



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 №4 по курсу "Функциональное и логическое программирование"

Тема Использование управляющих структур, работа со списками

Студент Цветков И.А.

Группа ИУ7-63Б

Оценка (баллы) _____

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

Москва — 2022 г.

1 Практические задания

1.1 Задание 1

Условие: чем принципиально отличаются функции `cons`, `list`, `append`? Пусть `(setf lst1 '(a b)); (setf lst2 '(c d))`. Каковы результаты вычисления следующих выражений?

```
1 (cons lst1 lst2) ; ((A B) C D)
2 (list lst1 lst2) ; ((A B) (C D))
3 (append lst1 lst2) ; (A B C D)
```

1.2 Задание 2

Условие: каковы результаты вычисления следующих выражений, и почему?

```
1 (reverse '()) ; Nil
2 (last ()) ; Nil
3 (reverse '(a)) ; (A)
4 (last '(a)) ; (A)
5 (reverse '((a b c))) ; ((ABC))
6 (last '((a b c))) ; ((A B C))
```

1.3 Задание 3

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

```
1 (defun get_last1 (lst) (first (reverse lst)))
2 (defun get_last2 (lst) (if (eq (cdr lst) Nil) (car lst)
3   (get_last2 (cdr lst))))
4 (defun get_last3 (lst) (cond ((eq (cdr lst) Nil) (car lst)) (T
5   (get_last3 (cdr lst)))))
```

1.4 Задание 4

Условие: написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.

```
1 (defun without_last1 (lst) (remove (first (reverse lst)) lst))
2 (defun without_last2 (lst) (if (eq (cdr lst) Nil) () (cons (car
  lst) (without_last2 (cdr lst)))))
3 (defun without_last3 (lst) (cond ((eq (cdr lst) Nil) ()) (T
  (cons (car lst) (without_last2 (cdr lst))))))
```

1.5 Задание 5

Условие: 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 имеет 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.

Игра в кости 1. Победа - выпало 7 или 11 2. Переброс - (1, 1) (6, 6) 3. Иначе - другой игрок (сумма больше - победа)

```
1 ; Задание 5
2
3 (defvar player1)
4 (defvar player2)
5 (setf player1 "user")
6 (setf player2 "computer")
7
8 (defvar dice1)
9 (defvar dice2)
10 (defvar tmp_dice)
11
12
13 (defun roll_dice()
14   (+ (random 6) 1)
15 )
```

```

16 (defun roll_dices()
17   (list (roll_dice) (roll_dice))
18 )
19
20 (defun sum_points(dice)
21   (+ (car dice) (cadr dice))
22 )
23
24 (defun is_win(dice)
25   (cond ((= (sum_points dice) 7) T) ((= (sum_points dice) 11)
26     T) )
27 )
28 (defun is_repeat(dice)
29   (cond ((= (car dice) (cadr dice) 1) T) ((= (car dice) (cadr
30     dice) 6) T) )
31 )
32 (defun is_player_won(result)
33   (= (cadr result) 1)
34 )
35
36 (defun print_res (player dice)
37   (format Nil "Win ~a, points = ~a, sum = ~a" player (car
38     dice) (sum_points (car dice)))
39 )
40
41 (defun print_info (player dice)
42   (format Nil "Player: ~a, points = ~a, sum = ~a" player
43     dice (sum_points dice))
44 )
45
46 (defun player_move (player)
47   (setf tmp_dice (roll_dices))
48   (print_info player tmp_dice)
49   (cond ((is_win tmp_dice) (list tmp_dice 1))
50     ((is_repeat tmp_dice) (player_move player))
51     (T (list tmp_dice 0)))
52 )

```

```
53
54
55 (defun start_game()
56   (setf dice1 (player_move player1))
57   (if (is_player_won dice1) (print_res player1 dice1)
58     (and (setf dice2 (player_move player2))
59       (cond ((is_player_won dice2) (print_res player2 dice2))
60         ((> (sum_points (car dice1)) (sum_points (car
61           dice2))) (print_res player1 dice1))
62         ((< (sum_points (car dice1)) (sum_points (car
63           dice2))) (print_res player2 dice2))
64         ((format Nil "Draw")) )))
65 )
```

2 Ответы на вопросы к лабораторной работе

2.1 Синтаксическая форма и хранение программы в памяти

TODO

2.2 Трактовка элементов списка

TODO

2.3 Порядок реализации программы

TODO

2.4 Способы определения функции

Построить функцию можно с помощью Lambda-выражения (базисный способ)

Lambda-определение безымянной функции:

```
1 (lambda <Lambda-список> <форма>),
```

где Lambda-список – список аргументов, а форма – тело функции.

Lambda-вызов функции:

```
1 (<Lambda-выражение> <формальные параметры>)
```

Функции с именем. В таких функциях `defun` связывает символьный атом с Lambda-определением:

```
1 (defun f <Lambda-выражение>)
```

Упрощенное определение:

1 (defun f(x1, ..., xk) (<формы>))