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 № 2
по дисциплине «Функциональное и логическое программирование»

Тема Определение функций пользователя.

Студент Калашников С.Д.

Группа ИУ7-63Б

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

Москва, 2023

СОДЕРЖАНИЕ

1 Теоретическая часть

1.1 Базис языка

Базис состоит из:

1. структуры, атомы;
2. встроенные (примитивные) функции (atom, eq, cons, car, cdr);
3. специальные функции и функционалы, управляющие обработкой структур, представляющих вычислимые выражения (quote, cond, lambda, label, eval).

1.2 Классификация функций

Функции в Lisp классифицируют следующим образом:

- чистые математические функции;
- рекурсивные функции;
- специальные функции — формы (сегодня 2 аргумента, завтра - 5);
- псевдофункции (создают эффект на внешнем устройстве);
- функции с вариативными значениями, из которых выбирается 1;
- функции высших порядков — функционал: используется для синтаксического управления программ (абстракция языка).

По назначению функции разделяются следующим образом:

- конструкторы — создают значение (cons, например);

- селекторы — получают доступ по адресу (car, cdr);
- предикаты — возвращают Nil, T.
- функции сравнения – такие как: eq, eql, equal, equalp.

1.3 Способы создания функций

Функции в Lisp можно задавать следующими способами:

Lambda-выражение

Синтаксис:
(lambda <λ-список> форма)
Пример:

Листинг 1.1 — Функция определенная Lambda-выражением

```
1 (lambda (a b) (sqrt (+ (* a a) (* b b))))
```

Именованная функция

Синтаксис:
(defun <имя функции> <λ-выражение>)
Пример:

Листинг 1.2 — Пример определения именованной функции

```
1 (defun hyp (a b) (sqrt (+ (* a a) (* b b))))
```

1.4 Функции car и cdr

car — функция получения первого элемента точечной пары.
Примеры:
cdr — функция получения второго элемента точечной пары.

S-выражение	Результат выполнения car
(A . B)	A
((A . B) . C)	(A . B)
A	ошибка

S-выражение	Результат выполнения cdr
(A . B)	B
(A . (B . C))	(B . C)
A	ошибка

1.5 Функции eq, eql, equal, equalp

(eq x y) является истиной тогда и только тогда, когда, x и y являются идентичными объектами. (В реализациях, x и y обычно равны eq тогда и только тогда, когда обращаются к одной ячейке памяти.)

Предикат eql истинен, если его аргументы равны eq, или если это числа одинакового типа и с одинаковыми значениями, или если это одинаковые буквы.

Предикат equal истинен, если его аргументы это структурно похожие (изоморфные) объекты. Грубое правило такое, что два объекта равны equal тогда и только тогда, когда одинаково их выводимое представление. Числа и буквы сравниваются также как и в eql. Символы сравниваются как в eq. Объекты, которые содержат другие элементы, будут равны equal, если они принадлежат одному типу и содержащиеся элементы равны equal.

Два объекта равны equalp, если они равны equal, если они буквы и удовлетворяют предикату char-equal, который игнорирует регистр и другие атрибуты символов, если они числа и имеют одинаковое значение, даже если числа разных типов, если они включает в себя элементы, которые также равны equalp.

Листинг 1.3 — Примеры

```
1 (eq 'a 'b) is false.
2 (eq 'a 'a) is true.
3 (eq 3 3) might be true or false, depending on the implementation.
4 (eq 3 3.0) is false.
```

```
(eq 3.0 3.0) might be true or false, depending on the
   implementation.
(eq #c(3 -4) #c(3 -4))
might be true or false, depending on the implementation.
(eq #c(3 -4.0) #c(3 -4)) is false.
(eq (cons 'a 'b) (cons 'a 'c)) is false.
(eq (cons 'a 'b) (cons 'a 'b)) is false.
(eq '(a . b) '(a . b)) might be true or false.
(progn (setq x (cons 'a 'b)) (eq x x)) is true.
(progn (setq x '(a . b)) (eq x x)) is true.
(eq #\A #\A) might be true or false, depending on the
   implementation.
(eq "Foo" "Foo") might be true or false.
(eq "Foo" (copy-seq "Foo")) is false.
(eq "FOO" "foo") is false.


(eql 'a 'b) is false.
(eql 'a 'a) is true.
(eql 3 3) is true.
(eql 3 3.0) is false.
(eql 3.0 3.0) is true.
(eql #c(3 -4) #c(3 -4)) is true.
(eql #c(3 -4.0) #c(3 -4)) is false.
(eql (cons 'a 'b) (cons 'a 'c)) is false.
(eql (cons 'a 'b) (cons 'a 'b)) is false.
(eql '(a . b) '(a . b)) might be true or false.
(progn (setq x (cons 'a 'b)) (eql x x)) is true.
(progn (setq x '(a . b)) (eql x x)) is true.
(eql #\A #\A) is true.
(eql "Foo" "Foo") might be true or false.
(eql "Foo" (copy-seq "Foo")) is false.
(eql "FOO" "foo") is false.


(equal 'a 'b) is false.
(equal 'a 'a) is true.
(equal 3 3) is true.
(equal 3 3.0) is false.
(equal 3.0 3.0) is true.
```

```
(equal #c(3 -4) #c(3 -4)) is true.
(equal #c(3 -4.0) #c(3 -4)) is false.
(equal (cons 'a 'b) (cons 'a 'c)) is false.
(equal (cons 'a 'b) (cons 'a 'b)) is true.
(equal '(a . b) '(a . b)) is true.
(progn (setq x (cons 'a 'b)) (equal x x)) is true.
(progn (setq x '(a . b)) (equal x x)) is true.
(equal #\A #\A) is true.
(equal "Foo" "Foo") is true.
(equal "Foo" (copy-seq "Foo")) is true.
(equal "FOO" "foo") is false.
```

