Глава 33. AutoLISP

- Математические выражения и тригонометрические функции в AutoLISP
- Основные функции AutoLISP и их применение
- Загрузка программ на AutoLISP и их выполнение
- Диалоговое окно загрузки программ
- Применение блок-схем алгоритмов
- Проверка выполнения условий и применение условных выражений

AutoLISP является одной из реализаций языка программирования LISP. Первое упоминание о LISP встречается в «The Communication of the ACM» в 1960.

Большинство языков программирования, созданных в начале 60-х годов, безнадежно устарели. Исключение составляют лишь Фортран и Кобол. Языку LISP в этом смысле повезло, он занял ведущие позиции среди языков программирования искусственного интеллекта. Диалектами языка программирования LISP являются Common LISP, BYSCO LISP, ExperLISP, GCLISP, IQLISP, LISP/80, LISP/88, MuLISP, TLCLISP, UO-LISP, Waltz LISP и XLISP. XLISP является свободно распространяемым интерпретатором языка LISP. Наибольшее сходство с AutoLISP имеет Common LISP. Начиная с версии 2.18, интерпретатор AutoLISP входит в состав программного обеспечения AutoCAD.

Программное обеспечение AutoCAD содержит большинство команд, применяемых при работе с чертежами. Однако некоторые операции AutoCAD не выполняет. Например, в AutoCAD отсутствует команда для вычерчивания прямоугольников, нельзя также выполнить общее редактирование текстовых объектов чертежа. Сделать это можно с помощью программ на языке AutoLISP. Любую AutoLISP-программу можно встроить в меню и таким образом существенно повысить эффективность работы с AutoCAD.

Язык программирования AutoLISP уже применялся сотнями сторонних разработчиков программного обеспечения для создания различных прикладных программ. Например, автор этой книги создал программу SMLayout, которая создает проекции на плоскость поперечного сечения водовода, днища котла, узла пересечения труб и т. п. Умение разрабатывать программы на AutoLISP чрезвычайно полезно при создании различных приложений и пользовательских меню.

Предполагается, что читатель, изучающий эту главу, знаком с командами и системными переменными AutoCAD. Знаний AutoCAD на уровне эксперта, равно как и специальных знаний в области программирования, не требуется. Разумеется, знакомство с какими-либо другими языками программирования поможет при

изучении AutoLISP. Надеемся, что основательное обсуждение различных функций языка и постепенное, пошаговое объяснение и разбор примеров сделают процесс освоения AutoLISP не слишком обременительным и для тех, кто никогда не писал программ. В этой главе рассматриваются основные функции языка AutoLISP и их применение в программах. С функциями, которые в настоящей главе не рассматриваются, можно познакомиться в документе под названием AutoLISP Programmers Reference Manual компании Autodesk. Какого-либо дополнительного аппаратного обеспечения для работы с AutoLISP не требуется. Если на компьютере работает AutoCAD, то будет работать и AutoLISP. Программы на языке AutoLISP (AutoLISP-программы) можно писать в любом текстовом редакторе.

Математические операции

Математические функции являются важной частью любого языка программирования. В AutoLISP поддерживается большинство математических функций, используемых в вычислениях. В AutoLISP выполняются четыре арифметических действия, вычисляются синусы и косинусы углов, выраженных в радианах; из обратных тригонометрических функций в AutoLISP присутствует вычисление значения арктангенса. Ниже рассматриваются основные математические функции, поддерживаемые AutoLISP.

Сложение

Формат:

```
(+ число1 число2 число 3…)
```

Эта функция вычисляет сумму всех чисел, стоящих справа от знака сложения (+). Числа могут быть вещественными или целыми. При вещественных слагаемых их сумма также будет вещественным числом. Если же складываются целые числа, то и сумма их будет целым числом. Если среди слагаемых присутствуют как целые, так и вещественные числа, то их суммой будет вещественное число. В приведенных ниже примерах в двух первых случаях все слагаемые — целые числа, целым числом выражается и результат их сложения. В третьей строке одно из слагаемых вещественное число (50,0), соответственно и сумма здесь будет числом вещественным.

Примеры:

Операция сложения в AutoLISP	**Значение**
Command: (+ 2 5)	7
Command: (+ 2 30 4 50)	86
Command: (+ 2 30 4 50.0)	86.0

Вычитание

Формат:

```
(- число1 число2 число3…)
```

В результате вычитания второе число вычитается из первого: `(число1 - число2)`. Когда эта операция выполняется над более чем двумя числами, второе число складывается с последующими, и их сумма вычитается из первого числа: `[число1 - (число2 + число3…)]`. В первом из приведенных примеров число 14 вычитается из 28. Поскольку оба числа целые, то и результатом является целое число (14). В третьем примере числа 20 и 10,0 складываются, в результате получается вещественное число 30,0, которое и вычитается из 50. В результате получается вещественное число 20,0.

Примеры:

Операция вычитания в AutoLISP	**Значение**
Command: (– 28 14)	14
Command: (– 25 7 11)	7
Command: (– 50 20 10.0)	20.0
Command: (– 20 30)	–10
Command: (– 20.0 30.0)	–10.0

Умножение

Формат:

```
(* число1 число2 число3…)
```

Эта функция возвращает значение произведения чисел, записанных справа от знака умножения (`*`). Если все сомножители являются целыми числами, их произведение также будет целым числом. Если хотя бы один из сомножителей — вещественное число, то и произведение будет вещественным числом.

Примеры:

Операция умножения в AutoLISP	**Значение**
Command: (* 2 5)	10
Command: (* 2 5 3)	30
Command: (* 5 2 3 2.0)	60.0
Command: (* 2 -5.5)	-11.0
Command: (* 2.0 -5.5 -2)	-22.0

Деление

Формат:

```
(/ число1 число2 число3…)
```

Эта функция возвращает значение частного от деления первого числа на второе:

(число1/число2). Если функция имеет более двух аргументов, то она возвращает значение частного от деления первого числа на произведение всех остальных чисел: [число1/(число2*число3*...)]. В четвертом примере вещественное число 200 делится на произведение вещественного числа 5,0 и целого числа 4: [200/(5,0*4)].

Примеры:

Операция деления в AutoLISP	**Значение**
Command: (/ 30)	30
Command: (/ 3 2)	1
Command: (/ 3 2.0)	1.5
Command: (/ 200.0 5.0 4)	10.0
Command: (/ 200 -5)	–40
Command: (/ -200 -5.0)	40.0

Инкремент, декремент и абсолютная величина числа

Инкремент числа

Формат:

```
(1+ число)
```

Оператор (1+) прибавляет к целому числу единицу.

Примеры:

Операция инкремента в AutoLISP	**Значение**
Command: (1+ 20)	21
Command: (1+ –10.5)	–9.5

Декремент числа

Формат:

```
(1- число)
```

Оператор (1–) вычитает из целого числа единицу.

Примеры:

Операция декремента в AutoLISP	**Значение**
Command: (1– 10)	9
Command: (1– –10.5)	–11.5

Абсолютная величина числа

Формат:

```
(abs число)
```

Функция `abs` возвращает значение абсолютной величины (модуля) числа. Аргумент может быть как целым числом, так и вещественным.

Примеры:

Функция модуля в AutoLISP	**Значение**
Command: (abs 20)	20
Command: (abs –20)	20
Command: (abs –20.5)	20.5

Тригонометрические функции

Синус (sin)

Формат:

```
(sin величина_угла)
```

Функция `sin` вычисляет синус угла, величина которого задана в радианах. Во втором примере вычисляется синус угла π радиан (180°) и возвращаемое значение, естественно, равно нулю.

Примеры:

Функция sin в AutoLISP	**Значение**
Command: (sin 0)	0.0
Command: (sin pi)	0.0
Command: (sin 1.0472)	0.866027

Косинус (cos)

Формат:

```
(cos величина_угла)
```

Функция `cos` вычисляет косинус угла, величина которого задана в радианах. В третьем примере вычисляется косинус угла π радиан (180°) и возвращаемое значение, естественно, равно –1,0.

Примеры:

Функция cos в AutoLISP	Значение
Command: (cos 0)	1.0
Command: (cos 0.0)	1.0
Command: (cos pi)	–1.0
Command: (cos 1.0)	0.540302

Арктангенс (atan)

Формат:

```
(atan число1)
```

Функция `atan` (арктангенс) вычисляет величину угла, тангенс которого равен ее аргументу (`число1`). Функция `atan` возвращает значение угла в радианах.

Примеры:

Функция atan в AutoLISP	Значение (радианы)
Command: (atan 0.5)	0.463648
Command: (atan 1.0)	0.785398
Command: (atan –1.0)	–0.785398

Функция может `atan` иметь и два аргумента.

Формат:

```
(atan число1 число2)
```

Если задан второй аргумент, функция `atan` возвращает значение угла (в радианах), образованного вектором с положительным направлением оси X. Аргументами функции `atan` здесь являются координаты вектора: `число1` — *Y*-координата вектора, `число2` — *X*-координата вектора. Функция `atan` возвращает значение угла в интервале от 0 до π и от –π до 0.Примеры:

Функция atan в AutoLISP	Значение (радианы)
Command: (atan 0.5 1.0)	0.463648
Command: (atan 2.0 3.0)	0.588003
Command: (atan 2.0 –3.0)	2.55359
Command: (atan –2.0 3.0)	–0.588003
Command: (atan –2.0 –3.0)	–2.55359
Command: (atan 1.0 0.0)	1.5708
Command: (atan –0.5 0.0)	–1.5708

Функция angtos

Формат:

```
(angtos величина_угла [единица_измерения_углов [точность]])
```

Функция `angtos` возвращает значение угла, выраженное в радианах, в строковом формате. Возвращаемое значение задается значениями параметров: типом единиц измерения углов и точностью (количеством десятичных знаков в дробной части).

Примеры:

Функция angtos в AutoLISP	**Значение**
Command: (angtos 0.588003 0 4)	"33.6901"
Command: (angtos 2.55359 0 4)	"146.3099"
Command: (angtos 1.5708 0 4)	"90.0000"
Command: (angtos –1.5708 0 2)	"90.00"

ПРИМЕЧАНИЕ

В оригинале функция angtos имеет формат: (angtos angle [mode [precision]]). Здесь angle — это значение угла в радианах, mode — параметр, определяеющий угловые единицы измерения в возвращаемом значении (его значение соответствует значению системной переменной AUNITS), precision — целое число, определяющее количество десятичных знаков в дробной части. Значение последнего параметра соответствует значению системной переменной AUPREC и может лежать в диапазоне от 0 до 4.

Системная переменная AUNITS может принимать следующие значения:

Параметр (mode)	**Единицы измерения углов в возвращаемом значении**
0	Градусы/доли градусов
1	Градусы/минуты/секунды
2	Грады[1]
3	Радианы
4	Геодезические единицы измерения углов

Функции сравнения

Обычно в любой программе присутствуют операции проверки выполнения определенных условий. Если условие выполняется, то программа выполняет определенные действия, если же условие не выполняется, то программа работает иначе. Например, функция сравнения `(if (< x 5))` проверяет истинность выражения `(x < 5)` в зависимости от значений, принимаемых переменной `x`. Проверки подобного типа часто встречаются в программировании. Функция сравнения имеет следующий формат:

```
if (условное_выражение)
```

---

[1] Градус астрономический, одна сотая прямого угла. — *Примеч. перев.*

Здесь:

- `if` — оператор AutoLISP;
- `условное_выражение` — выражение, содержащее проверку условия. Результат проверки может принимать два значения:
  - `T` (True, истина) — если условие выполняется;
  - `nil` (нуль, ложь) — если условие не выполняется.

