## § 8. ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ЛИСП

### 8.1. НАЗНАЧЕНИЕ И ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА

В программировании помимо процедурного подхода, представителями которого являются такие универсальные языки высокого уровня как Бейсик, Паскаль, Си, и логического подхода, представленного языком Пролог, существует еще одно направление - функциональное. Оно возникло в 1962 г. вместе с созданием Дж.Маккарти языка программирования Лисп (Lisp). Долгое время этот язык занимал особое место. Подавляющее большинство программ искусственного интеллекта составлено на языке Лисп. До сих пор он считается стандартным языком разработки систем искусственного интеллекта. Его популярность особенно велика в США. В нашей стране этот язык не получил широкого распространения (одна из причин - недостаток литературы о нем на русском языке), однако в настоящее время популярность этого языка быстро растет. Несмотря на то, что Лисп - один из самых старых используемых языков программирования, у него многое еще впереди.

Язык Лисп - один из первых языков обработки данных в символьной форме. Его название происходит от английских слов «list processing *» -* «обработка списков». В Лиспе и программа, и обрабатываемые ею данные представляются в одной и той же форме - в форме списка. Таким образом, программы могут обрабатывать и преобразовывать другие программы и даже самих себя.

Используемый в Лиспе, так называемый, функциональный подход к программированию основывается на той простой идее, что вся обработка информации и получение искомого результата могут быть представлены в виде вложенных и/или рекурсивных вызовов функций, выполняющих некоторые действия, так что значение одной функции используется как аргумент другой. Значение этой функции становится аргументом следующей и т.д. пока не будет получен конечный результат - решение задачи.

Программы строятся из логически расчлененных определений функций. Определения состоят из управляющих структур, организующих вычисления, и из вложенных вызовов функций. Основными методами функционального программирования являются композиция и рекурсия. Все это представляет собой реализацию идей теории рекурсивных функций.

Имеется большое число систем программирования на Лиспе, реализованных для компьютеров различных типов. Как правило, это интерпретирующие системы, работающие в интерактивном (диалоговом) режиме. Соответствующие описания и команды вводятся с клавиатуры после приглашения ("\_"), затем прочитывается результат.

### 8.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ НА ЛИСПЕ. СПИСКИ

Программы на языке Лисп строятся из простейших неделимых элементов, называемых атомами. Символы и числа представляют собой атомы, из них состоят все остальные структуры.

**Символ** - это имя, состоящее из букв, цифр и специальных знаков, которое обозначает какой-нибудь предмет или действие из реального мира, а также число, функцию (программу) и другие объекты. Наряду с символами используются и числа (значения), которые могут быть целыми (например, 543), десятичными (например, 3.789) и в представлении с мантиссой и порядком (например, 1.0243Е-6).

Главной структурой в Лиспе является список.

**Списком** называется упорядоченная последовательность, элементамикоторойявляются либо атомы, либо списки (подсписки). Списки заключаются в круглые списки, аих элементы разделяются пробелами. Например,

(ab(cd)e)

(В группе 18 студентов)

(((((первый) 2) третий) 4) 5).

Список, в котором нет ни одного элемента, называется пустым списком и обозначается "( )" или специальным символом NIL. Список - это структура данных, представляющая некоторую иерархическую связь (дерево) с помощью строго соответствующих друг другу открывающих и закрывающих скобок.

Имеется и альтернативный способ записи списков - с использованием, так называемой, точечной нотации. Точка при этом отделяет начальный элемент списка -его голову - от остальной части списка - хвоста: (голова, хвост) или

(а1 а2 ... aN) = (а1. (а2.... (aN.Nil)...)).

Здесь Nil - это предопределенная константа, означающая пустой список (и одновременно логическое значение «Ложь»).

Атомы и списки называются **S-выражениями**. Все вышесказанное можно обобщить в следующих формах Бэкуса - Наура

<S-выражение> :: = <атом> | <список>

<список> :: = (<внутренняя часть>)

<внутренняя часть> :: = NIL | <S-выражение> [{внутренняя часть}}

<атом> :: = цепочка алфавитно-цифровых символов без пробелов или специальных символов (,);.

Списки в Лиспе - основное средство представления знаний. Например,с помощью вложенных списков может быть представлена характеристика человека:

(сотрудник

(имя Петр)

(отчество Петрович )

(фамилия Иванов)

( образование ( среднее (с 1969 по 1979))

(высшее ( ВГУ г.Воронеж (с 1979 по 1982)

(МГУ г. Москва (с 1982 по 1984)) ( специальность

(техническая кибернетика)

(программирование)

(стаж (с 1984 по 1997)

)

### 8.3. ФУНКЦИИ

Функции в Лиспе аналогично математическим функциям ставят в соответствие элементам из одного множества - определения (аргументов) - единственный элемент из множества значений. В программах следует различать определение функций и вызов (применение) функции.

