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»

Отчет по лабораторной работе №2 по дисциплине «Функциональное и логическое программирование»

Тема Определение функций пользователя

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) _____

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

Москва — 2022 г.

Теоретические вопросы

1. Базис Lisp

Базис – это минимальный набор инструментов языка и стркутур данных, который позволяет решить любые задачи.

Базис Lisp :

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

Атомы:

- символы (идентификаторы) – синтаксически – набор литер (букв и цифр), начинающихся с буквы;
- специальные символы – T, Nil (используются для обозначения логических констант);
- самоопределимые атомы – натуральные числа, дробные числа, вещественные числа, строки – последовательность символов, заключенных в двойные апострофы (например, “abc”);

Более сложные данные – списки и точечные пары (структуры), которые строятся с помощью унифицированных структур – блоков памяти – бинарных узлов.

Определения:

Точечная пара ::= (<атом> . <атом>) | (<атом> . <точечная пара>) | (<точечная пара> . <атом>) | (<точечная пара> . <точечная пара>);

Список ::= <пустой список> | <непустой список>, где

<пустой список> ::= () | Nil,

<непустой список> ::= (<первый элемент> . <хвост>),

<первый элемент> ::= <S-выражение>,

S-выражение ::= <атом> | <точечная пара>,

<хвост> ::= <список>.

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

Функционалом, или функцией высшего порядка называется функция, аргументом или результатом которой является другая функция.

2.Классификация функций

Один из вариантов классификации функций:

- чистые математические функции (имеют фиксированное количество аргументов, сначала выяисляются все аргументы, а только потом к ним применяется функция);
- рекурсивные функции (основной способ выполнения повторных вычислений);
- специальные функции, или формы (могут принимать произвольное количество аргументов, или аргументы могут обрабатываться по-разному);
- псевдофункции (создают «эффект», например, вывод на экран);
- функции с вариантами значений, из которых выбирается одно;
- функции высших порядков, или функционалы – функции, аргументом или результатом которых является другая функция (используются для построения синтаксически управляемых программ);

