Функциональное и логическое программирование

Лекция 1

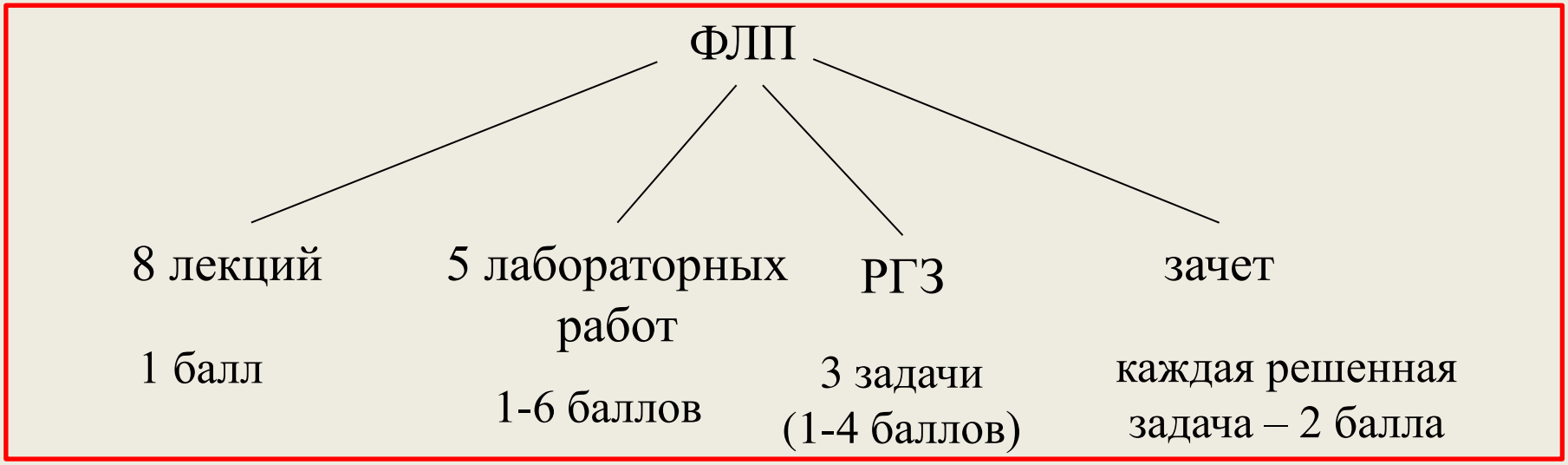
Лектор: Галкина Марина Юрьевна

Практические занятия ведет преподаватель:
Сороковых Дарья Анатольевна

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

Кодовое слово для групп ИВ: ИВ-<номер группы>_очное.
Кодовое слово для групп ИС: ИС-<номер группы>_очное.

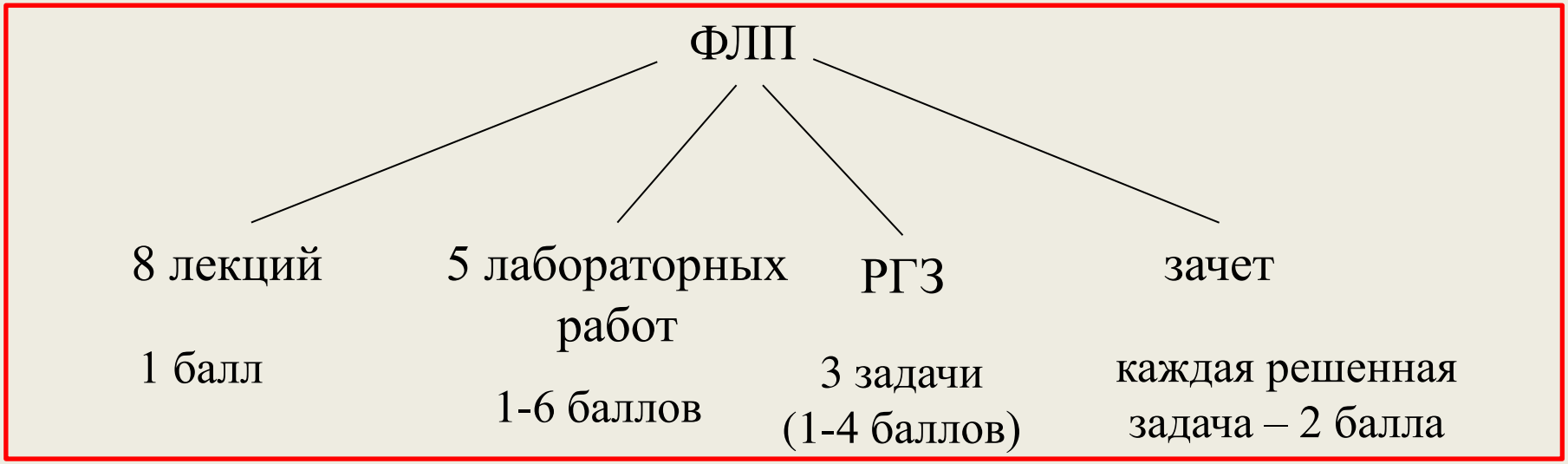
Например, для ИВ-121 кодовым словом является ИВ-121_очное.



