



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 по курсу «Функциональное и логическое программирование»

Тема Работа интерпретатора Lisp

Студент Динь Вьет Ань

Группа ИУ7И-64Б

Оценка (баллы)

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

Теоретические вопросы

1. Базис Lisp

Базис – это минимальный набор инструментов языка и структур данных, который позволяет решить любые задачи.

Базис Lisp :

- атомы и структуры (представляющиеся бинарными узлами);
- базовые (несколько) функций и функционалов: встроенные — примитивные функции (atom, eq, cons, car, cdr); специальные функции и функционалы (quote, cond, lambda, eval, apply, funcall).

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

Функционалом, или функцией высшего порядка называется функция, аргументом или результатом которой является другая функция.

2. Классификация функций

Функции в языке Lisp:

- Чистые математические функции (имеют фиксированное количество аргументов, сначала выясняются все аргументы, а только потом к ним применяется функция);
- Рекурсивные функции (основной способ выполнения повторных вычислений);
- Специальные функции, или формы (могут принимать произвольное количество аргументов, или аргументы могут обрабатываться по-разному);
- Псевдофункции (создают «эффект», например, вывод на экран);
- Функции с вариантами значений, из которых выбирается одно;
- Функции высших порядков, или функционалы – функции, аргументом или результатом которых является другая функция (используются для построения синтаксически управляемых программ)

- Базисные функции — минимальный набор функций, позволяющих решить любую задачу.

Также базисные и функции ядра можно классифицировать с точки зрения действий.

1. Селекторы — переходят по соответствующему указателю списковой ячейки.
2. Конструкторы — создают структуры данных.
3. Предикаты — позволяют классифицировать или сравнивать структуры.

3. Способы создание функций

- С помощью `lambda`. После ключевого слова указывается лямбда-список и тело функции.

```
1 (lambda (x y) (+ x y))
```

Для применения используются лямбда-выражения.

```
1 ((lambda (x y) (+ x y)) 1 2)
```

- С помощью `defun`. Используется для неоднократного применения функции (в том числе рекурсивного вызова).

```
1 (defun sum (x y) (+ x y))
2 (sum 1 2)
```

4. Работа функций `Cond`, `if`, `and/or`

- Функция `cond`

Синтаксис:

```
1 (cond
2   (condition-1 expression-1)
3   (condition-2 expression-2)
4   ...
5   (condition-n expression-n))
```

По порядку вычисляются и проверяются на равенство с `Nil` предикаты. Для первого предиката, который не равен `Nil`, вычисляется

находящееся с ним в списке выражение и возвращается его значение. Если все предикаты вернут `Nil`, то и `cond` вернет `Nil`.

Примеры:

```
1 (cond (Nil 1) (2 3)) -> 3
2 (cond (Nil 1) (Nil 2)) -> %\texttt{Nil}%
```

- Функция `if`

Синтаксис: `(if condition t-expression f-expression)`

Если вычисленный предикат не `Nil`, то выполняется `t-expression`, иначе - `f-expression`.

Примеры:

```
1 (if Nil 2 3) -> 3
2 (if 0 2 3) -> 2
```

- Функция `and`

Синтаксис: `(and expression-1 expression-2 ... expression-n)`

Функция возвращает первое `expression`, результат вычисления которого = `Nil`. Если все не `Nil`, то возвращается результат вычисления последнего выражения.

Примеры:

```
1 (and 1 Nil 2) -> %\texttt{Nil}%
2 (and 1 2 3) -> 3
```

- Функция `or`

Синтаксис: `(or expression-1 expression-2 ... expression-n)`

Функция возвращает первое `expression`, результат вычисления которого не `Nil`. Если все `Nil`, то возвращается `Nil`.

Примеры:

```
1 (or Nil Nil 2) -> 2
2 (or 1 2 3) -> 1
```

Практические задания

1. Задание 1

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

```
1 (defun f1 (x) (if (oddp x) (+ x 1) x))
2 (f1 0) => 0
3 (f1 1) => 2
4 (f1 -3) => -2
```

2. Задание 2

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

```
1 (defun f2 (x) (+ x (if (< x 0) -1 1)))
2 (f2 -5) => -6
3 (f2 2/3) => 5/3
```

3. Задание 3

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

```
1 (defun f3 (x1 x2) (if (> x1 x2) (list x2 x1) (list x1 x2)))
2 (f3 -1 2) => (-1 2)
3 (f3 3 1) => (1 3)
4 (f3 2 2.0) => (2 2.0)
```

4. Задание 4

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

```
1 (defun f4 (x1 x2 x3) (or (and (< x2 x1) (< x1 x3))
2                           (and (< x3 x1) (> x1 x2))))
3 (f4 2 1 3) => T
4 (f4 1 2 3) => NIL
5 (f4 3 1 2) => NIL
```

