Функциональное и логическое программирование

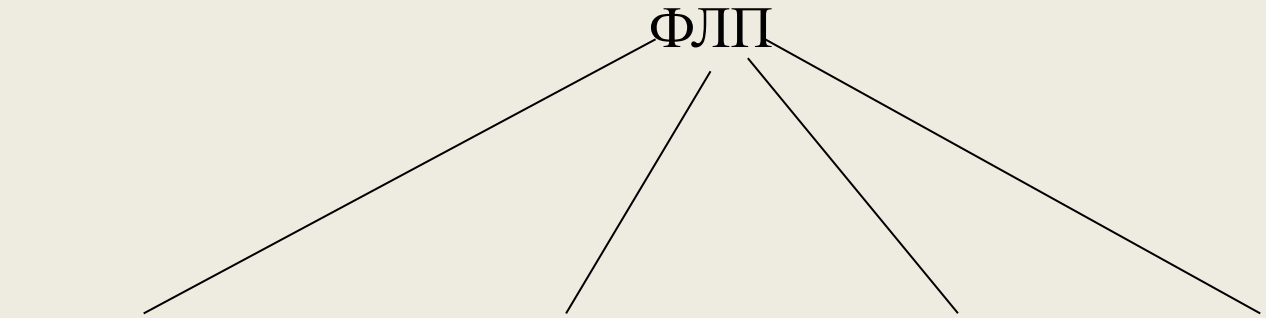
Лекция 1

Лектор: Галкина Марина Юрьевна

Практические занятия ведут преподаватели:
группы ИП-911-912 Галкина Марина Юрьевна
группы ИП-913-917 Белевцова Екатерина Андреевна

Кодовое слово для записи на курс в ЭИОС:
ИП-<номер группы>_очное.
Например, для ИП-911 кодовым словом является ИП-911_очное.

На каждой лекции отмечаем самостоятельно посещаемость через ЭИОС в течение 25 минут после начала лекции.



Максимальное количество баллов: 8+6·5=38

Зачет автоматом:

35-38 баллов + лабы + 2 задачи из РГЗ (на выбор)

30-34 балла + лабы + РГЗ

<30 баллов + лабы + РГЗ + зачет (добираем баллы до 30, 1 задача – 2 балла)

Все заработанные баллы можно смотреть в ЭИОС по ссылке Рейтинги.

Литература (можно найти в библиотеке и Интернете):

1. Э.Хювёнен, Й.Сеппянен, Мир Лиспа, т.1, 2

2. И.Братко, Программирование на языке Пролог для искусственного интеллекта

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.