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 начисленные за посещение лекций баллы обнуляются.



Максимальное количество баллов, которое может получить студент в течение семестра: $8+6\cdot5+10=48$.

Зачет автоматом можно получить, если к концу семестра заработать не менее 36 баллов:

42-48 баллов, должны быть сданы: все лабы + 2 задачи РГЗ;

36-41 баллов, должны быть сданы: все лабы + 3 задачи РГЗ.

Если студент набрал менее 36 баллов, то для получения зачета должны быть сданы: все лабы + 3 задачи РГЗ + зачет (добираем баллы до 36, одна решенная задача – 2 балла).

Все заработанные 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 в 1961 году (ориентирован на символьную обработку).

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

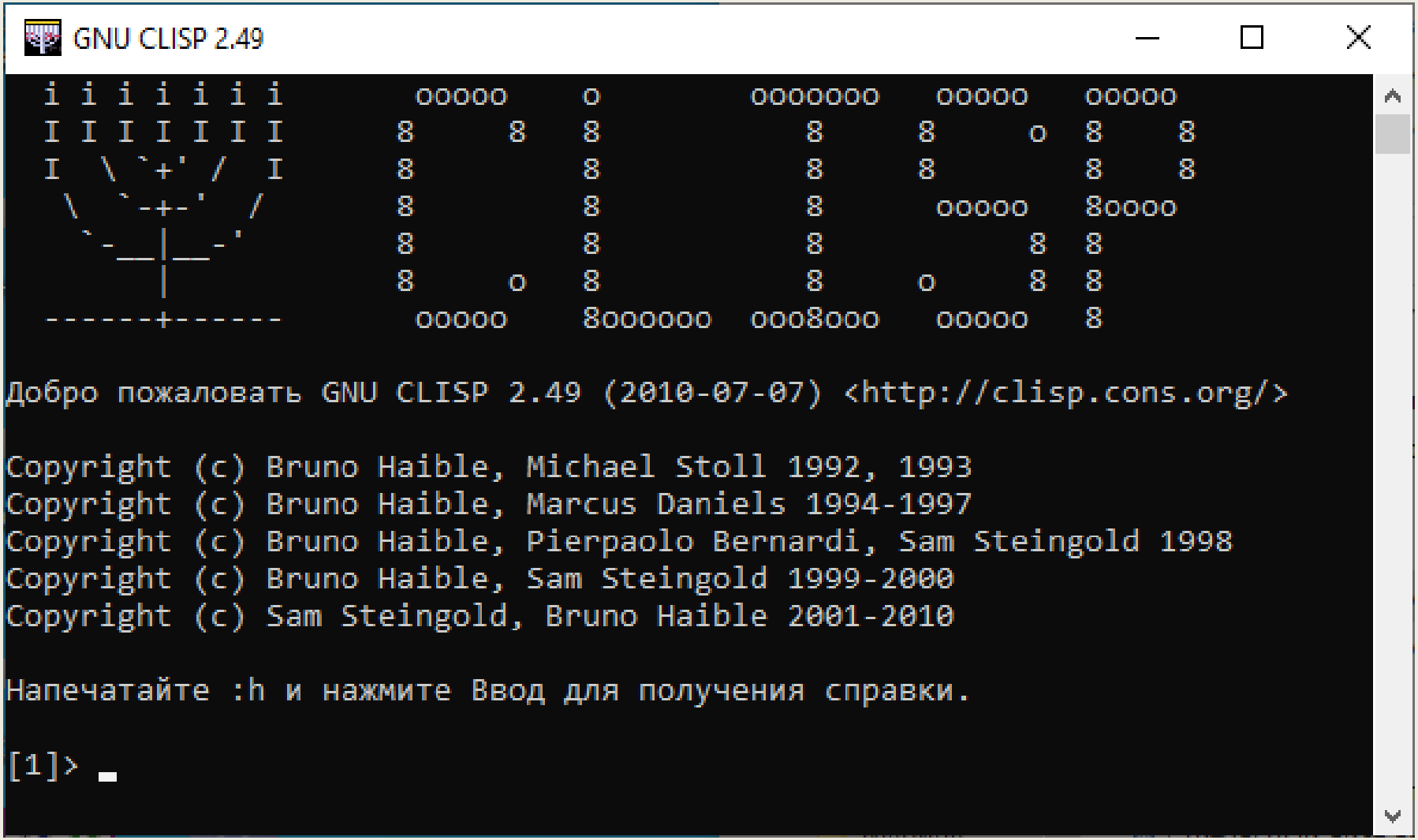
  (Lisp - Lots of Idiotic Silly Parentheses);

- множество диалектов.

