Министерство образования и науки 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»

Отчет по лабораторной работе № 5

По курсу: "Функциональное и логическое программирование"

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-66Б

Преподователи:

Толпинская Наталья Борисовна

Строганов Юрий Владимирович

Москва, 2021 г.

Содержание

Цель работы: приобрести навыки работы с управляющими структурами Lisp.

Задачи работы: изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

Введение

Многие стандартные функции Lisp являются формами и реализуют особый способ работы со своими аргументами. К таким функциям относятся функции, позволяющие работать с произвольным количеством аргументов: and, or, append, или особым образом обрабатывающее свои аргументы: функции cond, if, append, remove, reverse, substitute.

Если на вход функции подается структура данных(списков), то возникает вопрос: сохранится ли возможность в дальнейшем работать с исходными струтурами, или они изменятся в процессе реализации функции. В Lisp существуют функции, использующие списки в качестве аргументов и разрушающие структуру исходных аргументов при этом часть из них позволяет использовать произвольное количество аргументов, а часть нет.

Задание 1

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

```
(defun get\_even (x)
    (cond ((oddp x)(+ x 1)) (x))
)
```

Примеры:

- (get_even 4) -> 4
- (get_even 7) -> 8

Задание 2

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

```
(defun inc\_abs (x)
    (cond ((plusp x)(+ x 1))((- x 1)))
)
```

Примеры:

- (inc_abs 3) -> 4
- (inc_abs -5) -> -6

Задание 3

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

```
(defun get\_asc (x y)
    (cond ((> x y)(list y x))((list x y)))
)
```

Примеры:

- (get_asc 3 5) -> (3 5)
- (get_asc 5 -1) -> (-1 5)

Задание 4

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

```
(defun between (a b c)
    (or (and (> a b) (< a c)) (and (< a b) (> a c)))
)
```

Примеры:

- (between 8 7 10) -> T
- (between 2 3 0) -> T
- (between 1 5 8) -> NIL

Задание 5

Каков результат вычисления следующих выражений?

1. (and 'fee 'fie 'foe)
   Результат: FOE
2. (or 'fee 'fie 'foe)
   Результат: FEE

3. (and (equal 'abc 'abc) 'yes)

   Результат: YES

4. (or nil 'fie 'foe)

   Результат: FIE

5. (and nil 'fie 'foe)

   Результат: NIL

6. (or (equal 'abc 'abc) 'yes)

   Результат: T

Задание 6

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

```
(defun is\_greater
    (x y) (>= x y)
)
```

Примеры:

- (is_greater 5 2) -> T
- (is_greater 7 20) -> NIL
- (is_greater 6 6) -> T

Задание 7

Какой из следующих двух вариантов предикатов ошибочен и почему?

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

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

Ошибочен второй предикат, так как в нем сначала вычисляется предикат plusp, который может вернуть ошибку в случае, если x не является числом, а первый предикат в этом случает вычислит предикат numberp, вернет NIL и предикат plusp вычислен не будет.

Задание 8

Решить задачу 4, используя для ее решения конструкции:

- COND

```
1        (defun between (a b c)
2            (cond ((cond ((> a b) (< a c)) ((< a b) (> a c)))))
3        )
4
```

- IF

```
1        (defun between (a b c)
2            (if (> a b) (< a c) (if (< a b) (> a c)))
3        )
4
```

- AND/OR

```
1        (defun between (a b c)
2            (or (and (> a b) (< a c)) (and (< a b) (> a c)))
3        )
4
```

Задание 9

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

```
(defun how-alike (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
            )
        )
    )
)
```

Пример:

```
> (how_alike 1 1)
THE_SAME
> (how_alike 1 6)
DIFFERENCE
> (how_alike 8 4)
BOTH_EVEN
> (how_alike 5 7)
BOTH_ODD
```

Ответы на вопросы:

1) Классификация функций:

1. базовые функции:
   (a) селекторы (car, cdr);
   (b) конструкторы(cons);
   (c) предикаты (atom, Null, lisp, ..);
   (d) функции сравнения (eq, eql, =, equal, equalp).
2. формы;

3. функционалы.

4. математические.

5. рекурсивные.

Списки представлены с помощью списковых ячеек. Списочная ячейка состоит из двух частей, полей car и cdr. Каждое из полей содержит указатель. Указатель может ссылаться на другую списочную ячейку или на некоторый другой Lisp объект, как, например, атом.

Указатели между ячейками образуют цепочку, по которой можно из предыдущей ячейки попасть в следующую и так, наконец, до атомарных объектов. Каждый известный системе атом записан в определённом месте памяти лишь один раз.

Если это не стоит блокировка вычисления(quote, '), то первый элемент трактуется как имя функции, остальные — как аргументы.

2) Работа функций and, or, if, cond:

Функция **cond** принимает произвольное количество аргументов. Каждый аргумент функции должен быть списком ровно из двух элементов. Первый из этих элементов условно называется условием, а второй - результатом. Принцип работы заключается в выполнении каждого условия из каждого списка условий до тех пор пока не встретися условие равное T. Если такого выражения нет, то COND вернет NIL.

Функция **if**:

Оператор if обозначает то же, что и конструкция cond. Сначала выполняется условие. Если результат не равен nil, тогда выбирается форма then. Иначе выбирается форма else. Выбранная ранее форма выполняется, и if возвращает то, что вернула это форма. if более читабельный. Форма else может быть опущена. В таком случае, если значение условия является nil, тогда ничего не будет выполнено и возвращаемое значение формы if будет nil.

Функция **and**:

Логическая функция AND берет один или несколько аргументов. Она выполняет эти аргументы слева направо. Если она встречает аргумент, значение которого NIL, она возвращает NIL, не продолжая вычисления остальных. Если NIL аргументов не встретилось, то возвращается значение последнего аргумента.

Функция **or**:

Логическая функция OR берет один или несколько аргументов. Она выполняет эти аргументы слева направо и возвращает значение первого аргумента, который не NIL.Если все аргументы OR имеют значение NIL, то OR возвращает NIL.

3) Способы определения функций

Новые функции можно определить с помощью оператора defun. Он принимает три или более аргументов: имя, список параметров и ноль или более выражений, которые составляют тело функции.

(defun func_name (arg1 arg2 ... argN) func_body)

Но функция не обязательно должна иметь имя, для того, чтобы определить функцию, не имеющую имени, необходимо воспользоваться лямбда-выражением. Лямбда-выражение – это список, содержащий символ lambda и следующие за ним список аргументов и тело, состоящее из нуля или более выражений.

(lambda (arg1 arg2 .. argN) func_body)