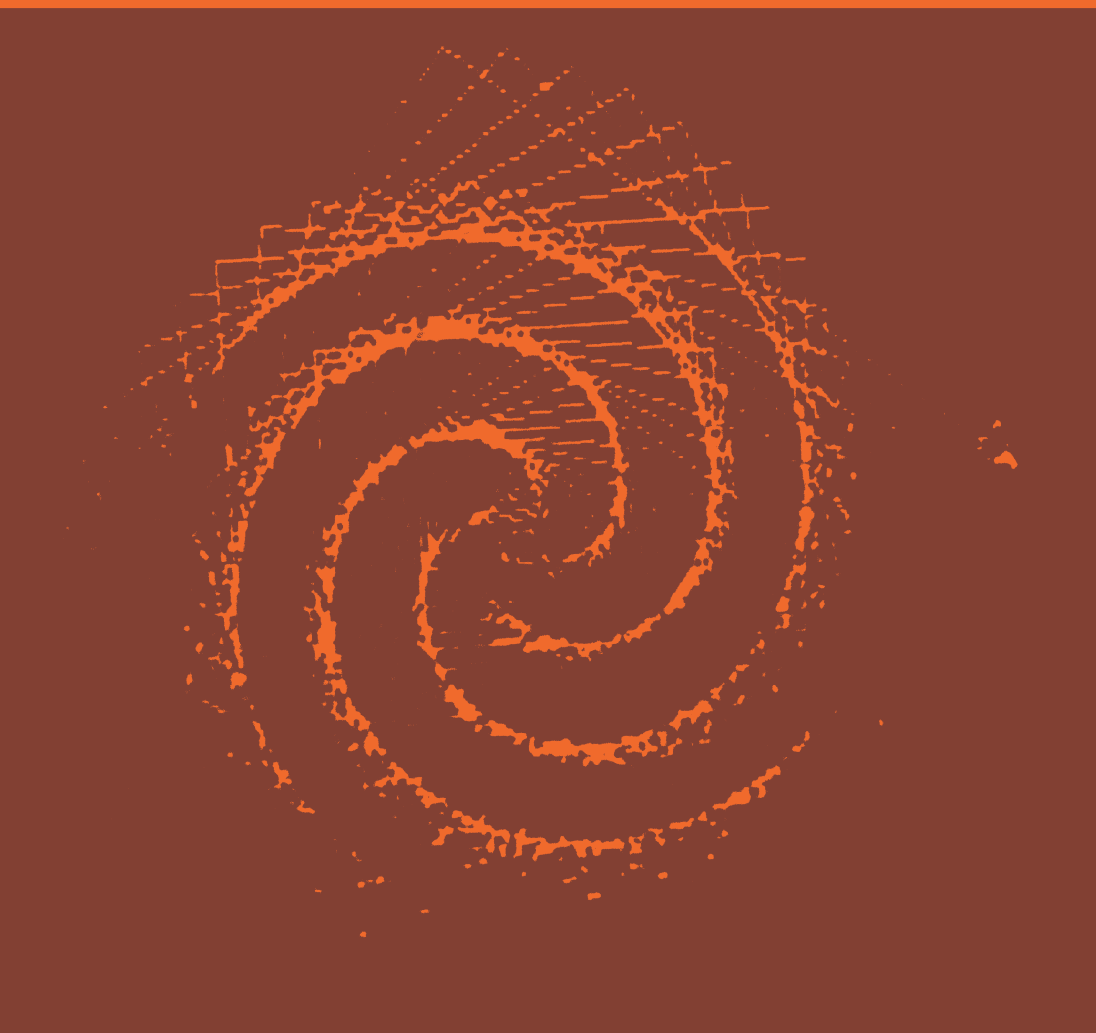


В.П. Дьяконов

# Язык программирования ЛОГО





В.П. Дьяконов

# Язык программирования Logo



Москва  
«Радио и связь»  
1991

ББК 22.18

Д 93

УДК 681.322.181.4.004.14

Рецензент: д-р техн. наук *И. М. Витенберг*

**Редакция литературы по информатике и вычислительной технике**

**Дьяконов В.П.**

**Д 93    Язык программирования Лого. — М.: Радио и связь, 1991. — 144 с.: ил.**

**ISBN 5-256-000343-7**

Рассмотрены вычислительные, логические и графические возможности языка Лого, сочетающего простоту освоения с хорошей структурированностью программ. Приведено более 250 процедур и функций для версий современных ПЭВМ (IBM PC, Apple-II, ZX-Spectrum, ЕС-1841, Искра-1030, Агат и др.). Особое внимание уделяется применению Лого в учебных и научно-технических приложениях, включая расчеты, построение графиков функций, фигур вращения (в том числе рекурсивных), гистограмм и др.

Для пользователей непрофессионалов. Может быть полезна для всех, интересующихся языками программирования.

Д 2404040000-055  
046(01)-91 133-91

**ББК 22.18**

**Производственное издание**

**ДЬЯКОНОВ ВЛАДИМИР ПАВЛОВИЧ**

**ЯЗЫК ПРОГРАММИРОВАНИЯ ЛОГО**

Заведующая редакцией **Г. И. Козырева**

Редактор **Т. М. Бердичевская**

Обложка художника **В. Ф. Громова**

Художественный редактор **Н. С. Шеин**

Технический редактор **Т. Г. Родина**

Корректор **Г. Г. Казакова**

**ИБ № 2287**

Подписано в печать с оригинал-макета 11.01.91. Формат 60×88/16. Бумага тип. гр. № 2. Гарнитура Пресс-роман. Печать офсетная. Усл. печ. л. 8,82. Усл. кр.-отт. 9,19. Уч.-изд. л. 6,83. Тираж 50 000 экз. Изд. № 23136. Зак. № 6267. Цена 2р. 20к.

Издательство "Радио и связь". 101000 Москва, Почтамт, а/я 693

Ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО "Первая Образцовая типография" Государственного комитета СССР по печати. 113054, Москва, Валуевская, 28.

**ISBN 5-256-000343-7**

© Дьяконов В. П., 1991

## *Предисловие*

Наше общество вступило в эпоху массовой компьютеризации. Появление относительно дешевых персональных ЭВМ (ПЭВМ) сделало их доступными не только для малых организаций, но и для отдельных лиц. Число ПЭВМ быстро растет. Они пришли в сферу просвещения и образования [1]. Как никогда раньше возрос интерес к языкам программирования ПЭВМ [2 – 4]. И это понятно – без них ПЭВМ не более чем "черный ящик". Особый интерес представляют языки, доступные миллионам пользователей-непрофессионалов, способные привить им навыки культурного программирования буквально с детского возраста. К сожалению, к ним нельзя отнести популярный Бейсик [4 – 6].

Среди десятка языков, получивших международное признание и подлинную массовость, видная роль принадлежит языку Лого. Он был разработан еще в 1972 г. С. Пейпертом (США) и предназначался для обучения детей основам программирования [2]. В Лого сразу была заложена концепция простоты и ясности языка, а также бескомпромиссная ориентация на структурное программирование. От Лиспа Лого унаследовал мощный и универсальный аппарат обобщенной обработки данных – чисел, слов, листов и списков (словом, объектов). Подлинной находкой оказалась Лого-графика, основанная на относительно перемещении в полярной системе координат особого графического объекта – черепашки, снабженного световым пером. Детям достаточно 10 – 20 мин. знакомства с основами Лого, чтобы тут же рисовать на экране дисплея забавные геометрические узоры, фигуры людей и зверушек, домики и т.д. На других языках это требует изощренных приемов программирования.

За рубежом Лого признан как лучший язык для изучения основ программирования и машинной графики. Трудно переоценить его возможности в создании обучающих программ по физике и математике, геометрии и черчению, информатике и лингвистике. За рубежом Лого посвящены десятки книг [13 – 31], издаются специальные журналы. Лого входит в типовое программное обеспечение наиболее массовых отечественных и зарубежных ПЭВМ: ЕС-1840, ЕС-1841, Искра-1030, Агат, IBM PC, MSX, ZX-Spectrum, Apple-II и др.

У нас в стране Лого известен мало. Достаточно отметить, что по нему не было опубликовано отечественных книг, а публикации в периодической печати буквально единичны [2, 3]. Отчасти это объясня-

ется тем, что многие до сих пор считают Лого детским языком программирования. Однако то, что Лого легко осваивают даже дети, его огромное достоинство. Вспомним, что Бейсик и Паскаль также были разработаны с ориентацией на начальное обучение программированию.

Современные версии Лого, описанные в этой книге (в частности, для ПЭВМ класса IBM PC, Apple-II и ZX-Spectrum), – мощные и универсальные средства общения с ПЭВМ. Так, версия Лого IBM PC (отечественные аналоги ЕС-1840, ЕС-1841 и Искра-1030) способна работать с числами "сверхвысокой" разрядности, с большим порядком (от –9999 до +9999), имеет обширный набор математических функций, прекрасные средства для работы с данными и общения с операционной системой.

Предлагаемая вниманию читателей книга и популярна, и серьезна. В ней по возможности доходчиво и в то же время полно описаны основные версии Лого. Особое внимание уделено уникальным графическим средствам Лого и их реальному применению. И школьник, полюбивший информатику, и школьный учитель найдут здесь множество интересных и поучительных примеров. Научные работники и инженеры наверняка также оценят уникальные математические и графические возможности Лого, программисты получают не общее, а вполне конкретное представление еще об одном современном языке программирования ПЭВМ. Если эти надежды автора оправдаются, значит замысел книги удался.

## 1. Краткое знакомство с языком программирования Лого

### 1.1. Что такое Лого?

Как известно, любая ЭВМ представляет собой *программно-управляемое вычислительное устройство*. Даже такие "невыхислительные" операции, как обработка текстов или построение графиков, в ЭВМ сводятся к вычислениям. Так, буквы могут быть представлены числовыми кодами, графики разложены на точки, координаты и цвет которых также задаются соответствующими числовыми кодами. Однако программирование ЭВМ в кодах крайне утомительно и требует глубокого знания особенностей каждой конкретной ЭВМ.

Поэтому сейчас для программирования ЭВМ используются в основном языки программирования *высокого уровня* с определенной проблемной ориентацией [1 – 6]. На этих языках управление ПЭВМ сводится к заданию команд, записываемых обычными или сокращенными словами, как правило, на английском языке. Последнее связано с международной стандартизацией языков (вспомним, что и в математике, и в физике, и в химии применяются формулы с латинскими буквами).

Каждая команда на языке высокого уровня запускает в действие несколько десятков, а то и сотен команд в машинных кодах. Запись программ на языках высокого уровня выглядит гораздо более компактно и понятно, чем в машинных кодах. Смысл команд, как правило, понятен из их обычного смыслового значения. Например, команда PRINT X на Бейсике, или PRINT :X на Лого буквально означает "Печать X" (print – в переводе печать). Программировать на языках высокого уровня гораздо проще, чем в машинных кодах.

Языки программирования делятся на два обширных класса – *компилирующие* и *интерпретирующие*. Компилирующие языки воспринимают сразу весь текст программы, проверяют его на наличие синтаксических ошибок и затем транслируют в машинные коды. Оттранслированная программа выполняется довольно быстро и занимает в памяти ЭВМ относительно немного места. Первые массовые языки – Фортран, Алгол, Паскаль – были компилирующего типа.

Другой класс языков – *интерпретирующие* – переводят в машинные коды каждую команду по мере ее поступления на вход интерпретатора и тут же исполняют ее. Хранить *машинные коды*, иначе говоря

*объективные*, здесь уже не требуется, но нужно постоянно держать в памяти сам интерпретатор. Очевидно, что такой способ замедляет вычисления, поскольку время трансляции входит в полное время вычислений. Но он дает одно важное достоинство – любую команду или группу команд можно тут же исполнить, а затем, убедившись в правильности работы, включить в общую программу. Такой ярко выраженный *диалоговый* характер общения пользователя с ЭВМ оказался особенно удобным для ПЭВМ, пользователи которых обычно не профессионалы в области программирования.