В следующем разделе рассматриваются различные условные выражения, применяемые в AutoLISP.

Проверка равенства

Формат:

```
(= элемент1, элемент2 …)
```

Это выражение проверяет равенство элементов между собой. Если все элементы равны, то условие равенства выполняется и выражение возвращает результат `T`. В противном случае выражение возвращает значение `nil`.

Примеры:

Выражение	**Возвращаемое значение**
Command: (= 5 5)	T (истина)
Command: (= 5 4.9)	nil (ложь)
Command: (= 5.5 5.5 5.5)	T
Command: (= "yes" "yes")	T
Command: (= "yes" "yes" "no")	Nil

Проверка неравенства

Формат:

```
(/= элемент1, элемент2 …)
```

Это выражение выполняет проверку неравенства сравниваемых элементов. Если они не равны, то условие выполняется и выражение возвращает значение `T`. В противном случае выражение возвращает значение `nil`.

Примеры:

Выражение	**Возвращаемое значение**
Command: (/= 50 4)	T

Command: (/= 50 50)	Nil
Command: (/= 50 –50)	T
Command: (/= "yes" "no")	T

Сравнение «меньше»

Формат:

```
(< элемент1, элемент2 …)
```

Это выражение сравнивает по величине первый элемент со вторым. Возвращает значение `T`, если первый элемент меньше второго, и значение `nil` в противном случае.

Примеры:

Выражение	**Возвращаемое значение**
Command: (< 3 5)	T
Command: (< 5 3 4 2)	Nil
Command: (< "x" "y")	T

Сравнение «меньше или равно»

Формат:

```
(<= элемент1, элемент2 …)
```

Это выражение сравнивает по величине первый элемент со вторым. Возвращает значение `T`, если первый элемент меньше или равен второму. Возвращает значение `nil` в противном случае.

Примеры:

Выражение	**Возвращаемое значение**
Command: (<= 10 15)	T
Command: (<= -2.0 0)	T
Command: (<= "c" "b")	Nil

Сравнение «больше»

Формат:

```
(> элемент1, элемент2 …)
```

Это выражение сравнивает по величине первый элемент со вторым. Возвращает значение `T`, если первый элемент больше второго. Возвращает значение `nil` в противном случае.

Примеры:

Выражение	Возвращаемое значение
Command: (> 15 10)	T
Command: (> 10 9 9)	Nil
Command: (> "c" "b")	T

Сравнение «больше или равно»

Формат:

```
(>= элемент1, элемент2 …)
```

Это выражение сравнивает по величине первый элемент со вторым. Возвращает значение `T`, если первый элемент больше или равен второму. Возвращает значение `nil` в противном случае.

Примеры:

Выражение	Возвращаемое значение
Command: (>= 78 50)	T
Command: (>= "x" "y")	T

Функции defun, setq, getpoint и Command

Функция defun

Оператор (функция) `defun` применяется для определения пользовательской функции в AutoLISP-программе.

Формат:

```
(defun имя_функции [список_аргументов])
```

Примеры:

```
(defun ADNUM())
```

В этом примере функция `ADNUM` определяется без аргументов. Это означает, что все переменные, используемые в этой программе, — глобальные. Глобальная переменная сохраняет свое значение после завершения программы.

```
(defun ADNUM(a b c))
```

Здесь функция `ADNUM` имеет три аргумента — `a`, `b` и `c`. Значения этим переменным присваиваются в другой части программы в момент обращения к функции `ADNUM`.

```
(defun ADNUM(/ a b))
```

Здесь функция ADNUM имеет две локальные переменные a и b. Локальная переменная используется только внутри программы и сохраняет свое значение только в течение ее выполнения.

```
(defun C:ADNUM())
```

Если перед именем определяемой функции стоят символы C:, то для ее вызова из командной строки AutoCAD достаточно просто ввести ее имя. Если символы C: не используются, то при вызове функции ее имя нужно заключать в круглые скобки.

ПРИМЕЧАНИЕ

AutoLISP содержит достаточно большое количество встроенных функций. Их имена нельзя использовать для переменных или функций, определяемых пользователем. Полный список встроенных функций имеется в AutoLISP Programmer's manual. Ниже приводится список основных встроенных функций.

abs	ads	alloc
and	angle	angtos
append	apply	atom
asci	assoc	atan
atof	atoi	distance
equel	fix	float
if	length	list
load	member	nil
not	nth	null
open	or	pi
read	repeat	reverse
set	type	while

Функция setq

Функция setq используется для присвоения значений переменным.

Формат:

```
(setq имя_переменной значение [имя_переменной значение]…)
```

Переменной может быть присвоено численное, строковое или буквенно-цифровое значение. Строковая переменная может состоять не более чем из 100 символов.

Примеры:

```
Command: (setq X 12)
```

```
Command: (setq X 6.5)
Command: (setq X 8.5 Y 12)
```

В последнем примере переменной `X` присваивается значение 8,5, а переменной `Y` — 12.

```
Command: (setq answer "YES")
```

В этом примере переменной answer присваивается строковая величина `"YES"`.

В качестве аргументов функции `setq` могут использоваться также и другие выражения. Например:

```
Command: (setq pt1 (getpoint "Enter start point:"))
Command: (setq ang1 (getangle "Enter included angle:"))
Command: (setq answer (getstring "Enter YES or NO:"))
```

Функция getpoint

Функция `getpoint` создает паузу для задания координат точки — *X*, *Y* или *X*, *Y*, *Z*. Координаты точки можно ввести с клавиатуры или указав точку на экране курсором. Функция `getpoint` имеет следующий формат:

```
(getpoint [point] [prompt])
```

Здесь:

- `point` — выбор точки курсором или ввод координат с клавиатуры;
- `prompt` — текст подсказки, отображаемый на экране.

Пример:

```
Command: (setq pt1 (getpoint))
Command: (setq pt1 (getpoint "Укажите начальную точку:"))
```

ПРИМЕЧАНИЕ

В качестве ответа на запрос функции getpoint нельзя указывать имя другой AutoLISP-подпрограммы.

Двумерные или трехмерные координаты точки всегда определяются в пользовательской системе координат (UCS).

Функция command

Функция `command` используется для выполнения команд AutoCAD внутри AutoLISP-программ. Имя команды и параметры выполнения при этом заключаются в кавычки. Функция `command` имеет следующий формат:

```
(command "Имя_команды")
```

Пример:

```
Command: (command "line" pt1 pt2)
```

Здесь:

- `"line"` — команда AutoCAD;
- `pt1` — начальная точка линии;
- `pt2` — конечная точка линии;
- `""` — пробел как знак возврата каретки.

ПРИМЕЧАНИЕ

В ранних версиях AutoCAD (12 и более ранние) функция command не могла использоваться для выполнения команды PLOT. Например, код AutoLISP-программы (command "plot") считался ошибочным. Использование команды PLOT в составе функции command в AutoCAD стало возможным начиная с версии v13.

Функция command не используется для ввода данных с командами DTEXT и TEXT, хотя использовать эти команды для задания высоты или угла поворота текста вместе с функцией command можно.

Функцию command нельзя использовать с функциями, связанными с присваиванием значений переменным. К этим функциям относятся: getpoint, getangle, getstring и getint. Например, выражение (command "getpoint") является некорректным. При запуске программы, содержащей такое выражение, появится сообщение об ошибке.

Пример 1

Создадим AutoLISP-программу, выполняющую построение треугольника по трем заданным пользователем точкам (рис. 33.1).

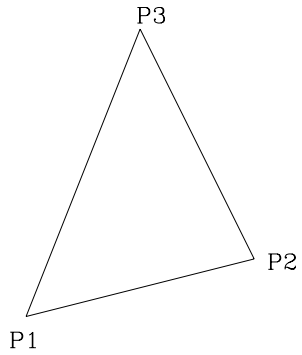

Рис. 33.1. Построение треугольника по трем точкам (P1, P2, P3)

Большинство программ состоит из трех основных частей: ввод данных, операции с данными и вывод результата (рис. 33.2). Перед написанием программы полезно определить эти три части. В рассматриваемом примере входными данными являются координаты трех точек. В результате выполнения программы (операции с данными) должен быть нарисован треугольник. Собственно выполнение программы состоит в проведении трех линий: от точки P1 к точке P2, от точки P2 к точке P3 и от точки P3 к точке P1. Представление программы в виде трех блоков делает ее более наглядной и способствует уменьшению вероятности внесения ошибок.

Раздел программы, в котором выполняется обработка входных данных, может быть как простым, так и сложным, включающим в себя много расчетов. Если в программе много вычислений, то раздел обработки данных целесообразно разбить на отдельные блоки. Текст программы должен быть написан таким образом, чтобы был виден порядок ее выполнения и можно было понять, что она делает на каждом из конкретных этапов. Если это возможно, предусматривайте контрольные точки для проверки промежуточных результатов.

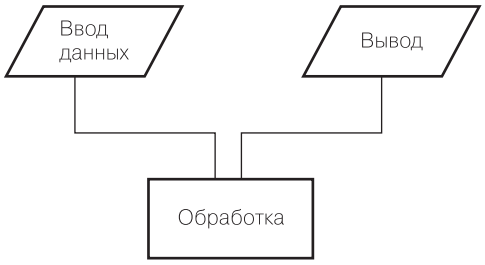

Рис. 33.2. Основные элементы программы

Вернемся к нашему примеру и сформулируем задачи для каждой из трех основных частей программы.

Ввод данных	Операции с данными	Вывод результата
Задание координат точки P1	Построение линии между точками P1 и P2	Отображение на экране треугольника с вершинами в точках P1, P2, P3
Задание координат точки P2	Построение линии между точками P2 и P3	
Задание координат точки P3	Построение линии между точками P3 и P1	

Листинг программы приводится ниже, номера строк справа не являются частью файла, а служат исключительно для удобства ссылок при пояснениях.

```
;Это программа построения треугольника по трем точкам.      1
;Координаты точек вводятся с клавиатуры или                 2
;указываются курсором на экране в ответ на запрос в        3
;командной строке.                                          4
                                                            5
(defun c:TRIANG1()                                          6
(setq P1 (getpoint "\n Введите первую вершину"))            7
(setq P2 (getpoint "\n Введите вторую вершину"))            8
(setq P3 (getpoint "\n Введите третью вершину"))            9
(command "LINE" P1 P2 P3 "C"                               10
)                                                          11
```

Пояснения

Строки 1–4

Первые четыре строки отведены под комментарии к программе. Строка комментария должна начинаться с символа точки с запятой.

Строка 5

Пустая строка отделяет одну часть программы от другой; вводится для улучшения зрительного восприятия текста программы.

Строка 6

```
(defun c:TRIANG1()
```

Это первая строка определения функции построения треугольника.

