



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

```
10 (defun my-last-recursive-2 (lst)
11   (int-last-recursive-2 lst nil))
12
13 (defun last-reduce (lst)
14   (reduce #'(lambda (acc e) e) lst))
```

Задание 4

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

Листинг 4 – Задание 4

```
1 (defun no-last-int (lst acc)
2   (if (cdr lst)
3       (no-last-int (cdr lst) (cons (car lst) acc))
4       (nreverse acc)))
5
6 (defun no-last (lst)
7   (and lst (no-last-int lst nil)))
8
9 (defun no-last-kernel (lst)
10  (and lst (nreverse (cdr (reverse 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
7   :initial-value initial-form))
8
9 (defconstant +auto-win-scores+ '(7 11))
10 (defconstant +rethrow-combinations+ '((1 1) (6 6)))
11
12 (defun throw-dices ()
13   (let ((first-throw (1+ (random 6)))
14         (second-throw (1+ (random 6))))
15     (list first-throw second-throw)))
16
17 (defun score-results (player-id &optional (score 0))
18   (let* ((dices (throw-dices))
19         (dices-sum (apply #'+ dices))
20         (result-score (+ score dices-sum)))
21     (format T "Player ~a throws ~a: ~a -> ~a ~%" player-id dices score result-score)
22     (cond ((member dices +rethrow-combinations+ :test #'equal)
23            (score-results player-id result-score))
24           ((member dices-sum +auto-win-scores+)
25            (cons player-id -1))
26           (T (cons player-id result-score)))))
27
28 (defun pick-a-winner (a b)
29   (let ((score-a (cdr a))
30         (score-b (cdr b)))
31     (cond ((= score-a -1) a)
32           ((= score-b -1) b)
33           ((>= score-a score-b) a)
34           (T b))))
35
36 (defun play (players)
37   (let ((init-player-id 0))
38     (->> (loop repeat players collect (incf init-player-id))
39         (mapcar #'(lambda (player-id)
40           (score-results player-id)))
41         (reduce #'pick-a-winner)
42         car
43         (format T "Winner: ~a")
44         not)))

```

Контрольные вопросы

1. Синтаксическая форма и хранение программы в памяти

В LISP формы представления программы и обрабатываемых ею данных одинаковы и представляются в виде **S-выражений**. Поэтому программы могут обрабатывать и преобразовывать другие программы и даже самих себя. В процессе трансляции можно введенное и сформированное в результате вычислений выражение данных проинтерпретировать в качестве программы и непосредственно выполнить. Так как программа представляет собой S-выражение, в памяти она представлена либо как атом (5 указателей; форма представления атома в памяти), либо списковой ячейкой (бинарный узел; 2 указателя).

2. Трактовка элементов списка

Первый аргумент списка, который поступает на вход интерпретатору, трактуется как имя функции, остальные — как аргументы этой функции.

3. Порядок реализации программы

Программа в языке LISP представляется S-выражением, которое передается интерпретатору — функции `eval`, которая выводит последний, полученный после обработки S-выражения, результат. Работа функции `eval` представлена на картинке ниже.

4. Способы определения функций

С помощью макро определения `defun` или с использованием Лямбда-нотации (функция без имени).