Функциональное и логическое программирование

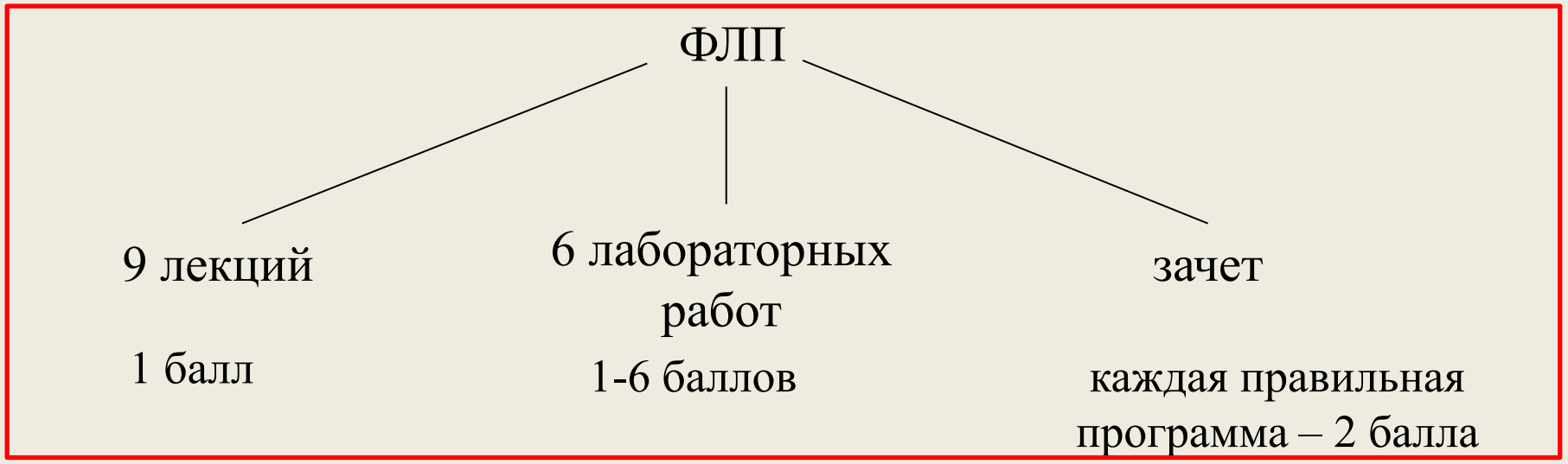
Лекция 1

Лектор: Галкина Марина Юрьевна

Практические занятия ведут преподаватели:
группы ИП311-314 Галкина Марина Юрьевна
группы ИП315-317 Новожилов Дмитрий Иванович

Ссылка на курс в ЭИОС https://eios.sibsutis.ru/course/view.php?id=672

Кодовое слово: ИП<номер группы>_очное.
Например, для ИП311 кодовым словом является ИП311_очное.



На каждой лекции студенты должны самостоятельно отметить посещаемость через ЭИОС в течение 20 минут после начала лекции.

Если студент забыл или не смог отметиться самостоятельно, то на перерыве необходимо подойти к лектору и попросить отметить.

Если обнаруживается, что студент отметился в ЭИОС, но при этом отсутствовал на лекции, все предыдущие баллы, начисленные за посещения лекций, обнуляются.

9 лекций	6 лабораторных работ	зачет
1 балл	1-6 баллов	каждая правильная программа – 2 балла

Максимальное количество баллов, которое может получить студент в течение семестра: 9+6·6=45.

Зачет автоматом можно получить, если к концу семестра заработать не менее 39 баллов, при этом необходимо защитить все лабораторные работы.

Если студент набрал менее 39 баллов, то для получения зачета необходимо будет решать дополнительные задачи до тех пор, пока не будет набрано 39 баллов (одна правильно решенная задача – 2 балла).

Все полученные баллы можно смотреть в ЭИОС по ссылке Рейтинги.

