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»*

**Дисциплина: Функциональное и логическое программирование**

**Лабораторная работа №7**

Выполнил:

Левушкин Илья

ИУ7-62Б

Москва, 2020

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

**Пусть (setf fest! '( a b)) (sets Ist2 '( d)).**

**Каковы результаты вычисления следующих выражений?**

(cons Istl Ist2) = ((A B) C D)

(list lst! Ist2) = ((A B) (C D))

(append Ist1 Ist2) = (A B C D)

**CONS**

Это основной конструктор списка. Эта функция берет элемент (<новый первый элемент>) и <список>, и возвращает список с добавленным элементом в начало списка.

Например:

(cons 'a '(b c d)) возвращает (A B C D)

(cons '(a) '(b c d)) возвращает ((A) B C D)

Заметим, что первый элемент может быть атомом или списком.

**LIST**

Эта функция берет любое число выражений (<выражение>) и организует из них строку, возвращая список.

Например:

(list 'a 'b 'c) возвращает (A B C)

(list 'a '(b c) 'd) возвращает (A (B C) D)

(list 3.9 6.7) возвращает (3.90000 6.70000)

**APPEND**

Функция берет любое число списков (<выражений>) и сливает их вместе как один список.

(append '(a b) '(c d)) возвращает (A B C D)

(append '((a) (b)) '((c) (d))) возвращает ((A) (B) (C) (D))

APPEND требует, чтобы аргументы были списками.

**2. Каковы результаты вычисления следующих выражений?**

(reverse Q) = Nil

(last ()) = Nil

(reverse '(a)) = (a)

(last '(a)) = (a)

(reverse '((a b c))) = (a b c)

(last '((a b c))) = (a b c)

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

(defun f3\_last (lst)

(car (last lst))

)

(defun f3\_reverse (lst)

(car (reverse lst))

)

(defun f3\_recursion (lst)

(cond ((Null (cdr lst)) lst)

(T (f3\_recursion (cdr lst)))

)

)

Пример:

1. (f3\_last ‘(a b c)) = C
2. (f3\_last ‘(a b (c))) = (C)
3. (f3\_last ‘((a b c))) = (A B C)

Пример:

1. (f3\_reverse ‘(a b c)) = C
2. (f3\_reverse ‘(a b (c))) = (C)
3. (f3\_reverse ‘((a b c))) = (A B C)

Пример:

1. (f3\_recursion ‘(a b c)) = C
2. (f3\_recursion ‘(a b (c))) = (C)
3. (f3\_recursion ‘((a b c))) = (A B C)

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

(defun f4\_reverse (lst)

(reverse (cdr (reverse lst)))

)

(defun f4\_recursion (lst)

(cond ((Null (cdr lst)) Nil)

(T (cons (car lst) (f4\_recursion (cdr lst))))

)

)

Пример:

1. (f4\_reverse ‘(1 2 3)) = (1 2)
2. (f4\_reverse ‘(1 2 3 (4)) = (1 2 3)

Пример:

1. (f4\_recursion ‘(1 2 3)) = (1 2)
2. (f4\_recursion ‘(1 2 3 (4)) = (1 2 3)

**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 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 play ()

(let ((flip (list (+ (random 6) 1) (+ (random 6) 1) ) ))

(cond ((and (equal (car flip) 1) (equal (cadr flip) 1)) (cons flip (play)))

((and (equal (car flip) 6) (equal (cadr flip) 6)) (cons flip (play)))

(T (cons flip Nil))

)

)

)

(defun get\_last\_sum (player)

(+ (caar (last player)) (cadar (last player)))

)

(defun game ()

(let\* ((player1 (play) )

(player1\_last\_sum (get\_last\_sum player1))

)

(cond ( (or (equal player1\_last\_sum 7) (equal player1\_last\_sum 11))

`(,(print 'Winner1)

,(print `(Player1 ,player1))

)

)

(T (let\* ((player2 (play ) )

(player2\_last\_sum (get\_last\_sum player2))

)

(cond ((or (equal player2\_last\_sum 7) (equal player2\_last\_sum 11))

`(,(print 'Winner2)

,(print `(Player1 ,player1))

,(print `(Player2 ,player2))

)

)

(T (cond ((equal player1\_last\_sum player2\_last\_sum)

`(

,(print 'Draw)

,(print `(Player1 ,player1))

,(print `(Player2 ,player2))

)

)

((< player1\_last\_sum player2\_last\_sum)

`(

,(print 'Winner2)

,(print `(Player1 ,player1))

,(print `(Player2 ,player2))

)

)

(T

`(

,(print 'Winner1)

,(print `(Player1 ,player1))

,(print `(Player2 ,player2))

)

)

)

)

)

)

)

)

)

)