Лого является специальной программой-интерпретатором. При разработке Лого особое внимание уделялось простоте и наглядности команд, легкости освоения языка и его безукоризненной строгости с позиций структурного программирования. Лого легко изучать, следуя хорошо известному методическому приему – от простого к сложному.

Менее двух десятков команд, описанных уже в первой главе, достаточно, чтобы на Лого не только решать повседневные вычислительные задачи, но и строить довольно сложные графики и рисунки. Современные версии Лого содержат до 300 слов-команд, выполняющих самые разнообразные функции по обработке данных, управлению ПЭВМ и ее периферийным оборудованием.

От языка Лисп Лого унаследовал весьма мощный и гибкий аппарат по обработке разнообразных *объектов* – чисел, букв, слов, списков слов, инструкций и т.д. Объекты в Лого рассматриваются с высокой степенью обобщения. Так, в Лого нет задания специальных типов целочисленных, с плавающей точкой, символьных, коротких, длинных и прочих *переменных*, присущих таким языкам, как Бейсик или Паскаль. Значениями переменных являются просто объекты. Лого имеет множество команд по обработке и редактированию объектов и списков из них, например по выделению первого и последнего или заданного по порядку объекта из списка, удалению их, вводу новых объектов и т.д. Все это резко облегчает программирование задач, связанных с обработкой текстов, созданием каталогов, словарей и баз данных.

Лого – один из немногих языков, имеющих естественную *словарную организацию* команд. В любой момент пользователь Лого может вывести на экран дисплея полный список слов своего словаря. В них входят как встроенные в интерпретатор Лого *ключевые слова*, так и слова – имена задаваемых пользователем *внешних процедур*. Это роднит Лого с языком Форт. Как и Форт, Лого относится к классу *развиваемых* пользователем (адаптируемых) языков. Это означает, что пользователь может пополнять язык новыми определениями (процедурами), подстраивая его под решение своих специфических задач.

Правда, в отличие от Форта, у Лого не предусмотрена полная запись дополненной версии на магнитные носители. Комплекты дополнительных процедур хранятся в отдельных файлах на магнитном диске или другом носителе информации. При использовании дисковых накопителей это не встречает каких-либо неудобств.

Лого – язык *универсальный*. Это означает, что на нем можно решать самые разнообразные задачи как вычислительного, так и невычислительного характера. Уже отмечались уникальные возможности Лого-графики, осуществляющей построение графиков, подобное рисованию их вручную кистью или пером. Если на Бейсике построение графиков связано обычно с вычислением координат всех особых точек, то на Лого графики вычерчиваются заданием относительных перемещений черепашки в любом заданном направлении. Трудно переоценить возможности Лого-графики в отображении сложных геометрических фигур, подготовке обучающих и игровых программ. Общение с ПЭВМ на Лого развивает математическую смекалку и геометрические представления. И не только у детей!

Применение Лого способствует привитию пользователям подлинной культуры программирования, базирующегося не только на четких правилах *структурного программирования*, но и на высокой степени обобщения различных понятий, таких как объекты, типы данных, процедуры и т.д. В этом плане Лого – подлинный антипод Бейсику, бессистемность, неструктурированность и нестрогость которого не раз осуждались [4] вплоть до утверждений, что после начального обучения Бейсику обучающиеся навсегда лишаются возможности стать в будущем хорошими программистами и превращаются в "кретинов" в области программирования.

Не разделяя таких крайних точек зрения (хотя бы потому, что для автора Бейсик был первым, но отнюдь не последним изучаемым языком), отметим, однако, что перечисленные свойства Лого делают его вполне обоснованным кандидатом на лучший язык для обучения основам структурного программирования и элементам машинной графики. Это уже признано за рубежом. Элементы Лого-графики сейчас включаются практически во все современные языки программирования – Бейсик, Паскаль, Си, микро-Пролог и др.

Разумеется, указанные достоинства Лого не приобретаются даром. Основная потеря, которая ощущается при работе с Лого, это потеря в скорости вычислений. В целом Лого при решении вычислительных задач более медленный язык, чем даже Бейсик. Частично этот недостаток компенсируется введением в Лого новых, подчас весьма привлекательных возможностей. Например, в версии Лого ПЭВМ IBM [28, 30] имеется возможность работы как с очень малыми, так и очень большй-

ми числами (порядок от -9999 до +9999), установки любого числа знаков после запятой, проведения арифметических операций как в инфиксной, так и в префиксной формах и т.д.

Программы на Лого выглядят гораздо компактнее, чем на Бейсике (и тем более на Паскале). Они не засорены множеством номеров строк, меток и обращений к ним. Лого настраивает пользователя на составление программ в виде небольших функционально законченных *поименованных процедур*, которые можно накапливать и неоднократно использовать. Структура программ на Лого носит древовидный характер. В любой момент можно вызвать на контроль листинг любой процедуры или ряда процедур, а не просматривать постоянно "метры" строк, как на Бейсике.

## 1.2. Загрузка Лого в персональную ЭВМ

Лого загружается в *оперативное запоминающее устройство (ОЗУ)* ПЭВМ с внешнего накопителя на магнитном диске или (в простых ПЭВМ) с обычного кассетного магнитофона. Использование Лого не требует глубоких знаний о структуре ПЭВМ. Полезно, однако, знать, что ОЗУ содержит множество ячеек, каждая из которых имеет определенный номер-адрес и может хранить 1 байт информации. 1 байт это 8 двоичных разрядов, так что каждая ячейка памяти может вместить одно из  $2^8 = 256$  целых чисел (0 ... 255). Этими числами и задаются машинные коды либо кодируются знаки алфавита Лого. ОЗУ является вместилищем всех данных и программ, с которыми ПЭВМ в данный момент работает.

Ввод Лого в ПЭВМ зависит от ее типа. В ПЭВМ класса IBM PC после включения автоматически загружается *дискровая операционная система (MS DOS)*. С ее помощью надо найти нужный *файл* с интерпретатором Лого (он имеет название LOGO.COM), пометить его и ввести нажатием клавиши ENTER. Последние модели ПЭВМ класса IBM PC содержат специальную программу, выдающую подробные меню с операциями ДОС, так что достаточно пользоваться подсказками этой программы. Загрузка Лого с диска происходит в считанные секунды. Если предполагается снятие копии графиков с экрана дисплея принтером, перед загрузкой Лого надо загрузить файл с именем GRAPHICS.

В более простых ПЭВМ, например домашних ZX-Spectrum, интерпретатор Лого хранится записанным на магнитной ленте обычной магнитофонной кассеты. В этом случае его загрузка производится подачей команды

LOAD "LOGO"

или

## LOAD ""

В первой форме команда LOAD (загрузка) используется, если интерпретатор Лого находится где-то в середине кассеты и его нужно отыскать по имени. Этот процесс может быть довольно долгим – нужно перемотать ленту и дожидаться поиска программы с Лого. Рекомендуется записать интерпретатор Лого в начало кассеты, тогда легче найти его и использовать команду LOAD во второй форме.

Лого имеет различные версии и занимает часть ОЗУ. Так, мощная версия Лого IBM PC занимает в ОЗУ почти 60 Кбайт (1 Кбайт = 1024 байт). Тем не менее эта потеря не очень существенна, так как ОЗУ этих ПЭВМ обычно имеет емкость 256 – 640 Кбайт (64 Кбайт в минимальном варианте, который сейчас почти не встречается). В более простых ПЭВМ ОЗУ имеет меньшую емкость, например 48 Кбайт у ZX – Spectrum. Поэтому для них используются более простые интерпретаторы Лого, занимающие 20 ... 25 Кбайт в ОЗУ. Отметим, что, несмотря на это, загрузка Лого с кассетного магнитофона занимает несколько минут.

После загрузки Лого на экране появляется сообщение об этом и вопросительный знак:

?

Этот знак является *приглашением* пользователя к работе. Если появляется сообщение об ошибке, например, вида

Tape loading error  
(магнитофон вводит с ошибкой),

следует повторить ввод интерпретатора. Если ошибка не исчезает, значит запись интерпретатора сделана некачественно или испорчена. В этом случае нужно сменить кассету с интерпретатором и повторить его ввод. Загрузка Лого с диска происходит, как правило, без ошибок.

Интерпретатор Лого – сложная программа. Поэтому, приобретя ее, следует сразу сделать дубли, а оригинал хранить с необходимыми мерами предосторожности. В частности, нужно оберегать диск или кассету от воздействия магнитных и тепловых полей. Для записи копий интерпретатора следует использовать качественные диски и кассеты. Это резко уменьшит время ввода и сэкономит Ваши нервы. При частой работе с Лого-интерпретатором его полезно перенести на твердый диск, встроенный в ПЭВМ класса IBM PC.

### 1.3. Первые шаги

Вопросительный знак (?), появившийся на экране дисплея после загрузки интерпретатора Лого, указывает на то, что ПЭВМ работает в режиме *непосредственного исполнения* команд, т.е. в режиме *прямых вычислений* (не по программе).

Попробуем заставить Лого сложить два числа, например 2 и 3:

? 2 + 3 <ENTER>

Здесь обозначение <ENTER> указывает на нажатие клавиши ввода (у некоторых ПЭВМ она может иметь иное обозначение, например ENTER, RETURN, BK и т.д.).

Нажав клавишу ввода, получим следующее любопытное сообщение:

```
? 2+3  
I DON'T KNOW WHAT TO DO WITH 5  
?
```

В переводе на русский язык это сообщение гласит:

Я не знаю, что делать с 5?

В этом примере мы убеждаемся, что Лого довольно интеллигентный язык. В самом деле, в данном случае мы допустили две грубейшие ошибки. Первая: мы не знали, что ввод чисел и команд в Лого должен разделяться пробелами, и не ввели их. Однако Лого все же вычислил правильный результат  $5 = 2 + 3$ . Далее мы действительно забыли указать, что делать с результатом: то ли вывести его на печать (индикацию), то ли задать как значение переменной. Об этом и сообщил Лого, на всякий случай выдав значение результата в своем ответе.

Еще одна характерная особенность Лого, заметная уже в этом примере: дружественный характер сообщений Лого об ошибках. В Бейсике, например, мы получаем сообщение вроде Error 14 in 340 (ошибка номер 14 в строке 340) и вынуждены копаться в технической документации, отыскивая, что это за ошибка с таким-то номером. Лого дает прямое указание о том, к чему относится ошибка.

Чтобы вывести на печать результат операции в Лого используется команда PRINT. Ее можно вводить и в сокращенном виде PR. В нашем примере правильные действия соответствуют одной из двух первых нижеследующих записей:

```
?PRINT 2 + 3  
5  
?PR 2 + 3  
5
```

3

Теперь попробуем вычислить  $\sin 30^\circ$  и  $\cos 30^\circ$ :

4

Обратим внимание, что результат вычисления  $\cos 30^\circ$  содержит 11 знаков после разделительной точки (Бейсик лишь 7).

