



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»

Отчёт

по лабораторной работе №3

Название: Работа интерпретатора lisp.

Дисциплина: Функциональное и логическое программирование

Студент ИУ7-64Б
(Группа)

Л.Е.Тартыков
(Подпись, дата) (И.О. Фамилия)

Преподаватель

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

Преподаватель

Ю.В.Строганов
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

1 Теоретические вопросы

1.1 Базис Lisp

Базис - минимальный набор конструкций языка и структур данных, с помощью которых можно решить любую задачу.

Базис Lisp образуют: атомы, структуры, базовые функции, базовые функционалы.

Базисные функции – минимальный набор функций, которые позволяют решить любую задачу.

1.2 Классификация функций

1. Чистые математические функции.

Имеют фиксированное число аргументов и возвращают один результат. Сначала вычисляются все аргументы, затем к ним применяется исходная функция.

2. Специальные функции (формы).

Специальные функции – функции, у которых переменное число аргументов или они обрабатываются по-разному (один вычисляется, другой – нет).

3. Псевдофункции.

Псевдофункции – функции, которые создают "спецэффекты"; например, вывод на экран.

4. Функции с вариантами значений – выбирают какое-то одно значение.

5. Функционалы.

Функционалы – функции, которые в качестве аргументов используют функции или возвращают в качестве результата функцию. Также они называются *функциями более высокого порядка*. Позволяют создавать

синтаксически управляемые программы (программы, которые сами создают какие-то функции; эти функции затем выполняются).

6. Рекурсивные.

1.3 Способы создания функций

1. *lambda*-выражение. Данный способ представлен с помощью формулы (1.1).

$$(lambda \lambda\text{-список форма}), \quad (1.1)$$

где λ -список – список формальных параметров, форма – тело функции. *lambda*-выражение не хранится в памяти и не имеет имени. Вычисляется сразу же. Используется для повторных вычислений.

Вызов *lambda*-функции выполняется по формуле (1.2).

$$(\lambda\text{-выражение последовательность форм}) \quad (1.2)$$

2. С помощью *defun* по формуле (1.3).

$$(defun f \lambda\text{-выражение}) \quad (1.3)$$

Система по имени символьного атома находит его определение.

1.4 Работа функциями *cond*, *if*, and/or

1. Функция **cond** – средство разветвления вычислений. Вызов функции *cond* представлен по формуле (1.4).

$$(cond (p_1 e_1) (p_2 e_2) \dots (p_n e_n)[(T \text{ else-}e)]), \quad n \geq 1 \quad (1.4)$$

Обращение к функции `cond` называется **условным выражением**. Выражения (p_i, e_i) – **ветви** условного выражения; выражения-формы p_i – **условия** ветвей.

Если ни одно из условий не вернуло Т, то можно организовать ветку «else», явно указав в качестве условия Т.

Порядок вычисления условного выражения:

- (a) Последовательно вычисляются условия ветвей до тех пор, пока не встретится выражение-форма p_i , значение которой отлично от Nil.
- (b) Вычисляются выражения e_i соответствующей ветви и его значение возвращается в качестве значения функции `cond`.
- (c) Если все условия p_i имеют значение Nil, то значением условного выражения становится Nil.

2. Функция **if**. Вызов функции `if` представлен по формуле (1.5).

$$(if\ test_clause\ action_t\ action_f) \quad (1.5)$$

В случае, если условие `test` истинно, то вычисляется $action_t$, иначе – $action_f$; это говорит о том, что *if* является формой.

3. Логическая функция **and**.

Вызов функции `and` представлен по формуле (1.6).

$$(and\ e_1\ e_2\ \dots\ e_n), n \geq 0 \quad (1.6)$$

При работе функции `and` последовательно слева-направо вычисляются аргументы функции e_i – до тех пор, пока не встретится значение, равное nil – вычисление прерывается и значение функции равно nil. Если же были вычислены все значения e_i и оказалось, что все они отличны от nil, то результирующим значением функции `and` будет значение последнего выражения e_n .

При $n = 0$ значение функции `and` равно Т. Значением функции `and` может быть не только Т и nil, но и произвольный атом или список.

4. Логическая функция **or**.