GNU Clisp 2.49

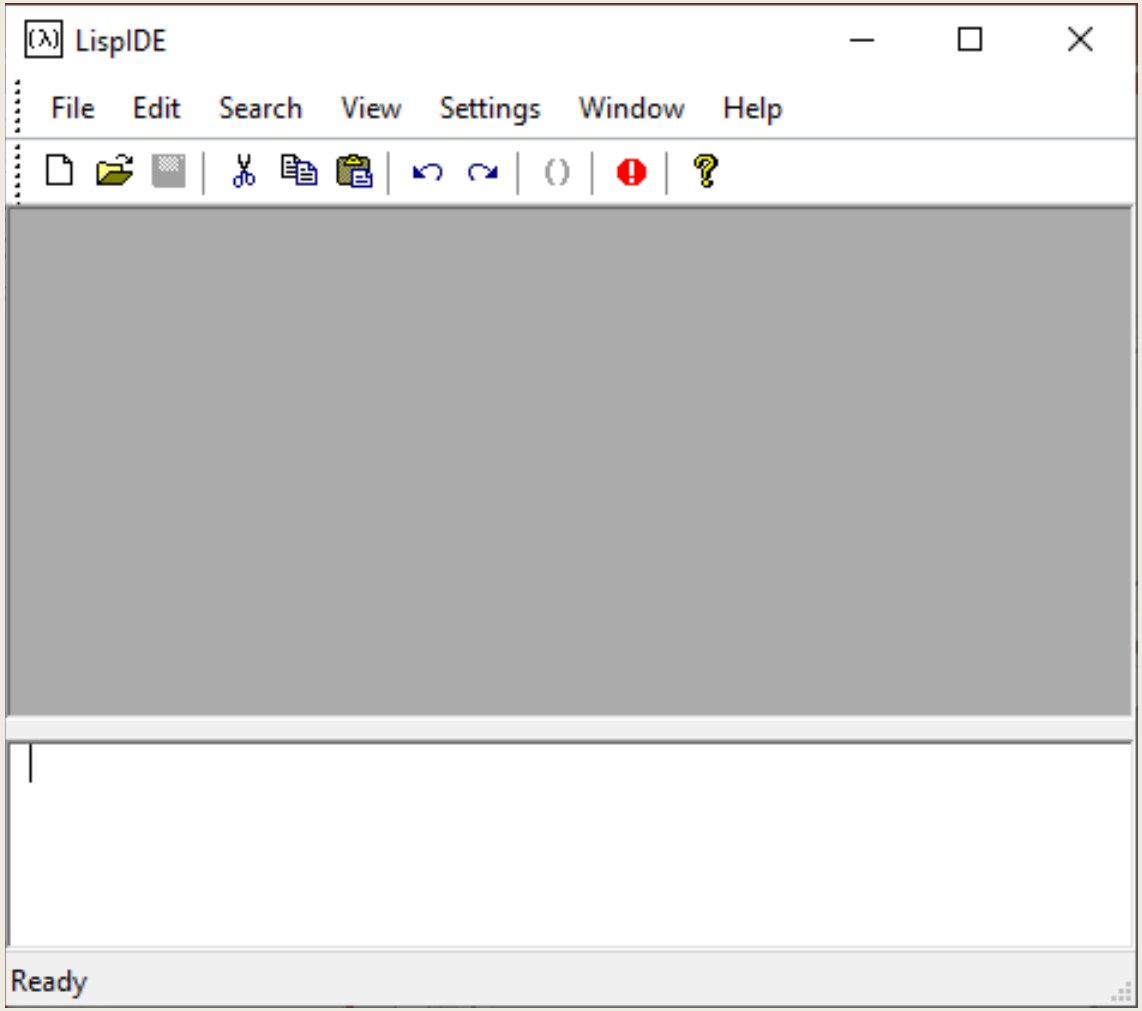
Реализован немецкими студентами Бруно Хайбле (Bruno Haible) и Михаэлем Штоллем (Michael Stoll). Он соответствует ANSI Common Lisp стандарту, работает под Unix, Windows.

Запуск интерпретатора: clisp.exe.