SETPRECISION n

[illegible]

Еще одно полезное свойство Лого IBM PC – возможность представления чисел необычно больших по модулю (порядок от  $-9999$  до  $+9999$ ). В этом нетрудно убедиться, выполнив следующие действия:

```
?PR 1E9999
1.000000000E+9999
?PR 1E9999/3
3.333333333E+9998
?PR 2E-157*5
1.000000000E-0156
?PR 256E-3*2/10
0.0512
?PR 2*(1/3E-7)
6666666.666
?PR 3.25E-6+2.64E-7*(4.67+2.25E3)
5.984828800E-0004
?
```

Попробуйте проверить выполнение следующих операций (не забывая нажать в конце клавишу ввода):

```
?PR 3 * 4 (результат  $3 \cdot 4 = 12$ )
?PR 3 - -5 (результат  $3 - (-5) = 8$ )
?PR 3/4 (результат  $3/4 = 0,75$ )
?PR (2 + 3) * 4 (результат  $(2 + 3) \cdot 4 = 20$ )
?PR SQRT 4 (результат  $\sqrt{4} = 2$ )
?PR INT 2.75 (результат  $\text{int } 2,75 = 2$ )
```

Вы убедитесь в том, что в режиме непосредственных вычислений ПЭВМ с интерпретатором Лого превращается в довольно мощный калькулятор, выполняющий арифметические операции, вычисляющий ряд элементарных функций и способный вычислять значения различных арифметических выражений, например  $(2 + 3) \cdot 4$ . В этом отношении Лого подобен Бейсику.

При начальном вводе Лого ПЭВМ устанавливается в один из четырех возможных режимов представления информации – *текстовый*. При этом на экране дисплея мы видим текст (он включает в себя и цифры). Экран делится на некоторое число строк с определенным числом знаков (литер) в каждой строке. Число строк и литер в строке зависит от конкретного типа ПЭВМ. Обычно число строк равно  $16 - 25$ , а число литер в строке – от  $32$  до  $80$ . Лого IBM PC содержит в строке  $40$  или  $80$  литер (при первой загрузке  $40$ ). Клавиша  $F_3$  вводит команду установки в строке  $80$  символов.

## 1.4. Черепашка Лого

Еще один режим Лого – *графический*. Он неотъемлемо связан с особым графическим объектом – *черепашкой*. Черепашка (TURTLE) у Лого – это похожее на ползущую черепаху изображение треугольника. В некоторых версиях Лого черепашка по виду больше напоминает обрезанный кончик кисточки (рис. 1.1,б). Светлый кончик кисточки (или головка черепашки) указывает на направление ее движения.

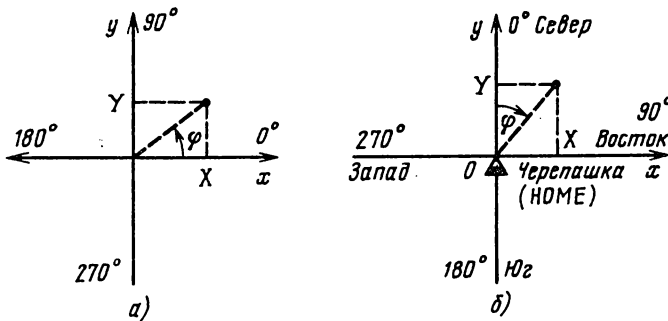


Рис. 1.1. Обычная координатная система (а) и координатная система Лого-графики с черепашкой в исходном состоянии (б)

Чтобы вывести черепашку в центр экрана (рис. 1.1,б), нужно подать команду

**? SHOWTURTLE**

или в русском переводе “показать черепашку”.

Черепашка в Лого способна на многое. Она может поворачиваться на заданный угол, перемещаться вперед и назад, “прыгать” в заданное место экрана, оставлять или не оставлять следа при своем движении, рисовать весьма замысловатые фигуры, раскрашивать их различными цветами и т.д.

Например, команда FORWARD (вперед), или сокращенно FD, заставит черепашку двигаться вперед на расстояние, указанное далее через пробел. Расстояние задается в *растровых* единицах экрана (пикселях, т.е. элементарных точках). Команда RIGHT φ или RT φ заставляет черепашку повернуться вправо на угол φ относительно своего исходного положения. Так, если задать исполнение команд

```
? FD 60 RT 90
? FD 60 RT 90
? FD 60 RT 90
```

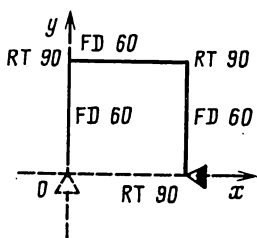


Рис. 1.2. Перемещения черепашки

то черепашка вычертит три стороны квадрата (рис. 1.2) трижды, двигаясь вперед на 60 пикселей и поворачиваясь на 90°. Если еще раз повторить строку этих команд, получим изображение квадрата.

В графическом режиме большая часть экрана освобождается под рисунки. Еще одна полезная команда CLEARSCREEN (стирание экрана), или сокращенно CS, служит для очистки экрана. Исполнив эту команду, мы сотрем результаты наших предыдущих экспериментов и получим чистый экран с черепашкой в центре и знаком вопроса в левом нижнем углу.

Читатель, вероятно, обратил внимание на то, что при построении квадрата операции FD 60 RT 90 повторились 4 раза. Этого можно было бы добиться с помощью очень полезной команды REPEAT (повторить), после которого указывается аргумент (число повторений), а в квадратных скобках – список повторяемых команд. Так, выполнив команду

? REPEAT 5 [FD 60 RT 360/5]

мы получим на экране изображение пятиугольника (рис. 1.3).

В этой команде надо отметить два новых для нас обстоятельства. Первое – повторяемый фрагмент команды заключается в квадратные скобки. Второе – аргумент (в нашем случае операции RT) может быть арифметическим выражением, а не только числом. Позже мы убедимся, что он может быть и переменной, имеющей численное значение.

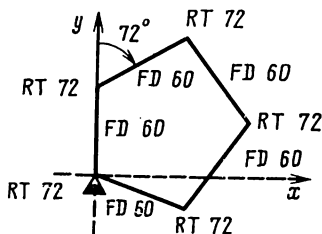


Рис. 1.3. Черепашка строит пятиугольник

## 1.5. Процедуры и их редактирование

Третий режим работы ПЭВМ с интерпретатором Лого – это *задание процедур*: ввод отдельных структурно законченных фрагментов программ, исполняемых впоследствии.

Любая программа на языке Лого записывается в виде некоторой процедуры, т.е. группы команд, исполняемых ПЭВМ. Процедуры, присущие Лого, называются примитивными, или *примитивами*. Так, процедуры вывода на печать PR, вывода черепашки SHOWTURTLE, ее движения вперед FD, поворота вправо RT, стирания экрана CS и многие другие есть примитивы.

Однако пользователь может задавать и *внешние процедуры*, выполняющие какие-либо новые функции. Процедура на Лого задается с помощью такой конструкции:

? TO Имя	Входные параметры	Заголовок процедуры
> .....		Тело процедуры
> END		Конец процедуры

Рассмотрим простейший пример – задать процедуру с именем DEMO, которая просто печатает (точнее, выводит на экран дисплея) слово QWERTY. Задание и исполнение такой процедуры выглядит следующим образом:

```
?TO DEMO
>PR "QWERTY
>END
DEMO DEFINED
?DEMO
QWERTY
?
```

Эта процедура не имеет *входных параметров*. После ввода заголовка TO DEMO в начале строки появляется знак >. Он означает, что Лого переходит в новый (третий) режим работы – *задания процедур*. Теперь последующие слова (кроме END) прямо не исполняются (так ввод PR "QWERTY уже не ведет к немедленной печати слова QWERTY), а лишь запоминаются как инструкции программы. Слово END – второе после TO – исполняемое слово в этом режиме. Оно просто завершает задание процедуры и переключает Лого-систему из режима задания процедуры в режим *немедленного исполнения*. Признаком этого является появление сообщения DEMO DEFINED (DEMO определено) и знака ? в начале строки. Слово DEMO пополнило словарь Лого.

Теперь для исполнения процедуры DEMO достаточно указать имя DEMO после знака ? и нажать клавишу ENTER (естественно, что она

нажимается в конце каждой строки). В результате печатается слово QWERTY и в новой строке появляется знак ? – приглашение к дальнейшей работе.

Как отмечалось, процедуры могут иметь входные параметры. Поэкспериментировав с Лого IBM PC, мы обнаружим, что в нем нет функции вычисления  $\operatorname{tg} x$ , но зато есть  $\sin x$  и  $\cos x$ . Попробуем задать процедуру TAN, вычисляющую  $\operatorname{tg} x = (\sin x)/(\cos x)$ . Очевидно, что эта процедура должна иметь входной параметр  $x$  и должна обеспечивать выход (возврат) результата. В Лого имеется команда выхода OUTPUT, или сокращенно OP. Она как бы превращает процедуру в функцию, преобразующую аргумент в результат. В таких случаях говорят, что функция возвращает результат.

Итак, процедуру TAN можно задать следующим образом:

```
?TO TAN :X
> OUTPUT SIN :X / COS :X
>END
TAN DEFINED
?PR TAN 45
0.896018936
?
```

Под определением процедуры показан и результат ее исполнения. Теперь слово TAN стало полноправным членом словаря Лого и может исполняться в любой другой программе точно так же, как, например, слова SIN или COS.

Процедуры очень удобно вводить и редактировать с помощью специального редактора. Так, задав команду

```
EDIT TAN
или
ED TAN
```

мы обнаружим, что экран очищается, а текст процедуры появляется в его верхней части без знаков ? и >. Это указывает на переход Лого в режим редактирования процедур (четвертый).

Редактор Лого IBM PC относится к полным экранным редакторам. Это означает, что в процессе редактирования курсор может перемещаться по всему экрану и устанавливаться на любой знак текста процедуры. После этого можно изменить этот знак, а с помощью специальных команд стереть его, вставив новый знак, перевести строку и т.д. Для выхода из редактора достаточно нажать клавишу Esc. Появится сообщение о задании процедуры или об ошибке, если текст процедуры с ошибкой. В последнем случае нужно вернуться в режим редакти-

рования (достаточно ввести команду ED) и внести исправления в текст процедуры. Этот процесс может повторяться несколько раз, пока не будут устранены все ошибки в тексте процедуры.

Запись процедуры на диск производится командой

SAVE "Имя процедуры

Если процедура была записана на диск ранее и ее нет в ОЗУ, то ввести процедуру можно, задав команду

LOAD "Имя процедуры

Команда

PO "Имя процедуры

выводит листинг процедуры на экран дисплея.

Теперь рассмотрим использование записанных на диск процедур. Так, если мы располагаем процедурой построения прямоугольника — щипка (BOX), то для ее загрузки подадим команду

LOAD "BOX

Далее, используя команду PO, выводим на экран текст (листинг) процедуры:

```
?PO "BOX
TO BOX :H :W
PD
FD :W
RT 90 FD :H
RT 90 FD :W
RT 90 FD :H
RT 90 PU
END
```

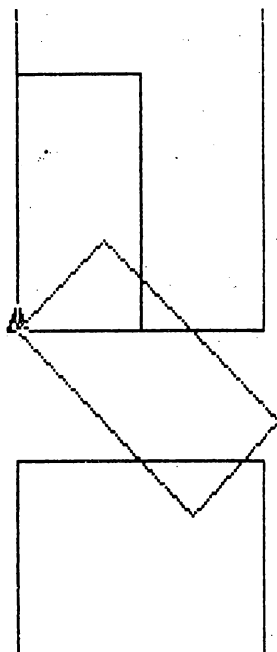
Легко проанализировать эту процедуру. В ней два входных параметра — высота прямоугольника W и его ширина H. Черепашка перемещается каждый раз на расстояние W, H, W, H, поворачиваясь вправо на 90°.

Теперь дадим команду CS (CLEARSCREEN). Экран очищается, и Лого переходит в графический режим с резервированием внизу строк под текст. Дальнейшие этапы работы показаны на рис. 1.4. Разберемся с полученной картинкой.

Вначале задан поворот черепашки вправо на 45°, затем построение невысокого длинного прямоугольника. На рисунке видно, что рисуется повернутый на 45° прямоугольник. Команда HOME (домой) возвращает черепашку в исходное состояние. После этого строятся два

Рис. 1.4. Исполнение процедуры BOX

```
?RT 45
?BOX 100 50
?HOME
?BOX 50 100
?BOX 100 200
?
```



прямоугольника с соотношением  $H/W = 1/2$ . Один (малый) построен нормально. А вот сторона второго не вместились в окно экрана, а потому оказалась достроенной снизу.