3. Способы создания функций

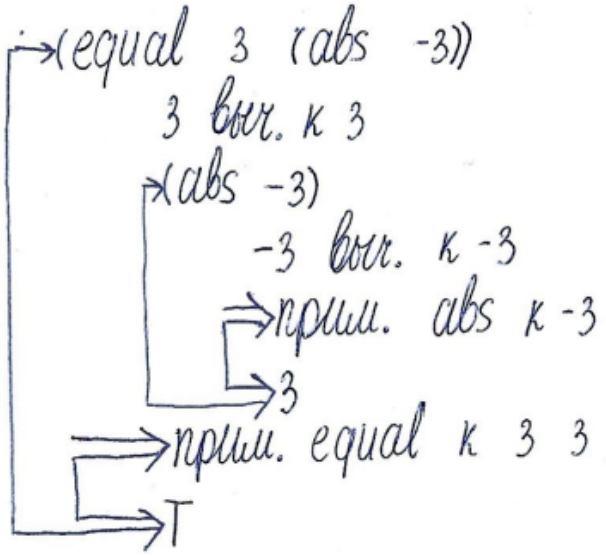
4. Функции Car и Cdr

5. Назначение и отличие в работе Cons и List

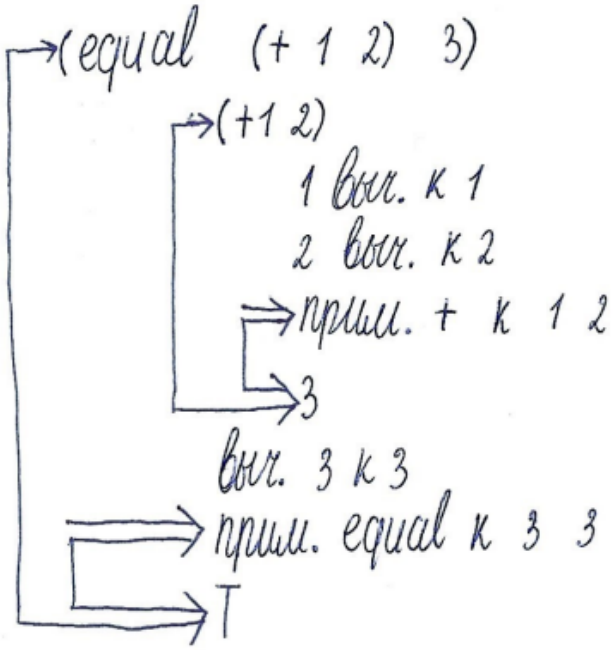
Практические задания

1. Составить диаграмму вычисления следующих выражений:

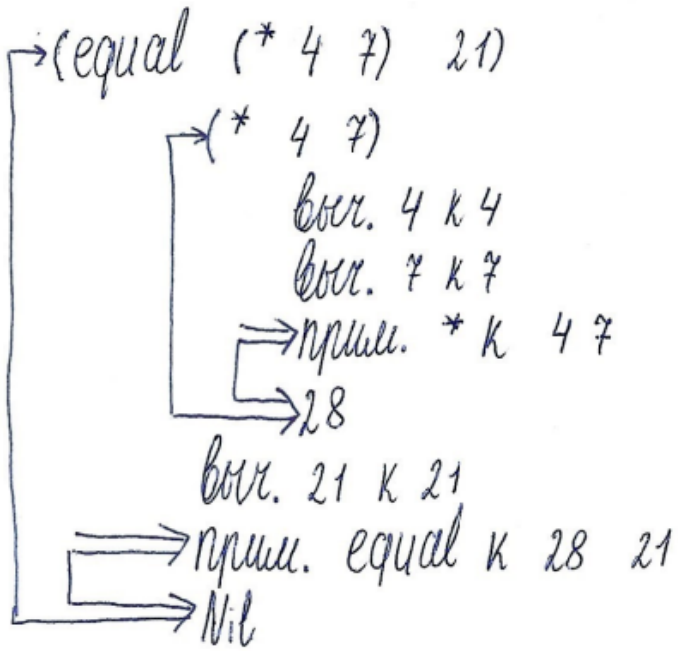
1. (equal 3 (abs - 3))



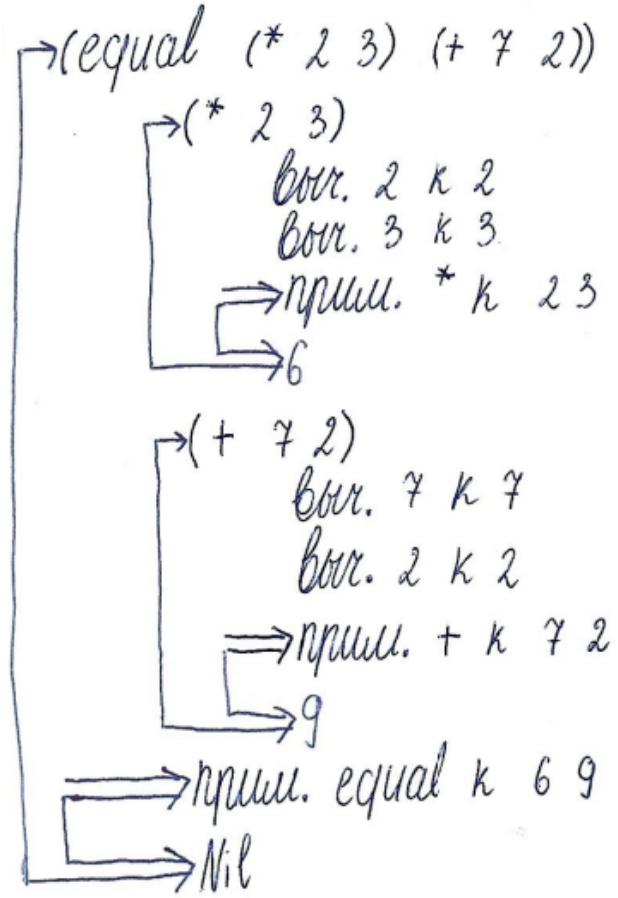
2. (equal (+ 1 2) 3)



