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

О т ч е т

по лабораторной работе № 5

Дисциплина: «Функциональное и логическое программирование»

Выполнила: Овчинникова А.П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н.Б.

Москва, 2020

Теоретическая часть.

Функция *(rem r1 r2)* возвращает остаток от деления *r1* на *r2* (второй результат вызова *truncate* с теми же аргументами).

Условный оператор if принимает три аргумента: *test-, then-* и *else-*выражение. Сначала вычисляется тестовое *test*-выражение. Если оно истинно, вычисляется *then*-выражение («то») и возвращается его значение. В противном случае вычисляется *else*-выражение («иначе»).

(if условие (t_body) (f_body))

Как и *quote, if* – это специальный оператор, а не функция, так как для функции вычисляются все аргументы, а у оператора *if* вычисляется лишь одно из двух последних выражений. Указывать последний аргумент *if* необязательно. Если он пропущен, то автоматически принимается за Nil.

Логические операторы *and* (и) и *or* (или) оба могут принимать любое количество аргументов, но вычисляют их до тех пор, пока не будет ясно, какое значение необходимо вернуть. Если все аргументы истинны, то оператор *and* вернет значение последнего. Но если один из аргументов окажется ложным, то следующие за ним аргументы не будут вычислены и будет возвращено nil. Так же действует и *or*, вычисляя значения аргументов до тех пор, пока среди них не найдется хотя бы одно истинное значение. В этом случае возвращается само значение, в противном случае – nil.

Функция *(equal object1 object2)* возвращает истину, если *object1* и *object2* равны с точки зрения *eql*, либо являются cons-ячейками, чьи *car* и *cdr* эквивалентны с точки зрения *equal*; либо являются строками или бит-векторами одной длины, чьи элементы эквиваленты с точки зрения *eql*. То есть *equal* сравнивает как *eql*, а также может корректно сравнивать списки.

Функция *(eql object1 object2)* возвращает истину, если *object1* и *object2* равны с точки зрения *eq* или являются одним и тем же знаком или числами, выглядящими одинаково при печати.

Функция *(eq object1 object2)* возвращает истину, если *object1* и *object2* идентичны. Выполняет проверку атомарных объектов на равенство.

Специальная форма

(cond (условие1) (результат1)

...

(условиеN) (результатN))

является некоторым аналогом case из других языков программирования. Она принимает на вход n пар условие-результат. Сначала она просматривает все условия в порядке следования, и если хоть одно из них истинно, то возвращается результат, связанный с этим условием. Если с условием не связан какой-либо результат, возвращается значение этого условия. Если ни одно условие не оказалось истинным, то возвращается Nil.

Атомы:

- символы (идентификаторы) – синтаксически это набор литер (букв и цифр), начинающихся с буквы;
- специальные символы T и Nil;
- самоопределимые атомы – натуральные числа, вещественные числа, строки – последовательности символов, заключенные в двойные кавычки.

На рисунке 1 показано, что в памяти атомы представлены структурой из пяти указателей: имя, пакет, значение связанной с ним переменной, значение связанной функции и список свойств.

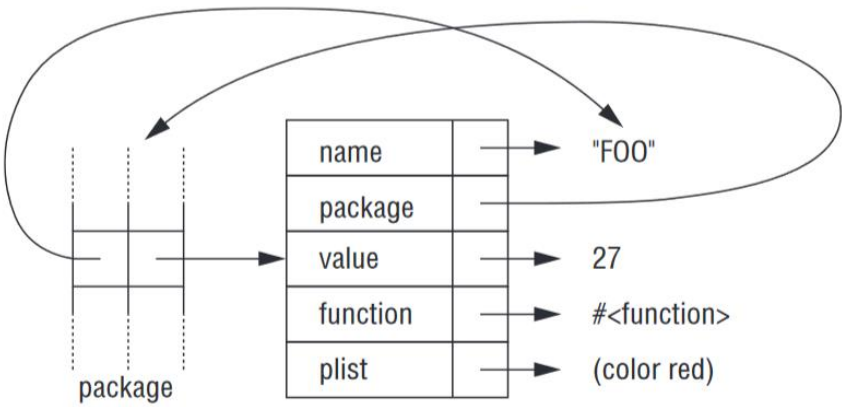

Рис. 1. Атом.

Логически пакеты – это таблицы, отображающие имена в символы. Любой символ принадлежит конкретному пакету. Символ, принадлежащий пакету, называют интернированным в него. Пакеты делают возможной модульность, ограничивая область видимости символов. Большие программы часто разделяют на несколько пакетов.

Большинство символов интернируются во время считывания. Когда вы вводите символ впервые, Лисп создаёт новый символьный объект и интернирует в текущий пакет (по умолчанию это common-lisp-user).

Не все символы являются интернированными.

Практическая часть.

Задание 1.

Написать функцию, которая принимает целое число и возвращает первое четное число, не меньшее аргумента.

```
(defun even-num (x)
(if (= (rem (+ x 1) 2) 1) (+ x 2) (+ x 1)))
```

Задание 2.

Написать функцию, которая принимает число и возвращает число того же знака, но с модулем на 1 больше модуля аргумента.

```
(defun mod_more (x)
(if (< x 0) (- x 1) (+ x 1)))
```

Задание 3.

Написать функцию, которая принимает два числа и возвращает список из этих чисел, расположенный по возрастанию.

```
(defun list2 (a b)
(if (< a b) (cons a (cons b ())) (cons b (cons a ()))))
```

Задание 4.

Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим.

```
(defun mycompare (a b c)
```

(if (or (and (<= a c) (>= a b)) (and (<= a b) (>= a c))) T Nil))

Задание 5.

(and ‘fee ‘fie ‘foe) = FOE

(or ‘fee ‘fie ‘foe) = FEE

(and (equal ‘abc ‘abc) ‘yes) = (and T ‘yes) = YES

(or nil ‘fie ‘foe) = FIE

(and nil ‘fie ‘foe) = NIL

(or (equal ‘abc ‘abc) ‘yes) = (or T ‘yes) = T

Задание 6.

Написать предикат, который принимает два числа-аргумента и возвращает T, если первое число не меньше второго.

(defun isGreater (a b)

(>= a b))

Задание 7.

Предикат

(defun pred2 (x)

(and (plusp x)(numberp x)))

ошибочен, потому что:

- предикат *(numberp object)* возвращает T, если объект object является числом;
- предикат *(plusp r)* возвращает истину, если r больше нуля;
- предикат *(plusp r)* работает только с числами;
- предикат *(plusp r)* вызывается до того, как с помощью предиката *(numberp object)* будет произведена проверка, является ли x числом, поэтому в случае, когда предикату *pred2* в качестве аргумента передается не число, возникает ошибка.

Задание 8.

Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим, используя COND, IF, AND/OR.

```
(defun mycompare3 (a b c)
(if (<= a c) (if (>= a b) T Nil) (if (<= a b) (if (>= a c) T Nil) Nil)))

(defun mycompare2 (a b c)
(or (and (<= a c) (>= a b)) (and (<= a b) (>= a c))))



(defun mycompare4 (a b c)
   (cond
      ((>= a b)
         (cond
            ((>= c a) T)))
      ((<= a b)
         (cond
            ((<= c a) T))))
)
```