Здесь:

- `TRIANG1` — имя функции;
- `c:` — знак отмены скобок при запуске программы из командной строки AutoCAD;

- `defun` — оператор определения функции в AutoLISP-программе.

Функция `TRIANG1` имеет три глобальных параметра: `P1`, `P2`, `P3`. При написании первой программы на AutoLISP полезно оставлять глобальные переменные, так как в этом случае значение переменной можно проверить после загрузки и запуска программы. Для этого в командной строке AutoCAD нужно ввести восклицательный знак и имя переменной.

```
Command: !P1
```

После отладки программы переменные можно сделать локальными. Для этого строку 6 следует записать следующим образом:

```
(defun c:TRIANG1(/ P1 P2 P3)
```

Строка 7

```
(setq P1 (getpoint "\n Введите первую вершину"))
```

В этой строке функция `getpoint` создает паузу для ввода координат первой вершины треугольника. В строке подсказки отображается текст `Введите первую вершину`. Можно ввести численные значения координат точки с клавиатуры или просто указать точку на экране курсором. После этого функция `setq` присвоит координаты точки переменной `P1`. Символ `\n` используется для перевода строки, чтобы текст, записанный после `\n`, начинался с новой строки. Ключ `n` означает здесь «new line» (новая строка).

Строки 8 и 9

```
(setq P2 (getpoint "\n Введите вторую вершину"))
(setq P3 (getpoint "\n Введите третью вершину"))
```

В этих строках пользователю предлагается ввести вторую и третью вершины треугольника. Введенные координаты точек присваиваются переменным `P2` и `P3`.

Строка 10

```
(Command "LINE" P1 P2 P3 "C"
```

Здесь функция `command` используется для выполнения команды `LINE`, которая соединит линиями три указанные точки. Параметр `C` (`Close`) показывает, что последняя точка (P3) соединяется с первой точкой (P1). При использовании в AutoLISP названия всех команд AutoCAD и их параметры заключаются в кавычки. Имена переменных разделяются пробелами.

Строка 11

```
)
```

В этой строке стоит закрывающая круглая скобка, которая указывает на завершение определения функции `TRIANG1`. В принципе, эту скобку можно было поставить и в предыдущей строке — это не противоречит синтаксису. Однако такая запись делает текст программы более наглядным и легко читаемым, позволяя легко найти место

окончания определения функции. В случае более сложных программ, с большим количеством модулей и определяемых функций следует постоянно применять пустые строки и строки со скобками для секционирования текста программы.

Загрузка AutoLISP-программы с помощью диалогового окна Load/Unload Application

С программами на AutoLISP связаны два вида имен — имена файлов и имена функций. Например, TRIANG.LSP является именем файла, `TRIANG1` — именем функции. Все файлы программ на AutoLISP имеют расширение .lsp. Внутри одного файла может быть определено несколько функций. Для выполнения функции файл программы, в котором она определена, необходимо загрузить. Это можно осуществить двумя способами: через диалоговое окно, либо путем ввода команды в командной строке.

Диалоговое окно Load/Unload Application используется для загрузки приложений, созданных не только на языке AutoLISP, но и другими средствами. С помощью этого диалогового окна можно загружать также приложения VLX, FAS, VBA, DBX и ObjectARX. Файлы приложений VBA DBX и ObjectARX загружаются сразу же после выбора их в окне. Файлы приложений LSP, VLX и FAS загружаются после закрытия диалогового окна. В верхней части диалогового окна приводится список файлов, находящихся в выбранной папке. Чтобы изменить тип файлов, отображаемых в этом списке, следует выбрать другой тип из раскрывающегося списка Files of type (Тип файлов). Для загрузки файла после его выбора в списке следует щелкнуть на кнопке Load (Загрузить). Ниже приводится описание основных элементов диалогового окна Load/Unload Application.

Кнопка Load

Кнопка Load используется для первой и повторной загрузки выбранных файлов. Кроме списка в верхней части диалогового окна файлы также могут быть выбраны и из списков, соответствующим вкладкам Loaded Application и History List, в нижней части диалогового окна.

Вкладка Loaded Application

При выборе этой вкладки отображается список приложений, которые к этому моменту времени уже загружены. Загрузку файла можно выполнить перетаскиванием имени файла из верхнего списка в список Loaded Application.

Вкладка History List

При выборе этой вкладки отображается список приложений, которые загружались во время предыдущих сеансов работы с AutoCAD. Чтобы файл попал в этот список, следует установить флажок Add to History.

Кнопка Unload

Кнопка Unload становится доступной при выборе файла из списка Loaded Application. Выберите нужный файл, щелкните на кнопке Unload, и приложение будет выгружено.

Кнопка Remove

Кнопка Remove становится доступной при выборе файла из списка History List. Для удаления файла из этого списка следует выбрать нужный файл и щелкнуть на кнопке Remove.

Группа Startup Suit

Файлы, находящиеся в этой группе, автоматически загружаются при каждом запуске AutoCAD. При щелчке на кнопке Contents (Содержимое) или значке портфеля в группе Startup Suit появляется диалоговое окно Startup Suit (Стартовый набор) со списком файлов. Чтобы приложение автоматически загружалось при запуске AutoCAD, следует щелкнуть на кнопке Add (Добавить). Откроется диалоговое окно Add File to Startup Suit. Следует выбрать нужный файл и щелкнуть на кнопке Add. Добавить файлы в список автоматически загружаемых файлов можно также их перетаскиванием в открытое диалоговое окно Startup Suit. Щелчок правой кнопкой мыши на имени файла в списке History List также приводит к добавлению этого файла в список Startup Suit.

В режиме редактирования чертежа файл программы можно загрузить из командной строки.

```
Command: (load"[path]file name")
```

Здесь:

- `load` — команда для загрузки файла программы;
- `"[path]file name"` — имя файла с указанием пути к нему.

Имя файла программы с указанием пути должно быть заключено в кавычки, а вся строка, включая команду `load`, — в круглые скобки. При отсутствии скобок AutoCAD будет пытаться загрузить текстовый файл или файл чертежа вместо AutoLISP-программы. Пробел между командой `load` и именем файла не нужен. В случае успешной загрузки имя функции отображается в командной строке AutoCAD.

Для запуска программы введите имя функции в командной строке и нажмите клавишу Enter.

```
Command: (TRIANG1)
```

или

```
Command: TRIANG1
```

Круглые скобки нужны в том случае, если в файле программы в имени функции после оператора `defun` отсутствуют символы `c:`.

ПРИМЕЧАНИЕ

При указании пути к файлу AutoLISP-программы можно использовать косую черту или двойную обратную косую черту. Например:

Command: (load "c:/lisp/triang") или Command: (load "c:\\lisp\\triang").

СОВЕТ

Для загрузки приложения можно использовать и стандартную для Windows процедуру перетаскивания объектов мышью. Выделите соответствующий файл в окне программы Проводник и перетащите значок файла в графическое окно AutoCAD.

Упражнение 1

Напишите программу на языке AutoLISP, выполняющую построение линии между двумя заданными точками (рис. 33.3). Программа должна предусматривать ввод пользователем координат точек P1 и P2.

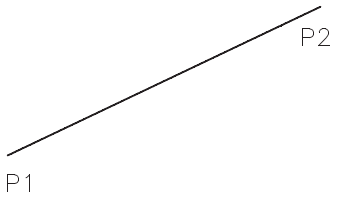

Рис. 33.3. Линия, проведенная между точками P1 и P2

Функции getcorner, getdist и setvar

Функция getcorner

Функция `getcorner` создает паузу для задания координат точки. Можно ввести координаты точки с клавиатуры или указать точку на экране курсором. Для функции `getcorner` требуется задать базовую точку. Точка, указываемая пользователем, становится противоположной вершиной воображаемого прямоугольника. При

движении курсора этот прямоугольник отображается на экране. Функция `getcorner` имеет следующий формат:

```
(getcorner point[prompt])
```

Здесь:

- `point` — базовая точка;
- `prompt` — подсказка, отображаемая на экране. Текст должен быть заключен в кавычки.

ПРИМЕЧАНИЕ

Координаты базовой точки и точки, введенной пользователем, отсчитываются в пользовательской системе координат (UCS).

Если точка, указанная пользователем, имеет три координаты *X*, *Y* и *Z*, то координата *Z* игнорируется.

Функция getdist

Функция `getdist` создает паузу для задания пользователем расстояния и затем возвращает значение расстояния в виде вещественного числа.

Функция gedist имеет следующий формат:

```
(getdist point[prompt])
```

Здесь:

- `point` — базовая точка;
- `prompt` — предложение, отображаемое на экране.

Примеры:

```
(getdist)
(setq dist (getdist))
(setq dist (getdist pt1))
(setq dist (getdist "Укажите расстояние:"))
(setq dist (getdist pt1 "Введите вторую точку для задания расстояния:"))
```

Расстояние может быть задано указанием двух точек на экране. В случае выражения `(setq dist (getdist))` пользователь может ввести число или указать две точки на экране. Если же используется выражение `(setq dist (getdist pt1))`, где одна из точек (`pt1`) уже определена, то выбора не остается, и следует указать вторую

точку. Функция `getdist` всегда возвращает значение расстояния в виде вещественного числа.

Функция setvar

Функция `setvar` присваивает значения системным переменным AutoCAD. Имя системной переменной должно быть заключено в кавычки. Функция `setvar` имеет следующий формат:

```
(setvar "variable-name" value)
```

Здесь:

- `variable-name` — имя системной переменной AutoCAD;
- `value` — значение, присваиваемое системной переменной.

Примеры:

```
(setvar "cmdecho" 0)
(setvar "dimscale" 1.5)
(setvar "ltscale" 0.5)
(setvar "dimcen" -0.25)
```

Пример 2

Напишем программу, строящую фаску между двумя линиями. Исходными данными являются угол фаски и ее длина. Обычно для создания фаски AutoCAD использует значения двух системных переменных — `CHAMFERA` и `CHAMFERB`. При запуске команды `CHAMFER` длины двух сторон фаски автоматически присваиваются этим двум переменным. При следующем запуске команды `CHAMFER` указывается два отрезка, являющиеся кромками, между которыми создается фаска. AutoCAD создает фаску, используя полученную ранее информацию (длины двух сторон фаски). В большинстве технических чертежей предпочтение отдается построению фаски по одной из сторон и углу по отношению к этой стороне (рис. 33.4).