В языке Лисп принята единообразная префиксная форма записи, при которой как имя функции или действия, так и аргументы записываются внутри скобок:

(f x)

(g x y) (сумма\_квадратов 2 3).

Аналогично записываются и арифметические действия:

(+ х у)

(\*x(+yz))

(+ (^ х х) (+ у у)).

Определение функций и их вычисление в Лиспе основано на **лямбда-исчислении Черча**. В 1-исчислении Черча функция записывается в виде

1(х1,х2,... ,xn) .fn

В Лиспе 1-выражение имеет вид (LAMBDA (xl, x2,..., xn).fn).

Символ LAMBDA означает, что мы имеем дело с определением функции. Символы xi являются формальными параметрами, они образуют список, называемый лямбда-списком; fn - это тело функции, которая может иметь произвольную форму, допускаемую интерпретатором Лиспа. Телом функции может быть, например, константа или композиция из вызовов функций. Функцию, вычисляющую сумму квадратов двух чисел, можно, например, определить так:

(lambda(xy)(+(\*xx)(\*yy))).

**Лямбда-выражение** - это безымянная функция, которая может быть использована для связывания формальных и фактических параметров на время вычислений. Вызов такой функции происходит по форме

(лямбда-выражение а1 а2 ... an)

Здесь ai - фактические параметры, с которыми происходит вычисление.

Например

((lambda (х у) (+ (\* х х) (\* у у))) 3 4).

Результат: 25.

Определить новую функцию и дать ей имя для последующих вызовов можно с помощью функции DEFUN (define function):

(DEFUN имя лямбда-список тело).

DEFUN соединяет символ с лямбда-выражением, и символ начинает представлять (именовать) определенные этим лямбда-выражением вычисления. Значением этой формы является имя новой функции:

(defun sumsquare (х у) (+ (\* х х) (\* у у))) .

Результат: sumsquare.

Вызов (применение) этой функции:

(sumsquare 34)

Результат: 25.

Определение функции задается списком, поэтому его можно модифицировать в ходе выполнения программы. Кроме того, некоторый символ может быть и именем функции и переменной.

В Лиспе передача параметров происходит по значению. Формальные параметры функций являются статическими и локальными, т.е. действительны только внутри той формы, в которой они определены.

Основу для построения различных функций образует набор небольшого числа примитивных встроенных функций. Базовыми функциями обработки S-выражений являются функции

CAR, CDR, CONS, ATOM, EQ, EQL, =

и другие, смысл которых отражен в табл. 3.7.

Таблица 3.7

**Базовые функции обработки S-выражений**

|  |  |  |  |
| --- | --- | --- | --- |
| **Функция** | **Вызов** | **Действие** | **Пример использования** |
| CAR | (CAR список) | Возвращает головною часть | (CAR(1 234)) |
|  |  | списка - его 1-й элемент | Результат:1 |
| CDR | (CDR список) | Возвращает хвостовую часть | (CDR(! 234)) |
|  |  | списка- все. кроме 1-го элемента | Результат:(2 3 4) |
| CONS | (CONS S-выра- | Строит список из переданных в | (CONS I (2 3 4)) |
|  | жение список) | качестве аргументов головы и хвоста | Результат: (1234) |
| ATOM | (ATOMS-выра- | Предикат; проверяет, является ли | (ATOM A) : t |
|  | жение) | аргумент атомом, и возвращает либо t | (ATOM (1 2 3)): Nil |
|  |  | (истина), либо Nil или ("(ложь) |  |
| EQ | (EQ символ | Предикат: проверяет тождественность | (EQ A A): t |
|  | символ) | символов-аргументов, неприменим | (EQ X (CAR (X Y Z))): t |
|  |  | для чисел |  |
| EQL | (EQL число | Предикат, проверяет тождественность | (EQL 3.0 3.0): t |
|  | число) | чисел одного типа |  |
| = | (= число | Предикат, проверяет тождественность |  |
|  | число) | чисел различных типов | (=30.3el):t |
| EQUAL | (EQUAL число | Аналогична EQL, | (EQUAL(xyz)(xyz)):t |
|  | или список | но, кроме того, проверяет идентичность |  |
|  | число или список) | Списков |  |
| EQUALP | (EQUALP | Проверка наиболее общего равенства |  |
|  | объект объект) |  |  |
| NULL | (NULL список) | Проверка, является ли аргумент |  |
|  |  | пустым списком |  |
| NOT | (NOT логическая | Логическое отрицание |  |
|  | величина) |  |  |
| NTH | (NTH n список) | Выделение n-го элемента списка | (NTH 2 (1 2 3)): 3 |
|  |  |  | (индексы начинаются с 0) |
| FIRST |  | Предикаты, выделяющие |  |
| SECOND |  | Соответствующие элементы списка |  |
| LAST |  |  |  |
| LIST | (LIST apr | Строит из аргументов список | (LIST a b (с)): (a b c) |
|  | арг2 ...) |  |  |

