1.1 СТРУКТУРЫ ДАННЫХ

1.1.1 Числа

Числа в языке Лисп подразделяются на **целые** и **дробные**. Целые числа в muLisp делятся на *малые целые* (-65536,+65536) и *большие целые* (произвольного размера). Различие между малыми и большими целыми числами влияет только на результат операции сравнения EQ.

Дробные числа представляют собой дробь, числитель и знаменатель которой являются целыми числами. При отображении дробные числа представляются в форме с десятичной точкой, при вводе возможны 2 варианта: форма с десятичной точкой и дробная форма.

Примеры:

-478	малое целое число
1234567890	большое целое число
4.789	дробное число
-1/5	дробное число
-1 / 5	не число (пробелы в записи числа недопустимы)

Все операции, кроме операции сравнения EQ, дают одинаковый результат для всех типов чисел.

В других диалектах Лиспа встречаются различные внутренние представления чисел. Иногда дробные числа в Лиспе хранятся в форме с плавающей точкой, как в других языках программирования (например, числа типа REAL в языке Паскаль). В этом случае в реализации языка предусматриваются функции приведения типов.

1.1.2 Символьные атомы

Символьные атомы представляют собой идентификаторы, т.е. последовательности латинских букв и цифр, начинающиеся с буквы. Заглавные и малые буквы в идентификаторах неразличимы (как в языке Паскаль). Кроме букв и цифр, имена символьных атомов могут включать и другие символы, такие как * (звездочка), + (плюс) и т.п. Примеры правильных идентификаторов: x1, X1, a, b2, *atom*.

Символьные атомы являются аналогами переменных величин в языках Паскаль и Си: они имеют **значение**, которое присваивается им специальными функциями SETQ и SET.

В отличие от переменных Паскаля и Си, символьные атомы Лиспа не являются *типизированными*, т.е. в Лиспе отсутствует понятие типа переменной (например, REAL, INTEGER в языке Паскаль). В разные моменты времени значением одного и того же символьного атома может быть число, строка, список, символьный атом и т.д. Для определения типа текущего значения символьного атома существуют встроенные функции. Подобным образом организованы переменные в СУБД FoxPro.

Значением символьного атома по умолчанию (до первого присваивания) в muLisp является сам этот атом. В классическом Лиспе только два символьных атома по умолчанию имеют в качестве значения самих себя -это атомы T (истина) и NIL (ложь). Попытка обращения к переменной, не имеющей значения, в классическом Лиспе вызывает сообщение об ошибке: UNDEFINED VARIABLE (неопределенная переменная).

Символьные атомы и числа вместе образуют группу **атомов**.

1.1.3 Списки

По определению список представляет собой выражение вида: (A_1 A_2 ...A_n), где все A_i - это атомы (символьные атомы или числа) или списки. Определение списка является примером **рекурсивного** определения, поскольку в нем имеется ссылка на само себя.

Примеры:

(A 3 1) -список из трех элементов;

((3.5)) -список из одного элемента, который является списком из числа 3.5;

((A 1) C ((4) 3)) -список из четырех элементов, причем первый элемент {(A 1)} является списком из двух элементов, второй {C} – символьным атомом, а третий {((4) 3)} – списком из 2 элементов, первый из которых {(4)} в свою очередь также является списком;

(A() -это выражение не является списком, т.к. количество открывающихся скобок не соответствует количеству

закрывающихся.

Пустой список (список, не содержащий элементов) обозначается пустыми скобками () или идентификатором NIL. Таким образом, NIL в Лиспе обозначает одновременно и символьный атом, и пустой список.

Списки и атомы вместе называются **s-выражениями** (от слова symbolic).

В классическом Лиспе отсутствуют другие типы данных. Наиболее общей структурой является **s-выражение**, которое может быть **атомом** или **списком**. **Атомы** делятся на **символьные атомы** и **числа**.

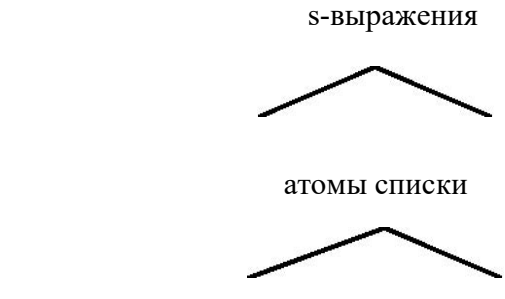

Рис. 1. Иерархия типов данных Лиспа.

В muLisp, как и в других современных вариантах языка Лисп (стандарт Common Lisp), добавлены типы данных, удобные для представления различных структур: строки, векторы (аналоги массивов), матрицы, файловые типы данных, классы и др.

