Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 7

Дисциплина	Функциональное и логическое программирование.
Тема	Работа с функционалом и рекурсией
Студент	Степанов А. О.
Группа	ИУ7-63Б
Оценка (баллы)	
Преподаватель	Толпинская Н.Б.

Москва, 2020 г.

ЗАДАНИЕ 1

Чем принципиально отличаются функции cons, lisp, append?

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

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

Функция cons создает списковую ячейку и кладет в голову первый аргумент, а в хвост – второй. Функция list создает списковые ячейки для каждого аргумента и соединяет их в один список. А функция append объединяет два списка в один, состоящий из элементов двух списков.

ЗАДАНИЕ 2

Каковы результаты вычисления следующих выражений?

```
(reverse ())          ;;; Nil
(last ())             ;;; Nil
(reverse '(a))        ;;; (A)
(last '(a))           ;;; (A)
(reverse '((a b c))) ;;; ((A B C))
(last '((a b c)))     ;;; ((A B C))
```

ЗАДАНИЕ 3

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

```
(defun last_red (lst)
    (if (null lst)
        Nil
        (reduce #'(lambda (_ el) el) lst)
    )
)

(last_red ()) ;;; Nil
(last_red '(1 2 3 4 5)) ;;; 5

(defun last_rec (lst)
    (cond
        ((null lst) Nil)
```

```
14          ((eql (length lst) 1) (car lst))
15          (T (last_rec (cdr lst)))
16      )
17  )
18
19  (last_rec ()) ;;; Nil
20  (last_rec '(1 2 3 4 5)) ;;; 5
```

ЗАДАНИЕ 4

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

```
1  (defun list_without_last_map (lst)
2      (mapcar #'(lambda (el _) el) lst (cdr lst))
3  )
4
5  (list_without_last_map ()) ;;; Nil
6  (list_without_last_map '(1 2 3 4 5)) ;;; (1 2 3 4)
7
8  (defun list_without_last_rec (lst)
9      (cond
10         ((null lst) Nil)
11         ((eql (length lst) 1) Nil)
12         (T (cons (car lst) (list_without_last_rec (cdr lst))))
13     )
14 )
15
16 (list_without_last_rec ()) ;;; Nil
17 (list_without_last_rec '(1 2 3 4 5)) ;;; (1 2 3 4)
```

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 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.

```
1  (defun roll_the_dice ()
2      (cons
```

```
3           (+ (random 6) 1)
4           (+ (random 6) 1)
5       )
6   )
7
8   (defun turn (n)
9       (let
10          (
11              (roll (roll_the_dice))
12          )
13          (and
14              (if (eql n 1)
15                  (print "Игрок 1 бросил:")
16                  (print "Игрок 2 бросил:")
17              )
18              (print roll)
19              (if (or
20                      (and (eql (car roll) 1) (eql (cdr roll) 1))
21                      (and (eql (car roll) 6) (eql (cdr roll) 6))
22                  )
23                  (cons roll (turn n))
24                  (list roll)
25              )
26          )
27      )
28  )
29
30  (defun sum(res)
31      (cond
32          ((null res) 0)
33          (T
34              (let*
35                  (
36                      (el1 (caar res))
37                      (el2 (cdar res))
38                      (plus (+ el1 el2))
39                  )
40                  (if (or (eql plus 7) (eql plus 11))
41                      '100000000
42                      (+ plus (sum (cdr res)))
43                  )
44              )
```

```
        )
    )
)

(defun main ()
    (let
        (
            (res1 (sum (turn 1)))
            (res2 (sum (turn 2)))
        )
        (cond
            ((eql res1 res2) (print "Ничья"))
            ((> res1 res2) (print "Выйграл игрок 1"))
            (T (print "Выйграл игрок 2"))
        )
    )
)

(main)
```

```
"Игрок 1 бросил:"
(1 . 4)
"Игрок 2 бросил:"
(1 . 4)
"Ничья"
---

"Игрок 1 бросил:"
(4 . 4)
"Игрок 2 бросил:"
(4 . 6)
"Выйграл игрок 2"
---

"Игрок 1 бросил:"
(3 . 2)
"Игрок 2 бросил:"
(2 . 2)
"Выйграл игрок 1"
---

"Игрок 1 бросил:"
(1 . 1)
```

```
24 "Игрок 1 бросил:"
25 (4 . 6)
26 "Игрок 2 бросил:"
27 (2 . 3)
28 "Выйграл игрок 1"
29 "Выйграл игрок 1"
```