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: Карпова Е. О.
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*, *quote*, *eq*, *eval*, *apply*, $funcall$.

1.2. Классификация функций языка Lisp.

Среди функций в языке Lisp выделяют бызисные, пользовательские и функции ядра. Также, по реализации 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 $defun$.

$$(lambda \quad (x_1 \quad x_2 \quad ... \quad x_n) \quad f),$$

где f — тело функции, $x_i, i = \overline{1,n}$ — формальные параметры.

$$(defun \quad < \text{имя} > \quad [lambda] \quad (x_1 \quad x_2 \quad ... \quad x_n) \quad f),$$

где f — тело функции, $x_i, i = \overline{1,n}$ — 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 (condition-1 expression-1-1 expression-1-2 ...)
      (condition-2)
      (condition-3 expression-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 языков программирования. Сначала выполняется форма *condition*. 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 condition then else)
```

1.4.3. *and*

and последовательно слева направо вычисляет формы. Если какая-либо форма *expressionN* 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* — истину, так и для условных выражений.

```
(and expression1 expression2 ... )
```

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.

```
(or expression1 expression2 ... )
```

2. Практическая часть

2.1. Задание №1

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

Листинг 2.1 — Задание №1

```
[2]> (defun close_big_even (x) (if (evenp x) x (+ x 1)))
; или (defun close_big_even (x) (+ x (mod x 2)))
CLOSE_BIG_EVEN
[3]> (close_big_even 5)
6
[4]> (close_big_even 4)
4
```

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 — Задание №2

```
[6]> (defun bigger_abs (x) (if (< x 0) (- x 1) (+ x 1)))
BIGGER_ABS
[7]> (bigger_abs 3)
4
[8]> (bigger_abs -3)
-4
```

2.3. Задание №3

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

Листинг 2.3 — Задание №3

```
[12]> (defun asc_list (x y) (if (< x y) (list x y) (list y x)))
ASC_LIST
[13]> (asc_list 2 3)
(2 3)
[14]> (asc_list 3 2)
(2 3)
[15]> (asc_list 0 -1)
(-1 0)
[16]> (asc_list 0 0)
(0 0)
```

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 — Задание №4

```
[8]> (defun first_between (x y z) (if (or (and (< z x) (< x y))
(and (< y x) (< x z))) t nil))
FIRST_BETWEEN
[9]> (first_between 1 2 3)
NIL
[10]> (first_between 2 1 3)
T
[11]> (first_between 2 3 1)
T
[12]> (first_between 0 -1 1)
T
[13]> (first_between 0 0 0)
NIL
```

2.5. Задание №5

Каков результат вычисления следующих выражений?

1. $(and \quad 'fee \quad 'fie \quad 'foe)$;
2. $(or \quad nil \quad 'fie \quad 'foe)$;
3. $(and \quad (equal \quad 'abc \quad 'abc) \quad 'yes)$;
4. $(or \quad 'fee \quad 'fie \quad 'foe)$;
5. $(and \quad nil \quad 'fie \quad 'foe)$;
6. $(or \quad (equal \quad 'abc \quad 'abc) \quad 'yes)$.

Листинг 2.5 — Задание №5

```
[7]> (and 'fee 'fie 'foe)
FOE ; если все выражения не-nil, возвращает результат последнего

[8]> (or nil 'fie 'foe)
FIE ; вернёт первый не-nil результат

[9]> (and (equal 'abc 'abc) 'yes)
YES ; если все выражения не-nil, возвращает результат последнего

[10]> (or 'fee 'fie 'foe)
FEE ; возвращает первый не-nil результат

[11]> (and nil 'fie 'foe)
NIL ; если встречает nil результат, сразу его возвращает

[12]> (or (equal 'abc 'abc) 'yes)
T ; возвращает первый не-nil результат
```

2.6. Задание №6

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

Листинг 2.6 — Задание №6