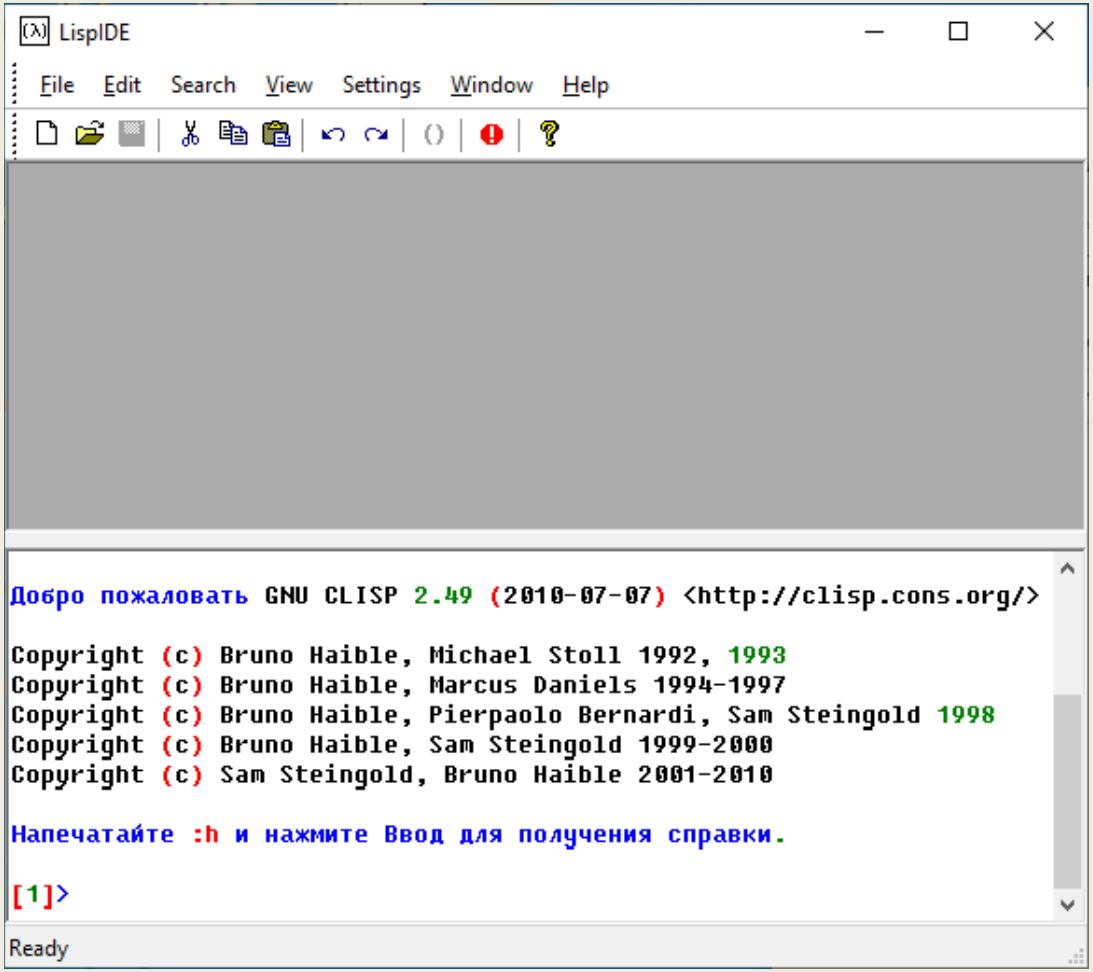

GNU CLISP 2.49
Добро пожаловать GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>
Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010
Напечатайте :h и нажмите Ввод для получения справки.
[1]>

Редактор LispIDE (работает под Windows)

При его первом запуске (файл LispIDE.exe), запрашивается имя файла, который запускает интерпретатор Lisp.



После запуска редактора можно установить путь к интерпретатору выполнив команду Setting - Set List Path.



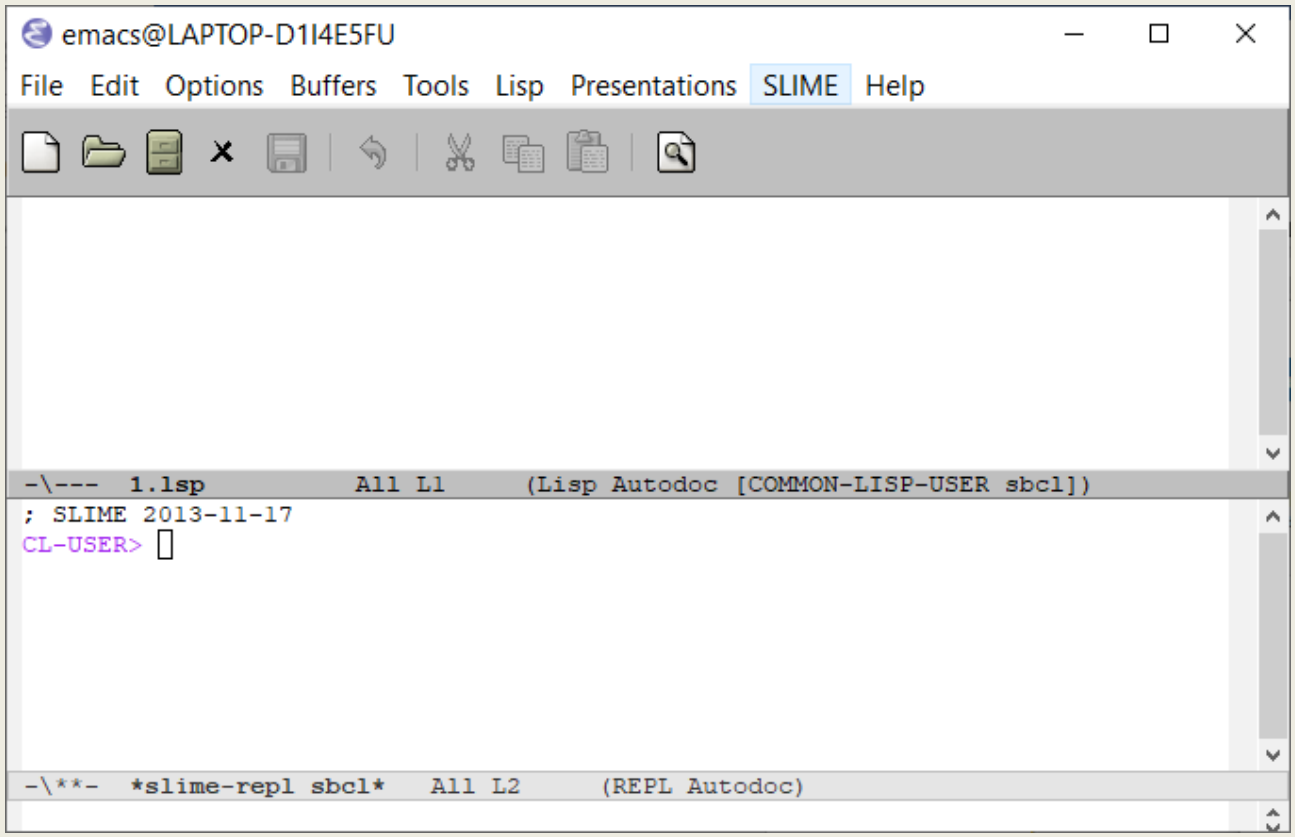
Иконки на панели инструментов:
() – отправить интерпретатору выделение до парной открывающейся скобки (если курсор стоит после закрывающейся скобки);
! – перезапустить интерпретатор;

