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 №3 по дисциплине «Функциональное и логическое программирование»

Тема Работа интерпретатора Lisp

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, функционалов и форм: встроенные — примитивные функции (atom, eq, cons, car, cdr), которые носят частичный характер; формы (quote, cond, lambda, eval); функционалы (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.

Функции всюду определены (то есть результат есть всегда), их аргументы и результаты – S-выражения.

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

Форма – функция, которая особым образом обрабатывает свои аргументы, т. е. требует специальной обработки.

2.Классификация функций

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

Классификаци базисных функций и функций ядра.

1. Селекторы: car и cdr (будут подробнее расссмотрены ниже).
2. Конструкторы: cons и list (будут подробнее расссмотрены ниже).
3. Предикаты – «логические» функции, позволяющие определить структуру элемента:
   - atom возвращает T, если значением её единственного аргумента является атом, иначе – NIL;
   - null возвращает T, если значение его аргумента – NIL (пустой список), иначе – NIL; listp возвращает T, если значением её аргумента является список, иначе – NIL; consp возвращает T, если значением её аргумента является структура, представленная в виде списковой ячейки, иначе – NIL.

4. Функции сравнения (принимают два аргумента, перечислены по мере роста «тщательности» проверки):

   - eq корректно сравнивает два символьных атома. Так как атомы не дублирутюся для данного сеанса работы, то фактически сравниваются соответсвующие указатели. Возвращает T, когда: 1) значением одного из аргументов является атом, и одновременно 2) значения аргументов равны (идентичны). В ином случае значением функции eq является NIL.
   - eql корректно сравнивает атомы и числа одинакового типа (синтетической формы записи). Например, (eql 1 1) вернет T, а (eql 1 1.0) – Nil, так как целое значение 1 и значение с плавающей точкой 1.0 являются представителями различных классов;
   - = корректно сравнивает только числа, причем числа могут быть разных типов. Например, и (= 1 1), и (= 1 1.0) вернет T;
   - equal работает идентично eql, но в дополнение умеет корректно сравнивать списки (считая списки эквивалентными, если они рекурсивно, согласно тому же equal, имеют одинаковую структуру и содержимое; считая строки эквивалентными, если они содержат одинаковые знаки);
   - equalp корректно сравнивает любые S-выражения.

3. Способы создания функций

Определение функций пользователя в Lisp-е возможно двумя способами.

- Базисный способ определения функции - использование λ-выражения (λ-нотации). Так создаются функции без имени.

  λ-выражение: (lambda λ-список форма), где λ-список – это формальные параметры функции (список аргументов), а форма – это тело функции.

  Вызов такой функции осуществляется следующим способом: (λ-выражение последовательность_форм), где последовательность_форм – это фактические параметры.

  Вычисление функций без имени может быть выполнено с использованием функционала apply: (apply λ-выражение последовательность_форм).

  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 па-

раметров для F. Значение функционала – результат применения F к этим фактическим параметрам.

- Другой способ определения функции – использование макро-определения defun:

  (defun имя_функции λ-выражение),

  или в облегченной форме:

  (defun имя_функции $(x_1, x_2, ..., x_k)$ (форма)), где $(x_1, x_2, ..., x_k)$ – это список аргументов.

  В качестве имени функции выступает символьный атом. Вызов именованной функции осуществляется следующим образом: (имя_функции последовательность_форм), где последовательность_форм – это фактические параметры

λ-определение более эффективно, особенно при повторных вычислениях.

Параметры функции, переданные при вызове, будут связаны с переменными в списке параметров из объявления функции. Еще один способ связывания формальных параметров с фактическими – использование функции let:

(let ((x1 p1) (x2 p2) ... (xk pk)) e),

где xi – формальные параметры, pi – фактические параметры (могут быть формами), e – формам (что делать).

4. Работа функций cond, if, and/or

cond

Общий вид условного выражения:

$(cond\ (p_1\ e_{11}\ e_{12}\ \ldots\ e_{1m_1})\ (p_2\ e_{21}\ e_{22}\ \ldots\ e_{2m_2})\ \ldots\ (p_n\ e_{n1}\ e_{n2}\ \ldots\ e_{nm_n})), m_i \geqslant 0, n \geqslant 1$

Вычисление условного выражения общего вида выполняется по следующим правилам:

1. последовательно вычисляются условия $p_1, p_2, \ldots p_n$ ветвей выражения до тех пор, пока не встретится выражение p_i, значение которого отлично от NIL;

2. последовательно вычисляются выражения-формы $e_{i1}\ e_{i2}\ \ldots\ e_{im_i}$ соответствующей ветви, и значение последнего выражения e_{im_i} возвращается в качестве значения функции cond;

3. если все условия p_i имеют значение NIL, то значением условного выражения становится NIL.