Введение.
Классификация языков программирования

- Процедурные (операторные);

  Бейсик, Паскаль, Си
- Непроцедурные:
  - Объектно-ориентированные;

  Object Pascal, C++, C#, Java, Python, Ruby
  - Декларативные:
    - функциональные (Lisp, Haskell, Erlang);
    - логические (Planner, Prolog).

Глава 1. Функциональное программирование. Основы языка Lisp

Язык Lisp (List processing) был разработан в Америке Дж.Маккарти и ориентирован на символьную обработку. Дж.Маккарти ввел термин «Искусственный интеллект».

Разработчики Lisp считали, что язык отражает устройство человеческого мозга и на нём можно закодировать любой интеллектуальный процесс.

Список проектов, в которых используются диалекты Lisp

- GNU Emacs (текстовый редактор написан на Emacs Lisp).
- Grammarly (онлайн-сервис для проверки качества англоязычных текстов).
- Piano (пакет программ, который используют Boeing и Airbus).
- ПО Лондонского метрополитена.
- AutoCAD.
- Apache Storm (система обработки больших данных в реальном времени).

Основа Lisp - лямбда-исчисление Черча, формализм для представления функций и способов их комбинирования.

Свойства Lisp:

- Однообразная форма представления программ и данных.
- Использование в качестве основной управляющей конструкции рекурсии.
- Широкое использование данных «список» и алгоритмов их обработки.

Достоинства и недостатки Лиспа

Достоинство: простота синтаксиса.

Недостатки:

- большое кол-во вложенных скобок

  (Lisp - Lots of Idiotic Silly Parentheses);

- множество диалектов.

GNU Clisp 2.49

Реализован немецкими студентами Бруно Хайбле (Bruno Haible) и Михаэлем Штоллем (Michael Stoll). Он соответствует ANSI Common Lisp стандарту, работает под Unix, Windows.

Будем использовать онлайн-интерпретаторы. Например,

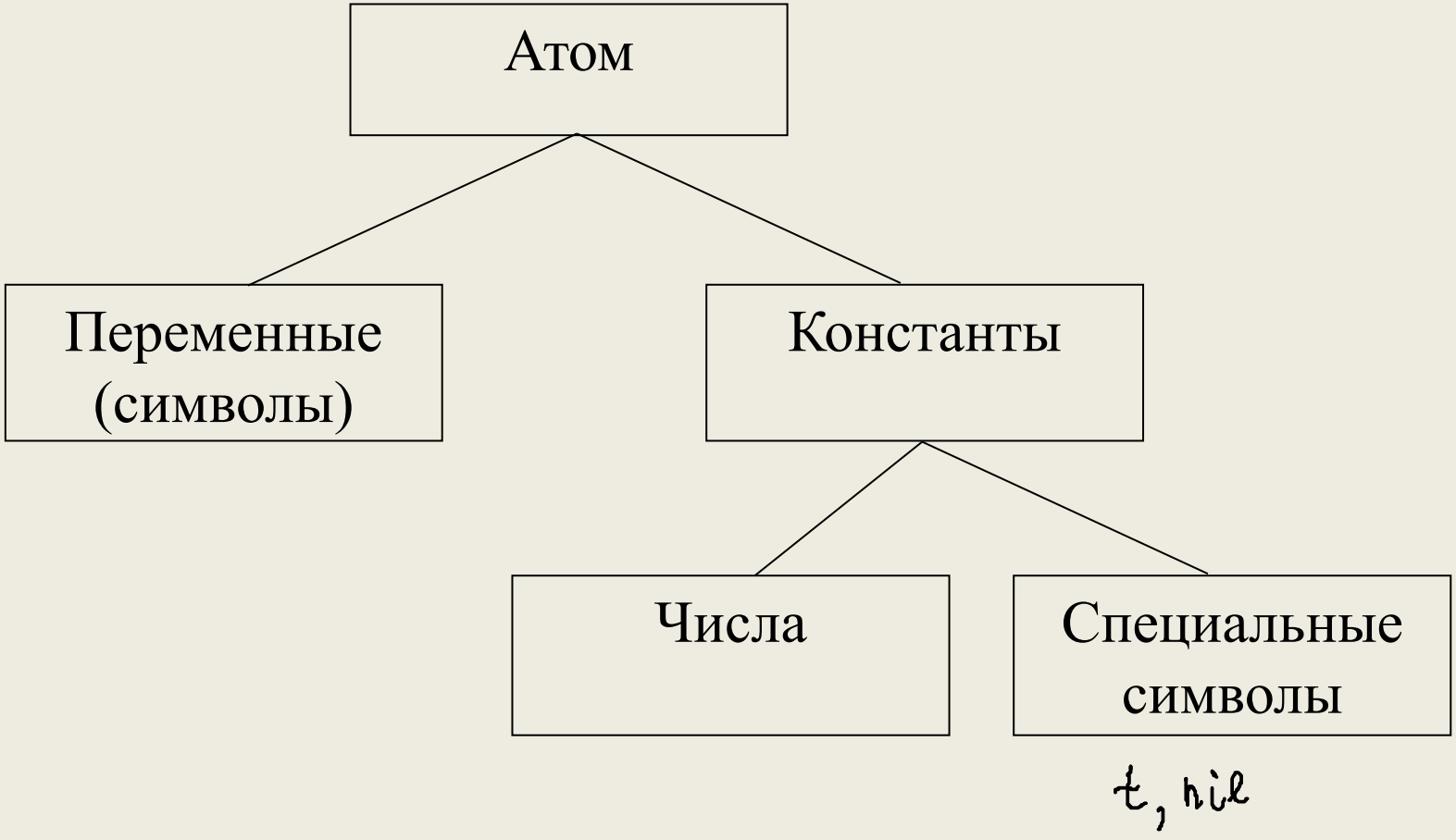
https://rextester.com/l/common_lisp_online_compiler