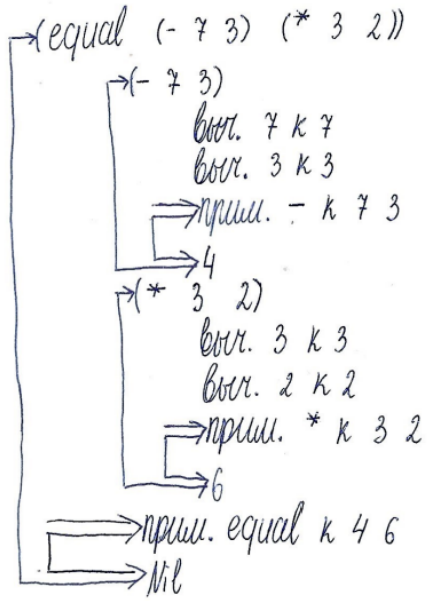
3. (equal (* 4 7) 21)



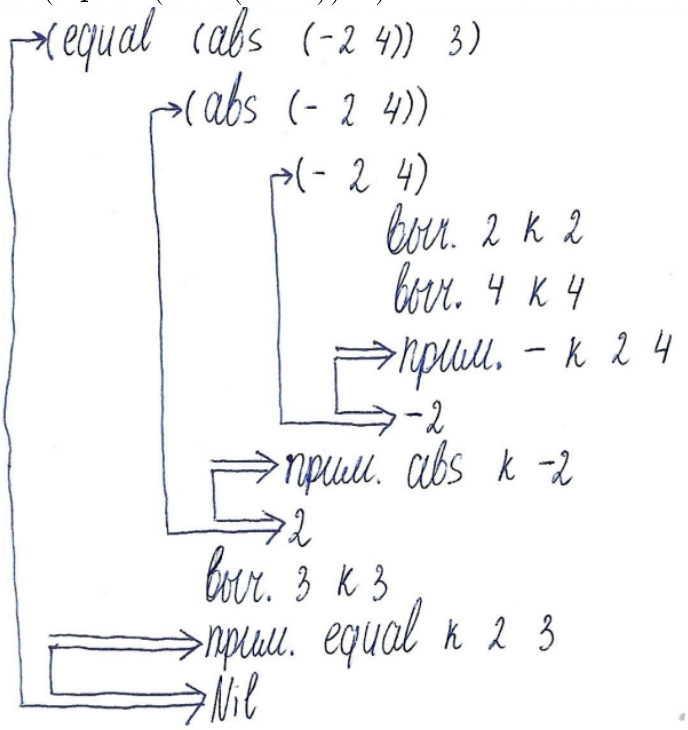
4. (equal (* 2 3) (+ 7 2))



5. (equal (- 7 3) (* 3 2))