Ветвь условного выражения может иметь вид (p_i), когда $m_i = 0$. Тогда если значение pi $\neq$ NIL, значением условного выражения cond становится значение pi.

В случае, когда pi $\neq$ NIL и $m_i \geqslant 2$, то есть ветвь cond содержит более одного выражения e_i, эти выражения вычисляются последовательно, и результатом cond служит значение последнего из них e_{im_i}. Таким образом, в дальнейших вычислениях может быть использовано только значение последнего выражения, и при строго функциональном программировании случай $m_i \geqslant 2$ обычно не возникает, т.к. значения предшествующих e_{im_i} выражений пропадают.

Использование более одного выражения e_i на ветви cond имеет смысл тогда, когда вычисление предшествующих e_{im_i} выражений даёт побочные эффекты, как при вызове функций ввода и вывода, изменении списка свойств атома, а также определении новой функции с помощью defun.

К примеру:

(cond ((< X 5)(print "Значение x меньше пяти") X) ((= X 10)(print "Значение x равно 10") X) (T(print "Значение x больше пяти, но не 10")X))

Значением этого условного выражения всегда будет значение переменной X, но при этом на печать будет выведена одна из трёх строк, в зависимости от текущего значения X.

if Макрофункция (If C E1 E2), встроенная в MuLisp и Common Lisp, вычисляет значение выражения E1, если значение выражения C отлично от NIL, в ином случае она вычисляет значение E2:

(defmacro If (C E1 E2) (list 'cond (list C E1) (list T E2)))

Этот макрос строит и вычисляет условное выражение cond, в котором в качестве условия первой ветви берётся выражение C (первый аргумент If), а выражения E1 и E2 (второй и третий аргумент If) размещаются соответственно на первой и второй ветви cond.

К примеру, для макровызова (If (numberp K) (+ K 10) K) на этапе макрорасширения будет построена конструкция (cond ((numberp K) (+ K 10) (T K)), а на этапе её вычисления в случае K=5 будет получено значение 15.

and/or

К логическим функциям-предикатам относят логическое отрицание not, конъюнкцию and и дизъюнкцию or. Первая из этих функций является обычной, а другие две – особыми, поскольку допускают произвольное количество аргументов, которые не всегда вычисляются.

Логическое отрицание not вырабатывает соответственно: (not NIL) => T и (not T) => NIL, и может быть определено функцией (defun not (x) (eq x NIL)).

Фактически действие этой функции эквивалентно действию функции null, работающей не только с логическими значениями T и NIL, но и с произвольными лисповскими выражениями. Поэтому, например: (not '(B ())) => NIL

Тем самым, определение функции not соответствует лисповскому расширенному пониманию логического значения истина.

Две другие встроенные логические функции также используют расширенное понимание истинного значения.

Вызов функции and, реализующей конъюнкцию, имеет вид (and e1 e2 ... en), n $\geqslant$ 0.

При вычислении этого функционального обращения последовательно слева направо вычисляются аргументы функции ei – до тех пор, пока не встретится значение, равное NIL. В этом случае вычисление прерывается и значение функции равно NIL. Если же были вычислены все значения ei и оказалось, что все они отличны от NIL, то результирующим значением функции and будет значение последнего выражения en .

Вызов функции-дизъюнкции имеет вид (or e1 e2 ... en), n $\geqslant$ 0.

При выполнении вызова последовательно вычисляются аргументы ei (слева направо) – до тех пор, пока не встретится значение ei, отличное от NIL. В этом случае вычисление прерывается и значение функции равно значению этого ei. Если же вычислены значения всех аргументов ei, и оказалось, что они равны NIL, то результирующее значение функции равно NIL.

При n=0 значения функций: (and)=>T, (or)=>NIL.

Таким образом, значение функции and и or не обязательно равно T или NIL, а может быть произвольным атомом или списочным выражением.

Из указаний к выполнению работы

проанализировать эффективность работы разных реализаций.

