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)
       (get_last2 (cdr lst))))
3    (defun get_last3 (lst) (cond ((eq (cdr lst) Nil) (car lst)) (T
       (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) (reverse (cdr (reverse 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.

```
1  (defvar player1 "user")
2  (defvar player2 "computer")
3
4  (defvar dice1)
5  (defvar dice2)
6  (defvar tmp_dice)
7
8
9  (defun roll_dice()
10     (+ (random 6) 1)
11 )
12 (defun roll_dices()
13     (list (roll_dice) (roll_dice))
14 )
15
16 (defun sum_points(dice)
17     (+ (car dice) (cadr dice))
```

```
18      )
19
20      (defun is_win(dice)
21          (cond ((= (sum_points dice) 7) T) ((= (sum_points dice) 11)
                T) )
22      )
23
24      (defun is_repeat(dice)
25          (cond ((= (car dice) (cadr dice) 1) T) ((= (car dice) (cadr
                dice) 6) T) )
26      )
27
28      (defun is_player_won(result)
29          (= (cadr result) 1)
30      )
31
32      (defun print_res (player dice)
33          (format Nil "Win ~a, points = ~a, sum = ~a" player (car
                dice) (sum_points (car dice)))
34      )
35
36
37      (defun print_info (player dice)
38          (format T "Player: ~a, points = ~a, sum =  ~a" player dice
                (sum_points dice))
39      )
40
41
42      (defun player_move (player)
43          (setf tmp_dice (roll_dices))
44          (print_info player tmp_dice)
45          (cond ((is_win tmp_dice) (list tmp_dice 1))
46                ((is_repeat tmp_dice) (player_move player))
47                (T (list tmp_dice 0)))
48      )
49
50
51      (defun start_game()
52          (setf dice1 (player_move player1))
53          (if (is_player_won dice1) (print_res player1 dice1)
54              (and (setf dice2 (player_move player2))
```

```
55              (cond ((is_player_won dice2) (print_res player2 dice2))
56                    ((> (sum_points (car dice1)) (sum_points (car
                         dice2))) (print_res player1 dice1))
57                    ((< (sum_points (car dice1)) (sum_points (car
                         dice2))) (print_res player2 dice2))
58                    ((format Nil "Draw")) )))
59      )
```

2 Ответы на вопросы к лабораторной работе

2.1 Синтаксическая форма и хранение программы в памяти

В LISP формы представления программы и обрабатываемых ею данных одинаковы и представляются в виде S-выражений. Из-за этого программы могут обрабатывать и преобразовывать другие программы и даже самих себя. В процессе трансляции можно введенное и сформированное в результатевычислений 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 (при этом память выделяется блоками)), либо списковой ячейкой (бинарный-узел; 2 указателя).

2.2 Трактовка элементов списка

Первый аргумент списка – имя функции, а остальные – аргументы этой функции (если при этом отсуствует блокировка вычислений).

2.3 Порядок реализации программы

Программа на языке Lisp – S-выражение. Данное S-выражение передается функции `eval`, которая обрабатывает программу. Схема работы функции `eval` представлена ниже.

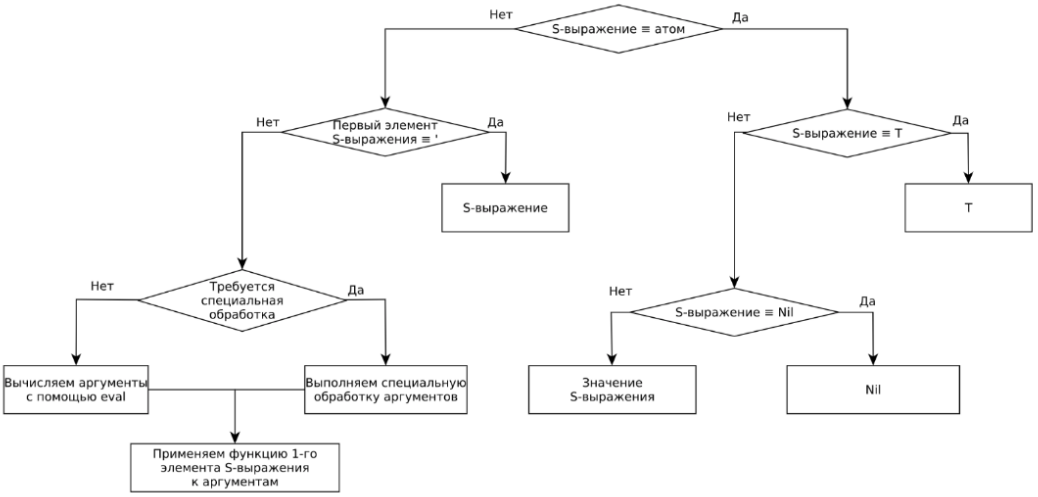

Рисунок 2.1 – Работа функции eval

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>))
```