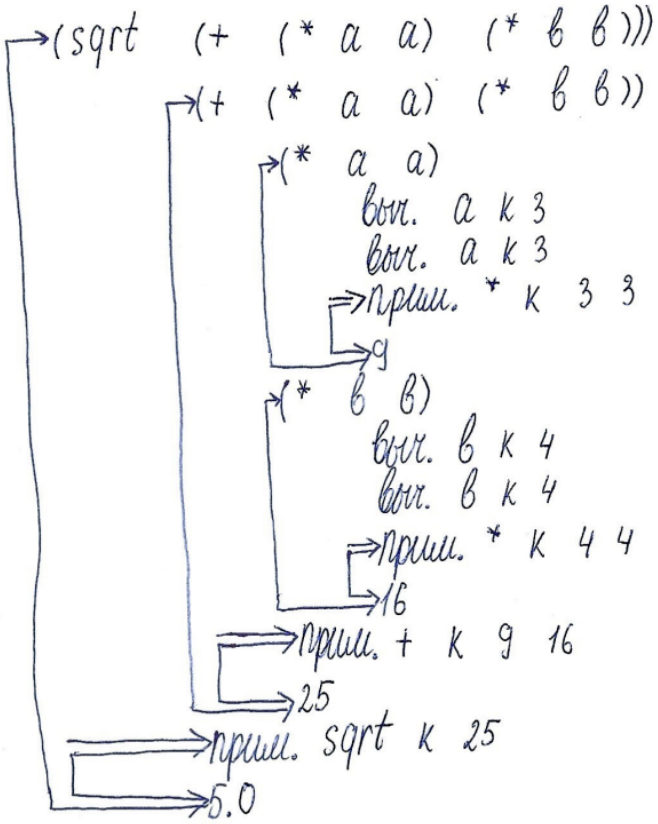


6. (equal (abs (- 2 4)) 3)



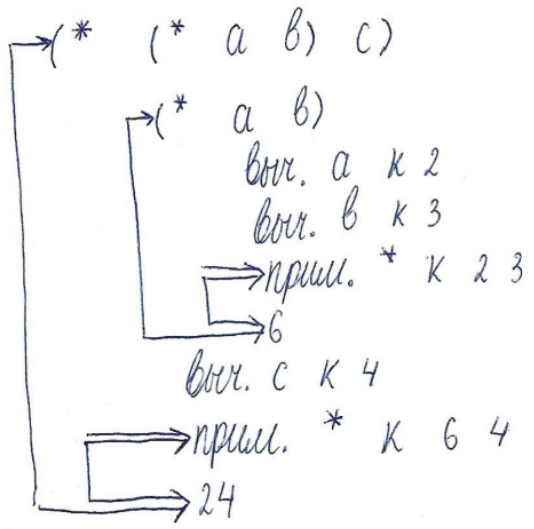
2. Написать функцию, вычисляющую гипотенузу прямоугольного треугольника по заданным катетам и составить диаграмму её вычисления.

```
(defun hypotenuse (a b) (sqrt (+ (* a a) (* b  b))))
(hypotenuse 3 4) => 5.0
```



3. Написать функцию, вычисляющую объем параллелепипеда по 3-м его сторонам, и составить диаграмму ее вычисления.

```
 (defun p_volume (a b c) (* (* a b) c))
(p_volume 2 3 4) => 24
;; or
(defun p_volume (a b c) (* a b c))
(p_volume 2 3 4) => 24
```



4. Каковы результаты вычисления следующих выражений?(объяснить возможную ошибку и варианты ее устранения)

1.

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

Одна из возможных ошибок: переменная с не связана со значением. Решение: задать переменнной с некоторое значение.

```
(let ((c 'c)) (list 'a c)) => (A C)
```

Другая из возможных ошибок: предполагалось, что с – это символ. Решение: использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
  (list 'a 'c) => (A C)
```

2.

```
   (cons 'a (b c)) => Undefined function: B, Undefined variable: C
```

Одна из возможных ошибок: функция b не связана со своим определением, а переменная с не связана со своимм значением. Решение: определить функцию b с одним аргументом (или переменным количеством аргументов), а переменной с задать некоторое значение.

```
  (defun b (c) (cons 'b (cons c Nil)))
  (b 'c) => (B C)
  (let ((c 'c)) (cons 'a (b c))) => (A B C)
```

Другая из возможных ошибок: предполагалось, что (b c) – это список из символов b и c. Решение: использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргументов.

```
(cons 'a '(b c)) => (A B C)
```

3.

```
(cons 'a '(b c)) => (A B C)
```

Ошибок нет

4.

```
(caddy (1 2 3 4 5))  => Undefined function: CADDY
```

Одна из возможных ошибок: функция caddy не связана со своим определением. Решение: определить функцию caddy, принимающую один аргумент (или переменное ччисло аргуентов), и использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
(defun caddy (arg) arg)
(caddy '(1 2 3 4 5)) => (1 2 3 4 5)
```

Другая из возможных ошибок: предполагалось вызвать функцию caddr для получения третьего элемента списка (1 2 3 4 5). Решение: исправить опечатку (заменить caddy на caddr) и использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
(caddr '(1 2 3 4 5)) => 3
```

5.