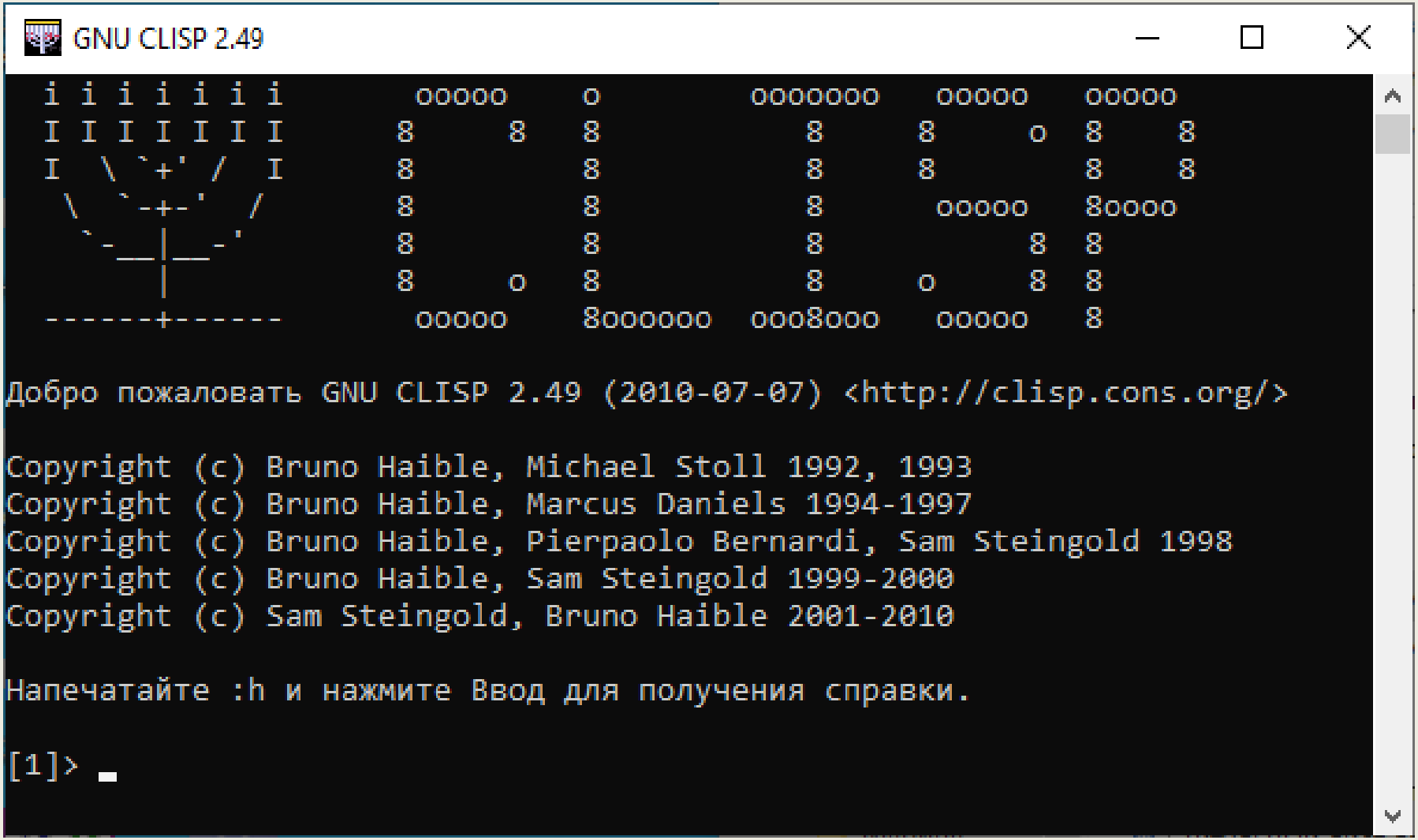
GNU Clisp 2.49

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.

