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»

Лабораторная работа №4 по дисциплине "Функциональное и логическое программирование"

Тема Использование управляющих структур, работа со списками

Студент Светличная А.А.

Группа ИУ7-63Б

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

Москва — 2023 г.

1 Теоритические вопросы

1.1 Синтаксическая форма и хранение программы в памяти

В Lisp и программа, и данные представлены S-выражениями, благодаря чему программа может обрабатывать и преобразовывать другие программы или саму себя. S-выражение — атом или точечная пара. Атом представляется в памяти 5-ю указателями: имя, значение, функция, свойство, пакет. Точечная пара представляется в памяти списковой ячейкой: бинарный узел, состоящий из двух указателей (car-указатель и cdr-указатель).

1.2 Трактовка элементов списка

По умолчанию список является формой (вычислимым выражением), в которой первый элемент трактуется как имя фунции, остальные элементы — как ее аргументы. Для возможности различия программы от данных создана функция quote и ее сокращенное обозначение — апостроф '. Функция quote и апостроф блокируют вычисление своего аргумента и возращают его текстовую запись.

1.3 Порядок реализации программы

Программа в языке Lisp представляется S-выражением, которое передается интерпретатору – функции eval, которая выводит последний, полученный после обработки S-выражения результат.

Работа функции eval представлена на рисунке 1.1.

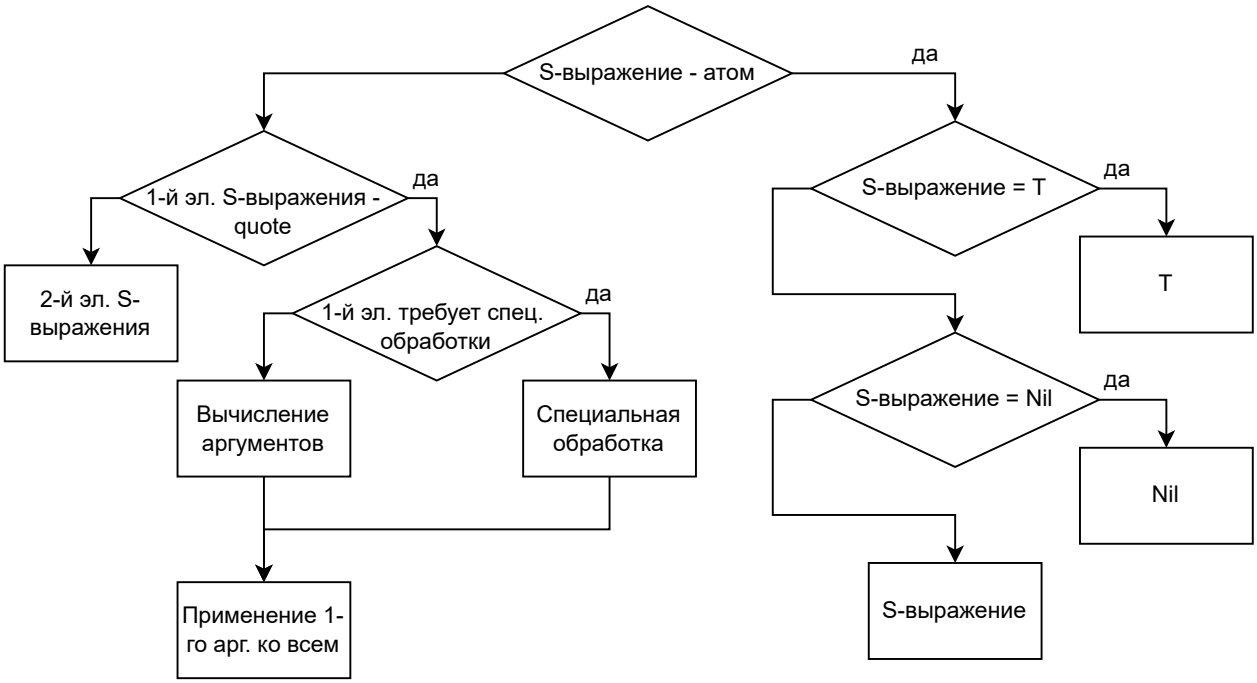

Рисунок 1.1 – Схема работы функции eval

1.4 Способы создания функций

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

- В Lisp можно определить функцию без имени с помощью λ**-выражений**. Lambda-определение безымянной функции:

  ```
  (lambda <lambda-список> <форма>)
  ```

  Lambda-вызов функции:

  ```
  (<lambda-выражение> <формальные параметры>)
  ```

- Также в Lisp можно определить функцию с именем с помощью **defun**. В таких функциях defun связывает символьный атом с Lambda-определением

  ```
  (defun f <lambda-выражение>)
  ```

Упрощенное определение:

```
(defun f(arg1, ..., argN) <формы>)
```

1.5 Работа со списками

Структуроразрушающие функции — функции, после использования которых теряется возможность работы изначальными списками. Эти функции изменяют сам объект. Чаще всего такие функции начинаются с префикса `n` (например, `nconc`, `nreverse` и т. п.).

