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»

Студент Рунов К.А.

Группа ИУ7-64Б

Оценка (баллы)

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

2024 г.

```
; 1. Чем принципиально отличаются функции cons, list, append?
; Пусть
(setf lst1 '(a b c))
(setf lst2 '(d e))
; Каковы результаты вычисления следующих выражений
(cons lst1 lst2) ; ((A B C) D E)
(list lst1 lst2) ; ((A B C) (D E))
(append lst1 lst2) ; (A B C D E)
```

```
; 2. Каковы результаты вычисления следующих выражений и почему?
(reverse '(a b c)) ; (C B A)
; (reverse l) возвращает список элементов l в обратном порядке, не
    разрушая структуру l
(reverse ()) ; NIL
(reverse '(a b (c (d)))) ; ((C (D)) B A)
; Встроенные функции lisp работают только со списковыми ячейками в
   ерхнего уровня
(reverse '((a b c))) ; ((A B C))
(reverse '(a)) ; (A)
(last '(a b c)) ; (C)
; (last l) возвращает последнюю списковую ячейку l
(last '(a)) ; (A)
(last '((a b c))) ; ((A B C))
; ((a b c)) - список, состоящий из одной списковой ячейки, у котор
   ой car-указатель указывает на список (a b c), a cdr-указатель
   - на nil
(last '(a b (c))) ; ((C))
; (a b (c)) - список, состоящий из трех списковых ячеек;
   car-указатель последней указывает на список (C)
(last ()) ; NIL
```

```
; 3. Написать, по крайней мере, два варианта функции, которая возв
   ращает последний элемент своего списка-аргумента
(defun easy (lst) (car (last lst)))

(defun hard (lst) ; FIXME (cdr lst) DRY
  (cond ((and (consp lst) (not (atom (cdr lst))) (consp (cdr
     lst)))
         (hard (cdr lst))) ; cdr until we can't
        ((null (cdr lst))
```

```
8           (car lst))
9         (t lst))) ; lst is dotted pair
```

```
1 ; 4. Написать, по крайней мере, два варианта функции, которая возв
    ращает свой список аргументов без последнего элемента
2 (defun f4v1 (lst)
3   (cond ((and (not (null lst)) (atom (cdr lst)))
4          nil)
5         (t (cons (car lst) (f4v1 (cdr lst))))))
6
7 (defun f4v2 (lst)
8   (cond ((and (not (not (not (null lst)))) (atom (cdr lst)))
9          nil)
10        (t (cons (car lst) (f4v2 (cdr lst))))))
```

```
1 ; 5. Напишите функцию swap-first-last, которая переставляет в спис
    ке-аргументе первый и последний элементы
2 (defun swap-first-last (lst)
3   (let ((tmp (cons (car lst) (car (last lst)))))
4     (setf (car lst) (cdr tmp))
5     (setf (car (last lst)) (car tmp)))
6   l)
```

```
1 ; 6. Написать простой вариант игры в кости, в котором бросаются дв
    е 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 второ
    му игроку, но запоминается сумма выпавших очков. Если второй иг
    рок не выигрывает абсолютно, то выигрывает тот игрок, у которог
    о больше очков. Результат игры и значения выпавших костей вывод
    ить на экран с помощью функции print.
2 (defun throw-one-dice ()
3   (+ 1 (random 6)))
4
5 (defun throw-two-dice ()
6   (cons (throw-one-dice) (throw-one-dice)))
7
8 (defun is-rethrow-possible (dice)
9   (cond ((and (equal (car dice) (cdr dice))
10              (or (equal (car dice) 1) (equal (car dice) 6)))
11          t)
12        (t nil)))
```

```
13
14 (defun get-dice-sum (dice)
15   (+ (car dice) (cdr dice)))
16
17 (defun is-absolute-win (dice)
18   (let ((sum (get-dice-sum dice)))
19     (cond ((or (equal sum 7) (equal sum 11)) t)
20           (t nil))))
21
22 (defun play-dice-rethrow (pr)
23   (let ((nr (cons nil nil))
24         (ans nil))
25     (setf nr (throw-two-dice))
26     (cond ((is-rethrow-possible nr)
27            ((lambda ()
28               (princ "Результат - ")
29               (princ nr)
30               (terpri)
31               (princ "Хотите перебросить? [y/n] > ")
32               (setf ans (read))
33               (cond ((or (eql ans 'y) (eql ans 'yes))
34                      (play-dice-rethrow pr))
35                     (t nr))
36               )))
37           (t nr))))
38
39 (defun play-dice ()
40   (let ((fpr (cons nil nil))
41         (spr (cons nil nil)))
42     (princ "Ход игрока 1.")
43     (terpri)
44     (setf fpr (play-dice-rethrow fpr))
45     (cond ((is-absolute-win fpr)
46            ((lambda ()
47                 (princ fpr)
48                 (princ " - ")
49                 (princ "Игрок 1 выигрывает абслютно.")
50                 (terpri)
51                 fpr
52             )))
53           (t ((lambda ()