```
[15]> (defun second_less (x y) (>= x y))
SECOND_LESS
[16]> (second_less 1 1)
T
[17]> (second_less 1 2)
NIL
[18]> (second_less 3 2)
T
```

2.7. Задание №7

Какой из следующих двух вариантов предиката ошибочен и почему?

1. $(defun\ \ pred1\ \ (x)\ \ (and\ \ (numberp\ \ x)\ \ (plusp\ \ x)))$;
2. $(defun\ \ pred2\ \ (x)\ \ (and\ \ (plusp\ \ x)\ \ (numberp\ \ x)))$.

Листинг 2.7 — Задание №7

```
[1]> (defun pred1 (x) (and (numberp x) (plusp x)))
PRED1
[2]> (pred1 2)
T
[3]> (pred1 -2)
NIL
[4]> (pred1 't)
NIL

[5]> (defun pred2 (x) (and (plusp x) (numberp x)))
PRED2
[6]> (pred2 2)
T
[7]> (pred2 -2)
NIL
[8]> (pred2 't)
*** - PLUSP: T is not a real number
```

Второй предикат ошибочен. Это связано с тем, что в *and* переданные выражения обрабатываются в порядке передачи, то есть проверка *plusp* на положительность аргумента будет выполнена раньше, чем проверка на то, что аргумент является числом. Если аргумент — не число, то такой порядок проверок приведёт к ошибке.

2.8. Задание №8

Решить задачу 4, используя для ее решения конструкции: только if, только *cond*, только and/or.

Задание №4: написать функцию, которая принимает три числа и возвращает только тогда, когда первое число расположено между вторым и третьим.

Листинг 2.8 — Задание №8

```
; только if
[2]> (defun first_between (x y z) (if (< y x) (< x z)
(if (< z x) (< x y) nil)))
FIRST_BETWEEN
[3]> (first_between 1 2 3)
NIL
[4]> (first_between 2 1 3)
T
[5]> (first_between 2 3 1)
T
[6]> (first_between 0 -1 1)
T
[7]> (first_between 0 0 0)
NIL
```

Листинг 2.9 — Задание №8

```
; только cond
[14]> (defun first_between (x y z) (
cond ((< y x) (< x z))
((< z x) (< x y))
))
FIRST_BETWEEN
[15]> (first_between 1 2 3)
NIL
[16]> (first_between 2 1 3)
T
[17]> (first_between 2 3 1)
T
[18]> (first_between 0 -1 1)
T
[19]> (first_between 0 0 0)
NIL
```

Листинг 2.10 — Задание №8

```
; только and/or
[20]> (defun first_between (x y z) (or (and (< z x) (< x y))
(and (< y x) (< x z))))
FIRST_BETWEEN
[21]> (first_between 1 2 3)
NIL
[22]> (first_between 2 1 3)
T
[23]> (first_between 2 3 1)
T
[24]> (first_between 0 -1 1)
T
[25]> (first_between 0 0 0)
NIL
```

2.9. Задание №9

Переписать функцию $how - alike$, приведенную в лекции и использующую $cond$, используя только конструкции if, and/or.

Вариант с $cond$ из лекций:

Листинг 2.11 — Задание №9

```
[8]> (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)
))
HOW_ALIKE
[9]> (how_alike 3 0)
DIFFERENCE
[10]> (how_alike 2 2)
THE_SAME
[11]> (how_alike 2 2.0)
THE_SAME
[12]> (how_alike 2 4)
BOTH_EVEN
[13]> (how_alike 3 3)
THE_SAME
[14]> (how_alike 3 1)
BOTH_ODD
```

Вариант с if, and/or:

Листинг 2.12 — Задание №9

```
[15]> (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))
))
HOW_ALIKE
[16]> (how_alike 3 0)
DIFFERENCE
[17]> (how_alike 2 2)
THE_SAME
[18]> (how_alike 2 2.0)
THE_SAME
[19]> (how_alike 2 4)
BOTH_EVEN
[20]> (how_alike 3 3)
THE_SAME
[21]> (how_alike 3 1)
BOTH_ODD
```