



КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Тема** Использование управляющих структур, работа со списками

**Студент**    Пересторонин П.Г.

Группа ИУ7-63Б

## Оценка

Преподаватель    Толпинская Н. Б.

Москва — 2021 г.

# Оглавление

<b>1</b>	<b>Задания</b>	<b>2</b>
1.1	Чем принципиально отличаются функции <code>cons</code> , <code>list</code> , <code>append</code> ? Пусть... . . . . .	2
1.2	Каковы результаты следующих выражений? . . . . .	2
1.3	Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента . . .	3
1.3.1	Рекурсия 1 . . . . .	3
1.3.2	Рекурсия 2 . . . . .	3
1.3.3	С помощью <code>reduce</code> . . . . .	3
1.4	Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента .	4
1.4.1	Рекурсия . . . . .	4
1.4.2	Функции ядра . . . . .	4
1.5	Написать простой вариант игры в кости, в котором... . . . .	4
<b>2</b>	<b>Ответы на вопросы к лабораторной работе</b>	<b>6</b>
2.1	Структуроразрушающие и не разрушающие структуру списка функции . . . . .	6
2.1.1	Не разрушающие структуру списка функции . . . . .	6
2.1.2	Структуроразрушающие функции . . . . .	8
2.2	Отличие в работе функций <code>cons</code> , <code>list</code> , <code>append</code> и в их результате	8

# 1 Задания

## 1.1 Чем принципиально отличаются функции cons, list, append? Пусть...

Продолжение задания:

Пусть

```
1 (setf lst1 '(a b))  
2 (setf lst2 '(c d))
```

Каковы результаты следующих выражений?

```
1 (cons lst1 lst2)
```

Результат: ((A B) C D)

```
1 (list lst1 lst2)
```

Результат: ((A B) (C D))

```
1 (append lst1 lst2)
```

Результат: (A B C D)

## 1.2 Каковы результаты следующих выражений?

```
1 (reverse ())
```

Результат: Nil

```
1 (last ())
```

Результат: Nil

```
1 (reverse '(a))
```

Результат: (a)

```
1 (last '(a))
```

Результат: (a)

```
1 (reverse '((a b c)))
```

Результат: ((a b c))

```
1 (last '((a b c)))
```

Результат: ((a b c))

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

### 1.3.1 Рекурсия 1

```
1 (defun my-last-recursive-internal (lst)
2   (if (cdr lst)
3       (my-last-recursive-internal (cdr lst))
4       (car lst)))
5 (defun my-last-recursive (lst)
6   (and lst (my-last-recursive-internal lst)))
```

### 1.3.2 Рекурсия 2

```
1 (defun my-last-recursive-internal-2 (lst last-el)
2   (if (eql nil lst) last-el (my-last-recursive-internal-2 (cdr lst) (car lst))))
3 (defun my-last-recursive-2 (lst)
4   (my-last-recursive-internal-2 lst nil))
```

### 1.3.3 С помощью reduce

```
1 (defun my-last-reduce (lst)
2   (reduce #'(lambda (acc el) el) lst))
```

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

### 1.4.1 Рекурсия

```
1 (defun no-last-internal (lst acc)
2   (if (cdr lst)
3       (no-last-internal (cdr lst) (cons (car lst) acc))
4       (nreverse acc)))
5 (defun no-last (lst)
6   (and lst (no-last-internal lst nil)))
```

### 1.4.2 Функции ядра

```
1 (defun no-last-kern (lst)
2   (and lst (nreverse (cdr (reverse lst))))))
```

## 1.5 Написать простой вариант игры в кости, в котором...

### Продолжение задания:

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 1.1: Игра в кости

```
1 (defconstant +dices-amount+ 2)
2 (defconstant +max-score+ 6)
3 (defconstant +magic-scores+ '(7 11))
4 (defconstant +rethrow-combinations+ '((1 1) (6 6)))
5
6 (defun iter-times-internal (times fn acc)
7   (if (<= times 0)
8       acc
9       (iter-times-internal (- times 1) fn (cons (funcall fn times) acc))))
10 (defun iter-times (times fn)
11   (iter-times-internal times fn nil))
12
13 (defun throw-dices (times)
14   (iter-times times #'(lambda (_x) (+ (random +max-score+) 1))))
15
16 (defun score-with-rules (i dices)
17   (let ((sum (reduce #'+ dices)))
18     (format T "Player~a~has~a!~%" i dices)
19     (cond ((member dices +rethrow-combinations+ :test #'equal)
20            (score-with-rules i (throw-dices +dices-amount+)))
21            ((member sum +magic-scores+) (cons i (+ (+dices-amount+ +max-score+) 1)))
22            (T (cons i sum)))))
23
24 (defun collect-throws (amount)
25   (mapcar #'(lambda (x) (score-with-rules (car x) (cdr x)))
26           (iter-times amount #'(lambda (i) (cons i (throw-dices +dices-amount+))))))
27
28 (defun is-next-better (prev next)
29   (or (< (cdr prev) (cdr next))))
30
31 (defun play (players_amount)
32   (let ((winner (reduce
33                #'(lambda (cur next) (if (is-next-better cur next) next cur))
34                (collect-throws players_amount))))
35     (format T "Winner:~a" (car winner))))
```

## 2 Ответы на вопросы к лабораторной работе

### 2.1 Структуроразрушающие и не разрушающие структуру списка функции

#### 2.1.1 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 `eq1` 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 `eq1`.



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

### 2.1.2 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.

## 2.2 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).`