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 дисциплине «Функциональное и логическое программирование»

Тема: Работа интерпретатора Lisp
Студент: Княжев А. В.
Группа: ИУ7-62Б
Оценка (баллы): _____
Преподаватели: Толпинская Н. Б., Строганов Ю. В.

Москва — 2023 г.

Оглавление

1. Теоретическая часть

1.1. Базис языка Lisp

Базис — это минимальный набор правил/конструкций языка, к которым могут быть сведены все остальные. Базис языка Lisp представлен атомами, структурами, базовыми функциями, базовыми функционалами. Некоторые базисные функции: `car`, `cdr`, `cons`, `eq`.

1.2. Классификация функций языка Lisp

Функции в Lisp бывают базисными, пользовательскими и функциями ядра. Также функции можно разделить на:

— чистые — не создающие побочных эффектов, принимающие фиксированное число аргументов, не получающие данные неявно, результат работы которых не зависит от внешних переменных;

— особые, или формы;

— функции более высоких порядков, или функционалы — функции, результатом и/или аргументом которых является функция.

1.3. Способы создания функций в языке Lisp

Функцию можно определить двумя способами: неименованную с помощью `lambda` и именованную с помощью `defun`.

```
(lambda (x_1 x_2 ... x_n) f)
```

— `f` — тело функции;

— `x_i` — формальные параметры.

```
(defun <имя> [lambda] (x_1 x_2 ... x_n) f)
```

— `f` — тело функции;

— `x_i` — формальные параметры.

Тогда имя будет ссылкой на описание функции.

1.4. Функции `cond`, `if`, `and`, `or`

Функции `cond`, `if`, `and`, `or` являются основными условными функциями в Lisp.

1.4.1. `cond`

Форма `cond` содержит некоторое (возможно нулевое) количество подвыражений, которые является списками форм. Каждое подвыражение содержит форму условия и ноль и более форм для выполнения. Например:

```
(cond (test-1 consequent-1-1 consequent-1-2 ...)
      (test-2)
      (test-3 consequent-3-1 ...)
      ... )
```

`cond` обрабатывает свои подвыражения слева направо. Для каждого подвыражения, вычисляется форма условия. Если результат `nil`, `cond` переходит к следующему подвыражению. Если результат `t`, `cdr` подвыражения обрабатывается, как список форм. После выполнения списка форм, `cond` возвращает управление без обработки оставшихся подвыражений. Оператор `cond` возвращает результат выполнения последней формы из списка. Если этот список пустой, тогда возвращается значение формы условия. Если `cond` вернула управление без вычисления какой-либо ветки (все условные формы вычислялись в `nil`), возвращается значение `nil`.

1.4.2. `if`

Оператор `if` обозначает то же, что и конструкция if-then-else в большинстве других языках программирования. Сначала выполняется форма `test`. 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 test then else)
```

1.4.3. `and`

`and` последовательно слева направо вычисляет формы. Если какая-либо форма `formN` вычислилась в `nil`, тогда немедленно возвращается значение `nil` без выполнения оставшихся форм. Если все формы кроме последней вычисляются в не-`nil` значение, and возвращает то, что вернула последняя форма. Таким образом, and может использоваться, как для логических операций, где `nil` обозначает ложь и не-`nil` значения истину, так и для условных выражений.

```
(and form1 form2 ... )
```

1.4.4. `or`

`or` последовательно выполняет каждую форму слева направо. Если какая-либо непоследняя форма выполняется в что-либо отличное от `nil`, or немедленно возвращает это не-`nil` значение без выполнения оставшихся форм. Если все формы кроме последней, вычисляются в `nil`, `or` возвращает то, что вернула последняя форма. Таким образом `or` может быть использована как для логических операций, в который `nil` обозначает ложь, и не-`nil` истину, так и для условного выполнения форм.

2. Практическая часть

2.1. Задание 1

Задание

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

Решение

```
(defun f1 (x) (+ x (mod x 2)))
```

```
(defun f1 (x) (if (oddp x) (+ x 1) x))
```

2.2. Задание 2

Задание

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

Решение

```
(defun f2 (x)
        (cond
                ((equal x 0) 0)
                ((< x 0) (- x 1))
                ((> x 0) (+ x 1))
        )
)
```

2.3. Задание 3

Задание

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

Решение

```
(defun f3 (x y)
        (if (< x y)
                (list x y)
                (list y x)
        )
)
```

2.4. Задание 4

Задание

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

Решение

```
(defun f4 (x y z)
        (if (or (and (< y x) (< x z)) (and (< z x) (< x y)))
                t
                nil
        )
)
```

2.5. Задание 5

Задание

Каков результат вычисления следующих выражений?

1. `(and 'fee 'fie 'foe)`
2. `(or nil 'fie 'foe)`
3. `(and (equal 'abc 'abc) 'yes)`
4. `(or 'fee 'fie 'foe)`
5. `(and nil 'fie 'foe)`
6. `(or (equal 'abc 'abc) 'yes)`

Решение

1. `FOE`
2. `FIE`
3. `YES`
4. `FEE`
5. `NIL`
6. `T`

2.6. Задание 6

Задание

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

Решение

```
(defun f5 (x y) (>= x y))
```

2.7. Задание 7

Задание

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)))`

Решение

1. предикат работает на любых входных данных, если передать строку, то `and` вернет `nil` на первой же проверке;
2. предикат не работает на данных, отличных от числовых, так как сначала идет проверка знака, потом уже проверка на то, что аргумент является числом, таким образом, если ввести строку, то произойдет ошибка на проверке знака.

2.8. Задание 8

Задание

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

Решение

`IF`

```
(defun f6 (x y z)
        (if (< y x)
                (< x z)
                (if (< z x)
                        (< x y)
                        nil
                )
        )
)
```

COND

```
(defun f7 (x y z)
        (cond
                ((< y x) (< x z))
                ((< z x) (< x y))
        )
)
```

AND/OR

```
(defun f8 (x y z)
        (or
                (and (< y x) (< x z))
                (and (< z x) (< x y))
        )
)
```

2.9. Задание 9

Задание

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

```
(Defun how_alike (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)))
```

Решение

```
(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
                        )
                )
        )
)
```