Из рис. 1.4 видно, что линии, которыми вычерчен наклонный прямоугольник, не совсем прямые: они состоят из отдельных маленьких прямоугольничков – пикселей. В вертикальных и горизонтальных линиях пиксели сливаются и линии имеют вид жирных отрезков прямых. Отмеченная погрешность графика – неотъемлемое свойство растровых изображений. Рис. 1.4 и текст на нем соответствуют графическому режиму среднего разрешения ( $320 \times 200$  точек растра). В режиме высокого разрешения ( $640 \times 200$  точек) дискретность наклонных линий заметна меньше.

Слева внизу экрана виден текст команд ввода, выводимый большими буквами (40 знаков в строке) в служебной части экрана. При необходимости этот текст можно убрать (см. гл. 3).

Известны предложения по разработке руссифицированных версий Лого. Например, в [3] описана версия, в которой черепашка управляется русскими словами: ВПЕРЕД, НАЗАД, ВПРАВО, ВЛЕВО и т.д. К сожалению, это лишает такие версии столь важного качества, как программная совместимость на международном уровне. Поэтому вряд

ни разумна замена общепринятых слов Лого, как и в математике или физике, русскими словами. Освоение небольшого набора английских слов не вызывает трудностей даже у детей дошкольного возраста. А понятие о новом для них языке и первые навыки его полезного применения несомненно привьют уважение к иностранным языкам, без знания которых немислима подлинная культура нашего будущего поколения.

Однако что является весьма желательным – это введение в Лого национальных алфавитов в дополнение к латинскому. Это открывает возможность к практическому созданию на Лого словарей, каталогов, редакторов текста, баз данных и других подобных программных средств, на долю которых ныне приходится львиная доля реальных применений ПЭВМ.

Вот так, к примеру, выглядят три процедуры словаря для перевода с английского языка на русский:

TO ANGLE	TO BEGIN	TO MAN
PR"УГОЛ	PR"НАЧАЛО	PR"ЧЕЛОВЕК
END	END	END

Такой словарь можно пополнять в произвольном порядке и оперативного редактировать. Стоит просто указать английское слово – имя процедуры, как тут же будет напечатан его русский перевод. Столь же просто создание на Лого записных книжек, телефонных справочников, кулинарных меню и т.д.

Лого открывает возможности наглядного изучения и гораздо более сложных приемов обработки текстовой информации: форматирование текста, вставка и уничтожение слов, добавление и исправление знаков, сортировки слов и т.д. Трудно переоценить эти возможности в создании обучающих программ для изучения иностранных языков. И не только! Правила дорожного движения, криптограммы спортивных состязаний, формы таблиц и деловых документов – это тоже объекты обучающих программ на Лого.

Словарная организация Лого, мощные средства задания и редактирования процедур и объектов, простая в использовании Лого-графика – все это позволяет резко снизить трудоемкость создания обучающих и игровых программ, остро необходимых нашему народному образованию.

## 2. Вычислительные и логические возможности Лого

### 2.1. Алфавит Лого, его объекты, слова и списки

В состав алфавита языка Лого входят 26 латинских прописных (больших) букв:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Могут использоваться и строчные (малые) буквы:

a b c d e f g h i j k l m n o p q r s t u v w x y z

С помощью этих букв записываются все инструкции языка Лого. В национальные версии Лого дополнительно включаются буквы соответствующего алфавита. Их можно использовать в текстах и комментариях.

С помощью букв образуются слова Лого – предписания ПЭВМ для выполнения тех или иных действий. Ряд слов заведомо включен в состав Лого (например, PRINT или PR) и не может быть исключен из словаря. Такие слова называются *ключевыми словами*, или *примитивами*.

В наборе примитивов и правилах их записи у различных версий Лого могут быть некоторые отличия. Например, Лого ПЭВМ ZX-Spectrum допускает запись примитивов как большими, так и малыми буквами. В Лого IBM PC примитивы должны записываться только большими буквами. Различия в составе примитивов для версий Лого ПЭВМ ZX-Spectrum, IBM – PC и Apple-II указаны в приложении.

Для ввода и индикации чисел в алфавит Лого включены десять арабских цифр (0, 1, 2, ..., 9) и разделительная точка. С их помощью можно задавать как целые числа (например, 123 или 456), так и дробные (например, 123.456). Знак минус используется как для указания знака числа (например, -123), так и для задания арифметической операции вычитания.

Числа могут задаваться и в *экспоненциальной форме*  $x = M \cdot 10^P$ , где M – мантисса и P – порядок. Порядок указывается после мантиссы и отделяется от нее буквой E:

? 2 \* 1.01 E - 9

2.02.E - 9

Помимо букв, цифр и указанных выше знаков в алфавит Лого входит ряд спецзнаков:

? – вопросительный знак,      = – знак равенства,

! – восклицательный знак,      > – знак неравенства (больше),

[ ] – квадратные скобки,	< – знак неравенства (меньше),
( ) – круглые скобки,	+ – знак арифметической операции
	сложения,
, – запятая,	* – знак умножения.
/ – знак деления,	

Знак кавычки ” обычно указывает, что последующая за ним цепочка знаков, заканчивающаяся пробелом, есть слово. Двоеточие : указывает на то, что такая цепочка знаков есть имя переменной. Списки слов заключаются в квадратные скобки. Круглые скобки используются обычно для задания двух параметров в командах. Таким образом, в Лого четко отмечается функциональная принадлежность знаков или слов.

Объектами Лого называются слова и списки, используемые в качестве входной и выходной информации. Слово – есть просто последовательность букв и цифр. Про любой знак (литеру) слова говорят, что он является элементом слова. А слова являются элементами списков.

Некоторые слова в Лого являются именами процедур (см. ниже). Чтобы отличить от них простое слово, перед ним ставится знак ”. Итак:

”SIN – простое слово (символьная строка),  
 SIN – служебное слово (название процедуры).

Пустое слово не содержит ни одного символа и отмечается знаком ”.

Числа в Лого также рассматриваются как слова. Но, учитывая их особый статус, их можно записывать без кавычек. Такое же исключение есть в отношении слов TRUE (истина) и FALSE (ложь) – результатов логических операций.

Список в Лого – последовательность слов, заключенных в квадратные скобки и отделенных друг от друга пробелами:

[ONE TWO THREE]	– список слов ONE, TWO и THREE,
[1 2 3 4 5]	– список цифр 1, 2, 3, 4, 5,
[A1 B2 C3 D4]	– список слов A1, B2, C3, D4, содержащих буквы и цифры,
[ ]	– пустой список, который следует понимать как содержащий одно пустое слово.

Ограничители – знаки, которыми ограничиваются слова в языке Лого и отделяются друг от друга. Обычно слова разделяются пробелами, но

могут использоваться и другие ограничители в виде следующих спецзнаков:

= + - \* %

Между словом и любой из этих литер пробел может не ставиться, хотя он допустим и целесообразен для единообразия синтаксиса.

## 2.2. Примитивы, процедуры, входные параметры и переменные

*Процедура* – объявляемая своим именем определенная последовательность операций, выполняемых ПЭВМ. Процедура является центральным понятием языка Лого, вытекающим из концепций структурного программирования. Образно говоря, процедура это блок в "крупнопанельном доме" программирования на языке Лого. Из таких блоков состоит *программа*.

Процедуры могут быть встроенными в язык – примитивами или внешними процедурами, создаваемыми пользователем. Программы на языке Лого в сущности являются набором процедур. Одна из них является главной процедурой и ее имя есть имя программы. Эта процедура обращается к другим процедурам, те к третьим и т.д. Таким образом, структура программы древовидная и нисходящая, что отвечает идее структурного программирования сверху вниз. С помощью входных и выходных параметров (см. ниже) процедуры обмениваются друг с другом данными.

Примитивы являются неотъемлемой частью языка Лого, и их изменение или уничтожение пользователем невозможно. Совокупность примитивов образуют *словарь* Лого, содержащий до 200 – 300 слов. Обычно имеется примитив, выводящий на экран дисплея или на печать список всех примитивов данной версии Лого. Таким примитивом в версии Лого IBM PC является примитив .CONTENTS, результат действия которого:

```
?PR .CONTENTS
LPT1 SETTC TEXTCOLOR TC ROUND DIFFERENC
E FILL SETSHAPE SHAPE TONE .DOS ARCTAN
EXP FORM LN POWER SETPRECISION PRECISIO
N ALLOPEN SETH ST HT ERASE WINDOW READP
OS WRITEPOS READER WRITER CAPS SETCAPS
.SCREEN .SETSCREEN SUM READCHARS RCS TE
XTSCREEN RW PALETTE SETPAL RERANDOM RAN
```

DOM .SETCOM COUNT PADDLE BUTTONP READCH  
 AR SETCURSOR CURSOR COS SIN QUOTIENT PR  
 ODUCT EDITFILE SETWIDTH POFILE ERASEFIL  
 E DIR / < > \* - -) ( + = PI EFORM SETY  
 SETX UNBURY SETTEXT THROW CLEARTEXT CT  
 OUTPUT LPUT FPUT BUTFIRST BF PLIST READ  
 LIST RL TEST BUTLAST BL NOT PRINT PR DO  
 T RIGHT RT WAIT EDIT .DEPOSIT IFTRUE ER  
 RACT SAVEPIC LOADPIC LEFT LT REPEAT MS  
 POTS .CONTENTS ERPS PPS POPS ERNS PONS  
 EDNS SETPOS SETWRITEPOS SETREADPOS POS  
 TOWARDS NODES ERROR PENCOLOR YCOR XCOR  
 REMAINDER ER CHAR EMPTYP STARTUP LISTP  
 FILEP NUMBERP MEMBERP KEYP STOP PENUP P  
 U SHOWNP STAMP PPROP REMPROP GPROP EQUA  
 LP READEOFF REDEFF PRIMITIVEP NAMEP WOR  
 DP DEFINEDP WRAP SNAP TO PO GO PENDOWN  
 RUN ERN MIXEDSCREEN TS CLEARSCREEN CS F  
 ULLSCREEN FS DRIBBLE SETPEN OPEN PEN CL  
 EAN .SYSTEM ITEM .CALL CLOSEALL ERALL P  
 OALL PKGALL TOPLEVEL LABEL LOCAL BACK B  
 K ASCII SHOW CATCH .SETSCRUNCH THING SE  
 THEADING VALPKG PROCPKG IF IFFALSE SETD  
 ISK COPYDEF .BSAVE TRUE IFT CO PENREVER  
 SE PX PAUSE REPARSE CLOSE FALSE SQRT SE  
 TWRITE NODRIBBLE TYPE INT PENERASE DEFIN  
 E .EXAMINE RECYCLE SHOWTURTLE HIDETURT  
 LE NAME HOME MAKE PACKAGE WRITEOFF FILE  
 LEN FENCE SENTENCE READWORD FORWARD FD  
 END AND ED PD SETBG BACKGROUND BG SETRE  
 AD .BLOAD RC DISK SETPC PC BURY TEXT FI  
 RST IFF LIST LAST OR OP LOAD WIDTH .SCR  
 UNCH HEADING SAVE SE PAL PE WORD

Пользователь, однако, может пополнить словарь, задав *внешние процедуры*, которые он может модифицировать и уничтожать. Имена их не должны совпадать с ключевыми словами, так как внешние процедуры используются точно так же, как и примитивы. Процесс задания внешних процедур был подробно описан в § 1.5 с примерами.

Подпроцедурой называется процедура, входящая в состав другой процедуры. Так, в первом примере примитив PRINT (PR) есть подпроцедура. По отношению к процедуре PRINT процедура DEMO является подпроцедурой.