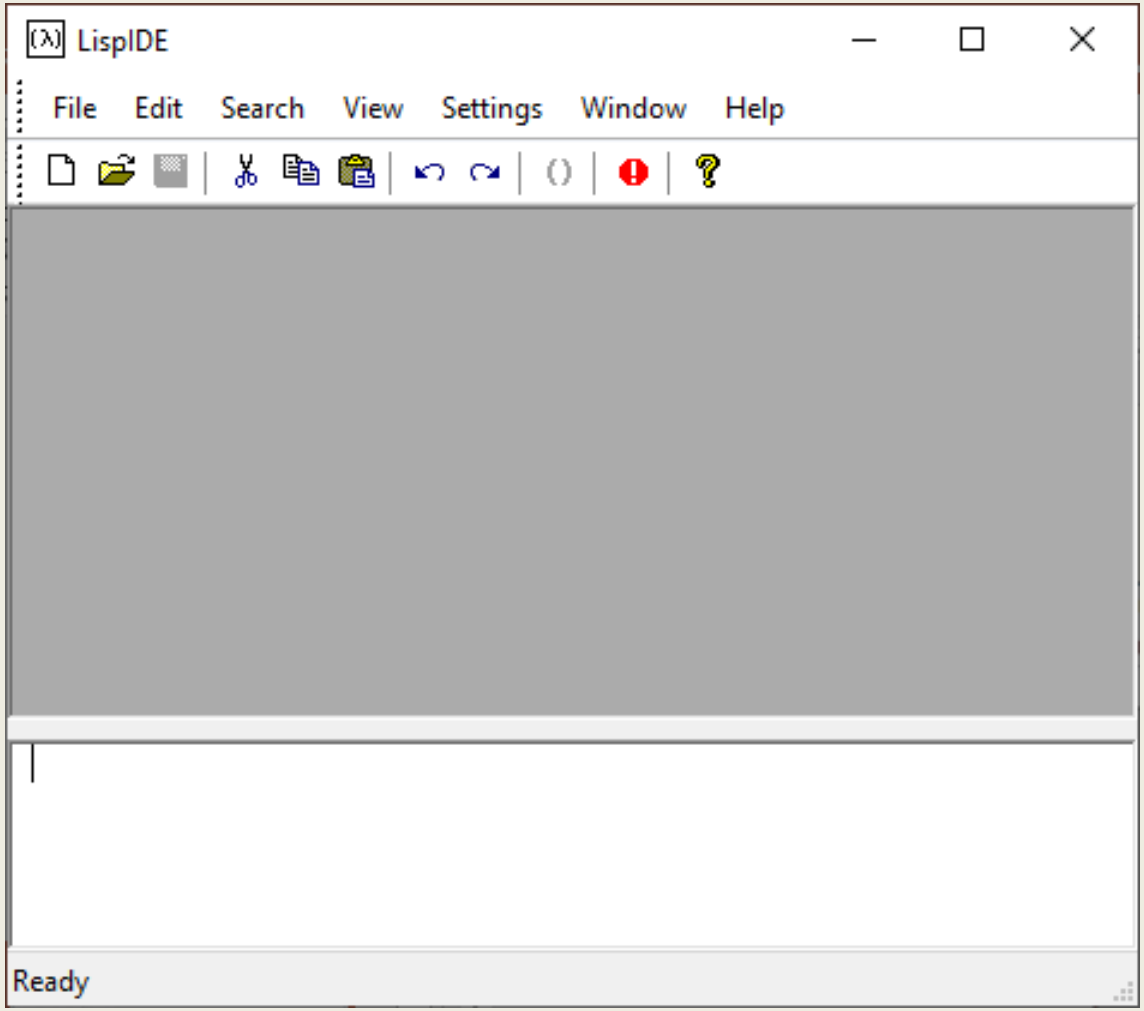
Если при выполнении команды возникла ошибка, то вернуться на предыдущий уровень (до ошибки) можно командой :r2.

Выход: (exit) или (bye).