Сохранение только в англоязычные папки (и сам путь к папке должен быть англоязычным).

Для работы под операционной системой Linux можно использовать связку Emacs+Slime+SBCL.

Slime (Superior Lisp Interaction Mode for Emacs) — режим Emacs для разработки приложений на Common Lisp).

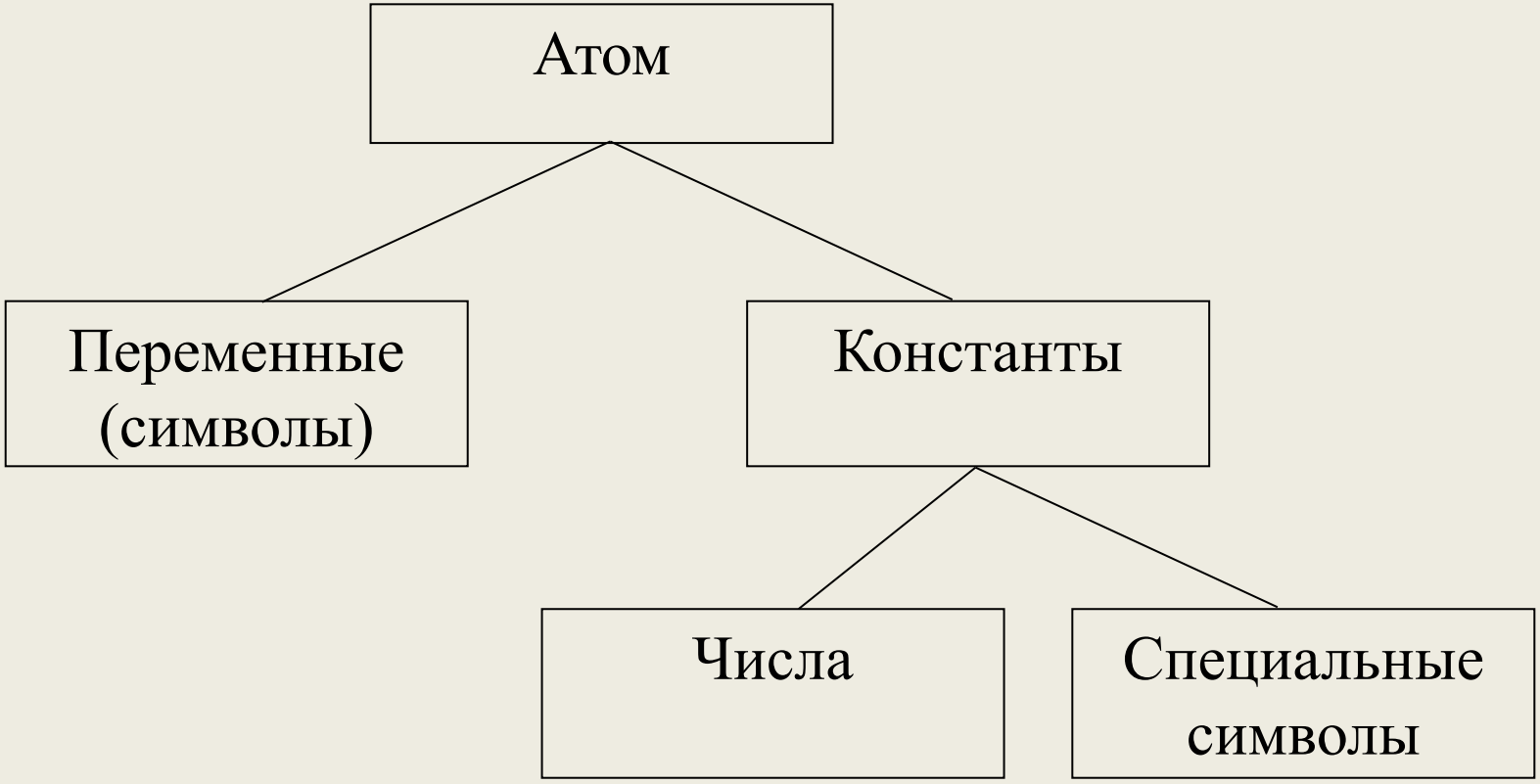
SBCL (Steel Bank Common Lisp) — свободная реализация языка программирования Common Lisp).



Допускается работа с любым online интерпретатором.

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:

(a b c) – список из 3-х эл-тов

(a (b c)) – список из 2-х эл-тов

() <=> 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.

