-acc dices-sum)))
24     (format T "Player ~a throws ~a: ~a -> ~a ~%" player-id dices sum-acc result-sum)
25     (cond ((member dices +rethrow-combinations+ :test #'equal)
26            (score-results player-id (throw-dices) result-sum))
27           ((member dices-sum +auto-win-scores+)
28            (cons player-id -1))
29           (T (cons player-id result-sum)))))
30
31 (defun pick-a-winner (a b)
32   (let ((sum-a (cdr a))
33         (sum-b (cdr b)))
34     (cond ((= sum-a -1) a)
35           ((= sum-b -1) b)
36           ((>= sum-a sum-b) a)
37           (T b))))
38
39 (defun play (players)
40   (let ((player-id 0))
41     (->> (init-n-throws players nil)
42          (mapcar #'(lambda (init-dices)
43                      (score-results (incf player-id) init-dices 0)))
44          (reduce #'pick-a-winner)
45          car
46          (format T "Winner: ~a")
47          not)))

```

# Контрольные вопросы

## 1. Структуроразрушающие и не разрушающие структуру списка функции

### Не разрушающие структуру списка функции

Данные функции не меняют сам объект-аргумент, а создают копию.

#### Функция `append`

Объединяет списки. Это форма, можно передать больше 2 аргументов. Создает копию для всех аргументов, кроме последнего.

Пример: `(append '(1 2) '(3 4))` — `(1 2 3 4)`.

#### Функция `reverse`

Возвращает копию исходного списка, элементы которого переставлены в обратном порядке. **В целях эффективности работает только на верхнем уровне.**

Пример: `(reverse '(1 2 3 4))` — `(4 3 2 1)`.

#### Функция `last`

Проход по верхнему уровню и возврат последней списковой ячейки.

Пример: `(last '(1 2 3 4))` — `(4)`.

#### Функция `nth`

Возврат указателя от n-ной списковой ячейки, нумерация с нуля.

Пример: `(nth 1 '(1 2 3 4))` — `2`.

#### Функция `nthcdr`

Возврат n-ого хвоста.

Пример: `(nthcdr 1 '(1 2 3 4))` — `(2 3 4)`.

### Функция `length`

Возврат длины списка (**только по верхнему уровню**).

Пример: `(length '(1 2 (3 4)))` — 3.

### Функция `remove`

Модифицирует, но работает с копией, поэтому не разрушает. Данная функция удаляет элемент по значению (Часто разрушающая аналогичная функция называется `delete`). По умолчанию используется `eq` для сравнения на равенство, но можно передать другую функцию через ключевой параметр `:test`.

Примеры:

1. `(remove 3 '(1 2 3))` — `(1 2)`;
2. `(remove '(1 2) '((1 2) (3 4)))` — `((1 2) (3 4))`;
3. `(remove '(1 2) '((1 2) (3 4)) :test #'equal)` — `((3 4))`;

### Функция `rplaca`

Переставляет `car`-указатель на 2 элемент-аргумент (*S*-выражение).

Пример: `(rplaca '(1 2 3) 3)` — `(3 2 3)`.

### Функция `rplacd`

Переставляет `cdr`-указатель на 2 элемент-аргумент (*S*-выражение).

Пример: `(rplacd '(1 2 3) '(4 5))` — `(1 4 5)`.

### Функция `subst`

Заменяет все элементы списка, которые равны 2 переданному элементу-аргументу на другой 1 элемент-аргумент. *По умолчанию для сравнения используется функция `eq`.*

Пример: `(subst 2 1 '(1 2 1 3))` — `(2 2 2 3)`.

## Структуроразрушающие функции

Данные функции меняют сам объект-аргумент, невозможно вернуться к исходному списку. Чаще всего такие функции начинаются с префикса **n-**.

### Функция `ncons`

Работает аналогично `append`, только не копирует свои аргументы, а разрушает структуру.

### Функция `nreverse`

Работает аналогично `reverse`, но не создает копии.

### Функция `nsubst`

Работает аналогично функции `nsubst`, но не создает копии.

## 2. Отличие в работе функций `cons`, `list`, `append` и в их результате

Функция `cons` — чисто математическая, конструирует списковую ячейку, которая может вовсе и не быть списком (будет списком только в том случае, если 2 аргументом передан список).

Примеры:

1. `(cons 2 '(1 2))` — `(2 1 2)` — список;
2. `(cons 2 3)` — `(2 . 3)` — не список.

Функция `list` — форма, принимает произвольное количество аргументов и конструирует из них список. Результат — всегда список. При нуле аргументов возвращает пустой список.

Примеры:

1. `(list 1 2 3)` — `(1 2 3)`;

2. `(list 2 '(1 2)) — (2 (1 2));`

3. `(list '(1 2) '(3 4)) — ((1 2) (3 4));`

Функция `append` — форма, принимает на вход произвольное количество аргументов и для всех аргументов, кроме последнего, создает копию, ссылая при этом последний элемент каждого списка-аргумента на первый элемент следующего по порядку списка-аргумента (так как модифицируются все списки-аргументы, кроме последнего, копирование для последнего не делается в целях эффективности).

Примеры:

1. `(append '(1 2) '(3 4)) — (1 2 3 4);`

2. `(append '((1 2) (3 4)) '(5 6)) — ((1 2) (3 4) 5 6).`