```

```
54                    (princ "Результат игрока 1 - ")
55                    (princ fpr)
56                    (terpri)
57                    (princ "Ход игрока 2.")
58                    (terpri)
59                    (setf spr (play-dice-rethrow spr))
60                    (cond ((is-absolute-win spr)
61                           ((lambda ()
62                                (princ spr)
63                                (princ " - ")
64                                (princ "Игрок 2 выигрывает абслютно
                                     .")
65                                (terpri)
66                                (princ "Счет игрока 1 - ")
67                                (princ fpr)
68                                  (princ " | ")
69                                  (princ (get-dice-sum fpr))
70                                  (terpri)
71                                  (princ "Счет игрока 2 - ")
72                                  (princ spr)
73                                  (princ " | ")
74                                  (princ (get-dice-sum spr))
75                                (terpri)
76                                spr
77                            )))
78                          (t ((lambda ()
79                                  (princ "Счет игрока 1 - ")
80                                  (princ fpr)
81                                  (princ " | ")
82                                  (princ (get-dice-sum fpr))
83                                  (terpri)
84                                  (princ "Счет игрока 2 - ")
85                                  (princ spr)
86                                  (princ " | ")
87                                  (princ (get-dice-sum spr))
88                                  (terpri)
89                                  (cond ((< (get-dice-sum fpr)
                                         (get-dice-sum spr))
90                                             ((lambda ()
91                                                  (princ "Игрок 2 побежд
                                                       ает.")
```

```
                                        spr
                                    )))
                                ((> (get-dice-sum fpr)
                                    (get-dice-sum spr))
                                 ((lambda ()
                                        (princ "Игрок 1 побежд
                                            ает.")
                                        fpr
                                    )))
                                (t ((lambda ()
                                        (princ "Ничья.")
                                        fpr
                                    )))
                            ))))
                ))))))))
```

```
; 7. Написать функцию, которая по своему списку-аргументу lst опре
   деляет, является ли он палиндромом (то есть равны ли lst и
   (reverse lst))
(defun listeq (lst1 lst2)
  (cond ((and (null lst1) (null lst2)) t)
        ((eql (car lst1) (car lst2))
         (listeq (cdr lst1) (cdr lst2)))
        (t nil)))

(defun is-palindrome (lst)
  (listeq lst (reverse lst)))
```

```
; 8. Напишите свои необходимые функции, которые обрабатывают табли
   цу из 4-х точечных пар: (страна . столица), и возвращают по стр
   ане - столицу, а по столице - страну.
(defun insert (key value table)
  (setf table (acons key value table)))

(defun get-value (key table)
  (cond ((null table) nil)
        ((eql (car (car table)) key)
         (cdr (car table)))
        (t (get-value key (cdr table)))))

(defun get-key (value table)
  (cond ((null table) nil)
```

```
        ((eql (cdr (car table)) value)
         (car (car table)))
        (t (get-key value (cdr table)))))
```

```
; 9. Напишите функцию, которая умножает на заданное число
   -аргумент первый числовой элемент списка из заданного 3-х элеме
   нтного списка-аргумента, когда а) все элементы списка - числа,
   б) элементы списка - любые объекты.
(defun strange-rec (x lst3 count)
  (let ((head (car lst3)))
    (cond ((numberp head)
           (setf (car lst3) (* x head)))
          ((> count 1)
           (strange-rec x (cdr lst3) (- count 1))))))

(defun strange (x lst3)
  ((lambda ()
    (strange-rec x lst3 3)
    lst3)))
```