https://www.jdoodle.com/execute-clisp-online

https://ideone.com/l/common-lisp-clisp

1.1 Типы данных в Lisp

Типы данных: атомы, списки, точечные пары.



Переменная – это последовательность из букв, цифр и специальных знаков. Переменные представляют другие объекты: числа, другие символы, функции.

Специальные символы: t и nil. Символ t обозначает логическое значение истина, а nil – имеет два значения: логическое значение ложь или пустой список.

Списком называется упорядоченная последовательность, элементами которой являются атомы или списки. Список заключается в скобки, а элементы разделяются пробелами.

Примеры списков:

(+ - * 3 4)

((1 2) a)

(((a b)))

() } пустой список

nil }

Атомы, списки – s-выражения.

1.2 Функции

Вызов функции записывается в префиксной нотации. Сначала идет имя функции, затем аргументы функции через пробел и все заключается в скобки.

$f(x, y, z) \Leftrightarrow (f\ x\ y\ z)$

Пример 1:

$5 * 8 \Leftrightarrow (*\ 5\ 8)$

Суперпозиция функций выполняется «изнутри наружу»!

Пример 2:

$3 + 7 * 9 \Leftrightarrow (+\ 3\ (*\ 7\ 9))$

По внешнему виду функция и список не различаются!

Чтобы выражение в скобках воспринималось как список используется специальная функция **QUOTE**. Эта функция блокирует вычисления и соответствует математической функции f(x)=x. Причем, значение аргумента не вычисляется.

(**QUOTE** x) можно записать как 'x

Пример 3:

(+ 1 3) → 4

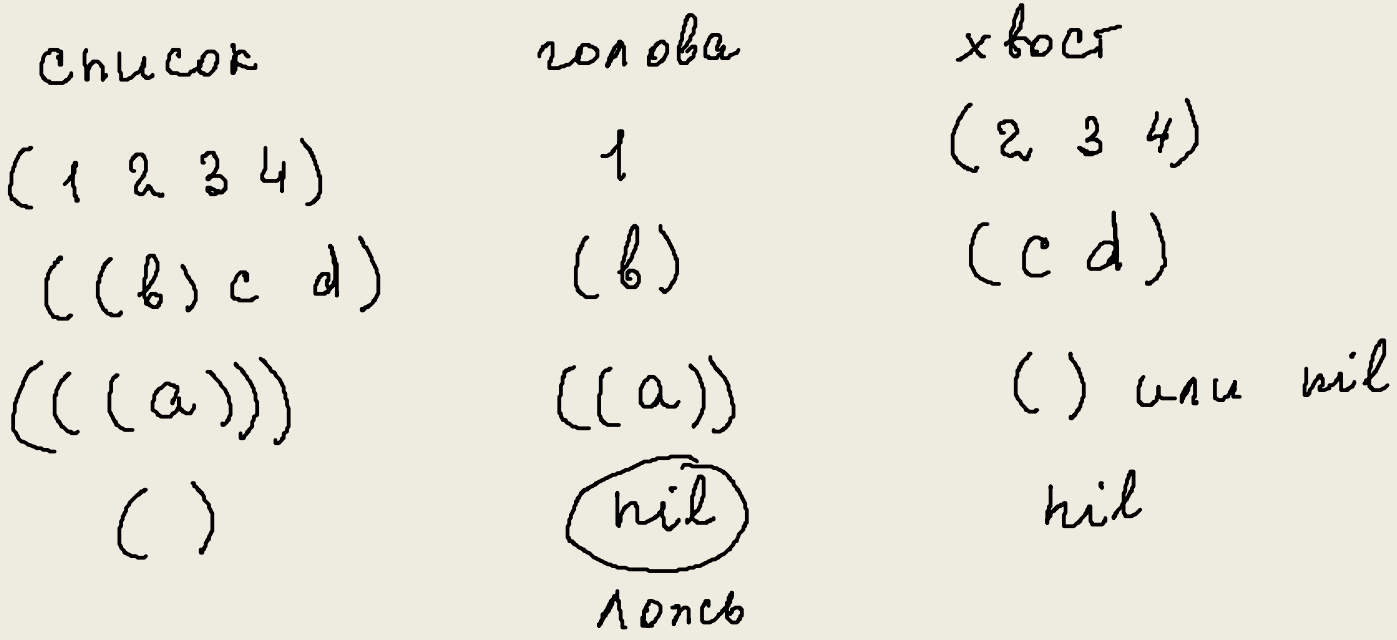
'(+ 1 3) → (+ 1 3)

'(+ 1 (* 4 5)) → (+ 1 (* 4 5))

Самая левая **QUOTE** блокирует все вычисления правее ее!

1.2.1 Функции обработки списков

Разделим список на голову и хвост. *Головой* назовем первый элемент списка, а *хвостом* – список без первого элемента. Пример 4:



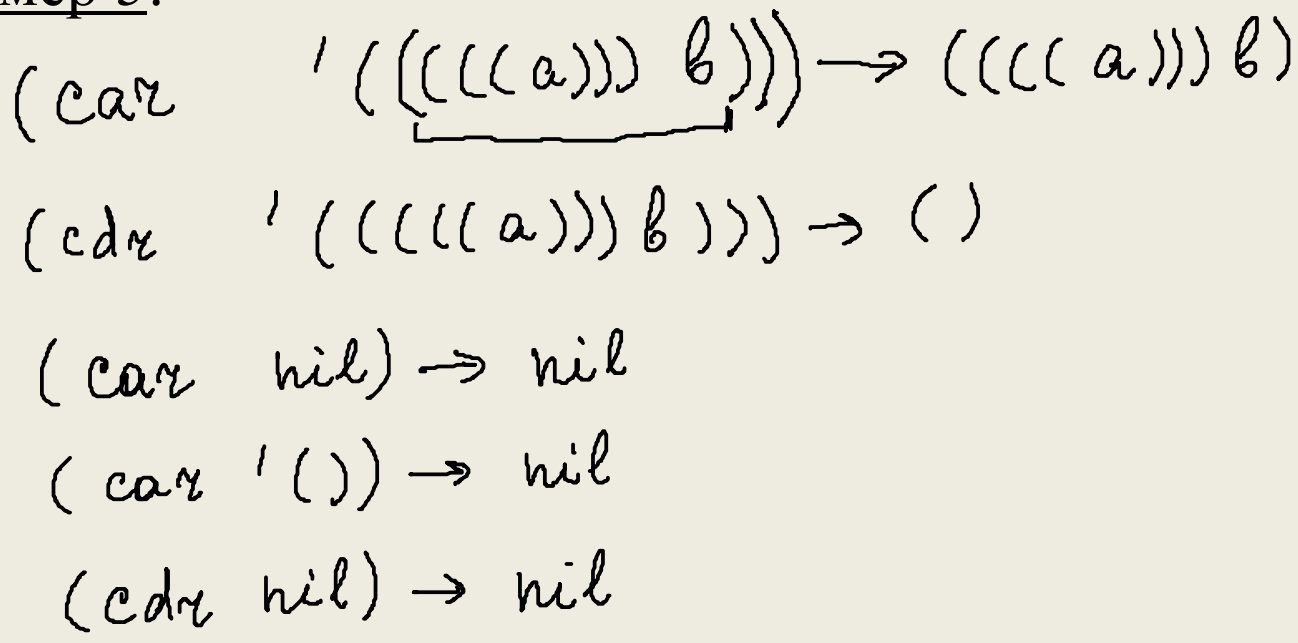
(CAR список)