1.6 Назначение и отличие list от cons

cons — функция конструирования точечной пары, на вход получает 2 значения и делает из них точечную пару.

list — функция конструирования списка. На вход получает произвольное количество элементов и делает из них список.

Вызовы (list 1 2 3 4) и (cons 1 (cons 2 (cons 3 (cons 4 Nil)))) эквивалентны, то есть дают одинаковый результат.

2 Практическая часть

2.1 Задание 1

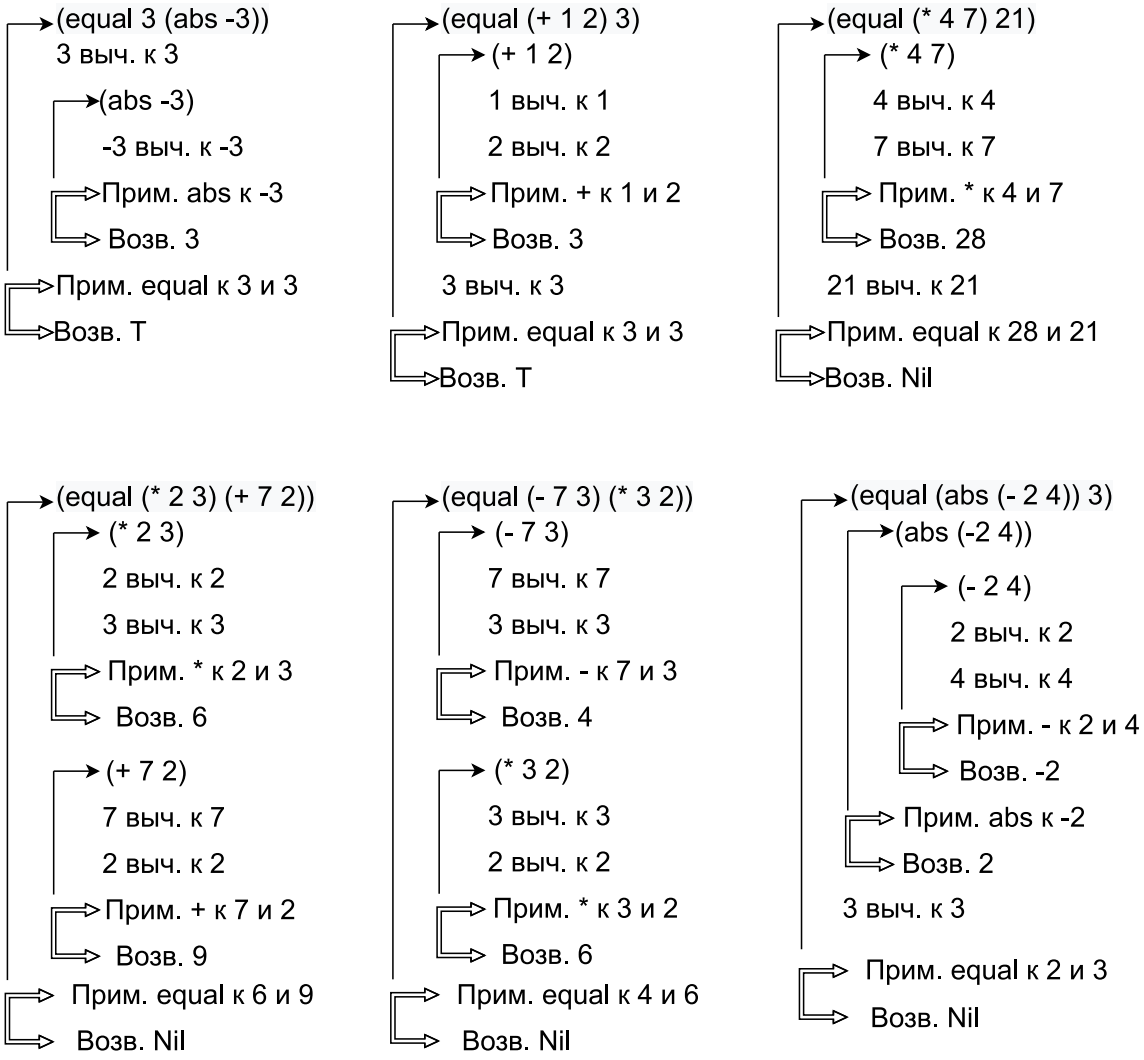

Рис. 2.1 – Схема A

2.2 Задание 2

Листинг 2.1 — Выражение 1

```
1 (defun gip (a b) (sqrt (+ (* a a) (* b b))))
```