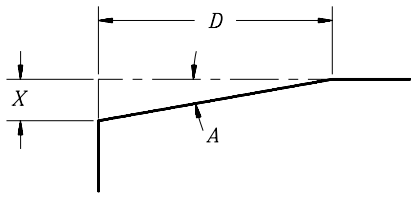

Рис. 33.4. Построение фаски по одной из сторон и углу

Таблица 33.1. Схема программы из примера 2

Ввод данных	**Операции с данными**	**Вывод результата**
Задание длины одной из сторон фаски	Определение длины второй стороны фаски	Фаска между двумя выбранными кромками
Задание угла фаски	Присвоение значений системным переменным CHAMFERA и CHAMFERB	
	Выполнение команды CHAMFER для создания фаски	
	Определение длины второй стороны фаски	
	Присвоение значений системным переменным CHAMFERA и CHAMFERB	
	Выполнение команды CHAMFER для создания фаски	
	Требуемые вычисления	
	x = d * tan a	
	Оператор AutoLISP	
	= d * [(sin a)/cos a)]	

Листинг программы приводится ниже, номера строк справа не являются частью файла, а служат исключительно для удобства ссылок при пояснениях.

```
;Это программа построения фаски по одной из ее              1
;сторон и углу.                                             2
;                                                           3
(defun c:chamf(/ d a)                                       4
(setvar "cmdecho" 0)                                        5
(graphscr)                                                  6
(setq d (getdist "\n Введите длину фаски:"))                7
(setq a (getangle "\n Введите угол фаски:"))                8
(setvar "chamfera" d)                                       9
```

```
(setvar "chamferb" (*d (/sin a) (cos a))))                10
(command "chamfer")                                       11
(setvar "cmdecho" 1)                                      12
(princ)                                                   13
)                                                         14
```

Пояснения

Строка 7

```
setq d (getdist "\n Введите длину фаски:"))
```

Функция `getdist` создает паузу для ввода пользователем длины одной из сторон фаски, затем функция `setq` присваивает введенное значение переменной `d`.

Строка 8

```
(setq a (getangle "\n Введите угол фаски:"))
```

Функция `getangle` создает паузу для ввода пользователем значения угла фаски, затем функция `setq` присваивает введенное значение переменной `a`.

Строка 9

```
(setvar "chamfera" d)
```

Функция `setvar` присваивает системной переменной `chamfera` значение `d`.

Строка 10

```
(setvar "chamferb" (*d (/sin a) (cos a))))
```

Функция `setvar` присваивает системной переменной `chamferb` вычисленное значение выражения `(*d (/sin a) (cos a))`.

Строка 11

```
("chamfer")
```

Функция `command` использует команду AutoCAD `CHAMFER` для создания фаски.

Упражнение 2

Напишите программу, при помощи которой можно создать чертеж, изображенный на рис. 33.5. Для работы программа должна запрашивать координаты точек P1 и P2, а также диаметров D1 и D2.

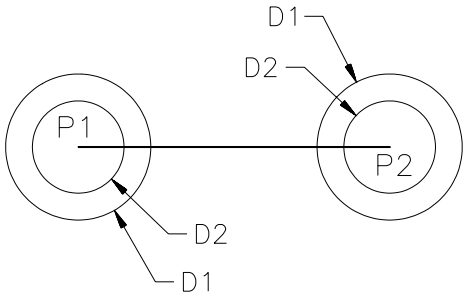

Рис. 33.5. Концентрические окружности, соединенные линией

Функция list

Функция `list` применяется в AutoLISP для задания точек на плоскости или в трехмерном пространстве. Вместо слова `list` в тексте программы можно ставить апостроф, если выражение не содержит других переменных или определяемых элементов.

Примеры:

```
(Setq x (list 2.5 3.56))

(Setq x '(2.5 3.56))
```

В обоих случаях возвращается пара чисел (координаты точки) — `(2.5 3.56)`

Функции car, cdr и cadr

Функция car

Функция `car` возвращает первый элемент списка. Формат: `(car list)`. Здесь слово `list` обозначает список элементов.

Функция	**Возвращаемое значение**
Command: (car '(2.5 3 56))	2.5
Command: (car '(x y z))	X
Command: (car '((15 20) 56))	(15 20)
Command: (car '())	Nil

В приведенных примерах вместо слова `list` используется апостроф.

Функция cdr

Функция `cdr` возвращает список без его первого элемента. Формат: `(cdr list)`.

Примеры:

Функция	Возвращаемое значение
Command: (cdr '(2.5 3 56))	(3 56)
Command: (cdr '(x y z))	(y z)
Command: (cdr '((15 20) 56))	(56)
Command: (cdr '())	Nil

Функция cadr

Функция `cadr` объединяет действия функций `cdr` и `car` и, таким образом, возвращает второй элемент списка. Функция `cdr` удаляет первый элемент списка, а функция `car` возвращает первый элемент нового списка (то есть, второй элемент исходного списка). Формат: `(cadr list)`.

Примеры:

Функция	Возвращаемое значение
Command: (cadr '(2 3))	3
Command: (cadr '(2 3 56))	3
Command: (cadr '(x y z))	Y
Command: (cadr '((15 20) 56 24)	56

ПРИМЕЧАНИЕ

Для извлечения различных элементов из списка к функциям car, cdr и cadr могут быть добавлены некоторые другие. Список этих функций приводится ниже. В этом списке функция f состоит из списка '((x y) z):

(setq f'((x y) z w))

Функция	Выполняемые операции	Возвращаемое значение
(caar f)	(car (car f))	X
(cdar f)	(cdr ((car f))	Y
(cadar f)	(car (cdr (car f)))	Y
(cddr f)	(cdr (cdr f))	W
(caddr f)	(car (cdr (cdr f)))	W

Функции graphscr, textscr, princ и terpri

Функция graphscr

Если к компьютеру подключен один монитор, то функция `graphscr` выполняет переключение из текстового окна в графическое. При наличии двух мониторов эта функция игнорируется.

Функция textscr

Если к компьютеру подключен один монитор, то функция `textscr` выполняет переключение из графического окна в текстовое. При наличии двух мониторов эта функция игнорируется.

Функция princ

Функция `princ` обеспечивает вывод на печать или отображение на экране значений переменных. Если какая-либо переменная или выражение заключены в кавычки, то на экран будут выведены (или напечатаны) все заключенные в кавычки символы.

Формат:

```
(princ [переменная_или_выражение])
```

Примеры:

Функция	**Выводит на экран или печатает**
Command: (princ)	Пустую строку
Command: (princ a)	Значение переменной a
Command: (princ "Welcome")	Слово Welcome

Функция terpri

Функция `terpri` определяет, что текст строки, расположенной после оператора `terpri`, будет печататься (выводиться на экран) в следующей строке. Действие функции `terpri` аналогично действию оператора `\n`.

Примеры:

```
(setq p1 (getpoint "Enter the first point:"))(terpri)
(setq p2 (getpoint "Enter the second point:"))
```

Первая строка `Enter the first point:` (Введите первую точку) будет напечатана в командной строке. Действие оператора `terpri` приведет к возврату каретки и переходу к новой строке. Таким образом, текст второй строки `Enter the second point:` (Введите вторую точку) будет напечатан на следующей строке.

При отсутствии оператора `terpri` оба предложения появятся в одной строке:

```
Enter the first point:Enter the second point:
```

Пример 3

Напишем программу, которая строит прямоугольник по двум задаваемым пользователем вершинам противоположных углов (рис. 33.6).

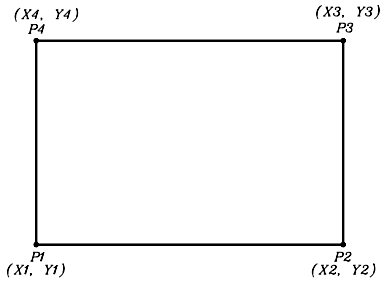

Рис. 33.6. Прямоугольник P1 P2 P3 P4

Ввод данных	**Операции с данными**	**Вывод результата**
Задание точки P1	Определение координат точек P2 и P4	Прямоугольник
Задание точки P3	Проведение линий:	
	От точки P1 к точке P2	
	От точки P2 к точке P3	
	От точки P3 к точке P4	
	От точки P4к точке P1	

X- и *Y*-координаты точек P2 и P4 вычисляются с помощью функций `CAR` и `CADR`. Для точки P2 функция `CAR` выделяет координату *X* из списка координат точки P3, затем функция `CADR` выделяет координату *Y* из списка координат точки P1. Аналогичным образом можно определить и координаты точки P4.

Определение *X*-координаты точки P2:

```
x2 = x3

x2 = car(x3 y3)

x2 = car P3
```

Определение *Y*-координаты точки P2:

```
y2 = y1
```

```
y2 = cadr(x1 y1)

y2 = cadr P1
```

Определение *X*-координаты точки P4:

```
x4 = x1

x4 = car(x1 y1)

x4 = car P1
```

Определение *Y*-координаты точки P4:

```
y4 = y3

y4 = cadr(x3 y3)

y4 = cadr P3
```

Таким образом, координаты точек P2 и P4 определяются следующим образом:

```
P2 = (list (car P3) (cadr P1))

P4 = (list (car P1) (cadr P3))
```

Листинг программы для примера 3 приводится ниже, номера строк справа не являются частью файла, а служат исключительно для удобства ссылок при пояснениях.

```
;Это программа построения прямоугольника по двум            1
;задаваемым пользователем противоположным вершинам.         2
;                                                           3
(defun c:RECT1(/ p1 p2 p3 p4)                               4
(graphscr)                                                  5
(setvar "cmdecho" 0)                                        6
(prompt "RECT1 команда, строящая прямоугольник")(terpri)    7
(setq p1 (getpoint "Введите первую вершину:"))(terpri)      8
(setq p3 (getpoint "Введите вторую вершину:"))(terpri)      9
(setq p2 (list (car p3) (cadr p1)))                        10
(setq p4 (list (car p1) (cadr p3)))                        11
(command "line" p1 p2 p3 p4 "c")                           12
(setvar "cmdecho" 1)                                       13
(princ)                                                    14
)                                                          15
```

Пояснения

Строки 1–3

Эти строки отводятся под краткое описание программы (комментарии). Строка с комментариями должна начинаться с символа точки с запятой.

Строка 4

```
(defun c:RECT1(/ p1 p2 p3 p4)
```

В этой строке определяется функция `RECT1`.

Строка 5

```
(graphscr)
```

Эта функция обеспечивает переключение из текстовой части экрана в графическую.

Строка 6

```
(setvar "cmdecho" 0)
```

Здесь функция `setvar` присваивает нулевое значение системной переменной `cmdecho`. Это означает, что текст подсказки не будет отображаться на экране при выполнении программы.

Строка 7

```
(prompt "RECT1 команда, строящая прямоугольник")(terpri)
```

Инструкция `prompt` отображает на экране текст строки, заключенный в кавычки. Оператор `terpri` создает пустую строку, с тем чтобы текст следующего сообщения был напечатан на новой строке.

Строка 8

```
(setq p1 (getpoint "Введите первую вершину:"))(terpri)
```

Функция `getpoint` создает паузу для ввода координат точки P1 (первой вершины прямоугольника). Функция `setq` присваивает значения координат переменной `p1`.

Строка 9

```
(setq p3 (getpoint "Введите вторую вершину:"))(terpri)
```

Функция `getpoint` создает паузу для ввода координат точки P3 (второй, противоположной вершины прямоугольника). Функция `setq` присваивает значения координат переменной `p3`.

Строка 10

```
(p2 (list (car p3) (cadr p1)))
```

Функция `cadr` выделяет координату *Y* точки P1, а функция `car` выделяет

координату *X* точки P3. Созданная действием функций `car` и `cadr` пара чисел представляет собой координаты точки P2, значения которых присваиваются переменной `p2` с помощью функции `setq`.