```
(cons 'a 'b 'c)  => The function CONS is called with three arguments,
    but wants exactly two.
```

Ошибка: функция cons вызвана с тремя аргументами, хотя она принимает два аргумента. Решение: предполагая, что автор хотел получить список из трех символов (a b c), можно либо первым аргументом передать символ a, а вторым – список (b c), либо использовать функцию list вместо cons.

```
(cons 'a '(b c)) => (A B C)
(list 'a 'b 'c) => (A B C)
```

6.

```
(list 'a (b c)) => Undefined function: B, Undefined variable: C
```

(Аналогично пункту 2)

Одна из возможных ошибок: функция b не связана со своим определением, а переменная c не связана со своимм значением. Решение: определить функцию b

с одним аргументом (или переменным количеством аргументов), а переменной с задать некоторое значение.

```
(defun b (c) (cons 'b (cons c Nil)))
(b 'c) => (B C)
(let ((c 'c)) (list 'a (b c))) => (A (B C))
```

Другая из возможных ошибок: предполагалось, что (b c) – это список из символов b и c. Решение: использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргументов.

```
(list 'a '(b c)) => (A (B C))
```

7.

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

Одна из возможных ошибок: переменная a не связана со значением. Решение: задать переменнной a некоторое значение.

```
(let ((a 'a)) (list a '(b c))) => (A (B C))
```

Другая из возможных ошибок: предполагалось, что a – это символ. Решение: использовать функцию quote (или сокращенную – апостроф) для блокировки вычисления аргумента.

```
(list 'a '(b c)) => (A (B C))
```

8.

```
(list (+ 1 '(length '(1 2 3)))) =>  Value of '(LENGTH '(1 2 3)) in
  (+ 1 '(LENGTH '(1 2 3))) is (LENGTH '(1 2 3)), not a NUMBER.
```

Функция + в качестве аргументов ожидает аргументы типа NUMBER. Вторымм аргументом ей передано '(length '(1 2 3)). Функция length возвращает длину переданного ей списка (тип NUMBER), однако апостроф блокирует вычисление. Таким образом, вместо длины списка типа NUMBER функции + в качестве второго аргумента передается значение (LENGTH '(1 2 3)). Решение - убрать апостроф, блокирующий вычисления аргумента.

```
(list (+ 1 (length '(1 2 3)))) => (4)
```

5. Написать функцию longer_then от двух списков-аргументов, которая возвращает T, если первый аргумент имеет большую длину

```
(defun longer_then (list1 list2) (> (length list1) (length list2)))
(longer_then '(1 2 3) '(1 2)) => T
(longer_then '(1 2) '(1 2 3)) => NIL
(longer_then Nil '(1)) => NIL
 (longer_then '(1 2) '(1 2)) => NIL
```

6. Каковы результаты вычисления следующих выражений?

```
(cons 3 (list 5 6)) => (3 5 6)
(cons 3 '(list 5 6)) => (3 LIST 5 6)
(list 3 'from 9 'lives (- 9 3)) => (3 FROM 9 LIVES 6)
(+ (length for 2 too)) (car '(21 22 23))) => The variable FOR is
   unbound.
(cdr '(cons is short for ans)) => (IS SHORT FOR ANS)
(car (list one two)) => Undefined variables: ONE TWO
(car (list 'one 'two)) => ONE
```

7. Дана функция (defun mystery (x) (list (second x) (first x))). Какие результаты вычисления следующих выражений?

first и rest – мнемоничные синонимичные названия для функций car и cdr. Функции от SECOND до TENTH извлекают соответствующие элементы списка. LAST возвращает последнюю cons-ячейку в списке (если вызывается с целочисленным аргументом n, возвращает n ячеек).