Отметим, что в программах на Лиспе надо тщательно отличать значения от их обозначений.

В Лиспе константы обозначают самих себя. Выражения типа (\* 2 2) сразу вычисляются. Чтобы избежать нежелательного вычисления выражения используется функция QUOTE или знак апострофа (') перед выражением:

(\* 2 2) : 4

' (\* 2 2) :' (\* 2 2) – список

Произвольный символ можно использовать как переменную, и он может обозначать произвольное выражение. При первом использовании символу должно быть присвоено или с ним связано некоторое значение с помощью функции SET, например,

(SET 'операции**'** (+ - \*/))

Знак ' используется для подавления вычисления аргументов функции SET. Функция SETQ не вычисляет значения 1-го аргумента (а 2-го вычисляет).

На значение символа можно сослаться, указав его без апострофа (').

Для занесения значений в ячейку памяти, связанной с символом, можно пользоваться обобщенной функцией присваивания SETF, размещающей значения в соответствующей ячейке памяти:

(SETF ячейка\_памяти значение).

Переменная «ячейка\_памяти» без апострофа указывает на ячейку памяти. Присвоение, выполняемое функциям» SET, SETQ и SETF, является побочным эффектом , этих функций, помимо того, данные функции возвращают присваиваемые значения.

### 8.4. ФОРМЫ. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ В ЛИСП-ПРОГРАММЕ

Программа состоит не только из функций, но и из форм. Простейшими формами являются константы, переменные, лямбда-вызовы, вызовы функций.

Остановимся более подробно на специальных формах, предназначенных для управления обработкой программы и контекстом. У каждой формы определенный синтаксис и семантика, основанные на едином способе записи и интерпретации.

Управляющие предложения Лиспа внешне выглядят как вызовы функций - в виде скобочных выражений, первый элемент которых действует как имя управляющей структуры, а остальные элементы - как аргументы. Наиболее важные формы можно разделить на следующие группы:

***работа с контекстом***

• QUOTE или блокировка вычисления,

• вызов функции и лямбда-вызов,

• предложения LET и LET\*;

***последовательное исполнение***

• предложения PROG1, PROG2 и PROGN;

***разветвление исполнения***

• условные предложения COND, IF, WHEN, UNLESS,

• выбирающее предложение CASE;

***итерации***

*•* циклические предложения DO, DO\*, LOOP, DOTIMES, DOUNTIL;

***передачи управления***

• предложения PROG, GO и RETURN;

***динамическое управление вычислением***

*•* THROW, CATCH, а также BLOCK.

Эти управляющие формы (кроме QUOTE и лямбда-вызова, а также вызовов функций), в основном, используются в теле лямбда-выражений, определяющих функции.

Предложение LET используется для создания связи переменных внутри формы:

(LET ((пep1 знач1) (пер2 знач2)...) форма1 форма2 ...).

При вычислении этого выражения статические переменные пep1, пер2, ... связываются (одновременно) с соответствующими значениями знач1, знач2, ..., а затем вычисляются значения форм форма1, форма2, ... Значение последней формы возвращается как общий результат. Форма LET\* отличается от LET лишь тем, что связывание переменных и вычисление форм происходит не одновременно, а последовательно, вначале 1-е, потом 2-е и т.д.

Например:

(let\*((x2)(y(\*3x)))

(list x у)

Результат: (2 6).

Предложения PROG1, PROG2 и PROGN позволяют организовывать последовательные вычисления из нескольких вычисляемых форм:

(PROG1 форма1 форма2 ... формаn)

(PROG2 форма1 форма2 .. формаn)

(PROGN форма1 форма2 . формаn).

Различие этих форм лишь в возвращаемых ими в качестве общего значения результатах. Форма PROG1 возвращает значение формы1, PROG2-формы2, PROGN -последней формы n.

Например:

(progn (setq x 2) (setq у (\* 3 х)))

Результат: 6.