Вызов функции **and** представлен по формуле (1.7).

$$(or\ e_1\ e_2\ \dots\ e_n), n \geq 0 \quad (1.7)$$

При выполнении вызова последовательно вычисляются аргументы e_i (слева-направо) – до тех пор, пока не встретится значение e_i , отличное от **nil**. В этом случае вычисление прерывается и значение функции равно значению e_i . Если же вычислены значения всех аргументов и они равны **nil**, то результирующее значение функции равно **nil**.

При $n = 0$ значение функции **or** равно **nil**. Значением функции **or** может быть не только **T** и **nil**, но и произвольный атом или список.

2 Практические задания

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

Листинг 2.1 – Задание 1

```
1 (defun first_even (number)
2   (if (evenp number) number (+ number 1)))
```

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

Листинг 2.2 – Задание 2

```
1 (defun abs_more (number)
2   (cond ((>= number 0) (+ number 1))
3         (T (- (+ (abs number) 1)))))
```

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

Листинг 2.3 – Задание 3

```
1 (defun inc_list (number_1 number_2)
2   (if (<= number_1 number_2) (list number_1 number_2)
3     (list number_2 number_1)))
```

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

Листинг 2.4 – Задание 4

```
1 (defun is_middle (number_1 number_2 number_3)
2   (cond ((and (> number_1 number_2) (< number_1 number_3)) T)
3         (T Nil)))
```

2.5 Каков результат вычисления следующих выражений?

1. $(\text{and } 'fee \ 'fie \ 'foe) \Rightarrow foe$
2. $(\text{or nil } 'fie \ 'foe) \Rightarrow fie$
3. $(\text{and } (\text{equal } 'abc \ 'abc) \ 'yes) \Rightarrow yes$
4. $(\text{or } 'fee \ 'fie \ 'foe) \Rightarrow fee$
5. $(\text{and nil } 'fie \ 'foe) \Rightarrow nil$
6. $(\text{or } (\text{equal } 'abc \ 'abc) \ 'yes) \Rightarrow T$

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

Листинг 2.5 – Задание 6

```
1 (defun is_bigger (number_1 number_2)
2   (if (>= number_1 number_2) T nil))
```

2.7 Какой из следующих двух вариантов предиката ошибочен и почему?

1. (defun pred1 (x) (and (numberp x) (plusp x)))
2. (defun pred2 (x) (and (plusp x) (numberp x)))

Вариант *pred2* является ошибочным ввиду того, что если в качестве аргумента будет не число, то функция *plusp* выдаст ошибку.

2.8 Решить задачу 4, используя для ее решения конструкции IF, COND, AND/OR.

Листинг 2.6 – Задание 8

```
1 (defun is_middle_cond (number_1 number_2 number_3)
2   (cond ((and (> number_1 number_2) (< number_1 number_3)) T)
3         (T Nil)))
4
5 (defun is_middle_if (number_1 number_2 number_3)
6   (if (and (> number_1 number_2) (< number_1 number_3)) T nil))
7
8 (defun is_middle_andor (number_1 number_2 number_3)
9   (and (> number_1 number_2) (< number_1 number_3)))
```


2.9 Переписать функцию how-alike, приведенную в лекции и использующую COND, используя только конструкции IF, AND/OR.

Листинг 2.7 – Задание 9

```
1 (defun how_alike_cond(x y)
2   (cond ((or (= x y) (equal x y)) 'the_same)
3         ((and (oddp x) (oddp y)) 'both_odd)
4         ((and (evenp x) (evenp y)) 'both_even)
5         (t 'difference)))
6
7 (defun how_alike_if_andor(x y)
8   (if (or (= x y) (equal x y)) 'the_same
9       (if (and (oddp x) (oddp y)) 'both_odd
10          (if (and (evenp x) (evenp y)) 'both_even 'difference))))
11
12 (defun how_alike_if (x y)
13   (if (if (= x y) (equal x y)) 'the_same
14       (if (if (oddp x) (oddp y)) 'both_odd
15           (if (if (evenp x) (evenp y)) 'both_even 'difference))))
16
17 (defun how_alike_andor (x y)
18   (or (and (= x y) (equal x y) 'the_same)
19       (and (oddp x) (oddp y) 'both_odd)
20       (and (evenp x) (evenp y) 'both_even)
21       'difference))
```