Возвращает голову списка

(CDR список)

Возвращает хвост списка

Пример 5:



Последовательно применяя функции **CAR** и **CDR** можно выделить любой элемент списка.

Пример 6**:** Выделить в списке ((a b c) (d e) (f)) элемент c.

(print (car(cdr(cdr(car '((a b c)(d e)(f)))))))

Допускаются сокращения, но подряд не может идти больше четырех букв A и D.

Для этого примера можно сделать так:

(print (caddar '((a b c)(d e)(f))))

Пример 7**:** Выделить в списке (1(2 3 ((4 *) 5) 6)) элемент *.

(print (cadaar(cddadr '(1(2 3((4 *)5)6)))))

(**CONS** s-выражение список)

Возвращает список, головой которого является первый аргумент функции, а хвостом – второй аргумент функции.

Пример 8:

(cons (+ 3 8) nil) → (11)

(cons '(+ 3 8) '(a)) → ((+ 3 8) a)

(cons 'a 'b) → (a . b) – точечная пара

Функции **CAR** и **CDR** являются обратными для **CONS**

(**LIST** s_1 … s_n)

Возвращает список, элементами которого являются аргументы функции, где s_i – s-выражение.

Пример 9:

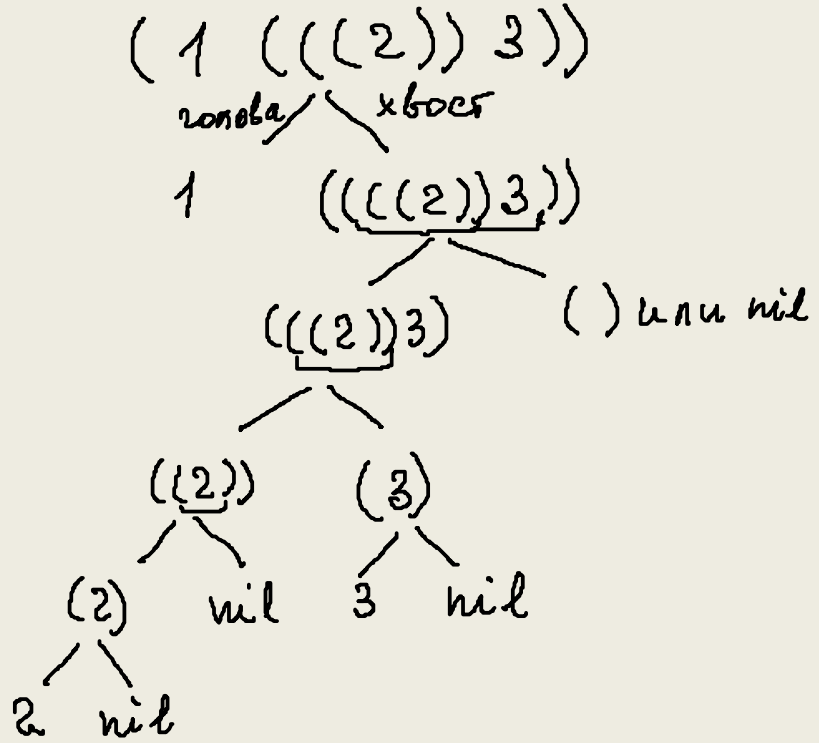
(list '(a b c) 1 'a) → ((a b c) 1 a)