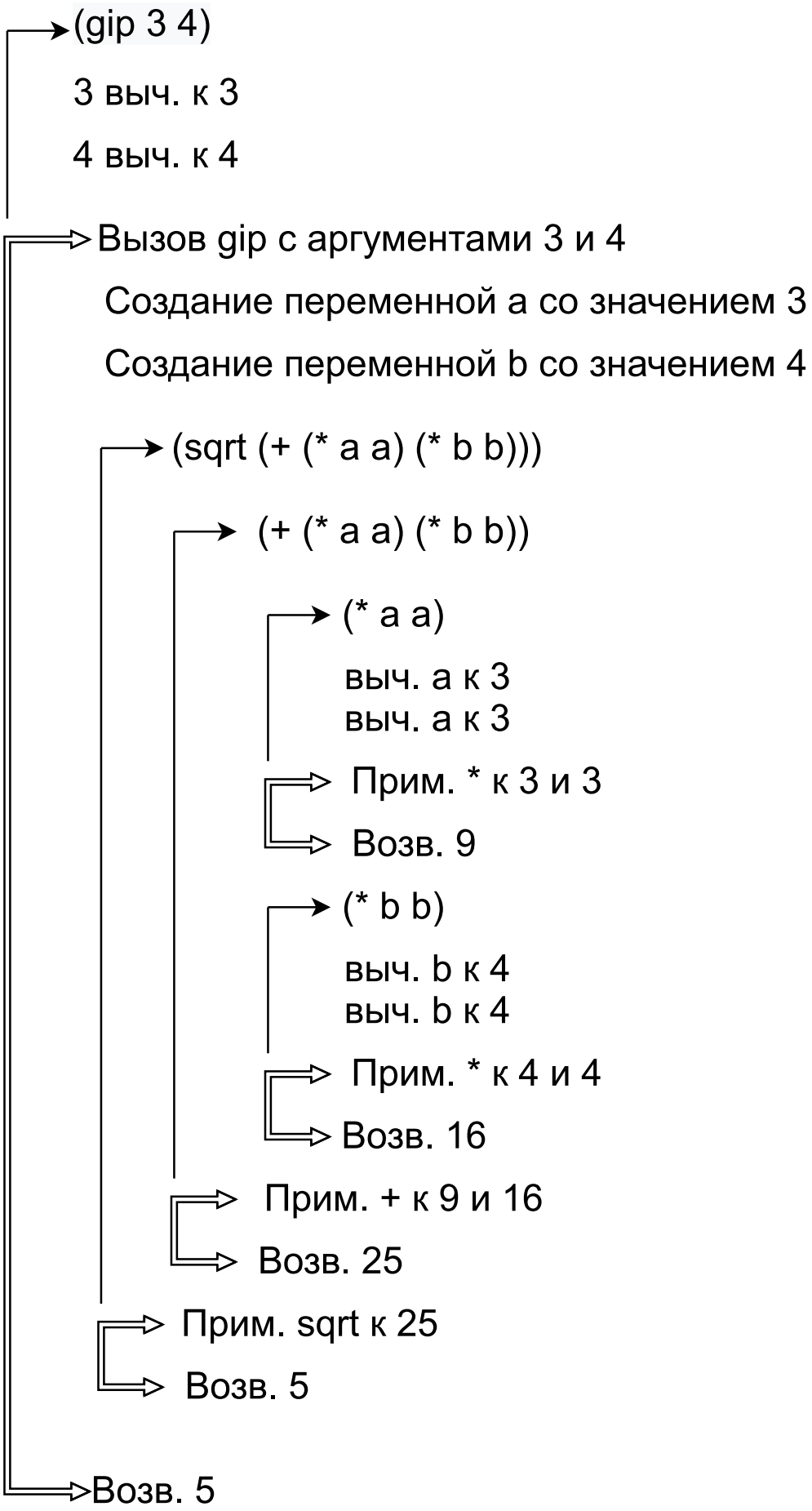

Рис. 2.2 – Схема Б

2.3 Задание 3

Листинг 2.2 — Выражение 1

```
(list 'a c) ;The variable C is unbound.; (list 'a 'c)
```

Листинг 2.3 — Выражение 2

```
(cons 'a (b c)) ;The variable C is unbound.; (cons 'a '(b c))
```

Листинг 2.4 — Выражение 3

```
(cons 'a '(b c))  ;(A B C)
```

Листинг 2.5 — Выражение 4

```
(caddr (1 2 3 4 5)) ;Compile-time error: illegal function call; (
   caddr '(1 2 3 4 5))
```

Листинг 2.6 — Выражение 5

```
(cons 'a 'b 'c) ;invalid number of arguments: 3; (cons '(a b) 'c)
    (cons 'a '(b c))
```

Листинг 2.7 — Выражение 6

```
(list 'a (b c)) ;The variable C is unbound.; (list 'a '(b c))
```

Листинг 2.8 — Выражение 7

```
(list a '(b c)) ;The variable A is unbound.; (list 'a '(b c))
```

Листинг 2.9 — Выражение 8

```
(list (+ 1 '(length '(1 2 3)))) ;  The value (LENGTH '(1 2 3)) is
    not of type NUMBER; (list (+ 1 (length '(1 2 3))))
```

2.4 Задание 4

Листинг 2.10 — Выражение 1

```
 (defun what (l1 l2) (> (length l1) (length l2)))
```

2.5 Задание 5

Листинг 2.11 — Выражение 1

```
(cons 3 (list 5 6)) ;(3 5 6)
```

Листинг 2.12 — Выражение 2

```
(list 3 'from 9 'lives (- 9 3)) ;(3 FROM 9 LIVES 6)
```

Листинг 2.13 — Выражение 3

```
(+ (length for 2 too)) (car '(21 22 23))) ;The variable FOR is
    unbound.
```

Листинг 2.14 — Выражение 4

```