Благодаря применению аппарата процедур Лого является расширяемым пользователем языком. Пользователь имеет возможность наряду с применением довольно развитого аппарата встроенных в язык Лого примитивов создавать свои процедуры и даже целые библиотеки из них. В отличие от языка Форт (также имеющего словарную организацию), процедуры Лого должны храниться отдельно от интерпретатора. При включении ПЭВМ и повторной загрузке Лого сохраняются лишь примитивы, а введенные пользователем процедуры должны загружаться отдельно.

Лого с введенными пользователем новыми процедурами по существу оказывается расширенной версией языка. Его возможности можно почти неограниченно наращивать. Далее будет показано, что процедура может вызывать сама себя (так называемая *рекурсивная процедура*). В состав процедур могут входить операции с числами, символами и словами, многочисленные графические операции, вызов звуковых эффектов и т.д. Все это делает язык Лого мощным инструментом эффективного использования ПЭВМ. Эти качества Лого полезны при решении расчетных задач и особенно при создании сложных обучающих и игровых программ.

Как и другие языки, Лого имеет аппарат задания *переменных*, т.е. объектов, несущих определенные числовые, символьные и другие значения. Однако в Лого нет деления переменных по типу (целочисленные, с плавающей точкой, символьные и др.). Все переменные задаются одинаково, и этим заданием фактически и определяется тип (и смысл) переменных.

Для задания переменных и присвоения им значений служит примитив MAKE. После него через пробел указывается отмеченное знаком " имя переменной, которое может содержать один или несколько знаков алфавита. Далее указывается числовое или символьное значение, присваиваемое переменной (оно также записывается после знака ").

В следующем примере показано задание трех переменных X, Ymax и CHAR с помощью примитива MAKE, а затем распечатка их с помощью примитива PR:

```
?MAKE "X 20
?MAKE "Ymax 199
?MAKE "CHAR "0123456789ABCDEF
?PR :X
20
?PR :Ymax
199
```

```
?PR :CHAR
0123456789ABCDEF
?
```

Для вывода значений переменных они указываются по имени с предшествующим знаком двоеточия `::`. Он является признаком того, что последующее имя есть имя переменной, а не процедуры (без знака) или символьной строки (помечается знаком `"`).

*Входные параметры процедур* – данные, которые указываются в заголовке процедуры и используются в процессе ее выполнения. Входные параметры могут быть объектами (словами или списками), числами или значениями переменных. Через входные параметры осуществляется передача данных от одной процедуры к другой.

Рассмотрим задание и исполнение процедуры вычисления гиперболического синуса  $\text{sh}(x) = (e^x - 1/e^x)/2$ :

```
?TO HSN :X
>MAKE "Y EXP :X
>OP (:Y - 1/:Y )/2
>END
HSN DEFINED
?PR HSN 1
1.175201194
?PR :Y
2.718281828
?PR :X
X HAS NO VALUE
?
```

Здесь после имени процедуры HSN указана переменная `:X`, свидетельствующая о задании входного параметра. При последующем исполнении процедуры после ее имени ставится числовое значение этого параметра (число 1). Это значение присваивается переменной `X`, после чего вычисляется значение  $e^x$ , присваивается вспомогательной переменной `Y`, а затем вычисляется  $\text{sh}(x) = (Y - 1/Y)/2$ . Примитив `OP` (или `OUTPUT`) означает возврат результата, после чего можно использовать примитив печати `PR` (или `PRINT`).

В конце примера сделана попытка вывести на печать значения `Y` и `X`. В отношении `Y` она оказалась успешной – выведено значение  $Y = e^1$ . А в отношении переменной `X` получено на первый взгляд странное сообщение (в переводе "X не имеет значения"). В чем тут дело?

Оказывается, Лого имеет два статуса переменных: они могут быть *локальными* или *глобальными*. Переменные, которые задаются как

входные параметры процедур, действуют только внутри их и являются локальными. Поскольку переменная X в нашем случае ранее не объявлялась, то по выходе из процедуры она и считается необъявленной, т.е. не имеющей значения.

В отношении статуса переменных, объявленных с помощью примитива MAKE, действуют несколько более сложные, но абсолютно определенные правила:

1) Переменная, объявленная в режиме непосредственных вычислений, всегда будет глобальной. Это означает, что она доступна в любом месте программы, в любой процедуре;

2) Переменная, созданная внутри процедуры и не являющаяся ее входным параметром, является глобальной;

3) Переменная, объявленная входным параметром процедуры, будет локальной для этой процедуры, т.е. действовать только в ее пределах.

Переменная Y соответствует правилу 2, становится глобальной, а потому и доступна после выхода из процедуры (ее значение  $e^1$  выведено на печать при выходе из процедуры).

Ряд версий Лого (IBM PC, Apple – II и др.) имеет примитив локализации переменных

```
LOCAL "Имя  
(LOCAL "Имя1 "Имя2 ... "Имя3)
```

с указанными именами. Следующий пример иллюстрирует это:

```
?TO HSN :X  
>LOCAL "Y  
>MAKE "Y EXP :X  
>OP ( :Y - 1/:Y )/2  
>END  
HSN DEFINED  
?PR HSN 1  
1.175201194  
?PR :Y  
Y HAS NO VALUE  
?PR :X  
X HAS NO VALUE  
?
```

Теперь уже при выходе из процедуры недоступной стала не только переменная X, но и переменная Y. Ибо она стала локальной.

Возможность локализации переменных облегчает программирование. Например, можно глобальной переменной X обозначать координаты

наты точки, а в отдельных процедурах под локальной переменной X. подразумевать аргумент какой-либо функции. Однако все же не рекомендуется давать одни и те же имена различным переменным, поскольку это затрудняет разбор программ и снижает их наглядность.

Аналогичным образом переменным можно присваивать значения списка числовых или символьных значений:

```
?MAKE "A [ 4 5 6 ]
?PR :A
4 5 6
?MAKE "D [ START FINISH ]
?PR :D
START FINISH
?
```

Из этих примеров видно, что список задается его компонентами, заключенными в квадратные скобки. При распечатке значений переменных примитивом PRINT скобки опускаются.

Имеется еще одна возможность задания поименованных переменных с помощью примитива

```
NAME "Имя объект
```

Например, NAME "MAN" BOBY порождает переменную MAN со значением (символьным) BOBY.

### 2.3. Арифметические и логические операции и функции

К простейшим арифметическим операциям в Лого относят сложение, вычитание, умножение и деление. Приоритет операций обратен приведенному, т.е. в первую очередь выполняется деление, в последнюю — сложение.

Для выполнения арифметических операций можно использовать примитив PRINT (PR). Арифметические операции Лого выполняет в двух формах. Инфиксная форма соответствует обычной алгебраической форме, когда знаки операций (сложение +, вычитание -, умножение \* и деление /) располагаются между числами — операндами:

```
?PR 2.1 + 3.75
5.85
?PR 3.75 - 2.1
1.65
?PR 2.1 * 3.75
7.875
```

?PR 2 / 3  
0.666666667  
?

При выполнении инфиксных операций не следует путать знак минус у числа (мантиссы или порядка) со знаком вычитания. Для этого знак минус отрицательного числа пишется слитно с ним и отделяется от знака операции пробелом:

? 2 - -7                      Операция  $2 - (-7) = 9$   
9

Естественный для Лого порядок выполнения арифметических операций можно изменять при помощи круглых скобок:

? 10 - 8 \* 2                      Вычисляется  $10 - (8 \cdot 2) = -6$   
-6  
? (10 - 8) \* 2                      Вычисляется  $(10 - 8) \cdot 2 = 4$   
4

*Префиксная* форма выполнения операций соответствует указанию вначале самой операции, а затем списка из двух или нескольких операндов. Эта форма выполнения операций удобна, когда, например, нужно сложить или умножить подряд несколько чисел или переменных. Однако на практике она применяется редко. Поскольку в последующем эта форма выполнения операций использоваться не будет, ограничимся несколькими примерами ее применения, приведенными ниже.

Для выполнения операций в префиксной форме используются следующие примитивы:

SUM $X_1 X_2$	Сумма $(x_1 + x_2)$ двух чисел
(SUM $X_1 X_2 \dots X_N$ )	Сумма $(x_1 + x_2 + \dots + x_N)$ $N$ чисел
DIFFERENCE $X_1 X_2$	Разность $(x_1 - x_2)$ двух чисел
PRODUCT $X_1 X_2$	Произведение $(x_1 \cdot x_2)$ двух чисел
(PRODUCT $X_1 X_2 \dots X_N$ )	Произведение $(x_1 \cdot x_2 \cdot \dots \cdot x_N)$ $N$ чисел
QUOTIENT $X_1 X_2$	Частное от деления $(x_1/x_2)$ двух чисел
POWER $X_1 X_2$	Возведение в степень $x_1^{x_2}$

Примеры префиксных операций:

?PR SUM 1.23 4.56  
5.79  
?PR DIFFERENCE 1.23 4.56  
-3.33  
?PR PRODUCT 3 4

```

12
?PR QUOTIENT 13 3
4.333333333
?PR POWER 2 3
8
?PR POWER 2 1/2
1.414213562
?
```

В некоторых версиях Лого префиксное деление задается примитивом DIV (ПЭВМ ZX-Spectrum).

Все эти операции могут выполняться и с переменными, имеющими числовые значения. Для этого они должны быть входными параметрами или определены с помощью примитива MAKE:

```

?MAKE "A 2
?MAKE "B 3
?MAKE "C 4
?PR SUM :A :B * :C
14
?
```

В последнем примере инфиксная форма вычисления  $b \cdot c$  комбинируется с префиксной формой (сложение  $a$  и  $b \cdot c$ ).

Лого имеет довольно большой набор *арифметических функций* — числовых, алгебраических, тригонометрических и обратных тригонометрических. Ниже они перечислены с примерами.

1. Функция INT  $x$  отделяет целую часть аргумента  $x$  путем отбрасывания дробной части.

2. Функция ROUND  $x$  округляет число  $x$  до ближайшего целого.

Примеры на действие функции INT и ROUND:

```

?PR INT 25.123
25
?PR INT 25.876
25
?PR INT -25.123
-25
?PR INT -25.876
-25
?PR ROUND 25.123
25
?PR ROUND 25.876
26
```

```
?PR ROUND -25.123
-25
?PR ROUND -25.876
-26
?
```

3. Функция REMAINDER  $X_1 X_2$  вычисляет остаток от деления  $x_1$  на  $x_2$ :

```
?PR REMAINDER 25.25 5
0.25
?PR REMAINDER 25 5
0
```

4. Функция SQRT  $X$  вычисляет квадратный корень из числа  $x$ :

```
?PR SQRT 16
4
?PR SQRT 17
4.123105626
?
```

5. Функция SIN  $\varphi$  вычисляет синус угла  $\varphi$ , заданного в градусах (рис. 2.1).

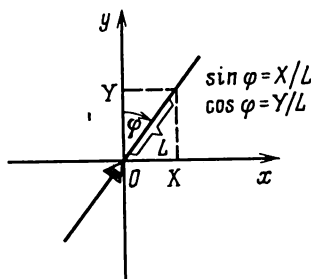


Рис. 2.1. К определению синуса и косинуса угла  $\varphi$ .

6. Функция COS  $\varphi$  вычисляет косинус угла  $\varphi$ , заданного в градусах (рис. 2.1).

7. Функция ARCTAN  $X$  вычисляет угол в градусах арктангенса с аргументом  $x$ .

8. Функция LN  $X$  вычисляет натуральный логарифм  $x - \ln x$ .

9. Функция EXP  $X$  вычисляет значение  $e^x$ .