$f(x, y) <=> (f\ x\ y)$

Пример 1:

$(3 + 2) * x <=> (* (+ 3\ 2)\ x)$

Суперпозиция функций выполняются «изнутри наружу»!

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 2:

$'(1\ (+\ 4\ 5)) \longrightarrow (1\ (+\ 4\ 5))$

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

1.2.1 Арифметические функции

- +
- -
- *
- /
- ABS

Пример 3:

1.2.2 Функции обработки списков

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

((a b) c d)

голова (a b)
хвост (c d)

((a))

голова (a)
хвост ( ) <=> nil

(**CAR** список)

Возвращает голову списка

(**CDR** список)

Возвращает хвост списка

Пример 5:

(car '(1 (2 3))) → 1

(cdr '(1 (2 3))) → ((2 3))

(car nil) → nil

(cdr nil) → nil

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

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

((a b c) (d e) (f)) $\xrightarrow{car}$ (a b c) $\xrightarrow{cdr}$ (b c) $\rightarrow$

$\xrightarrow{cdr}$ (c) $\xrightarrow{car}$ c

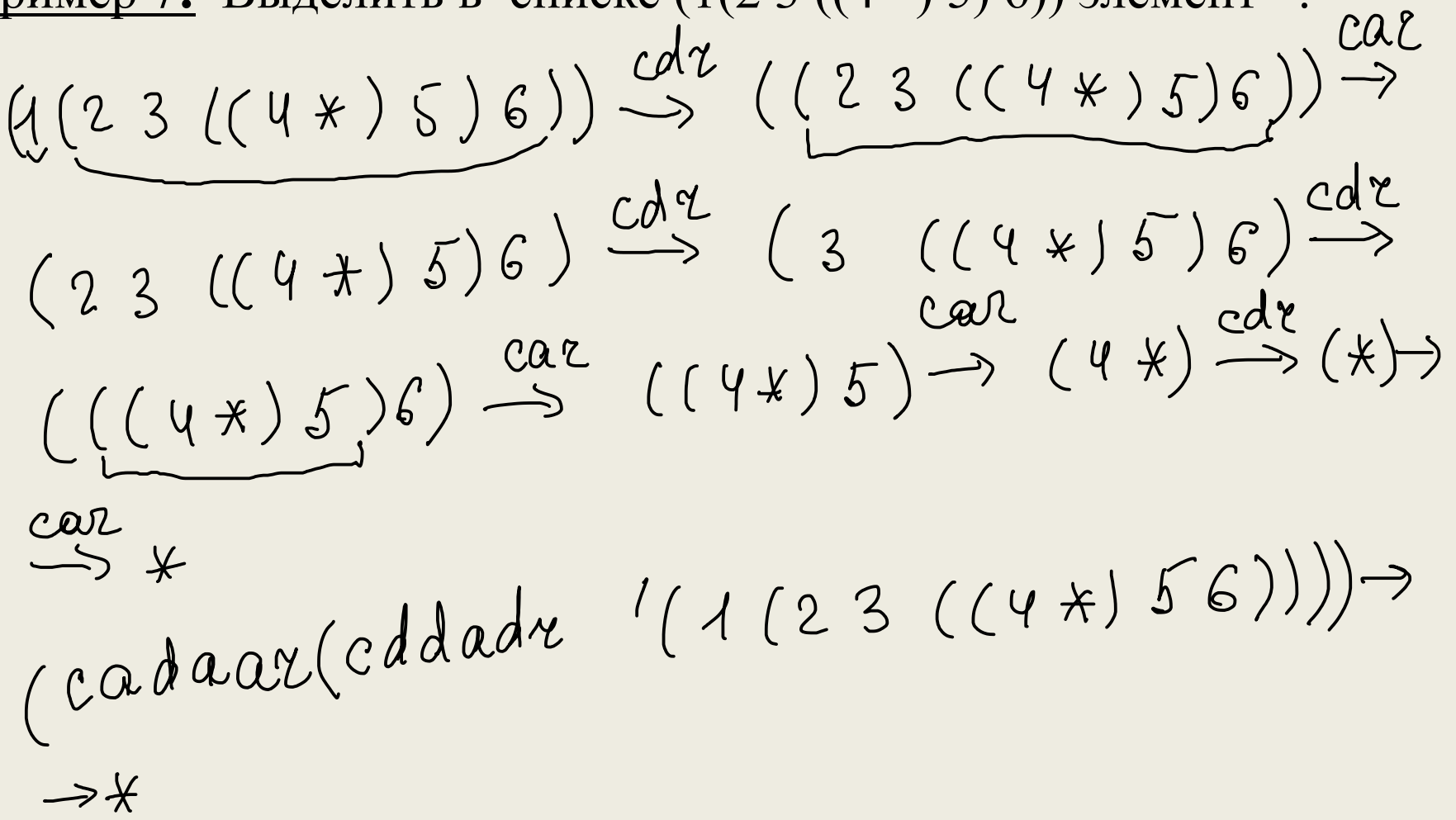
(car (cdr (cdr (car '((a b c) (d e) (f)))))) $\rightarrow$

$\rightarrow$ c

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

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

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



(**CONS** s-выражение список)

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

Пример 8:

(cons '(+ 5 6) '(* 7 8)) →

→ ((+ 5 6) * 7 8)