1.2 ПРОГРАММЫ НА ЛИСПЕ

Программа на Лиспе представляет собой последовательность s-выражений, поступающих на вход интерпретатора Лиспа. Каждое s-выражение **оценивается** интерпретатором и результат выводится. В интерпретаторе muLisp приглашением к вводу очередного s-выражения служит символ $.

Итак, в Лиспе программы и данные состоят из одних и тех же объектов, а именно из s-выражений. Любое s-выражение можно оценить, т.е. выполнить как программу.

Результатом оценки s-выражения всегда является s-выражение. Способ оценки конкретного s-выражения зависит от его типа:

1. 1. результатом оценки числа всегда является само это число;
2. 2. результатом оценки символьного атома -его значение;

3. результатом оценки списка -значение функции, при этом список должен иметь вид: (A B_1 B_2 ... B_n), где A -символьный атом, являющийся именем выполняемой функции, а B_1...B_n -произвольные s-выражения, представляющие собой аргументы функции. Сначала оцениваются аргументы по изложенным выше трем правилам, затем к результатам их оценки применяется соответствующая функция. Поскольку аргументами B_1...B_n могут быть списки, правила 1-3 являются рекурсивными.

Таким образом, оценивая s-выражения, Лисп вычисляет значения *функций*, поэтому он относится к языкам **функционального типа**, а способ программирования на подобных языках носит название **функционального программирования**.

При вычислении значений функций в чисто функциональных языках значения аргументов этих функций не изменяются. Лисп имеет ряд отличий от таких языков, в частности, некоторые функции Лиспа не оценивают своих аргументов (например, функции QUOTE, SETQ), другие меняют значения своих аргументов (функции присваивания SET, SETQ).

Функции могут быть встроенными или определенными пользователем. Если при оценке списка оказывается, что функция с заданным именем не существует, то по правилам Лиспа интерпретатор должен выдать сообщение об ошибке и прекратить вычисления. Интерпретатор muLisp ошибки не выдает и считает результатом оценки списка сам этот список.

1.3 ОСНОВНЫЕ ВСТРОЕННЫЕ ФУНКЦИИ ЛИСПА

1.3.1 функция (QUOTE A)

Функция QUOTE не оценивает свой аргумент и выдает его в качестве результата.

Примеры:

```
$ ( QUOTE A)

A

$ ( QUOTE (A B C))

(A B C)
```

Функция QUOTE встречается настолько часто, что для нее в Лиспе введено сокращенное обозначение:

```
(QUOTE A ) <==> 'A
(QUOTE ( A B)) <==> '(A B)
```

1.3.2 Арифметические функции

Арифметические функции обозначаются символами **“+”, “-”, “*”, “/”** и выполняют соответствующие действия после оценивания своих аргументов, значения которых должны быть числами.

Примеры:

```
$ (+ 3 7)
10
$ (+ 2 4 9)
15 ;Функция “+” может иметь произвольное число аргументов.
$ (- 5 2)
3
$ (- 5)
-5 ;Функция “-” может иметь один аргумент, в этом случае
результатом функции является число с противоположным
знаком.
$ (* 4 2 5)
40 ;Функция “*”, как и “+”, может иметь больше двух
аргументов.
$ (/ (+ 2 4) 3)
2
```

Рассмотрим, как интерпретируется последнее выражение. Сначала оценивается первый аргумент функции деления. Поскольку он является списком, начинается его интерпретация по третьему правилу. В выражении (+ 2 4) аргументы оцениваются сразу, так как числа равны самим себе. В результате функция **“+”** дает 6. Теперь оценивается второй аргумент функции деления: его значение равно 3 -и результат оценки выражения (/ 6 3) с оцененными аргументами равен 2.

1.3.3 Функции присваивания (SETQ A B) и (SET A B)

Функция SETQ не оценивает первый аргумент, который должен быть символьным атомом, оценивает второй аргумент, присваивает его значение первому аргументу и выдает это же значение в качестве результата функции. Функция SET, в отличие от SETQ, оценивает

и первый аргумент, результат оценки которого должен быть символьным атомом.

Примеры:

```
$ ( SETQ A 'B)
B

$ A
B
$ ( SETQ X 2)
2
```

В последнем присваивании не требуется апостроф (функция QUOTE) для второго аргумента, так как число равно самому себе.

```
$ ( SETQ A 'D)
D
$ ( SET A 3)
3
$ A
D
$ D
3
```

При выполнении функции SETQ атому A присваивается атом D. Затем при выполнении SET оценивается первый аргумент (атом A), значением которого теперь является D, поэтому атому D и присваивается 3 (оценка второго аргумента SET).