Предложение COND является основным средством разветвления обработки. Структура условного предложения такова:

(COND (р1 а1) (р2 а2)... (pn an)).

pi - это предикаты (выражения-условия, которые могут быть либо истинными (Т), либо ложными (NIL)). Их значения вычисляются слева направо, пока не будет получено значение «истина» (Т), затем вычисляется и возвращается в качестве результата результирующее выражение ai. соответствующее 1-му истинному предикату pi. Если истинного предиката нет. то значение COND - NIL. Форма ai для соответствующего предиката может отсутствовать (тогда возвращается значение этого предиката в случае его истинности), или, наоборот, может быть задана последовательность форм для предиката pi - тогда эти формы вычисляются последовательно и возвращается значение последней.

В следующем примере с помощью предложения COND определена функция, устанавливающая тип выражения:

(defun тип (1)

(cond ((null 1) 'пусто)

((atom 1) 'атом)

(t 'список)))

Результат: ТИП.

Примеры применения этой функции:

(тип ' (a b с))

Результат: СПИСОК.

(тип (atom ' (а т ом)))

Результат: ПУСТО.

Для организации ветвления можно использовать и формулы IF, WHEN, UNLESS:

(IF условие то-форма иначе-форма),

что эквивалентно

(COND (условие то-форма) (Т иначе форма));

(WHEN условие форма1 форма2 ...),

что эквивалентно

(UNLESS (NOT условие) форма! форма2 ...)

или

(COND (условие форма1 форма2 ...)).

Можно применять и выбирающее предложение CASE:

(CASE ключ (список ключей1 форма11 форма12 ...)

(список ключей2 форма21 форма22 . . .)

В этой форме сначала вычисляется значение ключевой формы «ключ», затем происходит сравнение с элементами списков ключей и, если найдено значение ключевой формы, вычисляется последовательность соответствующих форм, значение последней из которых возвращается как значение всего выражения CASE.

Предложения PROG, GO и RETURN аналогичны конструкциям неструктурных языков программирования (типа FORTRAN, Бейсик); пользоваться ими не рекомендуется.

### 8.5. РЕКУРСИЯ И ЦИКЛ В ПРОГРАММАХ НА ЛИСПЕ

В «чистом» функциональном программировании организация повторяющихся вычислений должна происходить лишь с помощью условных предложений и определения рекурсивных, вызывающих самих себя, функций. Рассмотрим в качестве примера функцию, просто определяемую через рекурсию, - факториал n!=1\*2 \* 3 \*...\* (n-1) \* n *=* (n-1)! т n (0! = 1 по определению):

(defun ! (n) (if(= п 0) 1 (\* п (! (. п 1))))) .

Имя функции - "!",ее аргументом является переменная n. Лямбда-выражение, определяющее функцию, представляет собой условную if-форму, которая в случае n=0 возвращает 1, а в противном случае вычисляет произведение n и результата вызова этой же функции ! для аргумента n-1.

Пример вызова этой функции:

(!5)

Результат: 120.

В случае повторяющихся вычислений в Лиспе могут быть использованы не только рекурсивные функции, но и известные по процедурным языкам циклы. Самым общим циклическим предложением в Лиспе является DO, имеющее следующую форму:

(DO ((nepi знач! шаг1) (пер2 знач2 шаг 2) ...)
(условие-окончания форма11 форма12 ...)

форма21 форма22 ...)

Вычисление предложения DO начинается с присваивания переменным пep1, пер2, ... начальных значений знач1, знач2, . . . соответственно; потом вычисляется условие окончания и, если оно истинно, последовательно вычисляются формы форма1i, и значение последней возвращается как результат DO-предложения. В противном случае вычисляются формы форма2i из тела предложения DO, затем значения переменных пep1, пер2, . . . изменяются на величину шага шаг1, шаг2, ... и все повторяется.

Для примера с помощью предложения DO определим функцию expt, вычисляющую n-ю степень числа х (n - целое положительное):

(defun expt (х n)

(do ((результат 1)) ; начальное значение

((= n 0 ) результат ) ; условие окончания

(setq результат (\* результат х))

(setqn(^nl))))

Результат задания функции: EXPT.

Пример вызова:

(expt 2 3)

Результат: 8.

Итеративные (циклические) и рекурсивные программы теоретически одинаковы по своим вычислительным возможностям, однако свойства итеративных и рекурсивных вариантов программ могут существенно различаться. Рекурсивные программы более короткие и содержательные. Особенно полезно использовать рекурсию в тех случаях, когда решаемая задача и обрабатываемые данные по своей сути рекурсивны, например, при обработке списков, так как списки могут рекурсивно содержать подсписки, при работе с другими динамическими структурами, которые заранее не полностью известны. Рекурсивные процедуры играют важнейшую роль почти во всех программах, связанных с искусственным интеллектом.