(cons (+ 5 6) '(* 7 8)) → (11 * 7 8)

(cons (+ 5 6) (* 7 8)) → (11 . 56)

(cons 'a nil) → (a)

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 'b '(1 2 3) 5) → (b (1 2 3) 5)

(list 1 nil) → (1 nil)

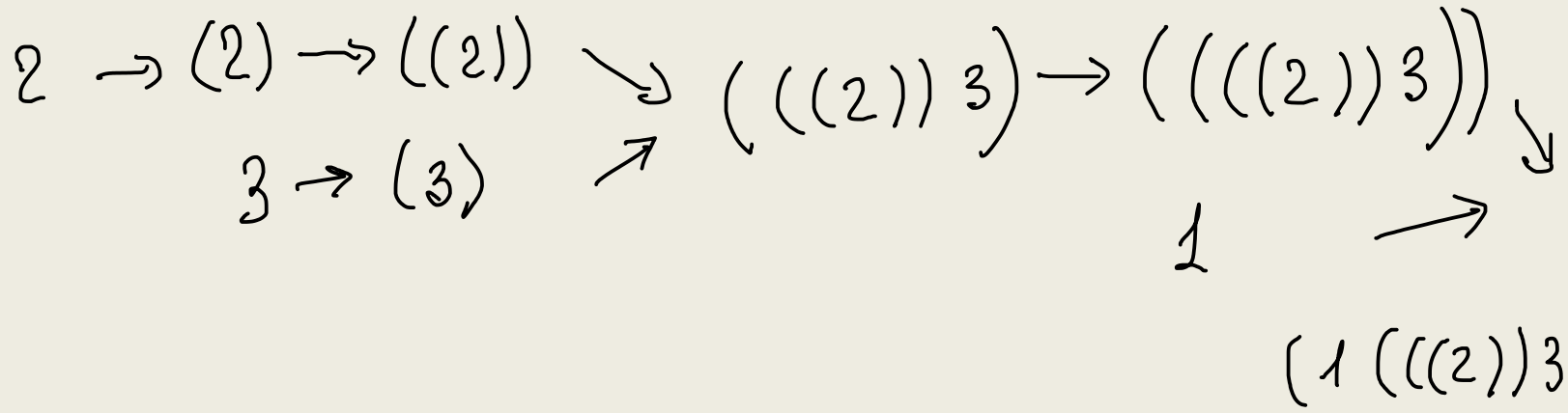
Пример 10: Из атомов 1, 2, 3, nil создать список (1 (((2)) 3)) двумя способами:

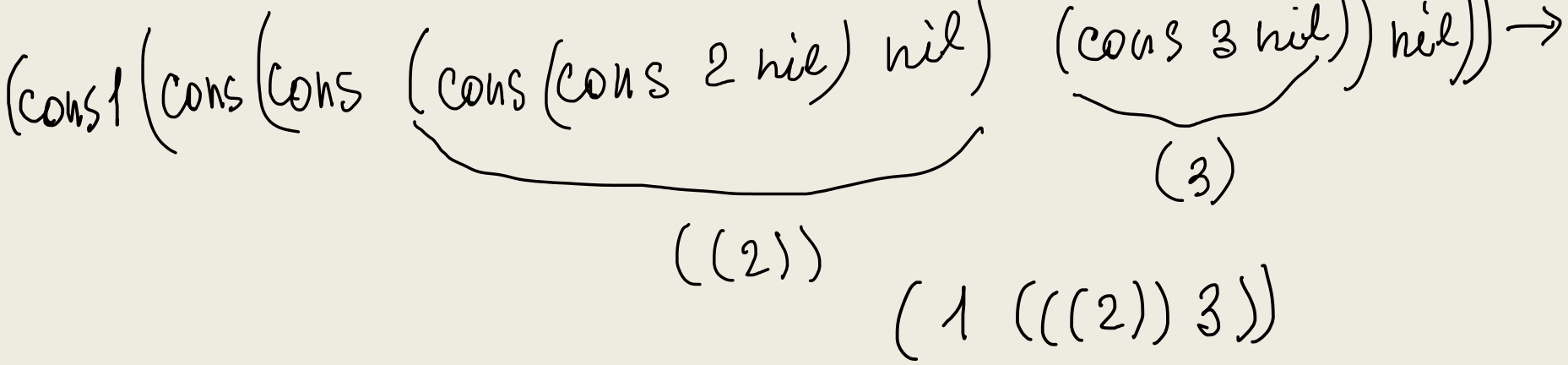
а) с помощью композиций функций **CONS**;

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

(1 (((2)) 3))