Заметим, что (SETQ A B) <==> (SET 'A B) для произвольных A и B, поэтому имя функции SETQ образовано от имен SET и QUOTE.

1.3.4 Функции обработки списков

Функция (CAR A) оценивает свой аргумент, который должен быть списком, и выдает в качестве значения первый элемент этого списка. Обратите внимание, что элемент списка может быть произвольным s-выражением, т.е. как атомом, так и списком. Примеры:

```
$ (CAR '(A B C))
A
$ (CAR '((A) B (X)))

(A)
$ (CAR '(((A))))
((A))
```

Функция (CDR A) оценивает свой аргумент, который должен быть списком, и выдает в качестве значения этот список без первого элемента. Примеры:

```
$ (CDR '(A B C))
(B C)
$ (CDR '((A) B (X)))
(B (X))
$ (CDR '(((A))))
NIL
```

Первый элемент списка называется **головой** списка, а список без первого элемента -**хвостом** списка. Таким образом, функция CAR выделяет голову списка, а функция CDR -хвост списка. Хвост списка всегда является списком, возможно, пустым.

Заметим, что функция CAR, как и большинство функций Лиспа, оценивает свой аргумент, поэтому при оценке выражения (CAR (A B C)) интерпретатор Лиспа будет считать аргументом функции CAR результат применения функции A к аргументам B, C. Если функция с именем A не была определена пользователем, такие действия приведут к ошибке “Неизвестная функция A””. Для предотвращения оценки аргумента необходимо поставить перед ним апостроф (т.е. применить к нему функцию QUOTE). Интерпретатор muLisp не выдает ошибки “Неизвестная функция””, а считает результатом оценки подобного списка сам этот список, т.е. как бы “подставляет”” пропущенный апостроф.

Функция (CONS A B) оценивает аргументы, причем значение первого аргумента может быть произвольным s-выражением, а значением второго должен быть список. Результатом функции является список с головой A и хвостом B. Примеры:

```
$ (CONS 'A '(B C))
(A B C)
$ (CONS '(A) '(B C))
((A) B C)
$ (CONS 'A NIL)

(A)
```

Для функций CAR и CDR существует сокращенный вариант записи: CxxR, CxxxR, CxxxxR (число символов x не больше 4), где вместо символов x можно подставить символы D или A. Примеры:

```
(CDAAR X) <=> (CDR(CAR(CAR X)))
(CADR X) <=> (CAR(CDR X))
```

Функция (LIST $A_1...A_n$) имеет произвольное количество аргументов и после их оценки строит список вида ($A_1...A_n$). Значениями $A_1...A_n$ могут быть произвольные s-выражения. Примеры:

```
$ (LIST 'A 2 '(B C))
(A 2 (B C))
$ (LIST (CAR '(A B))(CDR '(A B)))
(A(B))
$ (LIST 'A NIL)
(A NIL)
```

Функция (APPEND A B) выполняется следующим образом: если значениями A и B являются списки вида (A_1 ...A_n) и ($B_1...B_m$), то значение функции APPEND равно списку ($A_1...A_n$ $B_1...B_m$). Примеры:

```
$ (APPEND '(A B C) '(D E))
(A B C D E)
$ (APPEND '(A) '((B)))
(A(B))
$ (APPEND '(A) NIL)

(A)
```

1.3.5 Внутреннее представление данных в Лиспе

В Лиспе все списки хранятся в виде списочных ячеек, или **CONS-ячеек**, имеющих два указателя: на элемент и на следующую ячейку. Эти два указателя называются соответственно CAR и CDR:

Рис.2. Структура CONS-ячейки

При образовании списка ячейки, соответствующие элементам списка, объединяются в цепочку с помощью указателей CDR. Указатель CDR последнего элемента указывает на NIL. Ниже приведены примеры внутреннего представления списков.

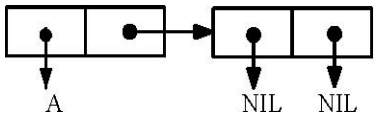

Рис.3. Структура внутреннего представления списка (A NIL)

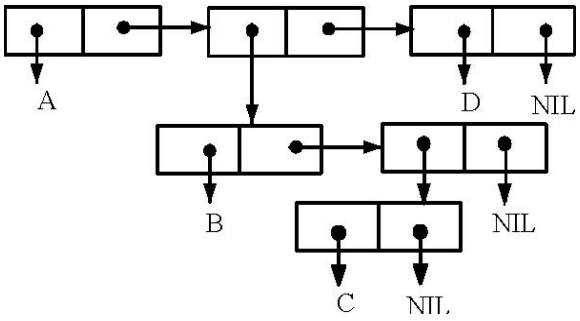

Рис.4. Структура внутреннего представления списка (A (B (C)) D)

Теперь становится ясным принцип работы описанных выше функций.

