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»

на тему: «Использование управляющих структур, работа со списками»

Студент ИУ7-61Б _____ Д. В. Шубенина
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Н. Б. Толпинская
(Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Ю. В. Строганов
(Подпись, дата) (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

1 Практические задания

1.1 Задание 1

Чем принципиально отличаются функции `cons`, `list`, `append`?

1) `cons` — базисная функция, принимающая 2 аргумента и объединяющая их значения в точечную пару;

2) `list` — функция, принимающая произвольное количество аргументов и возвращающая список из их значений;

3) `append` — функция, принимающая произвольное количество аргументов, производящая копирование всех переданных аргументов, кроме последнего, и возвращающая список, содержащий все переданные в качестве аргументов элементы.

Пусть:

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

Каковы результаты вычисления следующих выражений?

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

1.2 Задание 2

Каковы результаты вычисления следующих выражений и почему?

1) `(reverse '(a b c)) -> (C B A)`
`reverse` возвращает копию списка, содержащую элементы исходного списка в обратном порядке;

2) `(reverse '(a b (c (d)))) -> ((C (D)) B A)`
`reverse` работает только со списковыми ячейками верхнего уровня;

3) `(reverse '(a)) -> (A)`

4) `(reverse ()) -> NIL`

5) `(reverse '((a b c))) -> ((A B C))`
   `reverse` работает только со списковыми ячейками верхнего уровня, `(A B C)` — единственный элемент списка на этом уровне;

6) `(last '(a b c)) -> (C)`
   `last` возвращает последний элемент списка;

7) `(last '(a)) -> (A)`

8) `(last '((a b c))) -> ((A B C))`
   `last` работает только со списковыми ячейками верхнего уровня, `(A B C)` — единственный элемент списка на этом уровне;

9) `(last '(a b (c))) -> (C)`
   `last` работает только со списковыми ячейками верхнего уровня, `(C)` — последний элемент списка на этом уровне;

10) `(last ()) -> NIL`

1.3 Задание 3

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

```
(defun get-last-1 (lst)
    (nth (- (length lst) 1) lst))

(defun get-last-2 (lst)
    (cond ((null (cdr lst)) (car lst))
          (T (get-last-2 (cdr lst)))))

(defun get-last-3 (lst)
    (car (last lst)))

;; (get-last-1 '(a b (c))) -> (C)
;; (get-last-2 '(a b (c))) -> (C)
;; (get-last-3 '(a b (c))) -> (C)
```

1.4 Задание 4

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

```
(defun without-last-1 (lst)
    (reverse (cdr (reverse lst))))

(defun without-last-2 (lst)
    (cond ((null (cdr lst)) Nil)
          (T (cons (car lst) (without-last-2 (cdr lst))))))

;; (without-last-1 '(a b c)) -> (A B)
;; (without-last-2 '(a b c)) -> (A B)
```

1.5 Задание 5

Напишите функцию swap-first-last, которая переставляет в списке-аргументе первый и последний элементы.

```
(defun without-last (lst)
    (cond ((null (cdr lst)) Nil)
          (T (cons (car lst) (without-last (cdr lst))))))

(defun swap-first-last (lst)
    (setf head (without-last lst))
    (append (cons (nth (- (length lst) 1) lst) Nil)
            (append (cdr head)
                    (cons (nth 0 head) Nil))))

;; (swap-first-last '(a b c d)) -> (D B C A)
```

1.6 Задание 6

Написать простой вариант игры в кости, в котором бросается 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. Результаты игры и значения выпавших костей выводить на экран с помощью `print`.

```
(defun roll-dice ()
    (setf *random-state* (make-random-state t))
    (+ (random 6) 1))

```

```
(defun is-abs-win (result)
    (or (= result 7)
        (= result 11)))

(defun should-reroll (d1 d2)
    (and (= d1 d2)
         (or (= d1 6)
             (= d1 1))))

(defun make-a-move (iplayer)
    (let (
             (dice1 (roll-dice))
             (dice2 (roll-dice))
         )

    (print (list 'Игрок iplayer 'бросает 'кости dice1 dice2))
    (cond (
             (should-reroll dice1 dice2)
             (and
                 (print (list 'Игрок iplayer 'перебрасывает
                    'кости))
                 (make-a-move iplayer)
             )
          )
          (T
             (and
                 (print (list 'Выпало dice1 dice2 'очков))
                 (+ dice1 dice2)
             )
          ))
    ))

(defun play-game ()
    (let (
             (p1 (make-a-move 1))
             (p2 (make-a-move 2))
         )
    (cond (
             (is-abs-win p1)
             (print (list 'Игрок 1 'выиграл 'абсолютно))
          )
```

```
45              (
46                  (is-abs-win p2)
47                  (print (list 'Игрок 2 'выиграл 'абсолютно))
48              )
49              (
50                  (> p1 p2)
51                  (print (list 'Игрок 1 'выиграл))
52              )
53              (
54                  (< p1 p2)
55                  (print (list 'Игрок 2 'выиграл))
56              )
57              (
58                  (= p1 p2)
59                  (print 'Ничья)
60              )
61      )))
```

1.7 Задание 7

Написать функцию, которая по своему списку-аргументу `lst` определяет является ли он палиндромом.

```
1 (defun is-palindrome (lst)
2     (equal lst (reverse lst)))
```

1.8 Задание 8

Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна . столица), и возвращает по стране — столицу, а по столице — страну.

```
1  (defun country-by-capital (table capital)
2      (cond (
3                  (eq capital (cdr (car table)))
4                  (car (car table))
5              )
6              (
7                  T
8                  (country-by-capital (cdr table) capital)
9              )
10     ))
```

```
11
12 (defun capital-by-country (table country)
13     (cond (
14                 (eq country (car (car table)))
15                 (cdr (car table))
16           )
17           (
18                 T
19                 (capital-by-country (cdr table) country)
20           )
21     ))
```

1.9 Задание 9

Написать функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка-аргумента, когда

— все элементы списка — числа;

— элементы списка — любые объекты.

```
1  (defun mul-by-num (num lst)
2      (cond (
3                 (null lst)
4                 lst
5           )
6           (
7                 (numberp (car lst))
8                 (cons (* num (car lst)) (cdr lst))
9           )
10          (
11                T
12                (cons (car lst)
13                      (mul-by-num num (cdr lst)))
14          )
15     ))
16
17 ;; (mul-by-num 8 '(1 2 3)) -> (8 2 3)
18 ;; (mul-by-num 8 '(a 2 3)) -> (a 16 3)
19 ;; (mul-by-num 8 '(a 2 a)) -> (a 16 a)
20 ;; (mul-by-num 8 Nil) -> NIL
```