(cdr '(cons is short for ans)) ;(IS SHORT FOR ANS)
```

Листинг 2.15 — Выражение 5

```
(car (list one two)) ;The variable ONE is unbound.
```

Листинг 2.16 — Выражение 6

```
(cons 3 '(list 5 6)) ;(3 LIST 5 6)
```

Листинг 2.17 — Выражение 7

```
(car (list 'one 'two)) ;ONE
```

2.6 Задание 6

Листинг 2.18 — Выражение 1

```
  (mystery (one two)) ;The variable TWO is unbound.
```

Листинг 2.19 — Выражение 2

```
  (mystery (last one two)) ;The variable ONE is unbound.
```

Листинг 2.20 — Выражение 3

```
  (mystery free) ;The variable FREE is unbound.
```

Листинг 2.21 — Выражение 4

```
(mystery one 'two)) ;The variable ONE is unbound.
```

2.7 Задание 7

Листинг 2.22 — Выражение 1

```
(defun f-to-c (temp) (* (/ 5 9) (- temp 32.0)))
```

При переводе из 451 по Фаренгейту в Цельсии получится 232.77779 градуса.

2.8 Задание 8

Листинг 2.23 — Выражение 1

```
(list 'cons t NIL) ;(CONS T NIL)
```

Листинг 2.24 — Выражение 2

```
(eval (eval (list 'cons t NIL))) ;The function COMMON-LISP:T is
    undefined.
```

Листинг 2.25 — Выражение 3

```
(apply #cons "(t NIL)) ;illegal complex number format: #CONS
```

Листинг 2.26 — Выражение 4

```
(list 'eval NIL) ;(EVAL NIL)
```

Листинг 2.27 — Выражение 5

```
(eval (list 'cons t NIL)) ;(T)
```

Листинг 2.28 — Выражение 6

```
(eval NIL) ;NIL
```

Листинг 2.29 — Выражение 7

```
(eval (list 'eval NIL)) ;NIL
```