Функции CAR и CDR возвращают значение соответствующего указателя списочной ячейки, являющейся аргументом функции.

Функция CONS создает новую списочную ячейку (CONS-ячейку), устанавливает указатели CAR и CDR соответственно на первый и второй аргументы и возвращает указатель на созданную ячейку.

Действие функции (LIST A B) аналогично действию выражения (CONS A (CONS B NIL)), т.е. при выполнении функции LIST создается две новых списочных ячейки.

Функция APPEND ищет последнюю ячейку в списке (цепочке ячеек), соответствующей первому аргументу, и присваивает ее указателю CDR значение второго аргумента.

В стандартном Лиспе значение первого аргумента функции APPEND должно быть списком, в противном случае выдается соответствующее сообщение об ошибке.

Указание атома, т.е. не списочной ячейки, в качестве второго аргумента функций CONS и APPEND приведет к образованию **точечной пары**, т.е. списочной ячейки, указатель CDR которой указывает на атом.

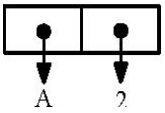

Рис.5. Списочная ячейка, представляющая точечную пару (A . 2)

Точечные пары записываются, как и списки, в круглых скобках, но элементы точечной пары разделяются символом **“.”** (точка).

Примеры:

```
$ (APPEND '(A) 'B)
(A . B)$ (CONS '(A B) 'C)
((A B) . C)
```

1.3.6 Функции -предикаты

Функции-предикаты предназначены для проверки логических условий и в качестве значений выдают T(истина) в случае выполнения условия и NIL (ложь) в противном случае.

Функция (NULL X) оценивает аргумент и выдает T, если значением X является NIL, и NIL в противном случае. Эта функция является аналогом NOT в языке Паскаль. Примеры:

```
$ (NULL 2)
NIL
$ (NULL (CDR '(R)))
T
$ (NULL (CAR '(T))
NIL
```

Функция (ATOM X) оценивает аргумент и выдает T, если его значением является атом, и NIL в противном случае. Примеры:

```
$ (ATOM 3)
T
$ (ATOM '(A))
NIL
$ (ATOM 'R)
T
$ (ATOM (CAR '(U 1)))
T

$ (ATOM NIL)
T
```

Функция (NUMBERP X) оценивает свой аргумент и выдает значение T, если значением аргумента является число и NIL в противном случае. Примеры:

```
$ (NUMBERP 2)
T
$ (NUMBERP (+ 2 3))
T
$ (NUMBERP (CAR '(R 4)))
NIL
$ (NUMBERP '(3))
NIL
```

Функция (LISTP X) оценивает свой аргумент и выдает T, если значением аргумента является список, и NIL в противном случае. Примеры:

```
$ (LISTP '(A D))
```

```
T
$ (LISTP (CAR '(A B)))
NIL
$ (LISTP (CDR '(A B)))
T
$ (LISTP NIL)
T
```

Так как NIL в Лиспе обозначает и пустой список и символьный атом, обе функции ATOM и LISTP возвращают истину (T) для этого аргумента.

Функция (CONSP X) оценивает свой аргумент и выдает T, если значением аргумента является CONS-ячейка (список или точечная пара), и NIL в противном случае.

```
$ (CONSP '((A B) . C))
T
$ (CONSP (CAR '(A 2)))
NIL ; атом не является CONS-ячейкой
$ (CONSP (CDR '(A 2)))
T ; список (2) является CONS-ячейкой
$ (CONSP (CDDR '(A 2)))
NIL ; NIL не является CONS-ячейкой, хотя является списком.
```

Функция сравнения двух атомов (EQ A B) оценивает аргументы и дает T, если они оба являются одним и тем же атомом.

Примеры:

```
$ (EQ 'A (CAR '(A B)))
T
$ (EQ 2 (+ 1 1))
T
$ (EQ '(A) '(A))
NIL
```

Функция EQ не позволяет сравнивать списочные ячейки, большие целые и дробные числа. Более общая функция сравнения (EQUAL A B) позволяет сравнить произвольные s-выражения и дает T, если они одинаковы и NIL в противном случае. Примеры:

```
$ (EQUAL '(A) '(A))
T
$ (EQUAL 2 (CAR '(2 U)))
T
$ (EQUAL T (ATOM 'Q))
T
```

Существует также несколько функций-предикатов для сравнения чисел, причем их отличие от остальных предикатов состоит в том, что значения аргументов должны быть числами, в противном случае выводится сообщение об ошибке.

