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»

Лабораторная работа №3 по дисциплине "Функциональное и логическое программирование"

Тема Работа интерпретатора Lisp

Студент Светличная А.А.

Группа ИУ7-53Б

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

Москва — 2023 г.

Оглавление

1 Теоритические вопросы

1.1 Базис Lisp

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

Базис языка Lisp содержит:

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

1.2 Классификация функций

Функции в Lisp классифицируют следующим образом:

- чистые математические функции (имеют фиксированное количесво аргументов и один результат);
- специальные функции — формы (прнимают произвольное число аргументов или по разному обрабатывают аргументы);
- функции высших порядков — функционалы (используются для создания синтаксически управляемых программ).

По базисные функции разделяются следующим образом:

- конструкторы — создают значение (cons, list);
- селекторы — получают доступ по адресу (car, cdr);
- предикаты — возвращают Nil, T.

1.3 Способы создания функций

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

- В Lisp можно определить функцию без имени с помощью **λ-выражений**.
  Lambda-определение безымянной функции:

  `(lambda <lambda-список> <форма>)`

  Lambda-вызов функции:

  `(<lambda-выражение> <формальные параметры>)`

- Также в Lisp можно определить функцию с именем с помощью **defun**.
  В таких функциях defun связывает символьный атом с Lambda-определением

  `(defun f <lambda-выражение>)`

  Упрощенное определение:

  `(defun f(arg1, ..., argN) <формы>)`

1.4 Работа функций cond, if, and/or

- **cond**

```
(cond (test1 body1)
    (test2 body2)
      ...
    (testN bodyN)
   [(T else-body)])
```

Список аргументов обрабатывается последовательно: вычисляется выражение `test_i`, и если не `Nil`, то вычисляется `body_i`, и работа функции завершается, если ни один тест не выполнился, то возвращается `Nil`, можно организовать ветку «`else`», явно указав в качестве `test - T`.

- **if**

```
1 (if test T-body F-body)
```

Работа функции `if` очевидна, с учетом, что всё что не nil, то T. Результат теста может быть как атомом (не обязательно Nil) так и списком. В зависимости от test, будет вычислен либо один либо другой аргумент.

- **and**

```
1 (and arg1 arg2 ... argN)
```

Функция `and` вычисляет аргументы, пока не станет очевидным результат (появится первый `nil`). Как только станет очевиден результат – возвращается последнее вычисленное значение.

- **or**

```
1 (or arg1 arg2 ... argN)
```

Функция `or` вычисляет аргументы, пока не станет очевидным результат (появится первый не `nil`). Как только станет очевиден результат – возвращается последнее вычисленное значение.

2 Практические задания

2.1 Задание №1

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

Листинг 2.1 – Выполнение задания №1

```
1 (defun f(x)
2     (if (evenp x)
3         x
4         (+ x 1) ))
```

2.2 Задание №2

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

Листинг 2.2 – Выполнение задания №2

```
1 (defun f(x)
2     (if (> x 0)
3         (+ x 1)
4         (- x 1) ))
```

2.3 Задание №3

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

Листинг 2.3 – Выполнение задания №3

```
1 (defun f(x y)
2     (if (< x y)
3         (list x y)
4         (list y x) ))
```

2.4 Задание №4

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

Листинг 2.4 – Выполнение задания №4

```
(defun f(x y z)
    (if (or (and (> x y) (< x z))
            (and (< x y) (> x z)))
        T
        NIL ))
```

2.5 Задание №5

Каков результат вычисления следующих выражений?

Листинг 2.5 – Выполнение задания №5

```
(and 'fee 'fie 'foe)          ;; FOE
(or nil 'fie 'foe)            ;; FIE
(and (equal 'abc 'abc) 'yes) ;; YES
(or 'fee 'fie 'foe)           ;; FEE
(and nil 'fie 'foe)           ;; NIL
(or (equal 'abc 'abc) 'yes)  ;; T
```

2.6 Задание №6

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

Листинг 2.6 – Выполнение задания №6

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

2.7 Задание №7

Какой из следующих двух вариантов предиката ошибочен и почему?

Листинг 2.7 – Условие задания №7

```
(defun pred1 (x)
    (and (numberp x) (plusp x)))

(defun pred2 (x)
    (and (plusp x) (numberp x)))
```

Предикат №2 неверен, так как проверка на число должна быть перед проверкой на положительность числа.

2.8 Задание №8

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

Листинг 2.8 – Выполнение задания №8

```
(defun f (x y z)
    (if (> x y)
        (< x z)
        (if (< x y)
            (> x z)) ))

(defun f (x y z)
    (cond ((> x y) (< x z))
          ((< x y) (> x z)) ))

(defun f (x y z)
    (or (and (> x y) (< x z))
        (and (< x y) (> x z))))
```

2.9 Задание №9

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

Листинг 2.9 – Условие задания №9

```
(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)))
```

Листинг 2.10 – Выполнение задания №9

```
(defun how_alike(x y)
    (if (if (= x y) T (if (equal x y) T NIL)) 'the_same
    (if (if (oddp x) (if (oddp y) T NIL) NIL)   'both_odd
    (if (if (evenp x) (if (evenp y) T NIL) NIL) 'both_even
     'difference))))

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

(defun how_alike(x y)
    (cond ((= x y) 'the_same) ((equal x y) 'the_same)
    ((cond ((oddp x) (cond ((oddp y) 'both_odd)))))
    ((cond ((evenp x) (cond ((evenp y) 'both_even)))))
        (T 'difference)))
```