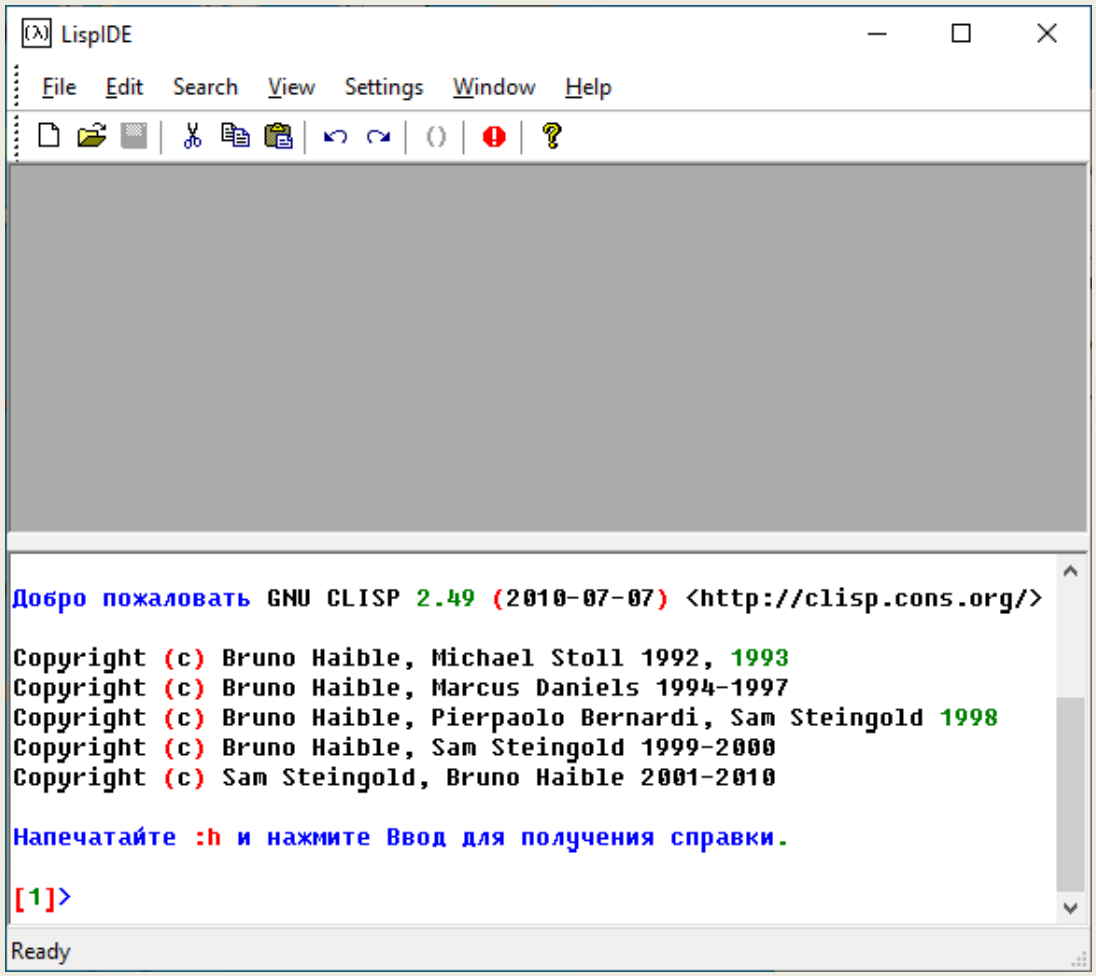

GNU CLISP 2.49
Добро пожаловать GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>
Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000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

Редактор LispIDE

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.