Примеры на действие функций SIN, COS, ARCTAN, LN и EXP:

```
?PR SIN 45
0.7071067812
```

```

?PR COS 30
0.8660254038
?PR ARCTAN 1
45
?PR LN 2
0.6931471806
?PR EXP 2
7.389056099
?
```

10. Функция RANDOM N генерирует целое случайное число в интервале  $[0; n]$ :

```

?PR RANDOM 10
5
?PR RANDOM 10
0
?PR RANDOM 10
3
?PR RANDOM 10
2
?PR RANDOM 10
5
?PR RANDOM 10
6
?
```

11. Примитив RERANDOM обеспечивает повторение последовательностей случайных чисел с одних и тех же значений.

12. Функция PI выдает число  $\pi$ .

Указанные функции характерны для версии Лого IBM PC. Другие версии могут иметь другой набор функций. Например, Лого для ПЭВМ /X Spectrum не имеет функций LN и EXP, но в нем имеются дополнительные функции: TANGENT  $\phi$  или TAN  $\phi$  (вычисление тангенса угла  $\phi$ ), COTANGENT  $\phi$  или COT  $\phi$  (вычисление котангенса угла  $\phi$ ), ARCSIN  $x$  (вычисление  $\arcsin x$ ) и ARCCOS  $x$  (вычисление  $\arccos x$ ). Функций POW, LN и EXP нет в версии Лого ПЭВМ Apple-II.

Набор арифметических функций в версии Лого IBM PC функционально достаточно полный. Это значит, что с их помощью можно вычислять многие отсутствующие функции, используя соответствующие формулы. При записи арифметических выражений можно использовать круглые скобки, как, например, в процедуре вычисления тангенса TAN:

```

?TO TAN :X
> OP (SIN :X)/(COS :X)
>END
TAN DEFINED
?PR TAN 45
1
?PR TAN 30
0.5773502692
?
```

Скобки применяются также в том случае, когда нужно изменить приоритет операций. Возможно применение двойных, тройных и т.д. круглых скобок. Но в арифметических выражениях нельзя использовать квадратные скобки, поскольку они зарезервированы под иные цели — объединение объектов в списки или команд в группы.

Лого имеет также стандартные *логические* операции и функции. К ним прежде всего относятся *операции сравнения*. Как видно из приведенного примера

```

?PR 5>3
TRUE
?PR 5<3
FALSE
?PR 2+3=1+4
TRUE
?PR 2*3>1+4
TRUE
?
```

результатом сравнения в Лого являются так называемые *предикаты* — утверждения TRUE (истинно) и FALSE (ложно). Соблюдение условия порождает предикат TRUE, а нарушение — FALSE. Имена предикатов Лого — ключевые слова.

Над предикатами могут выполняться логические операции: AND (логическое И), OR (логическое ИЛИ) и NOT (логическое НЕ). Операция AND задается в виде:

```

AND P1 P2
(AND P1 P2 ... )
```

где P1, P2, ... , PN — предикаты. Операция AND порождает предикат TRUE, если все входные предикаты TRUE, иначе она порождает предикат FALSE. Если предикатов — входных параметров — больше двух логическое выражение с AND заключается в круглые скобки.

Операция OR задается в виде

```
OR P1 P2  
( OR P1 P2 ... PN )
```

Результат TRUE, если хотя бы один входной предикат есть TRUE, иначе OR порождает предикат FALSE.

Операция

```
NOT P1
```

порождает TRUE, если P1 FALSE, и FALSE, если P1 TRUE, т.е. инвертирует логическое значение входного предиката.

Действие примитивов AND, OR и NOT иллюстрируется следующим примером:

```
?PR AND "TRUE "TRUE  
TRUE  
?PR AND "FALSE "FALSE  
FALSE  
?PR AND "FALSE "TRUE  
FALSE  
?PR OR "TRUE "FALSE  
TRUE  
?PR OR "FALSE "TRUE  
TRUE  
?PR OR "TRUE "TRUE  
TRUE  
?PR OR "FALSE "FALSE  
FALSE  
?PR NOT "TRUE  
FALSE  
?PR NOT "FALSE  
TRUE  
?
```

Входными параметрами примитивов AND, OR и NOT могут быть только предикаты, но не числа или символы. Если это правило нарушено, порождается сообщение об ошибке:

```
?PR AND 2 2  
2 IS NOT TRUE OR FALSE  
?PR OR 1 1  
1 IS NOT TRUE OR FALSE
```

```
?PR NOT 1
1 IS NOT TRUE OR FALSE
?
```

Однако имеется примитив эквивалентности EQUALP, входные параметры которого X1, X2, ..., XN могут быть числами или символами

```
EQUAL X1 X2
(EQUAL X1 X2 ... XN )
```

Этот примитив порождает TRUE, если все входные параметры эквивалентны, т.е. идентичны. Иначе он порождает FALSE:

```
?PR EQUALP 12 12
TRUE
?PR EQUALP 12 34
FALSE
?PR EQUALP "AB "AB
TRUE
?PR EQUALP "A "AB
FALSE
?
```

## 2.4. Обработка и модификация слов и списков

В основе программ для работы с текстами и сложными данными лежит обработка слов и списков. Для этого в Лого имеется ряд специальных примитивов.

Примитив

```
ASCII "Слово или литера
ASCII :Имя переменной
```

вырабатывает код в стандарте ASCII одиночной литеры, стоящей после знака ", либо первой литеры слова. Во втором случае вырабатывается код первой литеры слова – значения переменной. Примеры действия этого примитива:

```
?MAKE "C "ALLEN
?PR ASCII "A
65
?PR ASCII "ALLEN
65
?PR ASCII :C
65
```

**?PR CHAR 65**

**A  
?**

Примитив

**CHAR N**

где **N** лежит в пределах от 32 до 165, возвращает литеру, код которой в стандарте ASCII равен **n**.

Примеры:

**PR CHAR 66**                      Выдает на печать литеру **B** с кодом 66

**B**

**PR CHAR 88**                      Выдает на печать литеру **X** с кодом 88

**X**

У некоторых версий Лого допустимо указывать **n** в виде нецелых чисел. При этом дробная часть **n**, если она меньше 0,5, отбрасывается. Но, если она больше 0,5, возможно округление в большую сторону.

**PR CHAR 65.45**                  Выдает на печать литеру **A** с кодом 65

**A**

**PR CHAR 65.95**                  Выдает на печать литеру **B** с кодом 66

**B**

Примитив "за исключением первого"

**BUTFIRST**                      Объект

**BF**                              Объект

Порождает указанный объект, удалив из него первый элемент. Вторая форма имени примитива есть сокращенное наименование его. Пустой список **[ ]** нельзя использовать в качестве объекта для этого примитива. Примеры применения примитива **BF** даны ниже:

**?PR BF "ABCDEF**

**BCDEF**

**?PR BF [ALFA BETA GAMMA]**

**BETA GAMMA**

**?PR BF [ ]**

**BUTFIRST DOESN'T LIKE [ ] AS INPUT**

**?**

Ниже приведена рекурсивная процедура **VPRINT**, печатающая в столбик слова, полученные удалением первого символа из исходного слова:

```

?TO VPRINT :OBJECT
>PR :OBJECT
>VPRINT BF :OBJECT
>END
VPRINT DEFINED
?VPRINT "ABCDEF
ABCDEF
BCDEF
CDEF
DEF
EF
F

```

BUTFIRST DOESN'T LIKE AS INPUT:  
JUST BEFORE LEAVING VPRINT  
?

В данном случае исполнение процедуры VPRINT начинается с печати значения переменной :OBJECT в виде слова ABCDEF, введенного при начальном обращении к процедуре. Затем процедура обращается сама к себе, каждый раз удаляя первую литеру значения переменной :OBJECT. Так продолжается до тех пор, пока это слово не выродится в пустое, после чего исполнение процедуры прекращается из-за возникновения двух ошибок: пустой список неприменим как параметр к примитиву BF и к процедуре VPRINT.

Примитив BUTLAST	"за исключением последнего"
· BUTLAST	Объект
BL	Объект

порождает указанный объект с исключением последнего элемента:

```

?MAKE "C "ABCDEF
?PR BUTLAST "ABCDEF
ABCDE
?PR BL :C
ABCDE
?PR BL [ALFA BETA GAMMA]
ALFA BETA
?

```

Как видно из этого примера, если объектом является список слов, то его элементами будут слова. Если объект слово, то элементами его будут отдельные символы.

Примитив

COUNT Объект

вырабатывает целое число, равное числу элементов (литер в слове или слов в списке) объекта:

```
?MAKE "C "ABCDEF
?PR COUNT "ABCDEF
6
?PR COUNT :C
6
?PR COUNT [1 2 3]
3
?PR COUNT [ALFA BETA GAMMA]
3
?
```

Операция

WORD Слово1 Слово2 ... СловоN

выдает слово, составленное из указанных в списке входных параметров слов:

```
?PR WORD 123 456
123456
?PR WORD "ONE "TWO
ONETWO
?MAKE "A "ONE
?MAKE "B "TWO
?PR WORD :A :B
ONETWO
?
```

Примитив

WORDP Объект

вырабатывает слово TRUE (ИСТИНА), если объект является словом, и FALSE (ЛОЖЬ), если он не является словом:

```
?PR WORDP "123
TRUE
?PR WORDP 123
TRUE
?PR WORDP [ ]
```

```
FALSE
?PR WORDP [ 123 ]
FALSE
?PR WORDP [ ABC ]
FALSE
?
```

Примитив

MEMBERP Объект Список

вырабатывает слово TRUE, если объект является элементом указанного списка, и слово FALSE, если он не является элементом списка:

```
?PR MEMBERP "X "ZXC
TRUE
?PR MEMBERP "A "ZXC
FALSE
?PR MEMBERP "X [Z X C]
TRUE
?PR MEMBERP "THREE [ONE TWO]
FALSE
?PR MEMBERP "ONE [ONE TWO]
TRUE
?
```

Примитив

NUMBERP Объект

вырабатывает слово TRUE, если объект есть число, и FALSE, объект не число:

```
?PR NUMBERP 123
TRUE
?PR NUMBERP "ABC
FALSE
?PR NUMBERP [ 1 2 3 ]
FALSE
?PR NUMBERP [ ONE TWO ]
FALSE
?
```

Примитив

LISTP Объект

вырабатывает значение TRUE, если объект есть список, и FALSE – в противном случае:

```
?PR LISTP [1 2 3]
TRUE
?PR LISTP 123
FALSE
?PR LISTP [A B C]
TRUE
?PR LISTP "ABC"
FALSE
?
```

Примитив

EMPTYP Объект

вырабатывает слово TRUE, если объект пуст, и FALSE, если он не пуст:

```
?PR EMPTYP [ ]
TRUE
?PR EMPTYP [1 2 3]
FALSE
?PR EMPTYP [ABC]
FALSE
?
```

Примитив

FIRST (первый)

FIRST Объект

выделяет первый элемент (литеру в слове в списке) объекта.

Примеры (значение переменной : A есть слово QWERT):

PR FIRST 123	Выдает на печать литеру I (первую в слове 123)
I	
PR FIRST :A	Выдает на печать литеру Q (первую в слове QWERT – значение :A)
Q	
PR FIRST "ONE . TWO . THREE "	Выдает на печать литеру O (первую в слове ONE . TWO . THREE)
O	
PR FIRST [HAPPY NEW YEAR]	Выдает на печать слово HAPPY (первое в списке)
HAPPY	

Примитив

LAST Объект

выделяет последний элемент (литеру слова или слово списка) объекта:

```
?PR LAST 987
7
?PR LAST "ZXC
C
?PR LAST [ONE TWO THREE]
THREE
?
```

Наконец, примитив

ITEM N Список

выделяет N-й элемент из списка. Список должен содержать не менее N элементов. В противном случае выдается сообщение об ошибке.

Примеры:

```
?PR ITEM 2 [ONE TWO THREE]
TWO
?PR ITEM 3 [ONE TWO THREE]
THREE
?PR ITEM 3 [9 8 7 6 5]
7
?
```

Лого имеет эффективные и простые операции по модификации списков: устранению отдельных их элементов, включению новых элементов и т.д.

