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 (ИУ7)**

ОТЧЕТ

по лабораторной работе № 6

Название: Использование управляющих структур, работа со списками.

Дисциплина: Функциональное и логическое программирование

Студент ИУ7-63Б _____ В.П. Федоров
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2021

Задача 1.

Чем принципиально отличие функции cons, list, append?
Пусть (setf lst1 '(a b))(setf lst2 '(c d))
Каковы результаты вычисления следующих выражений?

выражение	**результат**
(cons lst1 lst2)	((lst1) lst2)
(list lst1 lst2)	((lst1) (lst2))
(append lst1 lst2)	(lst1 lst2)

cons всегда берет 2 аргумента и помещает первый в начало второго.
list берет 1 и больше аргументов и образует список, помещая аргументы в скобки.
append образует новый список, убирает скобки вокруг аргументов и помещает их в список.

Задача 2.

выражение	**результат**
(reverse ())	NIL
(last ())	NIL
(reverse '(a))	(A)
(last '(a))	(A)
(reverse '((a b c)))	((A B C))
(last '((a b c)))	((A B C))

Задача 3.

Листинг 3.1: Функция получает последний элемент списка

```
; изменяется структура ?
(defun get-last(lst)
        (last lst))
```

Листинг 3.2: Функция получает последний элемент списка

```
(defun get-last(lst)
        (car (reverse lst)))
```

Задача 4.

Листинг 4.1: Функция возвращает список без последнего элемента

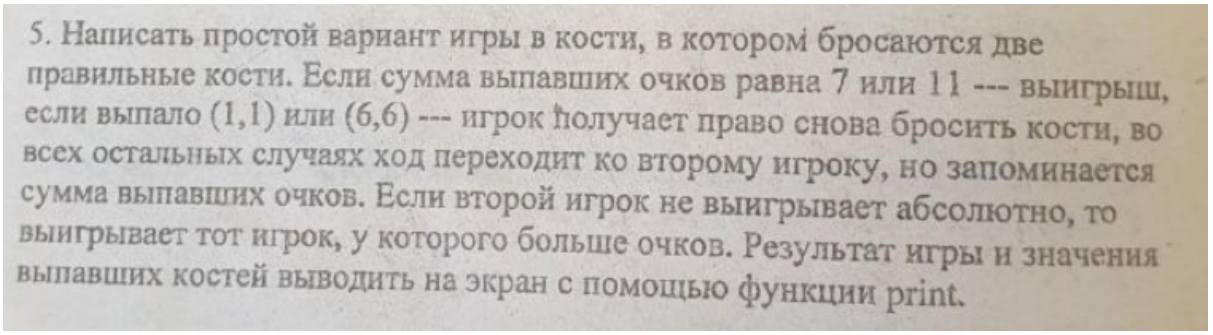
```
(defun get-list-without-last(lst)
        (reverse (cdr (reverse lst))))
```

Листинг 4.1: Функция возвращает список без последнего элемента (Вариант 2)

```
(defun get-lst-without-last(lst)
        (cond
                ((null (cdr lst)) nil)
                (t (cons (car lst) (get-lst-without-last (cdr lst))))))
```

```
* (GET-LST-WITHOUT-LAST '(1 2 3 4))
(1 2 3)
* (GET-LST-WITHOUT-LAST '())
NIL
* (GET-LST-WITHOUT-LAST '(1))
NIL
* (GET-LST-WITHOUT-LAST '(-3 d 3 e))
(-3 D 3)
```

Задача 5.