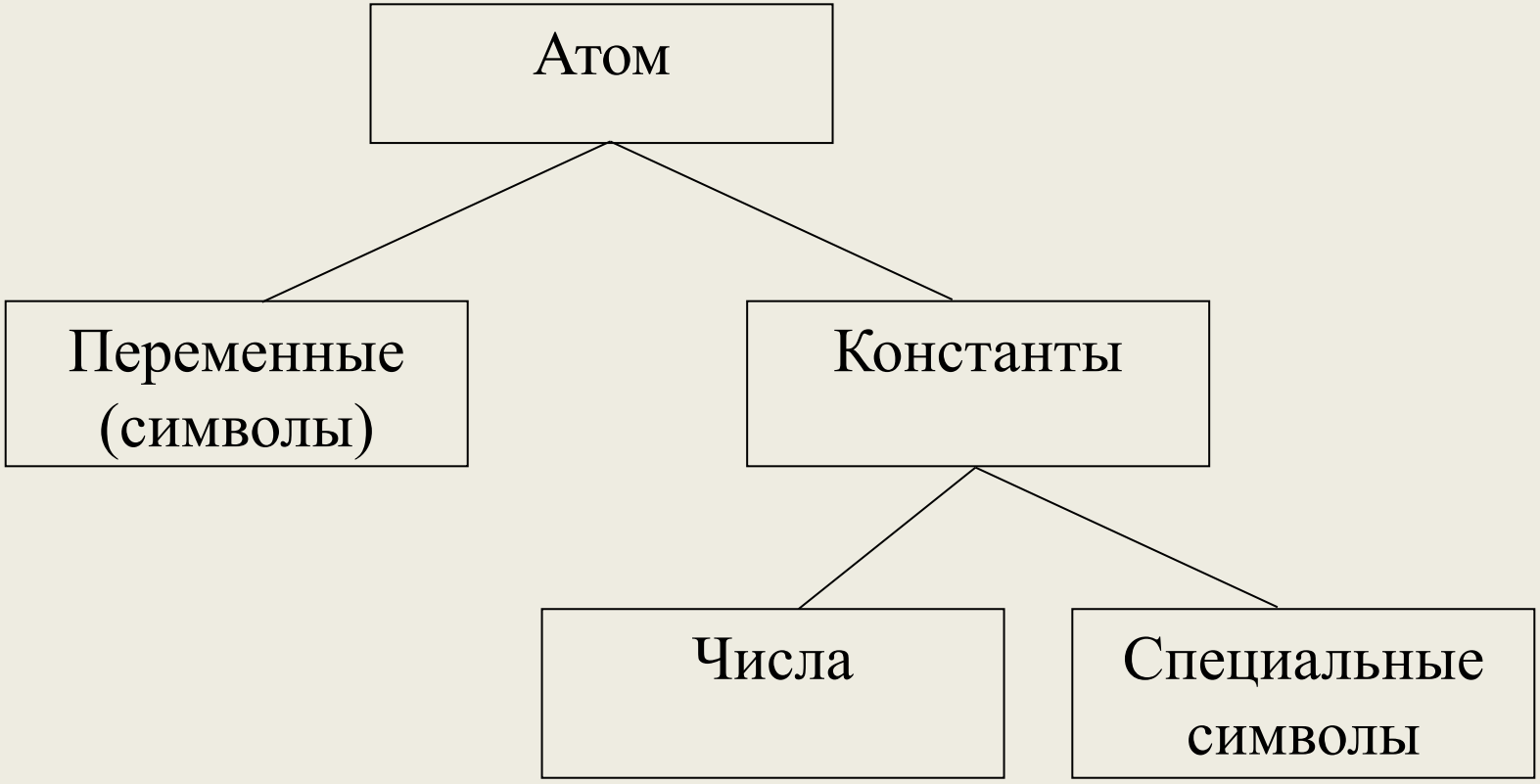


Иконки на панели инструментов:
() – отправить интерпретатору выделение до парной открывающейся скобки (если курсор стоит после закрывающейся скобки);
! – перезапустить интерпретатор;

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

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. abc , *

Числа состоят из цифр и точки, которым может предшествовать знак + или –. Число не может представлять других объектов, кроме самого себя.

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.
Пример:

(a 1 3) ((a b)) ((a b) c)

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. Суперпозиция функций выполняются «изнутри наружу».

Пример 1:

f(x,y) <=> (f x y) 1+2 <=> (+ 1 2)

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 2) → 3

'(+ 1 2) → (+ 1 2)

'(+ (* 3 4) 5) → (+ (* 3 4) 5)

1.2.1 Арифметические функции

- +
- -
- *
- /
- ABS

Пример:

(3 + 5) / (7 + 8) <=> (/ (+ 3 5) (+ 7 8))

1.2.2 Функции обработки списков

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

Пример 1:

(a b c) голова – a , хвост (b c)

((1)) голова – (1) , хвост ( ) <=> nil

(**CAR** список) возвращает голову списка

(**CDR** список) возвращает хвост списка

Пример 2:

(car '((a b c))) → (a b c)

(cdr '((a b c))) → nil

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

Пример 3**:** Выделить в списке ((a b c) (d e) (f)) элемент c.

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

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

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

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

(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 5:

(cons 3 '(1 2)) → (3 1 2)

(cons (+ 3 5) '((a b))) → (8 (a b))

(cons 'a nil) → (a)

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

(**LIST** s_1 … s_n), где s_i – s-выражение, возвращает список, элементами которого являются аргументы функции.

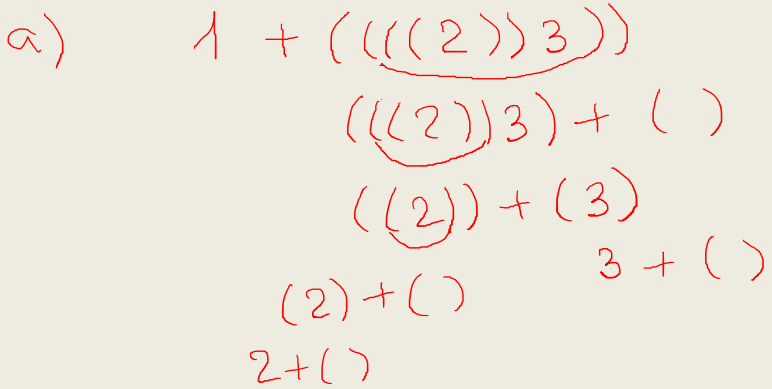
Пример 6:

(list '(1 2) 'a '(5 (6))) → ((1 2) a (5 6))

Пример 7: Из атомов 1, 2, 3, nil создадим 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))

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

(**APPEND** $sp_1 \ldots sp_n$), где sp_i – список, возвращает список элементами которого являются элементы списков - аргументов функции.

