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		Л.Е.Тартыков
	(Группа)	(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 в виде формулы (1.1).

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

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).

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

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

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

Система по имени символьного атома находит его определение.

1.4 Работа функций cond, if, and/or

1. Функция **cond** – средство разветвления вычислений. Вызов функции cond представлен в виде формулы (1.4).

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

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 T, то можно организовать ветку «else», явно указав в качестве условия T.

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 в виде формулы (1.5).

$$(if\ test\text{-}clause\ action\_t\ action\_f) \tag{1.5}$$

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 в виде формулы (1.6).

$$(and\ e_1\ e_2\ ...\ e_n), n \geq 0 \tag{1.6}$$

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 T. Значением функции and может быть не только T и nil, но и произвольный атом или список.

4. Логическая функция **or**.

   Вызов функции and представлен в виде формулы (1.7).

$$(or\ e_1\ e_2\ ...\ e_n), n \geq 0 \tag{1.7}$$

   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 T только тогда, когда первое число расположено между вторым и третьим.

Листинг 2.4 – Задание 4

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

2.5 Каков результат вычисления следующих выражений?

1. (and 'fee 'fie 'foe) ⇒ foe
2. (or nil 'fie 'foe) ⇒ fie
3. (and (equal 'abc 'abc) 'yes) ⇒ yes
4. (or 'fee 'fie 'foe) ⇒ fee
5. (and nil 'fie 'foe) ⇒ nil
6. (or (equal 'abc 'abc) 'yes) ⇒ T

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

Листинг 2.5 – Задание 6

```
(defun is_bigger (number_1 number_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

```
(defun is_middle_cond (number_1 number_2 number_3)
    (cond ((and (> number_1 number_2) (< number_1 number_3)) T)
          (T Nil)))

(defun is_middle_if (number_1 number_2 number_3)
    (if (and (> number_1 number_2) (< number_1 number_3)) T nil))

(defun is_middle_andor (number_1 number_2 number_3)
    (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

```
(defun how_alike_cond(x y)
    (cond ((or (= x y) (equal x y)) 'the_same)
          ((and (oddp x) (oddp y)) 'both_odd)
          ((and (evenp x) (evenp y)) 'both_even)
          (t 'difference)))

(defun how_alike_if_andor(x y)
    (if (or (= x y) (equal x y)) 'the_same
        (if (and (oddp x) (oddp y)) 'both_odd
            (if (and (evenp x) (evenp y)) 'both_even 'difference))))

(defun how_alike_if (x y)
    (if (if (= x y) (equal x y)) 'the_same
        (if (if (oddp x) (oddp y)) 'both_odd
            (if (if (evenp x) (evenp y)) 'both_even 'difference))))

(defun how_alike_andor (x y)
    (or (and (= x y) (equal x y) 'the_same)
        (and (oddp x) (oddp y) 'both_odd)
        (and (evenp x) (evenp y) 'both_even)
        'difference))
```