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 с со списками»

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) Руденко М. А. (И. О. Фамилия)

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

2024 г.

1 Практическое задание

1. Чем принципиально отличаются функции cons, list, append?

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

Каковы результаты вычисления следующих выражений?

```
(cons lstl lst2)
(list lst1 lst2)
(append lst1 lst2)
```

- cons объединяет значения своих аргументов в точечную пару. Если вторым аргументом будет передан список, то в результате получится список, в котором первый аргумент будет добавлен в начало: ((A B)C D)
- list составляет из своих аргументов список: ((A B) (C D))
- append создает копию всех аргументов, кроме последнего, т. е списковые ячейки. Связываются последними указателями. Результирующее значение: (A B C D)

2. Каковы результаты вычисления следующих выражений, и почему?

```
(reverse '(a b c))          --> (C B A)
(reverse '(a b (c (d))))    --> ((C (D)) B A)
(reverse '(a))              --> (A)
(last '(a b c))             --> (C)
(last '(a))                 --> (A)
(last '((a b c)))           --> ((a b c))
(reverse ())                --> Nil
(reverse '((a b c)))        --> ((A B C))
(last '(a b (c)))           --> ((c))
(last ())                   --> Nil
```

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

```
1 (defun return-last1(list1)
2 (car (reverse list1)))
3
4 (defun return-last2(list1)
5 (car (last list1)))
```

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

```
(defun without-last(list)
(reverse (cdr (reverse list))))

(defun without-last2(list)
(remove (car (last list)) list))
```

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

```
(defun swap-first-last (lst)
            (
                nconc
                (last lst)
                (reverse
                    (cdr
                        (reverse (cdr lst)))
                )
                (list (car lst))
            )
        )
```

6. 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.

```
(defvar first_player_list)
(defvar second_player_list)

(defun bones_throw ()

            (print "Enter first bone: ")
            (setq bone1 (read))
            (print "Enter second bone: ")
            (setq bone2 (read))
            (setq ret (list bone1 bone2))
```

```
            ret
        )

(defun check_absolute_win(list)
    (
        or (= (+ (first list) (last list) 7))
        (= (+ (first list) (last list) 11))
    ))

(defun check_rerun(list)
    (
        or (= (first list) (last list) 1)
        (= (first list) (last list) 6)
    ))


(defun check_not_absolute_win()
    (
        cond (
                (> (+ (first first_player_list) (last
                   first_player_list))
                    (+ (first second_player_list) (last
                       second_player_list)))
                (print "First player wins"))
            (
                (< (+ (first first_player_list) (last
                   first_player_list))
                    (+ (first second_player_list) (last
                       second_player_list)))
                (print "Second player wins"))
            (T (print "Draw in the game"))
    ))

(defun second_palyer_turn ()
    (
        (print "second player throws bones: ")
        (setq second_palyer_list (bones_throw))
        (print first_player_list)
```

```
        (cond ((check_absolute_win first_player_list) (print
           "Second player wins"))
        ((check_rerun(first_player_list)) (second_player_turn))
        (t check_not_absolute_win))
    ))

(defun first_player_turn ()
    (
        (print "first player throws bones: ")
        (setq first_palyer_list (bones_throw))
        (print first_player_list)

        (cond ((check_absolute_win first_player_list) (print
           "First player wins"))
        ((check_rerun(first_player_list)) (first_player_turn))
        (t (second_player_turn)))
    ))

(first_player_turn)
```

7. Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst и (reverse lst)).

```
(defun poly(list)
    (
        equal list (reverse list)
    ))
```

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

```
(defun countries_capitals (lst name)
    (cond ((assoc name lst) (cdr (assoc name lst)))
        ((rassoc name lst) (car (rassoc name lst)))
        (T Nil)))
```

9. Напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка-аргумента, когда а) все элементы списка — числа, б) элементы списка – любые объекты.

```
(defun mult_a (n lst)
    (cond ((and (numberp (car lst)) (numberp (cadr lst))
            (numberp (caddr lst)) (numberp n))
        (* (car lst) n))
        (T Nil)))

(defun mult_b (n lst)
    (cond ((numberp (car lst)) (* (car lst) n))
            ((numberp (cadr lst)) (* (cadr lst) n))
            ((numberp (caddr lst)) (* (caddr lst) n))))
```