Пример 8:

(append '(1 2) '((3)) '(* 5 6 10)) →
(1 2 (3) * 5 6 10)

(append '(a b c) nil) → (a b c)

(**LAST** список) возвращает список из одного элемента: последнего элемента списка – аргумента функции.

Пример 9:

(last '(a b c d)) → (d)

(**BUTLAST** список) возвращает список из всех элементов списка-аргумента, кроме последнего.

Пример 10:

(butlast '(a b c d)) → (a b c)

(**REVERSE** список) возвращает перевернутый список-аргумент (переворачивание только на верхнем уровне!).

Пример 11:

(reverse '((a b) ((c) d e))) →

(((c) d e) (a b))

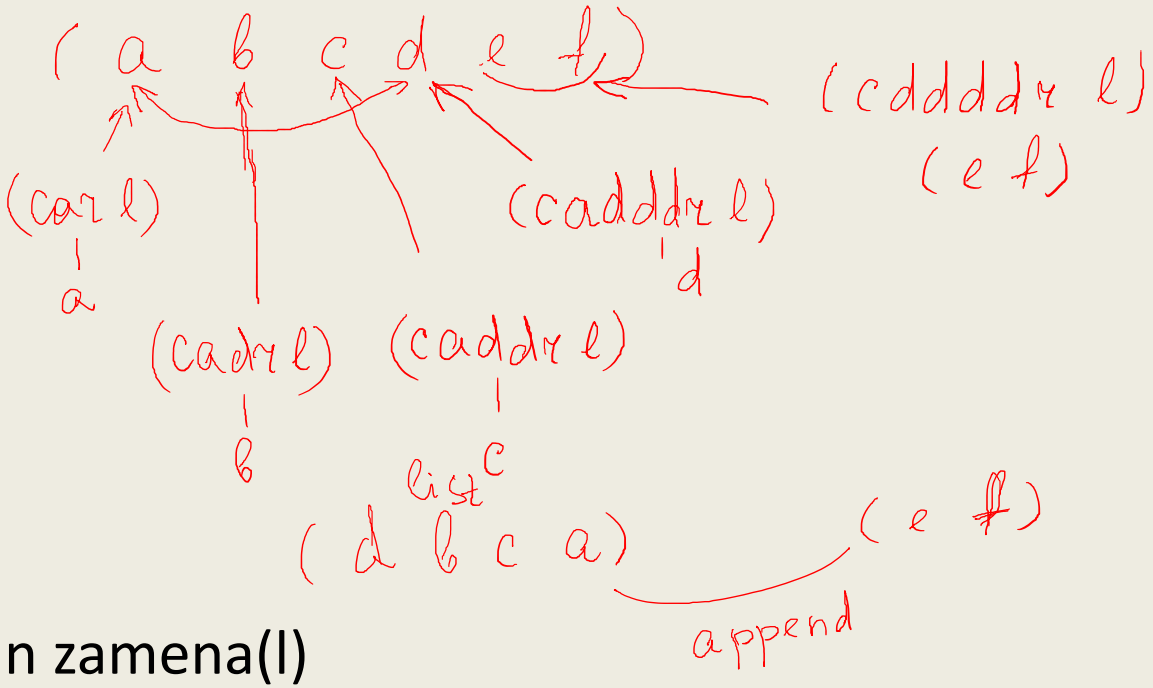
1.3 Определение функций пользователем

1.3.2 Определение функций с именем

(**DEFUN** имя-функции лямбда-список S_1 S_2 … S_k), где S_1 S_2 … S_k – тело функции, лямбда-список – список формальных параметров, возвращает имя функции. Имеет побочный эффект: связывание имени функции с лямбда-выражением (**LAMBDA** лямбда-список S_1 S_2 … S_k).
После такого определения можно обращаться к функции по имени.

Пример:

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




```
(defun zamena(l)
 (append (list(cadddr l)(cadr l)(caddr l)(car l))(cddddr l))
)
(zamena '(a s d f g h j k))
```