Практические задания

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

```
(defun f1 (x) (cond ((= (rem x 2) 1) (+ x 1)) (x)))
; or shorter
; (defun f1 (x) (if (oddp x) (+ x 1) x))
(f1 0) => 0
(f1 1) => 2
(f1 -3) => -2
```

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

```
(defun f2 (x) (+ x (cond ((< x 0) -1) (1))))
; or shorter
; (defun f2 (x) (+ x (if (< x 0) -1 1)))
(f2 -5) => -6
(f2 0.0) => 1.0
(f2 2/3) => 5/3
```

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

```
(defun f3 (x1 x2) (if (> x1 x2) (list x2 x1) (list x1 x2)))
(f3 -1 2) => (-1 2)
(f3 3 1) => (1 3)
(f3 2 2.0) => (2 2.0)
```

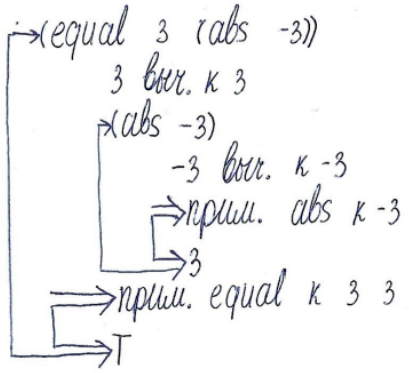
4. Написать функцию, которая принимает три числа и возвращает T только тогда, когда первое число расположено между вторым и третьим.

```
(defun f4 (x1 x2 x3) (and (< x2 x1) (< x1 x3)))
(f4 2 1 3) => T
(f4 1 2 3) => NIL
(f4 3 1 2) => NIL
(f4 1 1 2) => NIL
```

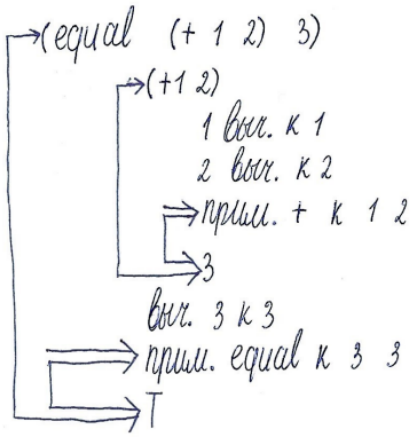
5. Каков результат вычисления следующих выражений?


```
(defun f4 (x1 x2 x3) (and (< x2 x1) (< x1 x3)))
(f4 2 1 3) => T
(f4 1 2 3) => NIL
(f4 3 1 2) => NIL
(f4 1 1 2) => NIL
```

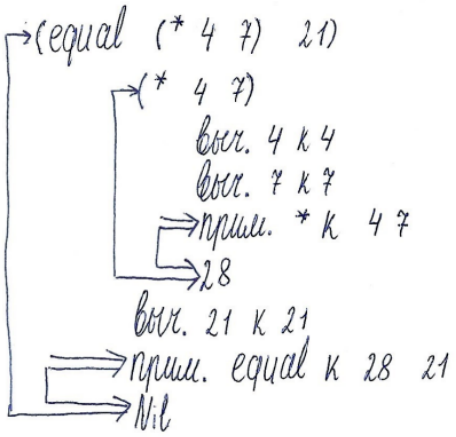

1. (equal 3 (abs - 3))



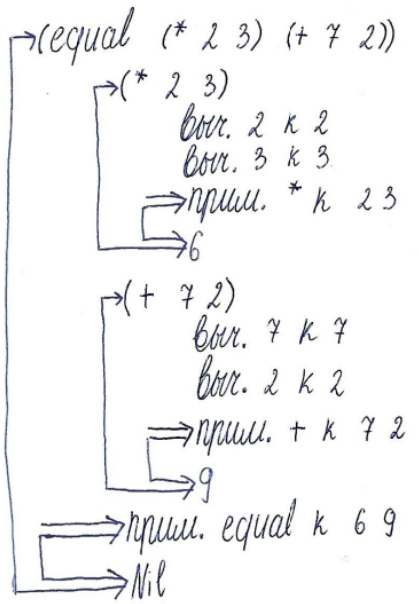
2. (equal (+ 1 2) 3)