Строка 11

```
(p4 (list (car p1) (cadr p3)))
```

Функция `cadr` выделяет координату *Y* точки P3, а функция `car` выделяет координату *X* точки P1. Созданная действием функций `car` и `cadr` пара чисел представляет собой координаты точки P4, значения которых присваиваются переменной `p4` с помощью функции `setq`.

Строка 12

```
(command "line" p1 p2 p3 p4 "c")
```

Функция `command` обеспечивает выполнение команды `LINE`, вычерчивающей линии между указанными точками. Параметр `"c"` обеспечивает соединение последней точки (`p4`) с первой (`p1`).

Строка 13

```
(setvar "cmdecho" 1)
```

Здесь функция `setvar` присваивает значение 1 системной переменной `cmdecho`. Это означает, что по окончании работы программы текст подсказки, как это и принято в AutoCAD, отобразится на экране.

Строка 14

```
(princ)
```

Оператор `princ` обеспечивает так называемое *чистое окончание программы*, то есть отсутствие каких-либо сообщений в командной строке. Иначе там появилось бы значение последнего выражения. На выполнение программы это никакого влияния не оказывает, но в некоторых случаях может вызвать замешательство, особенно у начинающих пользователей.

Строка 15

```
)
```

Закрывающая скобка свидетельствует о завершении определения функции `RECT1` и об окончании программы.

ПРИМЕЧАНИЕ

> В приведенной выше программе прямоугольник был построен после задания координат противоположных вершин. При этом не производилось растягивание исходного прямоугольника при перемещении курсора мыши от одной из вершин.

Построение прямоугольника с растягиванием мышью возможно при использовании функции getcorner. Листинг программы для этого случая приводится ниже.

```
;Программа построения прямоугольника с использованием
;операции перетаскивания курсора
;
(defun c:RECT2(/ p1 p2 p3 p4)
(graphscr)
(setvar "cmdecho" 0)
(prompt "RECT2 команда, строящая прямоугольник")(terpri)
(setq p1 (getpoint "Введите первую вершину:")(terpri)
(setq p3 (getcorner p1 "Введите вторую вершину:"))(terpri)
(setq p2 (list (car p3) (cadr p1)))
(setq p4 (list (car p1) (cadr p3)))
(command "line" p1 p2 p3 p4 "c")
(setvar "cmdecho" 1)
(princ)
)
```

Функции getangle и getorient

Функция getangle

Функция `getangle` создает паузу для ввода пользователем значения угла и возвращает это значение в радианах.

Формат:

```
(getangle [point][prompt])
```

Здесь:

- `point` — начальная точка угла;
- `prompt` — подсказка, отображаемая на экране. Текст должен быть заключен в кавычки.

Примеры:

```
(getangle)
(setq ang (getangle))
(setq ang (getangle pt1))
(setq ang (getangle "Введите угол конусности"))
(setq ang (getangle pt1 "Введите вторую точку угла"))
```

Результат измерения угла зависят от единиц измерения, а также от направления и

начала отсчета. Эти параметры могу быть изменены с помощью команды UNITS или изменением значений системных переменных ANGBASE и ANGDIR.

В AutoCAD по умолчанию приняты следующие начальные условия измерения углов:

- Угол измеряется относительно положительного направления оси *X*. Значение базы отсчета углов хранится в системной переменной ANGBASE.
- Положительное направление отсчета углов — против часовой стрелки. Установки направления отсчета углов хранятся в системной переменной ANGDIR.

Пример:

```
(setq ang (getangle "Введите угол"))
```

Возвращаемое значение: 3.92699.

Функция getorient

Функция getorient создает паузу для ввода пользователем значения угла и возвращает это значение в радианах.

Формат:

```
(getorient [point][prompt])
```

Здесь:

- point — начальная точка угла;
- prompt — подсказка, отображаемая на экране. Текст должен быть заключен в кавычки.

Примеры:

```
(getorient)
(setq ang (getorient))
(setq ang (getorient pt1))
(setq ang (getorient "Введите угол конусности"))
(setq ang (getorient pt1 "Введите вторую точку угла"))
```

Функция getorient во многом похожа на рассмотренную выше функцию getangle. Обе возвращают значение угла в радианах. Отличие состоит в том, что функция getorient всегда возвращает значение угла, отсчитанного в направлении против часовой стрелки от положительного направления оси *X*. Функция getorient игнорирует значение обеих системных переменных — ANGDIR и ANGBASE.

ПРИМЕЧАНИЕ

Значение угла для функций getorient и getangle можно вводить как с клавиатуры, так и указанием двух точек на экране. Если в программе присутствует выражение (setq ang (getorient pt1)), где первая точка уже определена, то отображается предложение Specify second point (укажите вторую точку). Эту точку также можно задать любым из двух указанных выше способов.

Для перевода угловых градусов в радианы используйте выражение:

Угол в радианах = (pi × (угол в градусах))/180, где pi — константа языка AutoLISP (pi = 3,14159).

Функции getint, getreal, getstring и getvar

Функция getint

Функция `getint` создает паузу для ввода числа и возвращает целое число, даже если было введено вещественное.

Примеры:

```
(getint)
(setq numx (getint))
(setq numx (getint "Введите число строк:"))
(setq numx (getint "\n Введите число строк:"))
```

Функция getreal

Функция `getreal` создает паузу для ввода числа и возвращает вещественное число, даже если было введено целое.

Примеры:

```
(getreal)
(setq realnumx (getreal))
(setq realnumx (getreal "Введите первое число:"))
(setq realnumx (getreal "\n Введите второе число:"))
```

Функция getstring

Функция `getstring` создает паузу для ввода строки и возвращает строку, даже если введенная строка состоит только из чисел.

Примеры:

```
(getstring)
(setq answer (getstring))
```

```
(setq answer (getstring "Введите Y при ответе Да, N при ответе Нет:"))
(setq answer (getstring "\n Введите Y при ответе Да, N при ответе Нет:"))
```

ПРИМЕЧАНИЕ

Вводимая строка может состоять не более чем из 132 символов. Если символов больше, то «лишние» будут игнорироваться.

Функция getvar

Функция `getvar` считывает и отображает значение системной переменной AutoCAD.

Формат:

```
(getvar "системная_переменная_AutoCAD")
```

Примеры:

Функция	**Возвращаемое значение**
Command: (getvar)	
Command: (getvar "dimcen")	0.09
Command: (getvar "ltscale")	1.0
Command: (getvar "limmax")	12.00,9.00
Command: (getvar "limmin")	0.00,0.00

ПРИМЕЧАНИЕ

Имя системной переменной обязательно должно быть заключено в кавычки.

В одном выражении можно считать значение только одной системной переменной. Чтобы определить значения нескольких системных переменных, для каждой из них необходимо составить отдельное выражение.

Функции polar и sqrt

Функция polar

Функция `polar` определяет точку по заданным углу поворота и расстоянию относительно заданной (базовой) точки. Угол задается в радианах, направление и начало отсчета — в соответствии с установками по умолчанию.

Формат:

```
(polar базовая_точка угол расстояние)
```

Пример:

```
Command: (polar (list 1 1) 0.785 1.414)
(2.00025 1.99945)
```

Здесь:

- `polar` — функция AutoLISP;
- `list (1 1)` — функция `list` преобразует координаты базовой точки в соответствии с требованиями синтаксиса AutoLISP;
- `0.785` — угол поворота относительно базовой точки;
- `1.414` — расстояние от базовой точки;
- `(2.00025 1.99945)` — координаты определяемой точки.

Функция sqrt

Функция `sqrt` вычисляет значение квадратного корня числа и возвращает найденное значение в виде вещественного числа.

Формат:

```
(sqrt число)
```

Примеры:

Функция	**Возвращаемое значение**
Command: (sqrt 144)	12.0
Command: (sqrt 144.0)	12.0
Command: (getvar "ltscale")	1.0
Command: (setq x (sqrt 57.25))	7.566373
Command: (setq x (sqrt(* 25 36.5)))	30.207615
Command: (setq x (sqrt(/ 7.5 (cos 0.75))))	3.2016035

Пример 4

Напишем программу построения равностороннего треугольника, описанного вокруг окружности (рис. 33.7). Программа должна предусматривать ввод пользователем центра и радиуса вписанной окружности.

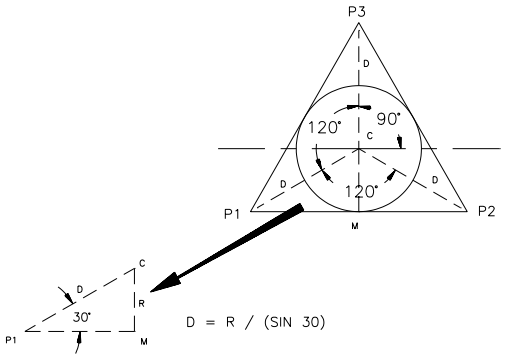

Рис. 33.7. Равносторонний треугольник, описанный вокруг окружности

Ниже приводится листинг программы, решающей задачу примера 4.

```
;Эта программа выполняет построение треугольника P1P2P3
;описанного вокруг окружности. Координаты центра
;окружности c и значение ее радиуса r вводятся пользователем.
;
(defun dtr(a)
      (* a(/ pi 180.0))
)
(defun c:trgcir(/ r c d p1 p2 p3)
(setvar "cmdecho" 0)
(graphdscr)
      (setq r(getdist "\n Enter circle radius:"))
      (setq c(getpoint "\n Enter center of circle:"))
      (setq d(/ r (sin(dtr 30))))
      (setq p1(polar c (dtr 210) d))
      (setq p2(polar c (dtr 330) d))
      (setq p3(polar c (dtr 90) d))
(command "circle" c r)
(command "line" p1 p2 p3 "c")
(setvar "cmdecho" 1)
(princ)
)
```

Упражнение 3

Напишите программу построения равнобедренного треугольника P1 P2 P3 (рис. 33.8). Основание треугольника — сторона P1 P2. Основание треугольника образует угол B с положительным направлением оси *X*. Программа должна

предусматривать ввод пользователем координат точки P1, длины основания L1 и величин углов A и B.

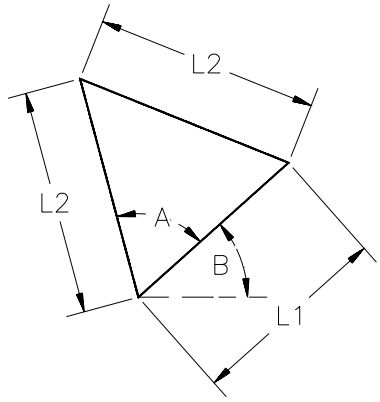

Рис. 33.8. Равнобедренный треугольник

Упражнение 4

Напишите программу построения паза с указанием осевых линий (рис. 33.9). Программа должна предусматривать ввод длины и высоты паза, а также имени слоя для построения осевых линий.