### 8.6. ВВОД-ВЫВОД ДАННЫХ

До сих пор рассматривался ввод и вывод данных в лисповских программах через параметры функций и свободные переменные. Для организации диалога человека с программой в Лиспе существуют специальные функции READ и PRINT.

Для вывода результатов можно использовать функцию PRINT. Это функция с одним аргументом, которая сначала вычисляет значение аргумента, а затем выводит это значение.

Например:

(PRINT (\* 2 2))

Результат: 4.

Перед выводом происходит переход на новую строку.

Функция READ читает и возвращает выражение: (READ). Как только интерпретатор встречает такое предложение, вычисления приостанавливаются до тех пор, пока не будет введен какой-либо символ или целиком выражение. Аргументов у функции READ нет, ее использование построено на побочном эффекте, состоящем именно во вводе выражения. Прочитанное выражение можно сохранить для следующего использования и обработки, например, так:

(setq input (read));

прочитанное READ выражение присваивается переменной input.

Лисповские операторы ввода-вывода очень гибки, их можно использовать в качестве аргументов других функций. Для более эстетичного оформления вывода можно использовать функции PRINC, печатающую строку без окаймляющих кавычек и со специальными символами, а также TERPRI, переводящую строку.

Для форматного вывода (в соответствии с некоторым образом) существует функция FORMAT, обладающая гибкими возможностями, описанными в руководствах по языку Лисп.

Помимо стандартных устройств ввода-вывода, может осуществляться обработка файлов на магнитных носителях, загружаться из файлов определения функций и т.д.

###

### 8.7. ПРИМЕР ПРОГРАММИРОВАНИЯ НА ЛИСПЕ

Рассмотрим в качестве примера программирования на Лиспе менее элементарную классическую задачу, носящую название игры в «ханойские башни».

Игра состоит в следующем. Используются три вертикальных стержня А, В, С и набор N дисков разного диаметра с отверстием посередине (так что их можно надевать на стержни). В начальном положении все диски надеты на стержень А по порядку убывания диаметров: внизу самый большой, над ним - поменьше и т.д., а наверху - самый маленький. Целью является перенос всех дисков со стержня А на стержень В по следующим правилам:

1) за один раз можно перенести только один диск;

2) больший по размеру диск нельзя положить на меньший;

3) третий стержень С можно использовать как вспомогательный. Алгоритм решения задачи можно представить в виде трех следующих рекурсивных подзадач:

1) перенести со стержня А N-1 дисков на вспомогательный стержень С;

1. перенести нижний диск со стержня А на стержень В;
2. перенести со стержня С N-1 дисков на стержень В.

Программа состоит из трех последовательно определяемых функций «ханойские-башни», «перенос», «выведи» и имеет вид:

*Программа 130*

(defun ханойские-башни (высота)

(рrоgn (перенос "а "Ь "с высота) "готово))

ХАНОЙСКИЕ-БАШНИ

(defun перенос (из в вспомогательный n)

(cond

((= п 1) ; ветвь 2

(выведи из в) (t (перенос из ; ветвь1 вспомогательный

в

(- n 1))

(выведи из в)

(перенос вспомогательный ; ветвь 3

в

из

(- п 1)))))

ПЕРЕНОС

(defun выведи (из в)

(format t "~S -> ~S~%"из в))

ВЫВЕДИ

Вызов функции «ханойские башни» дает такое решение:

(ханойские-башни 3)

А->В

А->С

В->C

А->В

С->А

С->В

А->В

ГОТОВО

Можно убедиться, что определенная нами функция дает правильное решение для произвольного числа дисков, однако время решения задачи с увеличением числа дисков быстро возрастает.

### 8.8. СВОЙСТВА СИМВОЛОВ

В Лиспе могут быть определены, так называемые, свойства символов. Список свойств имеет вид:

(имя\_свойства1 значение1 имя\_свойства2 значение2 . .. имя\_свойстваN значениеN).

Присваивание нового свойства или изменение значения существующего осуществляется с помощью функции PUTPROP (или просто PUT):

(PUTPROP символ свойство значение).

Выяснить значение свойства, связанного с символом, можно с помощью функции GET:

(GET символ свойство).

С использованием этой функции можно также присваивать свойства символам:

(SETF (GET символ свойство) значение).

Свойства символов глобальны Эта конструкция языка Лисп полезна во многих типичных случаях представления данных, в том числе семантических сетей, фреймов и объектов объектно-ориентированного программирования.