Лекция №3

Основные понятия языка

Форма Бэкуса-Науэра
- ::= «есть по определению»
- | – или
- {X}* – 0 или более вхождений
- {X}+ – 1 или более вхождений
- [X] – 0 или 1 вхождение

Алфавит Lisp
цифры ::= 0|1|2|3|4|5|6|7|8|9
буквы ::= A|B|C|…|X|Y|Z|a|b|c|…|x|y|z
спецзнаки ::= +|-|*|%|…

Имена в Lisp регистронезависимые.

Символы обозначают объекты, которыми манипулирует программа, и в этом смысле соответствуют именам переменных алгоритмических языков программирования.

символ ::= буква | символ буква | символ спецзнак | символ цифра | цифра буква | цифра спецзнак | цифра символ | спецзнак символ

Примеры символов: **A1**, **%1A**, **col13**

число ::= целое число | рациональное число | вещественное число

Примеры чисел: **537**, **2/3**, **2.0E-3**

атом ::= символ | число | ()

Примечание () – пустой список

Зарезервированы **T** (true) и **NIL** (он же пустой список () или false).

s-выражение ::= атом | (s-выражение**.**s-выражение)

Примеры s-выражений:

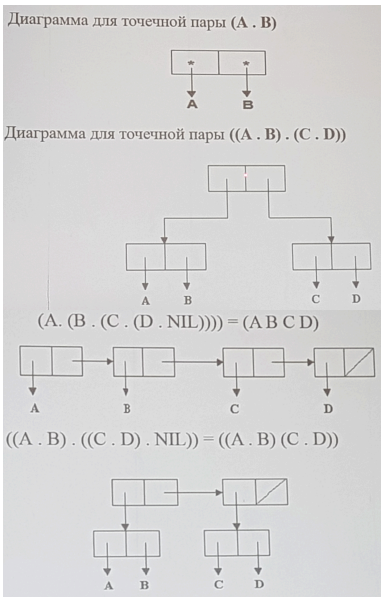
gruppa

(gruppa1.gruppa2)

((gruppa1.gruppa2).gruppa3)

Конструкция (s-выражение**.**s-выражение) называется точечной парой.

Оперативная память разбивается на маленькие области, называющиеся списочными ячейками (Lisp-овскими ячейками)



Каждая ячейка – указатель на атом или другую ячейку

список ::= NIL | (s-выражение.список)

Примеры списков:

(A.(B.(C.(D.NIL)))) **((A.B).((C.D).NIL))**

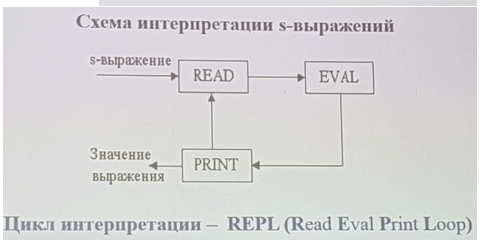
Любой список, заданный в виде последовательности элементов всегда можно представить в виде точечной записи. Обратное верно не всегда.

Интерпретация Lisp-программ

В Lisp не только данные представляются в виде s-выражений, но и программы. Выполнение программы состоит в вычислении s-выражений.

S-выражение, значение которого может быть вычислено называется формой.

- READ – читает одно выражение и преобразует его в соответствующую структуру данных в памяти;
- EVAL – принимает на вход структуру и вычисляет соответствующие ей выражение;
- PRINT – принимает результат вычисления и печатает его пользователю.



Программа = формы + функции
Функция = коллекция форм

Вызов функции выполняется по её имени из формы записанной в виде списка:
(**fn** a1 a2 … an)

Формы могут быть заданы константами, переменными и списками.

Константы соответствуют самоопределяемся формам, которые представляют самих себя и имеют фиксированные значения (числа, символы **T** и **NIL**, строки и др.)

Переменные в Lisp обозначаются символами. При вычислении такой формы возвращается значение символа (если оно есть!)

Форма, заданная в виде списка, может представлять: вызов функции, вызовы специальных форм, макровызовы.

Специальные формы позволяют выполнять действия, недостижимые с помощью обычных функций, например присваивать значения переменным, осуществлять условные вычисления. К таким формам относят: **SETQ**, **QUOTE**, **IF** и др.

Макровызовы внешне соответствуют вызову функции, но отличаются по способу вычисления. Они вычисляются в два этапа: сначала из аргументов макроса строится форма, а затем она вычисляется. **SETF** – это макрос.