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.

```
(defun game(lst)
	(cond
		((is-first-win-7-11 lst) (format t "~$ ~$" 'first-win lst))
		((is-second-win-by-7-11 lst) (format t "~$ ~$" 'second-win lst))
		((is-first-win-by-sum lst) (format t "~$ ~$" 'first-win lst))
		((is-second-win-by-sum lst) (format t "~$ ~$" 'second-win lst))
		((is-draw lst) (game (append (roll-dices) (roll-dices))))
		((first-get-double lst) (game (append (roll-dices) (list (nth '2 lst) (nth '3 lst)))))
		((second-get-double lst) (game (append (list (nth '0 lst) (nth '1 lst)) (roll-dices))))))

(defun roll-dices()
	(list (+ 1 (random 6)) (+ 1 (random 6))))

(defun get-first-player-sum(lst)
	(+ (car (subseq lst 0 2)) (car (cdr (subseq lst 0 2)))))

(defun get-second-player-sum(lst)
	(+ (car (subseq lst 2)) (car (cdr (subseq lst 2)))))

(defun is-draw(lst)
	(if (eq (get-first-player-sum lst) (get-second-player-sum lst)) t))



(defun is-first-win-7-11(lst)
	(if (OR (eq (get-first-player-sum lst) 7) (eq (get-first-player-sum lst) 11))
		t))

(defun is-second-win-by-7-11(lst)
	(if (OR (eq (get-second-player-sum lst) 7) (eq (get-second-player-sum lst) 11))
		t))

(defun first-get-double(lst)
	(if (OR (first-get-double-1 lst) (first-get-double-6 lst))
			t))
```

```
(defun first-get-double-1(lst)
        (if (AND (eq (car lst) 1) (eq (nth 1 lst) 1)) t))

(defun first-get-double-6(lst)
                (if (AND (eq (car lst) 6) (eq (nth 1 lst) 6)) t))

(defun second-get-double(lst)
        (if (OR (second-get-double-1 lst) (second-get-double-6 lst))
                        t))

(defun second-get-double-1(lst)
        (if (AND (eq (nth 2 lst) 1) (eq (nth 3 lst) 1)) t))

(defun second-get-double-6(lst)
        (if (AND (eq (nth 2 lst) 6) (eq (nth 3 lst) 6)) t))

(defun is-first-win-by-sum(lst)
        (if (> (get-first-player-sum lst) (get-second-player-sum lst))
                        t))

(defun is-second-win-by-sum(lst)
        (if (> (get-second-player-sum lst) (get-first-player-sum lst))
                        t))


(game (0 0 0 0))
```

```
* (game '(0 0 0 0))
FIRST-WIN (3 2 1 1)
NIL
* (game '(0 0 0 0))
SECOND-WIN (5 4 6 1)
NIL
* (game '(0 0 0 0))
FIRST-WIN (6 3 1 3)
NIL
* (game '(0 0 0 0))
SECOND-WIN (2 3 2 4)
NIL
* (game '(0 0 0 0))
SECOND-WIN (6 6 6 1)
NIL
* (game '(0 0 0 0))
FIRST-WIN (6 1 2 1)
```

```
NIL
* (game '(0 0 0 0))
SECOND-WIN (5 3 2 5)
NIL
* (game '(0 0 0 0))
FIRST-WIN (5 3 1 5)



* (game '(0 0 0 0))
  0: (GAME (0 0 0 0))
    1: (GAME (5 1 1 5))
      2: (GAME (6 1 4 5))
FIRST-WIN (6 1 4 5)      2: GAME returned NIL
    1: GAME returned NIL
  0: GAME returned NIL
```

Теоретические вопросы.

1. *структуроразрушающие и не разрушающие структуру списка функции.*

Функции в Лисп могут по разному взаимодействовать с передаваемыми им списками. Функции могут создать копию переданного списка и продолжить работу с ним, сохраняя исходный список не измененным. А могут изменять исходный список.

Первый вариант используется в случае, если исходный список может потребоваться нам в дальнейшем. Второй вариант используется, если первоначальная структура списка нам больше не потребуется.

Имена функций разрушающих структуру начинаются с буквы n. Например reverse и nreverse. Это нужно, чтобы отличать структуроразрушющие функции от не разрушающих.

reverse - создаст и возвратит новую структуру, при этом не изменит исходную структуру.
nreverse - будет изменять исходную структуру.

2. *отличие в работе функций cons, lisp, append и в их результате.*

cons всегда берет 2 аргумента и помещает первый в начало второго.
list берет 1 и больше аргументов и образует список, помещая аргументы в скобки.
append образует новый список, убирает скобки вокруг аргументов и помещает их в список.