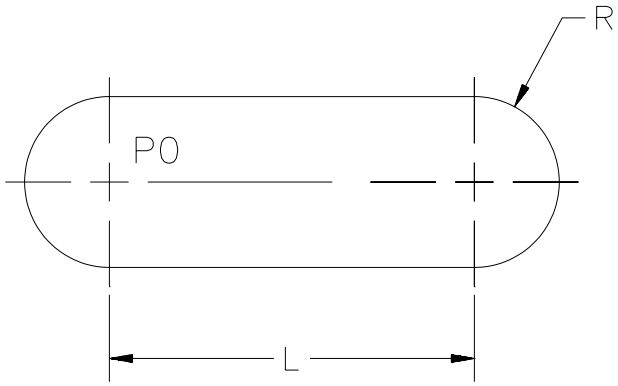

Рис. 33.9. Паз длины L и радиуса R

Функции itoa, rtos, strcase, prompt

Функции itoa

Функция `itoa` преобразует целое число в строку и возвращает значение в виде строки.

Формат:

```
(itoa целое_число)
```

Примеры:

Функция	**Возвращаемое значение**
Command: (itoa 89)	"89"
Command: (itoa –356)	"–356"
Command: (setq intnum 7)	
Command: (itoa intnum)	"7"
Command: (setq intnum 345)	
Command: (intstring (itoa intnum))	"345"

Функция rtos

Функция `rtos` преобразует вещественное число в строку и возвращает значение в виде строки.

Формат:

```
(rtos вещественное_число)
```

Примеры:

Функция	**Возвращаемое значение**
Command: (rtos 50.6)	"50.6"
Command: (rtos –30.0)	"–30.0"
Command: (setq realstrg(rtos 5.25))	"5.25"
Command: (setq realnum 75.25)	
Command: (realstring (rtos realnum))	"75.25"

Функция strcase

Функция `strcase` преобразует символы нижнего регистра в символы верхнего и наоборот.

Формат:

```
(strcase string [true])
```

Здесь:

- `string` — строка символов, регистр которых будет изменяться. Изменяемая строка должна быть заключена в кавычки;
- `true` — необязательный параметр. Если он отсутствует или значение `true` равно `nil`, все символы строки преобразуются в символы верхнего регистра. При любом другом значении параметра все символы строки преобразуются в символы нижнего регистра.

Примеры:

Функция	**Возвращаемое значение**
Command: (strcase "Добро Пожаловать")	"ДОБРО ПОЖАЛОВАТЬ"
Command: (strcase "Добро Пожаловать" 0)	"добро пожаловать"
Command: (strcase "Добро Пожаловать" 1)	"добро пожаловать"

Функция prompt

Функция `prompt` используется для отображения сообщения (подсказки) в командной строке AutoCAD. Текст сообщения должен быть заключен в кавычки.

Формат:

```
(prompt сообщение)
```

Примеры:

```
(prompt "Введите диаметр окружности:")
(setq d (getdist (prompt "Введите диаметр окружности:")
```

Пример 5

Напишем программу построения двух окружностей радиусов R1 и R2 и двух касательных к ним (табл. 33.2). Окружности изображают два шкива, а касательные — ремень. Расстояние между центрами шкивов — d. Линия, соединяющая центры шкивов, расположена под углом alpha к оси *X* (рис. 33.10).

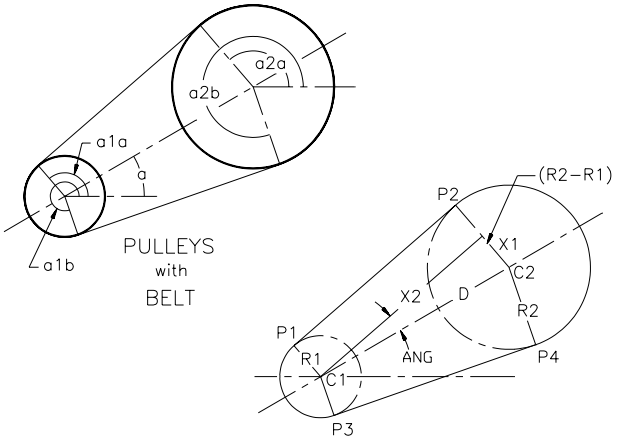

Рис. 33.10. Эскиз механизма с двумя шкивами и ремнем

Таблица 33.2. Схема программы из примера 5

Ввод данных	**Операции с данными**	**Вывод результата**
Радиус малого шкива R1	Вычисление расстояний x1 и x2	Окружность радиуса R1
Радиус большого шкива R2	Вычисление угла ang	Окружность радиуса R2
Расстояние между центрами шкивов d	Определение положения точки C2 относительно точки C1	Два отрезка линий, касательных к окружностям
Угол наклона линии, соединяющей центры шкивов (alpha)	Определение положения точек P1, P2, P3, P4	
	Построение малой окружности радиуса R1 с центром в точке C1	
	Построение большой окружности радиуса R2 с центром в точке C2	
	Построение линии между точками P1 и P2	
	Построение линии между точками P3 и P4	

Листинг программы для примера 5 приводится ниже, номера строк справа не являются частью файла, а служат исключительно для удобства ссылок при пояснениях.

```
;Это программа построения эскиза двух шкивов с                1
;ремнем. Заданы радиусы шкивов (R1,R2), расстояние            2
;между ними (d) и угол наклона осевой линии (alpha).          3
```

```
;Функция перевода градусов в радианы                          4
(defun dtr (alpha)                                            5
(* alpha (/pi 180.0))                                         6
)                                                             7
;Конец функции dtr                                            8
;Функция belt вычерчивает окружности и две касательные        9
(defun c:belt(/ R1 R2 d alpha C1 x1 x2 C2 P1 P2 P3 P4)       10
(setvar "cmdecho" 0)                                         11
(graphscr)                                                   12
(setq R1(getdist "\n Введите радиус малого шкива:"))         13
(setq R2(getdist "\n Введите радиус большого шкива:"))       14
(setq d(getdist "\n Введите расстояние между ними:"))        15
(setq alpha(getangle "\n Введите угол наклона:"))            16
(setq C1(getpoint "\n Введите центр малого шкива:"))         17
(setq x1(- R2 R1))                                           18
(setq x2(sqrt(- (*d d) (*(- R2 R1) ((- R2 R1)))))           19
(setq ang(atan (x1 x2)))                                     20
(setq C2(polar C1 a d))                                      21
(setq P1(polar C1 (+ ang alpha (dtr90)) R1))                 22
(setq P3(polar C1 (- (+ alpha (dtr270)) ang) R1))            23
(setq P2(polar C2 (+ ang alpha (dtr90)) R2))                 24
(setq P4(polar C2 (- (+ alpha (dtr270)) ang) R2))            25
                                                             26
;Построение окружностей и касательных                        27
(command "circle" C1 P3)                                     28
(command "circle" C2 P2)                                     29
(command "line" P1 P2 "")                                    30
(command "line" P3 P4 "")                                    31
(setvar "cmdecho" 1)                                         32
(princ)                                                      33
)                                                            34
;Конец функции belt, конец программы                         35
```

Пояснения

Строка 10

```
(defun c:belt(/ R1 R2 d alpha C1 x1 x2 C2 P1 P2 P3 P4)
```

В этой строке определяется функция `belt`, выполняющая построение двух окружностей и касательных к ним. В круглых скобках после косой черты следует перечень формальных параметров.

Строка 18

```
(setq x1(- R2 R1))
```

В этой строке значение разности радиусов окружностей присваивается переменной x1.

Строка 19

```
(setq x2(sqrt(- (*d d) (*(- R2 R1) ((- R2 R1)))))
```

В этой строке по теореме Пифагора определяется длина гипотенузы вспомогательного прямоугольного треугольника и вычисленное значение присваивается переменной x2.

Строка 20

```
(setq ang(atan (x1 x2)))
```

В этой строке с помощью функции арктангенса вычисляется угол вспомогательного прямоугольного треугольника. Значение угла в радианах присваивается переменной ang.

Строка 21

```
(setq C2(polar C1 a d))
```

В этой строке с помощью функции polar определяется центр большой окружности C2.

Строка 22

```
(setq P1(polar C1 (+ ang alpha (dtr90)) R1))
```

Здесь с помощью функции polar определяется точка P1, расположенная на окружности радиуса R1. Угол между осью *X* и направлением из точки C1 в точку P1 составляет ang + alpha + 90°.

Строка 28

```
(command "circle" C1 P3)
```

Здесь функция command обеспечивает выполнение команды построения малой окружности по заданным центру и точке, лежащей на окружности.

Строка 30

```
(command "line" P1 P2 "")
```

Здесь функция command обеспечивает проведение линии, проходящей через точки P1 и P2. Символы "" означают возврат каретки, завершающий команду LINE.

Упражнение 5

Напишите программу построения двух линий, являющихся касательными к двум окружностям, как это показано на рис. 33.11. Программа должна предусматривать

ввод диаметров окружностей (D1 и D2), а также расстояния между их центрами DIS.

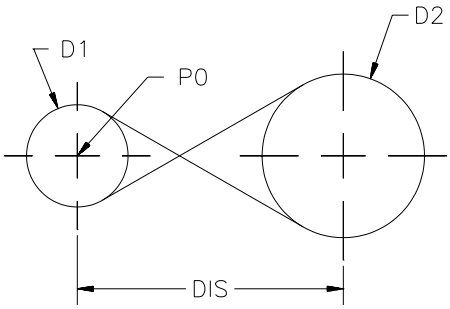

Рис. 33.11. Окружности с касательными линиями

Блок-схемы алгоритмов

Блок-схема позволяет наглядно представить алгоритм решения задачи. Применение блок-схем полезно при анализе возникающих проблем, особенно при исследовании ситуаций, связанных с наличием в программе условных выражений. Блок-схемы состоят из стандартных геометрических фигур, обозначающих различные этапы работы программы. Например, прямоугольник используется для представления процесса обработки данных при выполнении программы. Элементы блок-схемы соединяются линиями в соответствии с последовательностью операций в программе. Основные элементы блок-схем алгоритмов приведены на рис. 33.12.

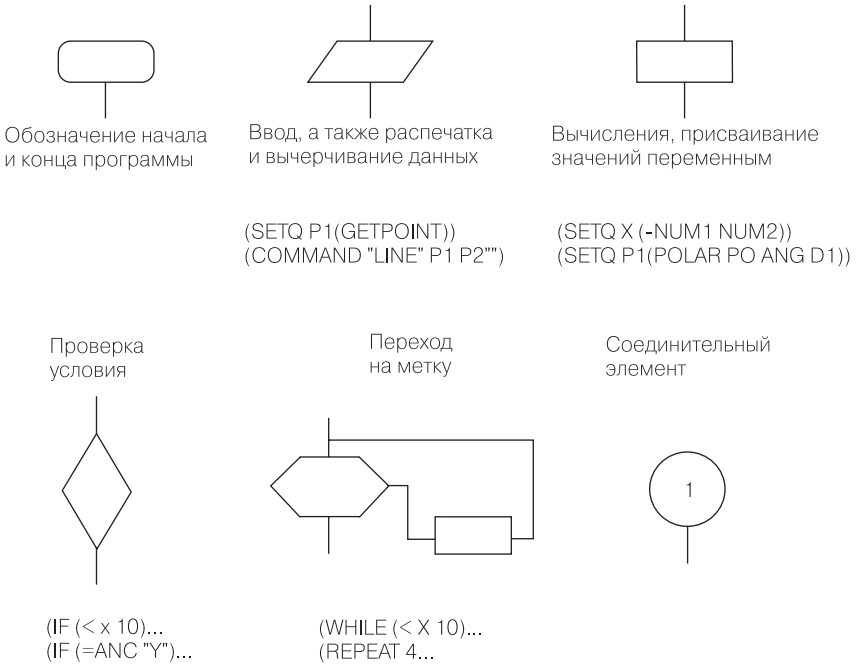

Рис. 33.12. Основные элементы блок-схем

Функции принятия решений

Рассмотренные ранее в этой главе функции сравнения определяют соотношение между двумя сравниваемыми элементами. Например, функция `if (< x y)` просто проверяет соотношение между переменными `x` и `y` и сама по себе никаких дальнейших действий не производит. Чтобы полноценно использовать функции сравнения в реальной программе, в AutoLISP применяются сконструированные на их основе более сложные функции — функции принятия решения. Например, группа операторов `(if(< x y) (setq z (- y x)) (setq z (- x y)))` описывает действия, выполняемые программой в зависимости от значения, возвращаемого функцией сравнения `if (< x y)`. Если возвращается значение `T` (истина), то переменной `z` присваивается значение `y – x`. Если возвращается значение `nil` (ложь), то переменной `z` присваивается значение `x – y`. В этом разделе рассматриваются основные функции принятия решений, без которых невозможно создать действительно полезную в реальной работе программу.