(list (* 8 9) nil) → (72 nil)

Пример 10: Из атомов 1, 2, 3, nil создать список (1 (((2)) 3)) двумя способами:

а) с помощью композиций функций **CONS**;

б) с помощью композиций функций **LIST**.

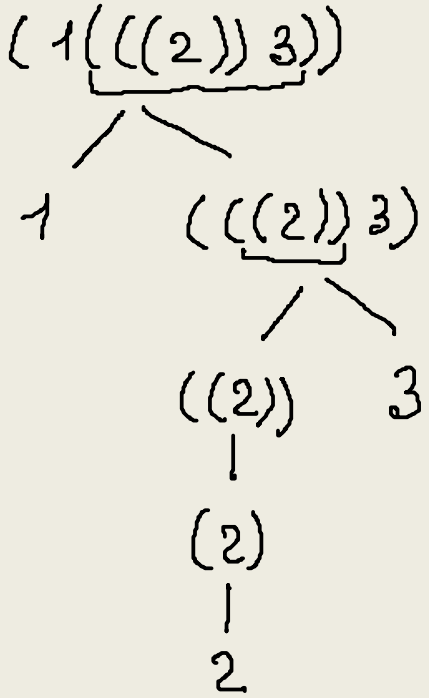


(cons 1 (cons (cons (cons (cons 2 nil) nil) (cons 3 nil)) nil))

(print (cons 1 (cons(cons(cons(cons 2 nil) nil)(cons 3 nil))nil))) --> (1 (((2))3))

(1 (((2)) 3))

б) с помощью композиций функций **LIST:**



(list 1 (list (list (list 2)) 3))

(print (list 1(list(list(list 2))3))) → (1 (((2)) 3))

(**APPEND** $sp_1 \ldots sp_n$)

Возвращает список элементами которого являются элементы списков - аргументов функции, где sp_i – список.

Пример 11:

(append '((a b) c) '(3 4)) → ((a b) c 3 4)

(append '(1 2 3) nil) → (1 2 3)

(**LAST** список)

Возвращает список из одного элемента: последнего элемента списка – аргумента функции.

Пример 12:

(last '(a b (c))) → ((c))

(last '(a b c)) → (c)

(**BUTLAST** список)

Возвращает список из всех элементов списка-аргумента, кроме последнего.

Пример 13:

(butlast '(a b (c))) → (a b)

(butlast '(a b c)) → (a b)

(**REVERSE** список)

Возвращает перевернутый список-аргумент (переворачивание только на верхнем уровне!).

Пример 14:

(reverse '((a b c) d e)) → (e d (a b c))

1.3 Определение функций пользователем

1.3.1 Лямбда-функции

Основа определения и вычисления функций – лямбда-исчисление Черча (формализм описания функций).

Для описания функции используется *лямбда-выражение*:

(**LAMBDA** (x_1 x_2 … x_n) S_1 S_2 … S_k), где

Формальные параметры

s-выражения, образуют тело функции

Список формальных параметров называется *лямбда-списком.*

Лямбда-выражение соответствует определению функции.

Пример 1: $f(x,y,z) = x + y : z$

```
(lambda (x y z) (+ x (/ y z)))
```

Лямбда-выражение нельзя вычислить, оно не имеет значения. Но можно организовать *лямбда-вызов* (соответствует вызову функции):

(лямбда-выражение a_1 a_2 … a_n),

где a_1, a_2, …, a_n – вычислимые s-выражения, задающие вычисления фактических параметров.