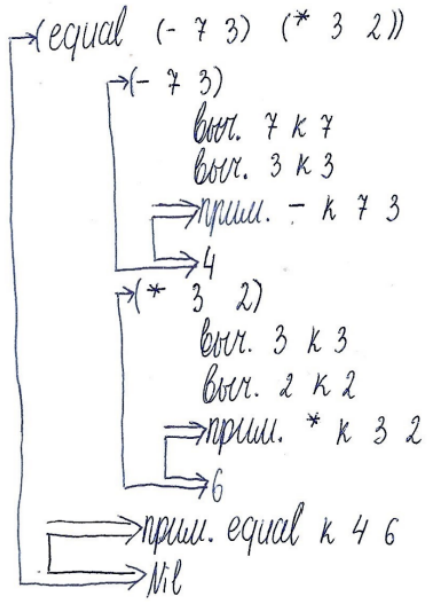
3. (equal (* 4 7) 21)


(equal (* 4 7) 21)
(* 4 7)
выч. 4 к 4
выч. 7 к 7
прим. * к 4 7
28
выч. 21 к 21
прим. equal к 28 21
Nil

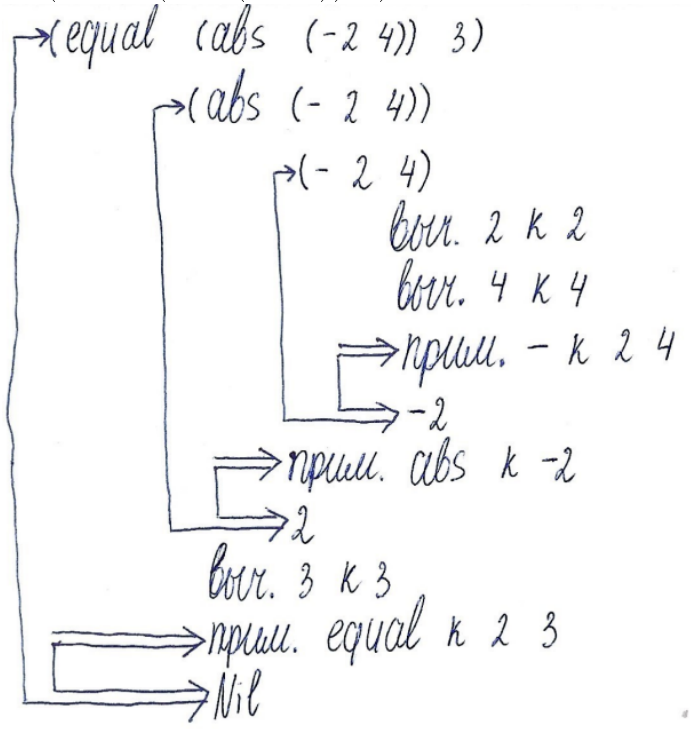
4. (equal (* 2 3) (+ 7 2))



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

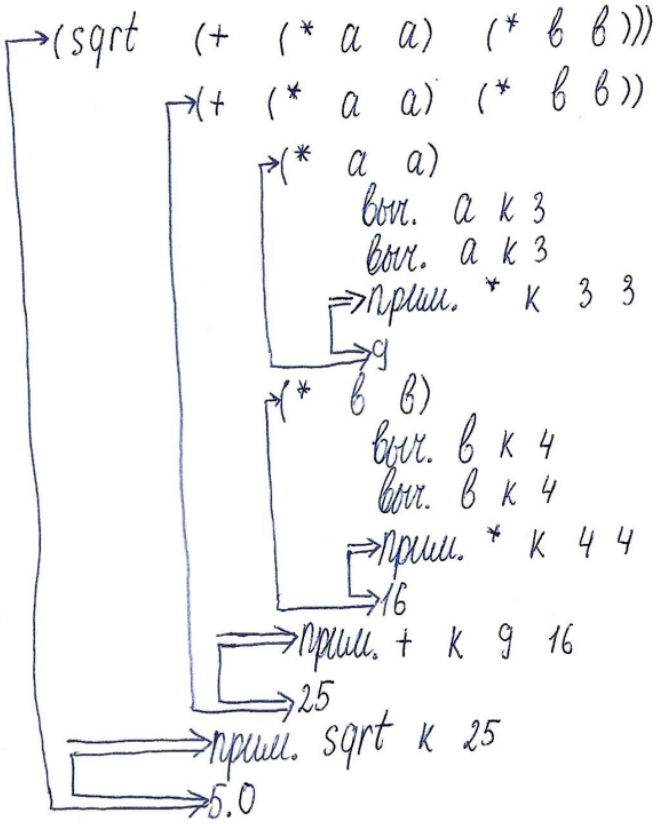


6. (equal (abs (- 2 4)) 3)