Интерпретатор **EVAL** реализует следующие упрощённые правила:

a) если s-выражение – это константа, то **EVAL** возвращает такое s-выражение без изменений;
   > 12
   12
b) если s-выражение – это переменная, представленная символом, то функция **EVAL** возвращает последнее значение, которое было связанно с этим символом: если символ не имеет значения, то выдаётся сообщение об ошибке;
   >(setf x `(1 2 (rrr) 23))
   …
   >x
   (1 2 (RRR) 23)
   >y
   ERRORRRRRRRR!!!1!111
c) если s-выражение – это список, и первый элемент списка символ, то **EVAL** интерпретирует его либо как имя функции, либо как имя специальной формы, либо как имя макроса. Указанные имена должны быть известны системе. Если первый элемент списка – имя функции, то **EVAL** интерпретирует оставшиеся элементы списка как её аргументы, которые подлежат вычислению с помощью **EVAL**.
   >(list `aa `bb 1 2 3)
   (AA BB 1 2 3)
   > (car `(a b c d f))
   A
   > (cdr `(a b c d f))
   (B C D F)

Базовые функции Lisp: **CAR**, **CDR**, **CONS**

- **CAR** выделяет первый элемент списка или точечной пары
  (car `(a b)) → A
  (car `(a.b)) → A
- **CDR** – выделяет хвост списка или второго элемента точечной записи
  (cdr `(a b)) → (B)
  (cdr `(a.b)) → B
- Для выделения произвольных элементов можно использовать композиции **CAR** и **CDR**
  (car (car `((a b) (b c)))) → A
  (car (cdr (cdr `(1 2 3)))) → 3
- Композиции **CAR** и **CDR** часто используются, потому есть спец имена:
  (car (car x)) ↔ (caar x)
  (cdr (cdr x)) ↔ (cddr x)
  (car (cdr x)) ↔ (cadr x)
  (cdr (cdr (cdr (cdr x)))) ↔ (cddddr x)
  Произвольное количество **d** и **a** между **c** и **r**, обозначают **CDR** и **CAR** соответственно.

- **CONS** – объединяет в точечную пару
  (cons `a `b) → (A.B)
  (cons `x (cons `y (cons `z NIL))) → (X Y Z)

Предикат – это функция, которая возвращает истина или ложь

- **ATOM** – является ли аргумент атомом
  (atom `a) → T
  (atom nil) → T
- **EQ** – сравнивает значения двух своих аргументов. Создан для сравнения только 2х символов, ссылающихся на одну и ту же физическую структуру.
  (eq `a `a) → T
  (eq `(a b c) `(a b c)) → NIL
  (eq 5.0 5.0) → NUL
- **EQL** – позволяет ещё сравнивать числовые значения
  (eql 5.0 5.0) → T
  (eql `a `a) → T
- **EQUAL** – символы, строки, числа
  (equal `a `a) → T
  (equal `(a b c) `(a b c)) → T
  (equal «lisp» «lisp») → T
  (equal 5.0 5.0) → T
  (equal 5.0 5) → NIL
- **EQUALP**
  (equalp 5.0 5) → NIL

Встроенные функции обработки списков:

- REST – эквивалент CDR, возвращает хвост;
- FIRST – эквивалент CAR, возвращает первый элемент;
- SECOND – эквивалент CARD, возвращает второй элемент списка;
- THIRD – 3й
- …
- TENTH – 10й

Для создания списков:

- **LIST**
  (list 1 `b `c) → (1 B C)
  (list `abc 1 2 `(3 4 5)) → (ABC 12 (3 4 5))
- **APPEND**
  (append `(1 2 a) `(b 4 5) `(a b c)) → (1 2 A B 4 5 A B C)

Другие функции:

- **LAST** – выделяет последний элемент списка (результат – список);
  (last `(a b c d)) → (D)
- **NTH** – выделяет n-й элемент списка (с 0);
  (nth 3 `(a b c d) → D
- **SUBST** – выполняет замену элементов в списке;
  (subst `a `b `(a b c)) → (A A C)
- **BUTLAST** – выделяет список без последних элементов;
  (butlast `(a b c d) 2) → (A B)
- **MEMBER** – проверяет принадлежность элемента списку;
  (member `c `(a b c d)) → (C D)
- **LENGTH** – вычисляет длину списка;
  (length `(a b (2 3))) → 3
- **REVERSE** – выполняет обращение списка;
  (reverse `(1 2 e 4)) → (4 E 2 1)
- **ADJOIN** – добавляет элемент в множество, представленное списком;
  (adjoin `a `(a b c d)) → (A B C D)

- **REMOVE** – удаляет элемент из списка.
  (remove `a `(a b a b a b)) → (B B B)

Все эти функции не изменяют аргументы, таким образом реверс списка в переменной вернёт реверсированный список, но переменная не изменяется.

; однострочный комментарий
#| многострочный комментарий |#