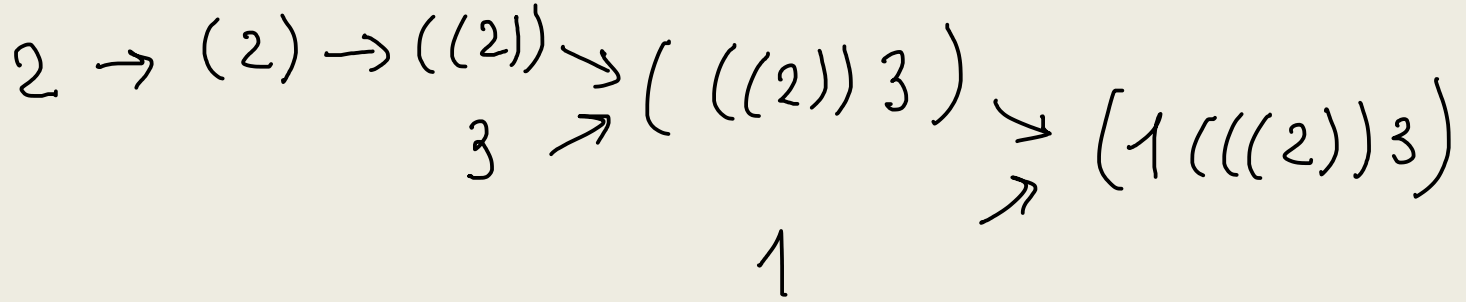
а) с помощью композиций функций **CONS**;





(1 (((2)) 3))

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



(list1 (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 d) '((e))) →

→ (a b c d (e))

(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 '(1 2 3 4)) → (1 2 3)

(**REVERSE** список)

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

Пример 14:

(reverse '((1 2) a (3 4 5))) →

((3 4 5) a (1 2))

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:

$f(x,y) = (x+y)\cdot y$

(lambda (x y) (* (+ x y) y))

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, 3) – ?

((lambda (x y) (* (+ x y) y)) 2 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

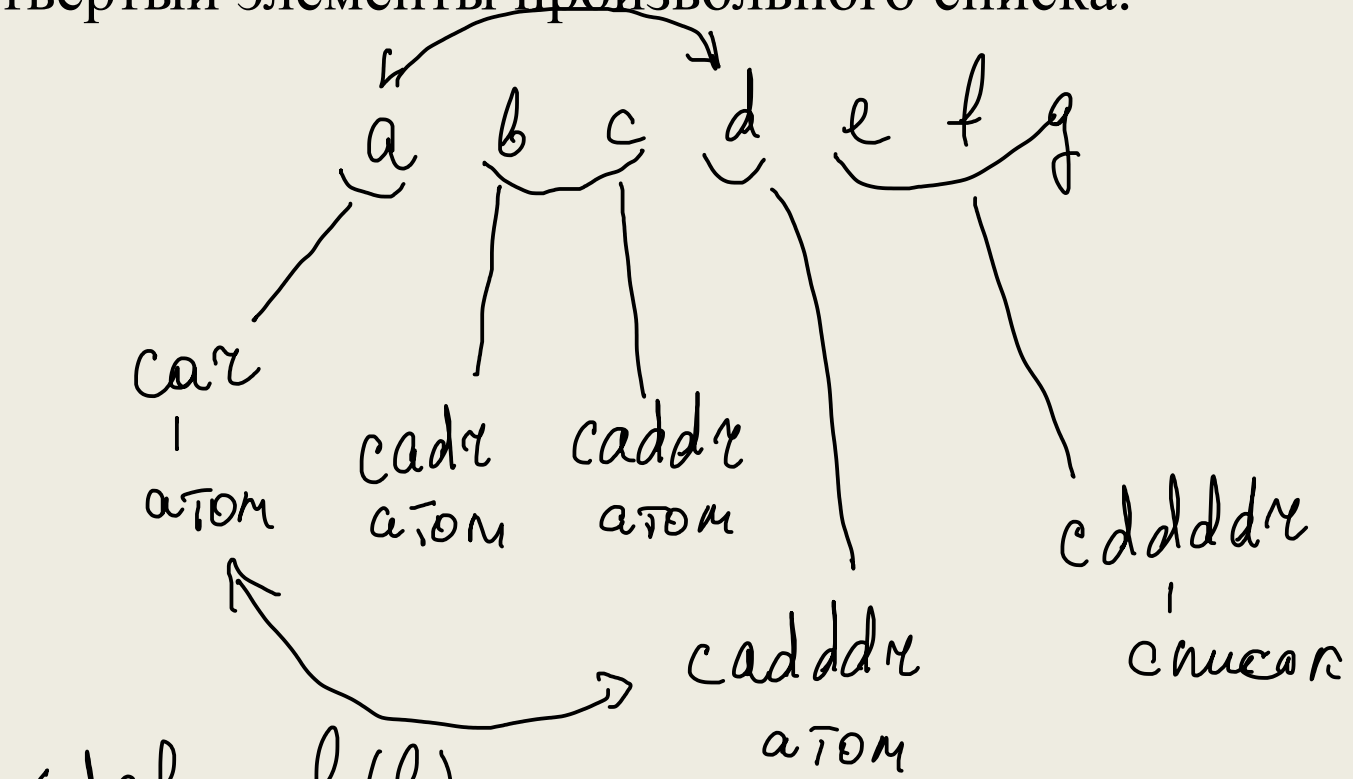
(**DEFUN** имя-функции лямбда-список S_1 S_2 … S_k) возвращает имя функции.

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.




```
(defun f (l)
  (append
    (list (cadddr l) (cadr l) (caddr l) (car l))
    (cddddr l)
  )
) → f
```

После определения функции, можно делать к ней обращение.

(f '(a b c d e f g h)) → (d b c a e f g h)