Примитив

FPUT Объект Список

вырабатывает новый список, полученный из старого с включением в его начало нового объекта:

```
?MAKE "A FPUT "ONE [TWO THREE]
?PR :A
ONE TWO THREE
?PR FPUT "FOUR :A
FOUR ONE TWO THREE
?
```

Примитив

LPUT Объект Список

вырабатывает новый список, полученный из старого с включением в его конец нового объекта:

```
?PR LPUT "THREE [ONE TWO]
ONE TWO THREE
?MAKE "A LPUT "THREE [ONE TWO]
?PR :A
ONE TWO THREE
?
```

Примитив

```
LIST Объект1 Объект2 ... ОбъектN
(LIST Объект1 Объект2 ... ОбъектN)
```

вырабатывает список, элементами которого являются указанные объекты.

Например,

MAKE "X LIST [ONE] [TWO]	Переменной :X присвоено
PR :X	значение в виде списка [ONE]
[ONE TWO]	[TWO], которое выведено на
	печать
MAKE "X (LIST [ONE][TWO][THREE])	Переменной :X присвоено
PR :X	значение в виде списка [ONE]
[ONE] [TWO] [THREE]	[TWO][THREE], которое выве-
	дено на печать

Примитив

```
SENTENCE Объект1 Объект2 ... ОбъектN
(SE Объект1 Объект2 ... ОбъектN)
```

вырабатывает список, составленный из перечисленных объектов:

```
?PR SENTENCE "ONE "TWO
ONE TWO
?PR SE [ONE TWO]
ONE TWO
?PR SENTENCE 123 456
123 456
?
```

Ниже сравниваются четыре описанных примитива, объединяющие слова и списки:

Операция	Объект1	Объект2	Выход операции
FPUT	"ONE	"TWO	Ошибка
LIST	"ONE	"TWO	ONE TWO
LPUT	"ONE	"TWO	Ошибка
SENTENCE	"ONE	"TWO	ONE TWO
FPUT	"ONE	[TWO]	ONE TWO
LIST	"ONE	[TWO]	ONE [TWO]
LPUT	"ONE	[TWO]	ONE TWO
SENTENCE	"ONE	[TWO]	ONE TWO
FPUT	[ONE]	[TWO]	[ONE] TWO
LIST	[ONE]	[TWO]	[ONE][TWO]
LPUT	[ONE]	[TWO]	[TWO][ONE]
SENTENCE	[ONE]	[TWO]	[ONE][TWO]
FPUT	"ONE	[ ]	ONE
LIST	"ONE	[ ]	ONE [ ]
LPUT	"ONE	[ ]	ONE
SENTENCE	"ONE	[ ]	ONE

Выходом операций являются списки. Приведен вид их распечатки при исполнении примитива PRINT (PR).

Для создания и преобразования сложных объектов служит ряд дополнительных примитивов (их нет в версии Лого ПЭВМ ZX-Spectrum).

Примитив

PROP "Имя "Признак Объект

придает объекту с заданным именем некоторый признак.

Примитив-функция

GPROP "Имя "Признак

выдает объект с заданными именем и признаком (см. примитив PROP).

Примитив

REMPROP "Имя "Признак

ликвидирует объект с заданным именем и признаком.

Примеры:

```
?PPROP "FAMILY "RUS [IVANOV PETROV SIDOROV]
?PR GPROP "FAMILY "RUS
IVANOV PETROV SIDOROV
?REMPROP "FAMILY "RUS
?PR GPROP "FAMILY "RUS
```

?

В первом примере примитив PROP создал объект из трех фамилий с именем FAMILY и признаком RUS (русские фамилии). Второй пример выполняет на индикацию (с помощью примитива GPROP) объект с именем FAMILY и признаком RUS. В третьем примере примитив REMPROP ликвидирует объект. Наконец, четвертый пример показывает, что объект уже не существует.

Примитив-функция

PLIST "Имя

выдает признак листа, ассоциированный с указанным именем.

Примитив

CATH "Имя Лист инструкций

создает цепочку (лист) инструкций с заданным именем. Примитив

THROW "Имя

используется для сравнения имени с CATH-именем. Если имя не существует, выдается сигнал ошибки.

Примитив

ERROR

выдает лист с сообщением об ошибках.

Таким образом, Лого имеет весьма развитый аппарат по обработке и заданию различных объектов, в том числе текстовых. Это резко облегчает создание на Лого баз данных, текстовых редакторов, каталогов и т.д. Для полной реализации этих возможностей в Лого должны быть включены знаки национального алфавита (к сожалению, большинство версий Лого зарубежных ПЭВМ не допускает использование знаков кириллицы – русского алфавита). Однако интерпретаторы Лого легко дорабатываются с учетом включения в них знаков других алфавитов.

## *2.5. Условные выражения и передача управления*

Внутри всякой процедуры в языке Лого инструкции выполняются строка за строкой снизу вверх. Если встречается подпроцедура, то происходит переход к ее выполнению, после чего выполняется следующая инструкция основной процедуры.

Порядок выполнения инструкций может быть изменен: поставлен в зависимость от выполнения некоторого условия или повторен указанное число раз.

Условные выражения позволяют выполнить некоторые инструкции, если определенное условие истинно (дает TRUE). Эти инструкции игнорируются, если условие не выполняется (дает FALSE). Такие инструкции вводятся примитивом IF (если):

IF Предикат [Список инструкций 1] [Список инструкций 2]

Первым параметром примитива IF является логическая операция (предикат), которая анализируется примитивом IF. Напомним, что предикаты – операции, создающие значения истинности (TRUE) или неистинности (FALSE). Если предикат создает значение TRUE, то будут выполняться инструкции из списка инструкций 1. Если предикат создает значение FALSE, то будут выполняться инструкции из списка инструкций 2 (инструкции из списка 1 игнорируются). В обоих случаях затем происходит выполнение инструкций следующих далее строк процедуры (их нельзя путать со строками экрана дисплея).

Возможна и иная организация условных выражений. Для этого используется примитив

TEST Условие

вырабатывающий предикат TRUE, если условие выполняется, и FALSE, если оно не выполняется.

После него используются примитивы

IFTRUE [Список инструкций]

или

IFT [Список инструкций]

и

IFFALSE [Список инструкций]

или

IFF [Список инструкций]

При использовании примитива IFTRUE или IFT список условий выполняется, если TEST дает TRUE, а при использовании примитива IFFALSE или IFF список условий выполняется, если TEST дает FALSE.

Примитив

END

завершает задание процедуры и действует как команда. Как отмечалось, он фиксирует окончание процедур и возвращает Лого в режим исполнения процедур (прямых вычислений).

Примитив

STOP

прекращает выполнение текущей процедуры. При этом происходит передача управления следующей части надпроцедуры (если она есть) или остановка вычислений. Этот примитив обычно используется для остановки вычислений при определенных условиях, например при возникновении ошибки или просто завершении работы с программой.

Как отмечалось, при программировании на языке Лого нет необходимости в использовании меток и безусловных переходов по ним, которые противоречат концепции структурного программирования. Однако в некоторых версиях Лого (например, для IBM PC и Apple-II или Агат) имеется возможность вводить метки с именем вида

```
LABEL "Имя
```

Безусловный переход к такой метке выполняется командой GO

```
GO "Имя
```

Безусловные переходы неверно рассматривать как абсолютно необходимые. Если они ведут к упрощению программ и применяются не слишком часто, их использование (в версиях Лого с ними) может быть вполне целесообразным.

Рассмотрим примеры применения описанных выше примитивов. Процедура VERT обеспечивает вертикальную печать любого слова – ее единственный параметр:

```
?TO VERT :OBJECT
>IF EMPTY? :OBJECT [STOP]
>PR FIRST :OBJECT
>VERT BF :OBJECT
>END
VERT DEFINED
?VERT "HAPPY
H
A
P
P
Y
?
```

Слово задает значение переменной OBJECT. Затем с помощью условной конструкции IF и примитива EMPTY? анализируется содержание переменной OBJECT. Если она становится пустой, выполняется команда STOP. В противном случае печатается первая буква значения

OBJECT, затем она удаляется и процедура VERT рекурсивно исполняется вновь.

Применение примитивов LABEL и GO иллюстрирует процедура DEMOGO:

```
?TO DEMOGO :X
>LABEL "M
>PR :X
>IF :X>10 [STOP] [MAKE "X :X + 1 GO "M]
>END
DEMOGO DEFINED
?DEMOGO 5
5
6
7
8
9
10
11
?
```

Эта процедура печатает с новой строки значение X, а затем проверяет, больше оно 10 или нет. Если больше 10, то печать прекращается, если нет, то к X прибавляется 1 и задается переход к метке M. В этой процедуре используется полная конструкция IF.

Процедуры можно заставить выполняться заданное число раз с помощью примитива повторения (цикла)

REPEAT 'N Список инструкций

Здесь N – число повторений инструкций, стоящих в указанном списке.

В этом примере команда REPEAT 4 обеспечивает печать слова HELLO 4 раза:

```
?REPEAT 4 [ PR "HELLO ]
HELLO
HELLO
HELLO
HELLO
?
```

Примитив

RUN Список инструкций

используется для исполнения следующего за ним списка. Если список Лого задан входным параметром, то примитив RUN выполняет список инструкций как строку Лого.

В данном примере

```
?RUN [ REPEAT 4 [ PR "HELLO ] ]  
HELLO  
HELLO  
HELLO  
HELLO  
?
```

Также печатается 4 раза слово HELLO, но в заключенные во внешние квадратные скобки команды являются списком инструкций для примитива RUN.

Примитив выхода

OUTPUT Объект

или

OP Объект

обеспечивает остановку выполнения текущей процедуры и передает свой объект вызывающей процедуре для его последующего использования при необходимости. Таким образом, примитив OUTPUT действует как операция выхода. Включение его в процедуру превращает последнюю в функцию, преобразующую входные параметры в результат, который можно использовать для арифметических операций или печати (примеры этого неоднократно приводились выше). Аналогичную OUTPUT функцию по отношению к переменной с заданным именем выполняет примитив

THING Имя

Например, THING :X дает выход для значения переменной X.

Примитив

.DOS

переводит ПЭВМ из Лого-системы и переводит ее в дисковую операционную систему (MS DOS для IBM PC). Для ряда версий Лого аналогичную функцию выполняет примитив BYE. Встречается также примитив TOPLEVEL, прекращающий выполнение текущей процедуры и передающий управление на верхний уровень.

### 3. Графические возможности Лого

#### 3.1. Команды управления экраном и цветом

Команды управления экраном и цветом рассмотрим на примере версии Лого IBM PC. Дисплей этих ПЭВМ может отображать информацию в текстовом и графическом режимах. При вводе интерпретатор Лого вначале устанавливается текстовый режим, при котором на экране можно наблюдать цветные алфавитно-цифровые знаки.

Цветное текстовое изображение характеризуется цветами самих знаков и их основы, т.е. знакоместа, отведенного под соответствующий знак. Эти цвета задаются кодами цвета основы  $C_0$  и цвета знака  $C_3$ . В текстовом режиме можно задавать 16 цветов с кодами от 0 до 15: черный – 0, синий – 1, зеленый – 2, голубой – 3, красный – 4, пурпурный – 5, золотистый – 6, белый – 7, темно-серый – 8, светло-синий – 9, светло-зеленый – 10, светло-голубой – 11, розовый – 12, светло-пурпурный – 13, желтый – 14, ярко-белый – 15.

Для задания цветов знака и основы служит примитив

SETTC [ $C_3$   $C_0$ ]

А примитив-функция

TEXTCOLOR

или

TC

возвращает коды цветов в виде списка [ $C_3$   $C_0$ ].