Функция (= A B) дает T, если значения аргументов -равные числа, и NIL, если числа не равны. Примеры:

```
$ (= 4 (* 2 2))
T
$ (= 3 (CAR '(4 5)))
NIL
```

Функции (< A B) (<= A B) (> A B) (>= A B) позволяют сравнивать числа между собой. Примеры:

```
$ (<= 3 (+ 1 4))
T
$ (< 2 6)
T
$ (> 3 5)
NIL
$ (>= 5 5)
T
```

1.3.7 Логические функции AND и OR

Функция (AND $A_1 ... A_n$) последовательно оценивает свои аргументы, и если какой-нибудь аргумент равен NIL, то значение всей функции AND становится равным NIL, последующие аргументы при этом не оцениваются, иначе значением функции AND является значение последнего аргумента.

Функция (OR $A_1 ... A_n$) последовательно оценивает свои аргументы, и если какой-нибудь аргумент не равен NIL, то значением функции OR становится значение этого аргумента, а остальные аргументы не оцениваются, в противном случае значением функции OR является NIL.

Таким образом, при невыполнении условий значение функций OR и AND равно NIL (ложь), но оно не обязательно равно T (истина) при выполнении условий. Такие функции Лиспа называются *расширенными* предикатами, и этим они отличаются от аналогичных операций Паскаля.

Примеры:

```
$ (AND (ATOM 'A) 'B 2)
2
```

```
$ (OR (NULL (ATOM '(A B))) '(1 2))
T
```

1.3.8 Функции проверки условия

Функция (COND $A_1 ... A_n$) оценивает последовательно свои аргументы, которые должны иметь вид $A_i = (B_i\ C_i)$.

Если значение очередного B_i не равно NIL, то производится оценка C_i и его значение становится значением всей функции COND, при этом остальные пары A_i не рассматриваются. Для того, чтобы у функции COND значение всегда было определено, пара A_n обычно имеет вид (T C_n).

Примерным аналогом COND в Паскале является оператор CASE. Выражение (T C_n) при этом аналогично выражению ELSE C_n.

Функция (IF A B C) имеет 3 аргумента, причем A является условием, при выполнении которого (значение A не равно NIL) оценивается B, и значение B становится значением всей функции IF. Если же значение A равно NIL, значением функции IF будет оценка C. Функция IF является аналогом выражения на Паскале:

```
IF A THEN B
     ELSE C
```

Примеры:

```
$ (COND ((NULL 'A) 1) ((ATOM '(Q)) 2) (T 3))
3
$ (IF (= 5 (+ 2 3)) T NIL)
T
```

Легко заметить, что функция IF следующим образом выражается через COND:

```
(IF A B C) <=> (COND (A B) (T C))
```

1.3.9 Функция определения новых функций DEFUN

Функция (DEFUN A B C) не оценивает своих аргументов, ее назначение -определить новую функцию, которую в дальнейшем можно будет использовать так же, как и встроенные.

A -символьный атом, являющийся именем определяемой функции;

B -список из символьных атомов, являющихся формальными параметрами

функции, если их нет, необходимо поставить пустой список NIL; C - s-выражение, являющееся телом определяемой функции. Функция DEFUN в качестве результата возвращает имя определяемой функции.

Рассмотрим ряд простых примеров определения новых функций.

а) Вычисление квадрата функции:

```
$ (DEFUN SQR (X) (* X X))
SQR
$ (SQR 5)
25
```

б) Функция, вычисляющая сумму квадратов двух чисел:

```
$ (DEFUN SUMS (X Y) (+ (SQR X) (SQR Y)))
SUMS
$ (SUMS 3 4)
25
```

В данном определении используется ранее определенная функция SQR.

в) Вычисление второго элемента произвольного списка:

```
$ (DEFUN SECOND (X) (CAR (CDR X)))
SECOND
$ (SECOND '(1 2 3))
2
г) Функция, возвращающая минимум из двух чисел:
$ (DEFUN MIN2 (X Y) (IF (< A B) A B))
MIN2
$ (MIN2 3 8)
3
```

д) Функция, дающая в качестве значения число 2:

```
$ (DEFUN TWO NIL 2)
TWO
$ (TWO)
2
```