5. Задание 5

Каков результат вычисления следующих выражений?

```
1 (and 'fee 'fie 'foe) => FOE
2 (or 'fee 'fie 'foe) => FEE
3 (or nil 'fie 'foe) => FIE
4 (and nil 'fie 'foe) => NIL
5 (and (equal 'abc 'abc) 'yes) => YES
6 (or (equal 'abc 'abc) 'yes) => T
```

6. Задание 6

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

```
1 (defun gep (x1 x2) (>= x1 x2))
2 (gep 2 2.0) => T
3 (gep -1 2/3) => NIL
```

7. Задание 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)))
```

Предикат `numberp` вырабатывает Т, если значение его аргумента – числовой атом, и NIL в противном случае. `plusp` проверяет, является ли оди-ночное вещественное число большим чем ноль.

Таким образом, идея приведенных вариантов предиката заключается в том, чтобы проверить, является ли переданный ему аргумент числом, большим нуля.

При вычислении функционального обращения `(and e1 e2 ... en)` последовательно слева направо вычисляются аргументы функции `ei` – до тех пор, пока не встретится значение, равное NIL. В этом случае вычисление прерывается и значение функции равно NIL. Если же были вычислены все значения `ei` и оказалось, что все они отличны от NIL, то результирующим значением функции `and` будет значение последнего выражения `en`.

Таким образом, корректным является первый вариант предиката. Пер-вым будет вычислено значение аргумента `e1 = (numberp x)`, которое прове-рит, является ли переданный аргумент числовым атомом. Если это не так, то `e1 = (numberp x)` вернет Nil, на чем вычисление функции `and` прервется,

и результатом всего предиката `pred1` будет `NIL`. Если же переданный аргумент является числовым атомом, то следующим будет вычислено значение аргумента `e2 = (plusp x)`. Это выражение проверит, является ли переданный числовой атом (здесь мы уже уверены, что переданный аргумент – числовой атом) больше нуля, и станет результатом всего предиката `pred1`.

Второй вариант же является ошибочным. В нем первым будет вычислено значение аргумента `e1 = (plusp x)`, но `plusp` принимает только числовой атом, и если `x` не является числовым атомом, то вычисление всего предиката `pred2` завершится с ошибкой "неверный тип".

8. Задание 8

Решить задачу 4, используя для ее решения конструкции: только `IF`, только `COND`, только `AND/OR`.

`(and x y)` можно представить как `(cond (x y))`; `(or x y)` можно представить как `(cond (x) (y))`.

Все приведенные ниже функции на тестах из предыдущего номера выдают те же результаты.

```
1  ; if
2  (defun is_first_between (first second third)
3    (
4      if (> first second)
5        (
6          if (< first third)
7            t
8            nil
9        )
10     (
11       if (< first second)
12         (
13           if (> first third)
14             t
15             nil
16         )
17     )
18   )
19 )
20 )
```

```

1      ; cond
2      (defun is_first_between (first second third)
3      (
4          cond
5          (
6              (> first second)
7              (
8                  cond ((< first third) t)
9                  (T nil)
10             )
11          )
12
13          (
14              (< first second)
15              (
16                  cond ((> first third) t)
17                  (T nil)
18             )
19          )
20      ))
21      ; or
22      (defun is_first_between (first second third)
23      (
24          or
25          (and (> first second) (< first third))
26          (and (> first third) (< first second))
27      )
28      )

```

9. Задание 9

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

Исходная функция:

```

1      (defun how-alike (x y)
2          (cond
3              ((or (= x y) (equal x y)) 'the_same)
4              ((and (oddp x) (oddp y)) 'both_odd)
5              ((and (evenp x) (evenp y)) 'both_even)
6              (t 'diff))
7          )
8      )
9      (how-alike 1 1) => THE_SAME
10     (how-alike 2.5 2.5) => THE_SAME
11     (how-alike -2/3 -4/6) => THE_SAME
12     (how-alike 3 5) => BOTH_ODD
13     (how-alike -4 6) => BOTH_EVEN
14     (how-alike 1 2) => DIFF

```

(= работает только с числами, причем они могут быть различных типов; oddp и evenp работают только с целыми числами)

Переписанная функция:


```

1 ; if, and, or
2 (defun how-alike (x y)
3   (if (or (= x y) (equal x y)) 'the_same
4       (if (and (oddp x) (oddp y)) 'both_odd
5           (if (and (evenp x) (evenp y)) 'both_even
6               'diff)
7         )
8     )
9   )
10 )
11 ; and, or
12 (defun how-alike (x y)
13   (or
14     (and
15       (or (= x y) (equal x y)) 'the_same
16     )
17     (and
18       (and (oddp x) (oddp y)) 'both_odd
19     )
20     (and
21       (and (evenp x) (evenp y)) 'both_even
22     )
23     'diff
24   ))
25
26 ; if
27 (defun how-alike (x y)
28   (if
29     (if (= x y) T (equal x y))
30     'the_same
31     (if
32       (if (oddp x) (oddp y))
33       'both_odd
34       (if
35         (if (evenp x) (evenp y))
36         'both_even
37         'diff
38       )
39     )
40   )
41 )

```