Функции, не разрушающие структуру списка — функции, после использования которых сохраняется возможность работы с изначальными списками. Эти функции не изменяют сам объект, а создают копии (например, `append`, `reverse`, `length` и т. п.).

2 Практические задания

2.1 Задание №1

Чем принципиально отличаются функции `cons, list, append`? Каковы результаты вычисления следующих выражений?

Листинг 2.1 – Выполнение задания №1

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

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

2.2 Задание №2

Каковы результаты вычисления следующих выражений, и почему?

Листинг 2.2 – Выполнение задания №2

```
(reverse '(a b c))          ;; (C B A)
(reverse ())                ;; NIL
(reverse '(a b (c (d))))    ;; ((C (D)) B A)
(reverse '((a b c)))        ;; ((A B C))
(reverse '(a))              ;; (A)
(last '(a b c))             ;; (C)
(last '(a b (c)))           ;; ((C))
(last '(a))                 ;; (A)
(last ())                   ;; NIL
(last '((a b c)))           ;; ((A B C))
```

2.3 Задание №3

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

Листинг 2.3 – Выполнение задания №3

```
(defun my-last(lst)
    (car (last lst)))

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

2.4 Задание №4

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

Листинг 2.4 – Выполнение задания №4

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

(defun without-last(lst)
    (remove (car (last lst)) lst)) :test #'equal
```

2.5 Задание №5

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

Листинг 2.5 – Выполнение задания №5

```
(defun swap-first-last(lst)
    (setf tmp (car (last lst)))
    (rplaca (last lst) (car lst))
    (rplaca lst tmp)
)
```

2.6 Задание №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.

Листинг 2.6 – Выполнение задания №6

```
(defvar name-first  'first_player)
(defvar name-second 'second_player)

(defvar dice-first)
(defvar dice-second)
(defvar tmp-dice)

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

(defun sum (dice)
  (+ (car dice) (cadr dice)))

(defun is-win (dice)
  (cond ((= (sum dice) 7 ))
        ((= (sum dice) 11))
    )
)

(defun repeat-roll (dice)
  (cond ((= (car dice) (cadr dice) 6))
        ((= (car dice) (cadr dice) 1))
    )
)

(defun print-res (name dice)
    (print '(Win ,name)))

```

```
29 (defun players-move (name)
30     (setf tmp-dice (roll-dice)
31     )
32     (print `(,name ,tmp-dice sum = ,(sum tmp-dice))
33     )
34     (cond
35         ((is-win tmp-dice)
36             (list tmp-dice 1))
37         ((repeat-roll tmp-dice)
38             (players-move name))
39         (T (list tmp-dice 0))
40     )
41 )
42
43 (defun continue-game (dice-first)
44     (setf dice-second (players-move name-second)
45     )
46     (cond
47         ((= (cadr dice-second) 1)
48             (print-res name-second dice-second))
49         ((> (sum (car dice-first)) (sum (car dice-second)))
50             (print-res name-first dice-first))
51         ((< (sum (car dice-first)) (sum (car dice-second)))
52             (print-res name-second dice-second))
53         (T (print '(Draw)))
54     )
55 )
56
57 (defun play ()
58     (setf dice-first  (players-move name-first)
59     )
60     (cond
61         ((= (cadr dice-first) 1)
62             (print-res name-first dice-first))
63         (T (continue-game dice-first))
64     )
65     (terpri) (terpri)
66 )
```

2.7 Задание №7

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

Листинг 2.7 – Условие задания №7

```
(defun palindrome (lst)
    (if (equal lst
               (reverse lst))
        T
        NIL)
)
```

Предикат №2 неверен, так как проверка на число должна быть перед проверкой на положительность числа.

2.8 Задание №8

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

Листинг 2.8 – Выполнение задания №8

```
(defun get-capital (table country)
    (cond ((null table) Nil)
          ((equal (caar table) country) (cdar table))
          (T (get-capital (cdr table) country))
    )
)

(defun get-country (table capital)
    (cond ((null table) Nil)
          ((equal (cadar table) capital) (caar table))
          (T (get-country (cdr table) capital))
    )
)
```

2.9 Задание №9

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

Листинг 2.9 – Выполнение задания №9

```
(defun mult-first-num-arg (lst num)
    (setf (car lst) (* (car lst) num))
    lst
)


(defun mult-first-num-arg (lst num)
    (cond
        ((null lst) NIL)
        ((numberp (car lst))
            (setf (car lst) (* (car lst) num)))
        (T (mult-first-num-arg (cdr lst) num))
    )
    lst
)
```