```
(mystery (one two)) => The variable TWO is unbound
(mystery (last one two)) => The variable ONE is unbound
(mystery free) => The variable FREE is unbound
(mystery one 'two) => The variable ONE is unbound
```

Примеры корректной работы:

```
(mystery '(one two)) => (TWO ONE)
(mystery '(last one two)) => (ONE LAST)
(mystery (last '(one two))) => (NIL TWO)
(mystery '(free)) => (NIL FREE)
```

8. Написать функцию, которая переводит температуру в системе Фаренгейта температуру по Цельсию (defun f-to-c (temp)...)

Формулы: c = 5/9*(f-32.0); f = 9/5*c+32.0.

Как бы назывался роман Р. Брэдбери "451 по Фаренгейту"в системе по Цельсию?

```
(defun f-to-c (temp) (* (/ 5 9) (- temp 32.0)))
(f-to-c 451) => 232.77779

(defun c-to-f (temp) (+ (* (/ 9 5) temp) 32.0))
(c-to-f 232.77779) => 451.0
```

Ответ: "232.77779 по Цельсию"

9. Что получится при вычисления каждого из выражений?

Eval выполняет двойное вычисление своего аргумента. Эта функция является обычной, и первое вычисление аргумента выполняет так же, как и любая обычная функция. Полученное при этом выражение вычисляется ещё раз. Такое двойное вычисление может понадобиться либо для снятия блокировки вычислений (установленной функцией quote), либо же для вычисления сформированного в ходе первого вычисления нового функционального вызова.

Функционал apply является обычной функцией с двумя вычисляемыми аргументами, обращение к ней имеет вид: (apply F L), где F – функциональный аргумент и L – список, рассматриваемый как список фактических параметров для F. Значение функционала – результат применения F к этим фактическим параметрам.

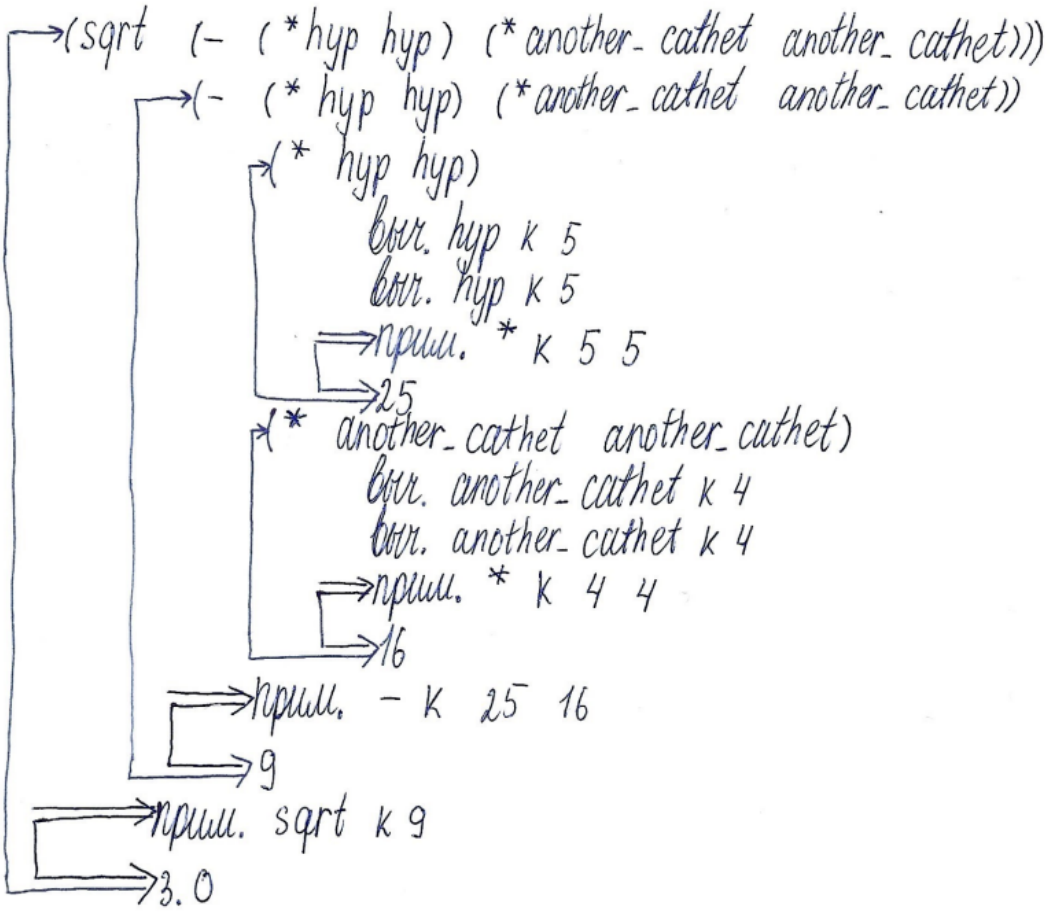
В диалекте Common Lisp для замыкания функционального аргумента встроена специальная форма (function F), где F – определяющее выражение функции. Эту форму часто называют функциональной блокировкой, поскольку она аналогична по действию функции quote, но не просто квотирует аргумент, а как бы замыкает значения используемых в функциональном аргументе F свободных переменных, фиксируя их значения из контекста его определения. Функциональную блокировку можно записывать короче, с помощью двух знаков #'.

