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

Название «Использование управляющих структур, работа со списками»

Дисциплина «Функциональное и логическое программирование»

Студент ИУ7-65Б _____ (подпись, дата) Бугаенко А.П. (Фамилия И.О.)

Преподователь _____ (подпись, дата) Толпинская Н.Б. (Фамилия И.О.)

Москва, 2022

1 Цели и задачи работы

Цель работы — приобрести навыки работы с управляющими структурами Lisp. Задачи работы —изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

2 Теоретические вопросы

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. 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).

2.2 Трактовка элементов списка

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

2.3 Порядок реализации программы

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

2.4 Способы определения функции

Определение именованной функции - синтаксис:
(defun имя список_аргументов лямбда-выражение)

Определение неименованной функции - синтаксис:
(lambda список_аргументов лямбда-выражение)
(lambda $(x_1, ..., x_k)$ форма)

3 Практические задания

3.1 Задание 1

Чем принципиально отличаются функции cons, list, append?
Принципиальное отличие cons list и append состоит в способе работы со списками. cons и list создают новые списки, изменяя ссылки, в то время как append работает с копией списка, что позволяет не разрушать структуру списка, поданного на вход.

Пусть: (setf lst1 '(a b))
(setf lst2 '(c d))

Каковы результаты вычисления следующих выражений?
(cons lstl lst2) → ((a b) c d)
(list lst1 lst2) → ((a b) (c d))
(append lst1 lst2) → (a b c d)

3.2 Задание 2

Каковы результаты вычисления следующих выражений, и почему?
(reverse ()) → nil
(last ()) → nil
(reverse '(a)) → (a)
(last '(a)) → (a)
(reverse '((a b c))) → ((a b c))
(last '((a b c))) → ((a b c))

3.3 Задание 3

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

```
1 (defun last_elem (x)
2   (if (cdr x)
3       (last_elem (cdr x))
4       (car x)))
```

```
1 (defun last_elem (x)
2   (car (reverse x)))
```

3.4 Задание 4

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

```
(defun remove_last (lst)
  (and lst (nreverse (cdr (reverse lst)))))
```

```
(defun delete_last_rec (lst acc)
                (if (not (eql (cdr lst) nil))
                    (delete_last_rec (cdr lst) (cons (car lst) acc))
                    (reverse acc)))

(defun delete_last (lst)
  (delete_last_rec lst nil))
```

3.5 Задание 5

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 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.

```
(defun throw_bones ()
  (if  (setf *random-state* (make-random-state t))
       (list (random 7) (random 7))
       ))

(defun sum_score (x)
  (+ (car x) (cadr x)))

(defun player_1_throw (scores)
  (if (setf score (throw_bones))
      (and (setf scores
                 (list
                  (sum_score score)
                  (cadr scores)))
           (or (format t  "~&~&Player 1 throws~&results:~&~A ~A~&scores:~&Player 1:
              ~A~&Player 2: ~A~&" (car score) (cadr score) (car scores) (cadr
              scores)) t)
           (if (or (eql (car scores) 11)
                   (eql (car scores) 7))
               (format t  "~&>Player 2 wins<")
               (if (or
                    (and
```

```
21                      (eql (car score) 1)
22                      (eql (cadr score) 1)
23                      )
24                     (and
25                      (eql (car score) 6)
26                      (eql (cadr score) 6)))
27                    (or (format t  "~&Player 1 can get another try ~&Throw again?
                          [y/n] ")
28                        (if (eql (read) 'y)
29                            (player_1_throw scores)
30                            (player_2_throw scores)))
31                    (player_2_throw scores)
32                    ))
33            )))
34
35 (defun player_2_throw (scores)
36   (if (setf score (throw_bones))
37       (and (setf scores
38                  (list
39                   (car scores)
40                   (sum_score score)
41                   ))
42            (or (format t  "~&~&Player 2 throws~&results:~&~A ~A~&scores:~&Player 1:
                   ~A~&Player 2: ~A~&" (car score) (cadr score) (car scores) (cadr
                   scores)) t)
43            (if (or (eql (cadr scores) 11)
44                    (eql (cadr scores) 7))
45                (format t  "~&>Player 2 wins<")
46                (if (or
47                     (and
48                      (eql (car score) 1)
49                      (eql (cadr score) 1)
50                      )
51                     (and
52                      (eql (car score) 6)
53                      (eql (cadr score) 6)))
54                    (or (format t  "~&Player 2 can get another try ~&Throw again?
                          [y/n] ")
55                        (if (eql (read) 'y)
56                            (player_2_throw scores)
57                            (player_2_throw scores)))
58                    (if (>= (car scores) (cadr scores))
59                        (if (eql (car scores) (cadr scores))
60                            (format t  "~&Draw")
61                            (format t  "~&Player 1 wins"))
62                        (format t  "~&Player 2 wins"))
63                    ))
```

```
            )))

(defun play ()
  (player_1_throw '(0 0)))
```