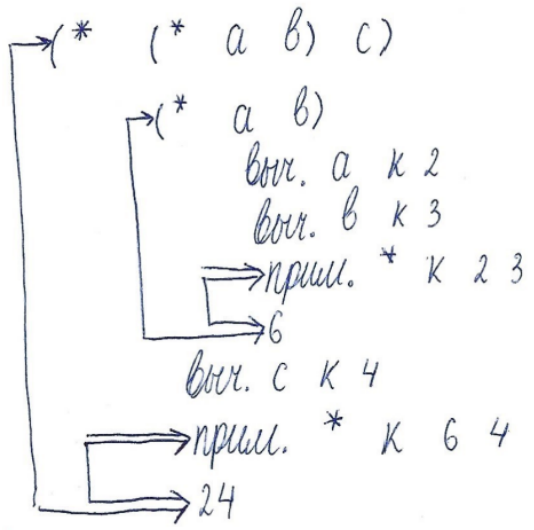
2. Написать функцию, вычисляющую гипотенузу прямоугольного треугольника по заданным катетам, и составить диаграмму её вычисления.

```
1 (defun hypotenuse (a b) (sqrt (+ (* a a) (* b  b))))
2 (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 c не связана со своимм значением. Решение: определить функцию b с одним аргументом (или переменным количеством аргументов), а переменной c 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 c 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?

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"

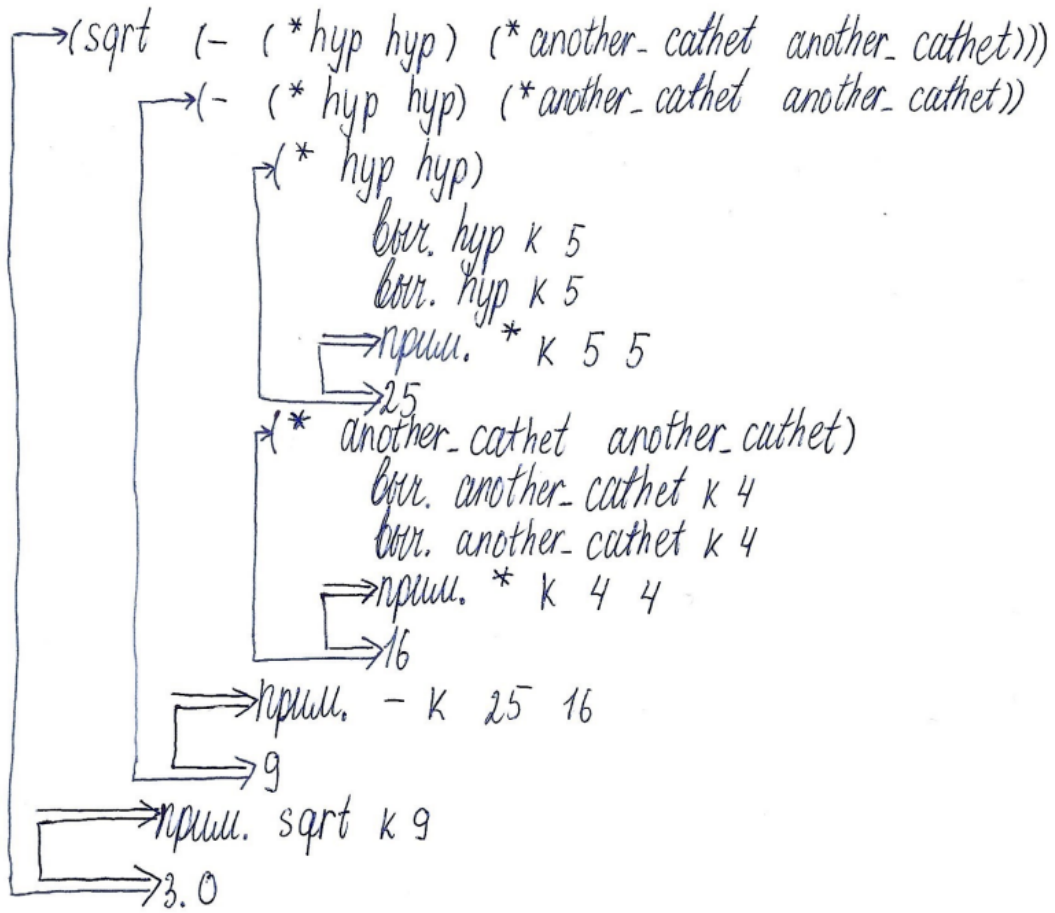
9. Что получится при вычисления каждого из выражений?

```
(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
```