Функция if

Алгоритм действия функции `if` представлен блок-схемой на рис. 33.13. По этой блок-схеме видно, что дальнейший ход выполнения программы зависит от значения, возвращаемого функцией сравнения, входящей в функцию `if`. Функция `if` имеет следующий формат:

```
(if condition then [else])
```

Здесь:

- `condition` — функция сравнения;
- `then` — выражение, которое вычисляется, если условие сравнения выполняется;
- `else` — выражение, которое вычисляется, если условие сравнения не выполняется.

Пример:

```
(setq num1 8)
(setq num2 10)
      if(>num1 num2)
         (setq x (-num1 num2)
         (setq x (-num2 num1)
)
```

В результате выполнения этой группы операторов переменной `x` будет присвоено значение 2.

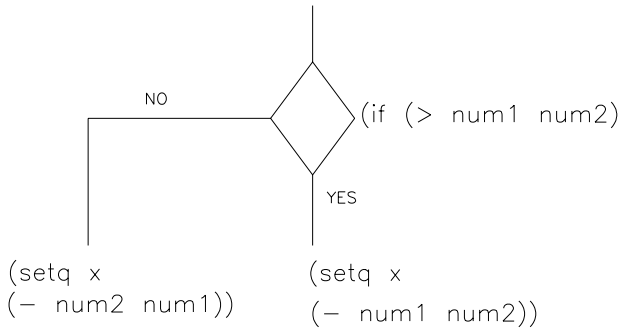

Рис. 33.13. Блок-схема алгоритма функции if

Пример 6

Напишем программу, выполняющую вычитание меньшего числа из большего. В программе должен быть предусмотрен ввод пользователем двух чисел.

Ввод данных	**Операции с данными**	**Вывод результата**
Ввод первого числа (num1)	Если num1 > num2, то x = num1 - num2	x = num1 - num2
		или

x = num2 - num1

Ввод второго числа (num2) Если num1 < num2, то x = num2 - num1

Процесс выполнения этой программы показан на рис. 33.14 с помощью блок-схемы алгоритма. Листинг программы приводится ниже, номера строк справа не являются частью файла, а служат исключительно для удобства ссылок при пояснениях.

```
;Эта программа выполняет сравнение двух чисел и                1
;вычитание меньшего числа из большего.                         2
;                                                              3
(defun c:subnum()                                              4
(setvar "cmdecho" 0)                                           5
(setq num1 (getreal "\n Введите первое число:"))               6
(setq num2 (getreal "\n Введите второе число:"))               7
(if(> num1 num2)                                               8
(setq x (-num1 num2))                                          9
(setq x (-num2 num1))                                         10
)                                                             11
(setvar "cmdecho" 1)                                          12
(princ x)                                                     13
)                                                             14
```

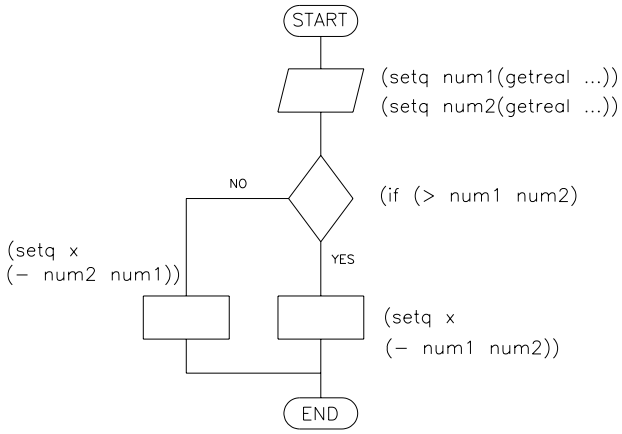

Рис. 33.14. Блок-схема алгоритма решения задачи из примера 6

Пояснения

Строка 8

```
(if(> num1 num2)
```

В этой строке функция сравнения проверяет условие `num1 > num2` и возвращает

результат T в случае выполнения условия или nil в противном случае.

Строка 9

```
(setq x (-num1 num2))
```

Вычисления, определяемые в этой строке, выполняются, если функция if(> num1 num2) возвращает значение T. В этом случае число num2 вычитается из числа num1 и результат вычитания присваивается переменной x.

Строка 10

```
(setq x (-num2 num1))
```

Вычисления, определяемые в этой строке, выполняются, если функция if(> num1 num2) возвращает значение nil. В этом случае число num1 вычитается из числа num2 и результат вычитания присваивается переменной x.

Строка 11

```
)
```

Закрывающая скобка завершает определение функции.

Пример 7

Напишем программу, выполняющую умножение и деление двух чисел. Ввод чисел и выбор арифметического действия должен сделать пользователь. В случае некорректного выбора на экран должно быть выведено соответствующее сообщение. Блок-схема алгоритма приведена на рис. 33.15, листинг программы приводится ниже.

```
;Эта программа выполняет деление или умножение двух чисел
;
(defun c:mdnum()
(setvar "cmdecho" 0)
(setq num1 (getreal "\n Введите первое число:"))
(setq num2 (getreal "\n Введите второе число:"))
(prompt "Будет выполняться умножение или деление? Введите УМН или ДЕЛ:)"
(setq ans (strcase (getstring)))
if(= ans "УМН")
      (setq x (* num1 num2)
)
if(= ans "ДЕЛ")
      (setq x (/ num1 num2)
)
(if(and (/= ans "УМН") (/=ans "ДЕЛ"))
(prompt "Ошибка выбора действия. Повторите попытку")
)
(setvar "cmdecho" 1)
```

```
(princ x))
```

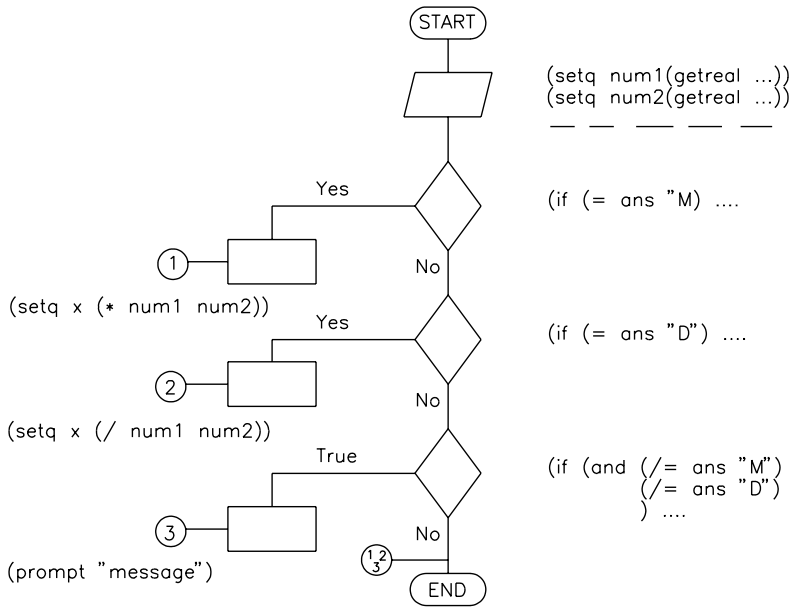

Рис. 33.15. Блок-схема алгоритма решения задачи из примера 7

Функция progn

Функция `progn` используется вместе с функцией `if` для вычисления нескольких выражений. Функция имеет следующий формат:

```
(progn выражение1 выражение2 ...)
```

Если функция `if` возвращает значение `T`, то вычисляется только одно выражение. Для вычисления нескольких выражений в связке с функцией `if` применяется функция `progn`.

Пример:

```
(if(= ans "yes")
      (progn
      (setq x (sin ang))
      (setq y (cos ang))
      (setq tanang (/ x y))
))
```

Функция while

Функция `while` оценивает результат сравнения в условном выражении. Операторы, следующие за оператором `while`, повторяются до тех пор, пока результат сравнения в условном выражении не станет равным `nil`. Функция `while` имеет следующий формат:

```
(while условное_выражение операторы)
```

Блок-схема алгоритма функции `while` приведена на рис. 33.16.

Пример:

```
(while(= ans "Да")
      (setq x (+ x 1))
      (setq ans (getstring "Введите Да или Нет:"))
)
(while(< n 3)
      (setq x (+ x 10))
      (setq n (1+ n))
)
```

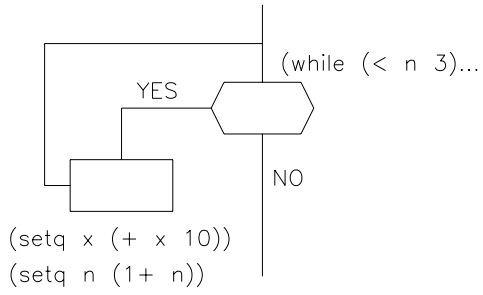

Рис. 33.16. Блок-схема алгоритма функции while

Пример 8

Напишем программу возведения числа в степень *n*. Основание и показатель степени вводятся пользователем. Показатель степени — целое число. Блок-схема алгоритма программы приведена на рис. 33.17.