Пример 2: f(2, 7, 7) - ?

((lambda (x y z) (+ x (/ y z))) 2 7 7) -> 3

2 этапа вычисления лямбда-вызова:

1. Вычисляются значения фактических параметров и соответствующие формальные параметры связываются с полученными значениями.
2. С учетом новых связей вычисляется тело функции, и последнее вычисленное значение возвращается в качестве значения лямбда-вызова.

После завершения лямбда-вызова фактические параметры получают те связи, которые были у них до вычисления лямбда-вызова, т.е. происходит передача параметров по значению.

Лямбда-вызовы можно ставить как на место тела функции, так и на место фактических параметров.

Лямбда-выражение является чисто абстрактным механизмом для определения и описания вычислений. Это безымянная функция, которая пропадает сразу после вычисления значения лямбда-вызова. Ее нельзя использовать еще раз, т.к. она не имеет имени.

1.3.2 Определение функций с именем

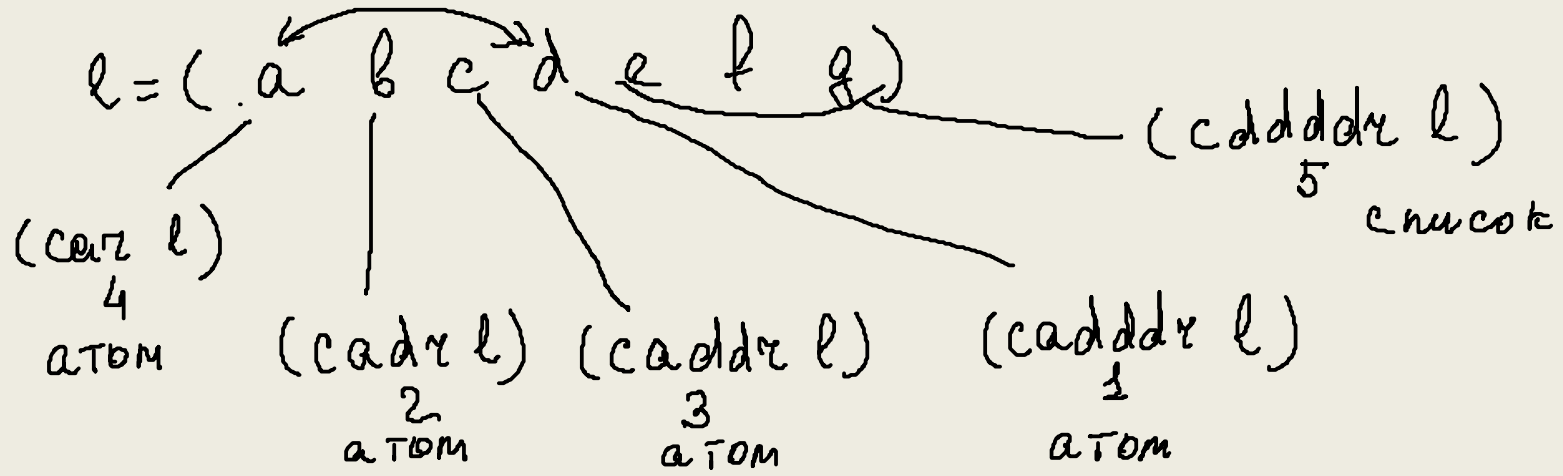
(**DEFUN** имя-функции лямбда-список S_1 S_2 … S_k) возвращает имя функции.

Побочный эффект: связывание символа-имени функции с лямбда-выражением:
(**LAMBDA** лямбда-список S_1 S_2 … S_k).

После такого определения можно обращаться к функции по имени.

Пример 3:

Определить функцию, которая меняет местами первый и четвертый элементы произвольного списка.




```
(append (list (cadddr l)
              (cadr l)
              (caddr l)
              (car l))
        (cddddr l))
```

```
(defun f(l)
  (append (list (cadddr l)
                (cadr l)
                (caddr l)
                (car l))
          (cddddr l))
)
(print(f '(a b c d e f g)))
```

→ (d b c a e f g)