```
(list 'cons t NIL) =>  (CONS T NIL)
(eval (list 'cons t NIL)) => (T)
(eval (eval (list 'cons t NIL))) => The function COMMON-LISP:T is
   undefined
;(eval t) => T
(apply #cons "(t NIL)) => illegal complex number format: #CONS
;(apply #'cons '(t NIL)) => (T)
(eval NIL) => NIL
(list 'eval NIL) => (EVAL NIL)
(eval (list 'eval NIL)) => NIL
;(eval (list 'eval NIL)) = (eval (eval NIL)) = (eval NIL) => NIL
```

Дополнительно

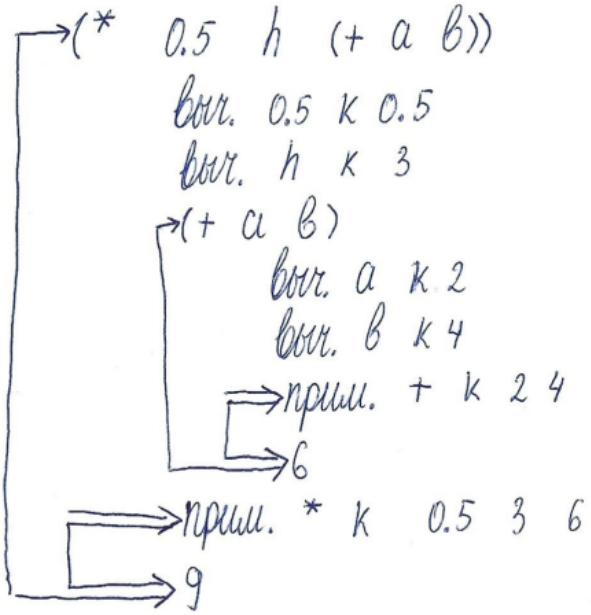
1. Написать функцию, вычисляющую катет по заданной гипотенузе и другому катету прямоугольного треугольника, и составить диаграмму ее вычисления.

```
(defun cathet (hyp another_cathet) (sqrt (- (* hyp hyp) (*
   another_cathet another_cathet))))
(cathet 5 4) => 3.0
```



2. Написать функцию, вычисляющую площадь трапеции по ее основаниям и высоте, и составить диаграмму ее вычисления.

```
(defun trapezoid_area (a b h) (* 0.5 h (+ a b)))
(trapezoid_area 2 4 3) => 9.0
```


(* 0.5 h (+ a b))
выч. 0.5 к 0.5
выч. h к 3
(+ a b)
выч. a к 2
выч. b к 4
прим. + к 2 4
6
прим. * к 0.5 3 6
9