Ввод данных	**Операции с данными**	**Вывод результата**
Число x (основание степени)	1. Присвоить начальные значения вспомогательным переменным t и c:	Вычисленное значение x^n
	t = 1 c = 1	
Число n	2. Вычислить произведение t*t и результат	

(показатель степени)	присвоить переменной t. Значение переменной c увеличить на 1
	3. Выполнять операции, предусмотренные пунктом 2, пока значение c не станет больше n

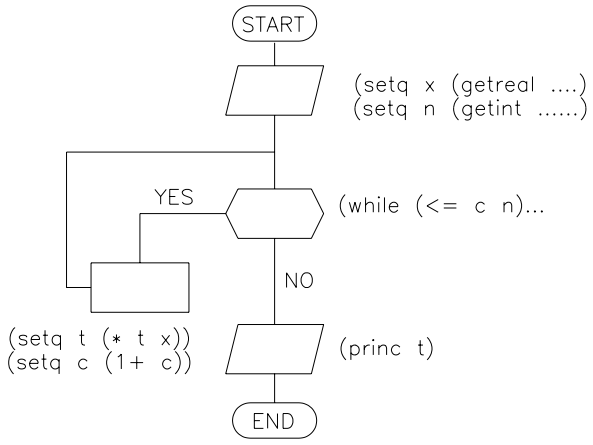

Рис. 33.17. Блок-схема алгоритма программы возведения числа в целую степень

Листинг программы для примера 8 приводится ниже.

```
;Программа возведения числа в целую степень
;Основание и показатель степени вводятся пользователем
;
(defun c:npower()
(setvar "cmdecho" 0)
(setq x(getreal "\n Введите основание степени:"))
(setq n (getint "\n Введите показатель степени:"))
(setq t 1)
(setq c 1)
(while (<= c n)
      (setq t (* t x)
      (setq c (1+ n)
)
(setvar "cmdecho" 1)
(princ t))
```

Пример 9

Напишем программу, выполняющую эскиз круглого сита (рис. 33.18). Пользователь должен указать центр сита, его диаметр, диаметр отверстий и угол смещения первого отверстия относительно оси *X*.

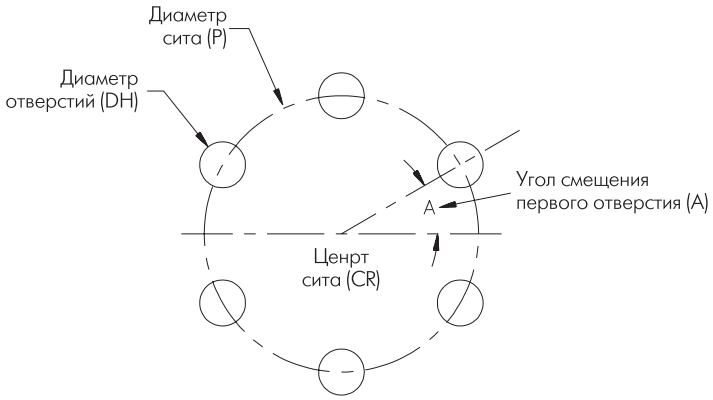

Рис. 33.18. Сито с шестью отверстиями

Ниже приводится листинг программы.

```
;Программа выполняет эскиз круглого сита
;
(defun c:bc1()
(graphscr)
(setvar "cmdecho" 0)
   (setq cr(getpoint "\n Задайте центр сита:"))
   (setq d(getdist "\n Задайте диаметр сита:"))
   (setq n(getint "\n Задайте число отверстий в сите:"))
   (setq a(getangle "\n Задайте начальный угол:"))
   (setq dh(getdist "\n Задайте диаметр отверстий в сите:"))
   (setq inc(/ (* 2 pi) n))
   (setq ang 0)
   (setq r(/ dh 2))
while (< ang (*2 pi))
   (setq p1(polar cr (+ a inc) (/ d 2)))))
   (command "circle" p1 r)
   (setq a (+ a inc))
   (setq ang (+ ang inc))
)
(setvar "cmdecho" 0)
(princ)
)
```

Функция repeat

Функция `repeat` (рис. 33.19) повторяет вычисление выражения `n` раз. Параметр `n` должен быть целым числом.

Формат:

```
(repeat n)
```

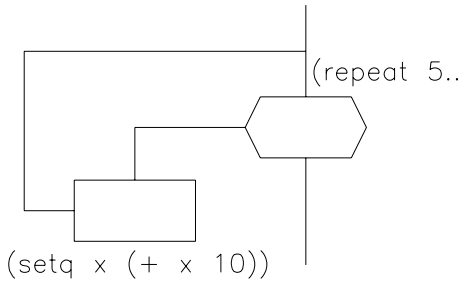

Рис. 33.19. Блок-схема действия функции repeat

Пример:

```
(repeat 5
   (setq x (+ x 10))
)
```

Пример 10

Напишем программу построения заданного числа концентрических окружностей. Пользователь должен задать центр окружностей, их количество, начальный радиус и приращение радиуса. Блок-схема алгоритма приведена на рис. 33.20. Листинг программы приводится ниже.

```
;Эта программа иллюстрирует применение функции repeat на
;примере построения семейства концентрических окружностей
;
(defun c:concir()
(graphscr)
(setvar "cmdecho" 0)
   (setq c(getpoint "\n Задайте центр окружностей:"))
   (setq n(getint "\n Задайте количество окружностей:"))
   (setq r(getdist "\n Задайте радиус первой окружности:"))
   (setq d(getdist "\n Задайте приращение радиуса:"))
(repeat n
   (command "circle" c r)
   (setq r (+ r d))
)
(setvar "cmdecho" 0)
(princ)
```

)

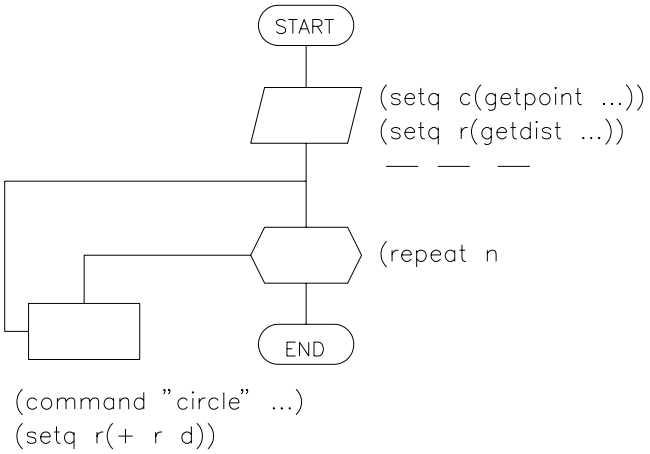

Рис. 33.20. Блок-схема алгоритма построения концентрических окружностей

Упражнения

Упражнение 6

Напишите программу, выполняющую построение трех концентрических окружностей (рис. 33.21) с центром в точке C1 и диаметрами D1, D2, D3.

В программе должен быть предусмотрен ввод пользователем координат центра окружностей и их диаметров.

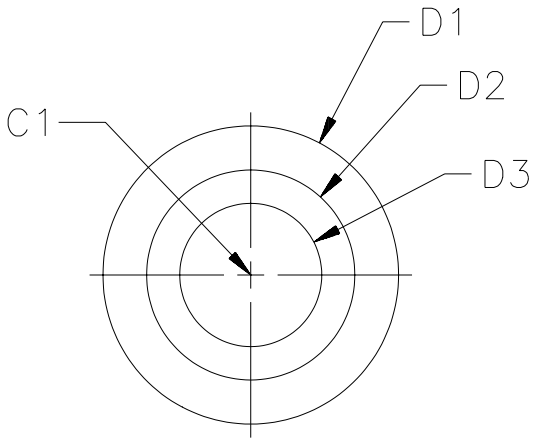

Рис. 33.21. Концентрические окружности с диаметрами D1, D2 и D3

Упражнение 7

Напишите программу, выполняющую построение линии от точки P1 до точки P2 и повернутой относительно оси *X* на угол A (рис. 33.22). Расстояние между концами отрезка равно L, диаметр окружностей — D1 (или L/4).

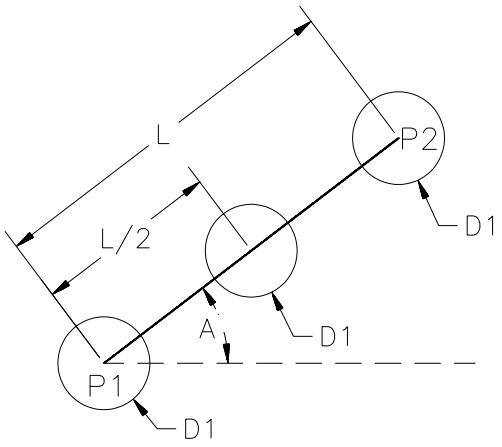

Рис. 33.22. Окружности и линия, проведенная под углом A к оси X

Упражнение 8

Напишите программу, выполняющую построение равнобедренного треугольника P1 P2 P3 (рис. 33.23). В программе должен быть предусмотрен ввод пользователем координат начальной точки P1, длины основания и угла при основании.

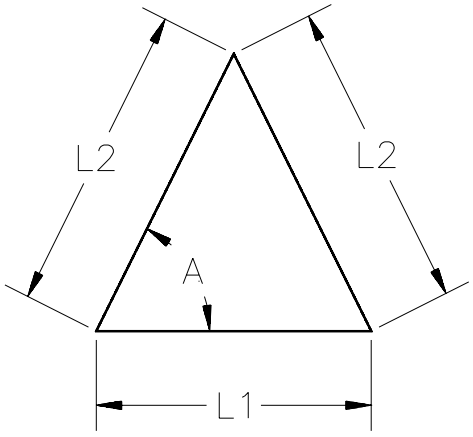

Рис. 33.23. Равнобедренный треугольник

Упражнение 9

Напишите программу, выполняющую построение паза с осевыми линиями (рис. 33.24). Программа должна запрашивать длину и высоту паза, а также имя уровня для построения осевых линий.

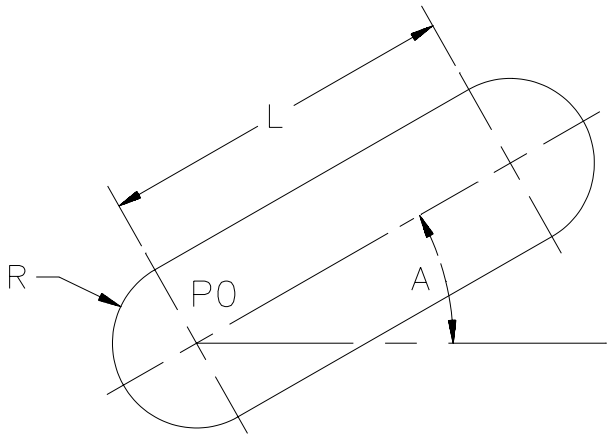

Рис. 33.24. Паз длиной L с радиусом K, повернутый к оси X на угол A

Упражнение 10

Напишите программу, выполняющую построение линии и дальнейшее создание заданного количества параллельных ей линий в количестве, равном N (рис. 33.25)

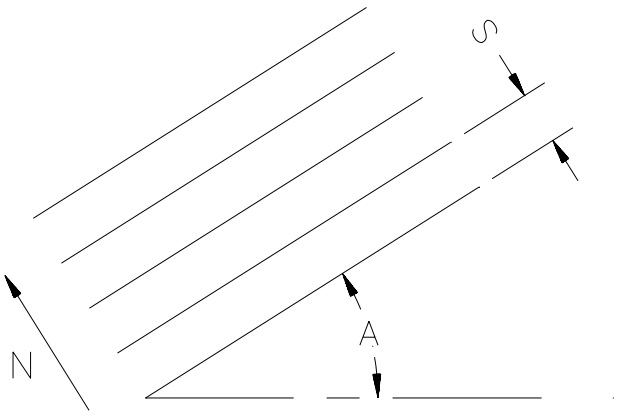

Рис. 33.25. Параллельные линии, смещенные друг относительно друга на расстояние S