В графический режим ПЭВМ переводится, как только указывается команда, относящаяся к выводу или управлению черепашкой (кроме команды сокрытия черепашки). Графический режим может быть высокого разрешения:  $640 \times 200$  элементарных элементов изображения – пикселей. В этом случае изображение получается монохроматическим, т.е. двухцветным. При среднем разрешении ( $320 \times 200$  пикселей) изображение может быть цветным. При этом могут устанавливаться два комплекта (или палитры) цветов по 4 цвета в каждом. Палитры задаются номером P (0 или 1) с помощью примитива

SETPAL P

Примитив-функция

PALLETE

или

PAL

обрабатывает номер цветовой палитры.

Черепашка Лого снабжена "световым пером". Если оно поднято, то черепашка при своем движении не оставляет следа. Если перо опущено, то черепашка оставляет след с цветом, заданным кодом С и номером цветовой палитры Р. В табл. 3.1 отражено соответствие между значением С и Р и цветом следа, оставляемого световым пером черепашки.

Таблица 3.1.

Выбор цветов светового пера черепашки в графическом режиме среднего разрешения

Номер палитры Р	Код цвета С			
	0	1	2	3
0	Цвет фона	Зеленый	Красный	Золотистый
1	Цвет фона	Голубой	Пурпурный	Белый

Таким образом, выбор цветов в графическом режиме ограничен. Это связано с тем, что ограничены затраты памяти под хранение  $320 \times 200$  пикселей и их цветовых признаков – атрибутов. Для задания цвета фона в графическом режиме среднего разрешения можно использовать 8 цветов: черный, синий, зеленый, голубой, красный, пурпурный, золотистый и белый. Им соответствуют коды  $C_0$  от 0 до 7 при темном переднем плане и фоне, от 8 до 15 при светлых переднем плане и фоне и от 16 до 23 при светлом переднем плане и темном фоне.

Рассмотрим ряд других примитивов Лого для управления экраном цветом. Примитив

SETTEXT N

открывает под текст N строк в нижней части экрана, т.е. создает текстовое окно.

Примитив

CLEARTEXT

или

CT

переводит ПЭВМ в текстовый режим и очищает от текста экран. Его удобно применять перед началом очередного этапа работы, например перед вводом новых процедур.

Примитив

TEXTSCREEN

или

TS

освобождает под текст весь экран. Он используется, в частности, после применения примитива SETTEXT или работы в графическом режиме.

Примитив

MIXEDSCREEN

или

MS

устанавливает смешанный режим отображения информации (n строк снизу используются под текстовую информацию, остальная часть экрана – под графическую информацию).

Печать текста в текстовом (и графическом) режиме осуществляется, начиная с текущего положения курсора.

Примитив

SETCURSOR [XY]

устанавливает курсор в положение с текстовыми (а не растровыми) координатами X и Y. Примитив

.SETSCREEN M

возвращает значения [XY] для текущего положения курсора.

Максимальные значения X зависят от режима экрана M. Она устанавливается примитивом

.SETSCREEN M

а примитив

.SCREEN

возвращает значение M.

Режимов экрана три:

- M = 0 – текстовый режим,
- M = 1 – графический режим среднего разрешения с отображением информации в цвете (при этом текст может содержать до 40 знаков в строке),
- M = 2 – графический режим высокого разрешения с монохромным изображением (текст может содержать до 80 знаков в строке).

В текстовом режиме размер знаков по горизонтали устанавливается примитивом

SETWIDTH N

где N = 40 или 80 в зависимости от нужного числа знаков в строке (допустимы только указанные значения N).

Любая команда вывода черепашки (см. § 3.2) ведет к установке графического режима для верхней части экрана (свободной от действия примитива SETTEXT). Примитив

FULLSCREEN

или

FS

открывает под графику весь экран, а примитив

CLEARSCREEN

или

CS

полностью очищает экран и переводит черепашку в исходное состояние. Еще один полезный примитив графического режима

CLEAN

стирает все изображение на экране, но оставляет черепашку на том месте, где она находилась перед исполнением примитива CLEAN.

Некоторые версии Лого имеют несколько иную систему команд управления экраном. Так, Лого ZX-Spectrum содержит следующие примитивы:

BACKGROUND или BG	– возвращает номер цвета фона (от 0 до 7),
BRIGHT N	– включает повышенную яркость знаков (N = 1) или нормальную (N = 0),
COPYSCREEN	– дает распечатку принтером копии изображения на экране,
FLASH	– включает мигание знаков,
INVERSE	– включает инвертирование цвета знаков,
NORMAL	– задает нормальный режим (без мигания и инвертирования),
OVER	– включает режим покрытия (новое изображение создается поверх старого),
SETBG C	– задает цвет основы рабочей части экрана с кодом C,
SETBORDER C или SETBR C	– задает цвет окаймления (бордюра) экрана.

У этой ПЭВМ внизу экрана всегда зарезервированы две строки. При снятии копии изображения экрана содержание этих служебных строк не копируется.

### 3.2. Управление черепашкой и ее световым пером

Как отмечалось, характерной особенностью Лого-графики является наличие специального графического объекта – черепашки. Черепашкой можно управлять, поднимая или опуская ее световое перо, устанавливая его цвет, поворачивая черепашку вокруг своей оси и перемещая ее в любом направлении. Управление черепашкой производится в полярной системе координат, как правило, указанием относительных перемещений.

Любой примитив, относящийся к выводу или управлению черепашкой (кроме операции сокрытия ее), ведет к установке графического режима ПЭВМ.

Вывод черепашки задается примитивом

SHOWTURTLE

или

ST

При этом черепашка появляется в центре экрана (рис. 3.1).

Команда

HOME

(домой) возвращает черепашку из любого места в центр экрана, если она была куда-то перемещена. Нарисованное ранее изображение при этом сохраняется.

Команда "спрятать черепашку"

HIDETURTLE

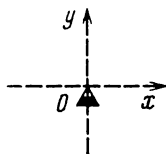
или

HT

делает черепашку невидимой.

Во всех этих случаях, если световое перо черепашки опущено, она оставляет след. Так, перемещение черепашки по команде HOME оставляет след этого перемещения. Невидимая после команды HIDE-TURTLE (HT) черепашка при своем движении также оставляет след. Невидимая черепашка двигается заметно быстрее, чем видимая, так как в этом случае нет затрат времени на построение изображения самой черепашки.

Рис. 3.1. Исходное положение черепашки (HOME)



Пользователь Лого имеет ряд возможностей по управлению движением черепашки. Они реализованы соответствующими командами-примитивами.

Примитив

FORWARD N

или

FD N

Перемещает черепашку вперед на N шагов относительно ее текущего положения. Направление движения определяется головкой черепашки, а не положением экрана дисплея ПЭВМ. Так, если головка черепашки находится сверху, как на рис. 3.1, то команда FD приведет к движению черепашки снизу вверх. В общем случае черепашка двигается вперед в направлении ее радиус-вектора (рис. 3.2).

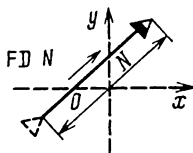


Рис. 3.2. Движение черепашки вперед (FORWARD)

Число N задается в растровых единицах (пикселях). Максимальные и минимальные значения N ограничены, причем эти значения различны у разных типов ПЭВМ. Пользователь может задавать свои ограничения на перемещение черепашки по экрану дисплея (соответствующие команды описываются ниже).

Команда

BACK N

или

BK N

Перемещает черепашку назад на N шагов вдоль радиус-вектора (рис. 3.3).

Соответственно, команда

RIGHT  $\phi$

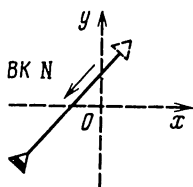


Рис. 3.3. Движение черепашки назад (BACK)

или

RT  $\varphi$

поворачивает черепашку по часовой стрелке (вправо) на  $\varphi$  градусов (рис. 3.4), а команда

LEFT  $\varphi$

или

LT  $\varphi$

поворачивает ее против часовой стрелки (влево) на  $\varphi$  градусов (рис. 3.5).

Рис. 3.4. Поворот черепашки вправо (RIGHT)

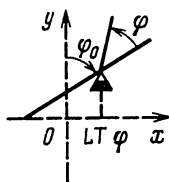
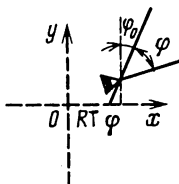


Рис. 3.5. Поворот черепашки влево (LEFT)

Команда перемещения черепашки вдоль оси  $x$

SET X  $N_x$

перемещает черепашку вдоль оси  $x$  на  $N_x$  шагов. При этом координата  $Y$  черепашки не меняется.

Команда перемещения черепашки по оси

SET Y  $N_y$

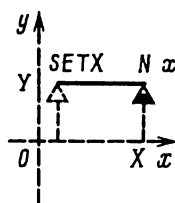
перемещает черепашку на  $N_y$  шагов вдоль оси  $y$ . При этом не меняется координата  $X$ .

Для перемещения черепашки в точку с координатами  $(x, y)$  служит команда

SETPOS [ $X$   $Y$ ]

Если световое перо опущено, то перемещение черепашки по командам SETX, SETY и SETPOS оставляет след (рис. 3.6).

Рис. 3.6. Перемещение черепашки из исходного положения в точку  $(x, y)$



Команда

DOT [X Y]

перемещает в точку с координатами  $(x, y)$  черепашку без оставления следа (рис. 3.7). След в виде точки появляется на месте остановки черепашки.

Команда

FENCE

(в переводе ограда) задает ограничение на перемещение черепашки в пределах заданного окна. После подачи этой команды Лого не позволяет задавать перемещения черепашки за пределами окна (рис. 3.8).

Команда

WINDOW

(окно) снимает ограничение на перемещение черепашки за пределами экрана. При этом черепашка подчиняется инструкциям управления, даже если она в пределах окна. Черепашку можно передвигать в любом направлении от центра (обычно это перемещение ограничено,

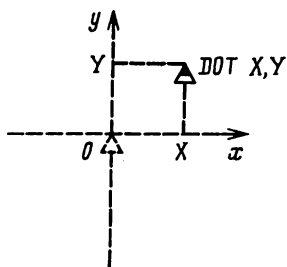


Рис. 3.7. Прыжок черепашки из исходного положения в точку  $(x, y)$

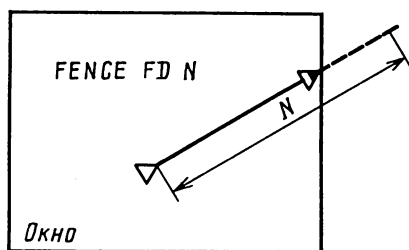


Рис. 3.8. Перемещение черепашки за пределами экрана или окна

но числом шагов намного большим, чем нужное для достижения границ окна).

Команда "замкнуть края"

WRAP

соединяет друг с другом противоположные края окна, в котором движется черепашка. Если она пересекает границу окна, она тут же появится на его противоположной границе (рис. 3.9).

Команда

SETHEADING  $\varphi$

или

SETH  $\varphi$

(установить направление) обеспечивает поворот черепашки на угол  $\varphi$  относительно исходного состояния  $\varphi = 0$  (рис. 3.10). Угол, как обычно, отсчитывается против часовой стрелки. Координатная система, в которой вращается черепашка, дана на рис. 3.10. Отметим, что в исходном состоянии черепашка находится в точке  $(0, 0)$  с координатами  $x = 0$  и  $y = 0$ .

Функция "направление"

TOWARDS [X Y]

дает угол  $\varphi$ , на который должна повернуться черепашка, чтобы при движении вперед попасть в точку с координатами  $(X, Y)$ . Эта точка может быть и за пределами экрана (рис. 3.11).

При первом выводе черепашки ее световое перо обычно опущено, так что всякое перемещение черепашки оставляет след.

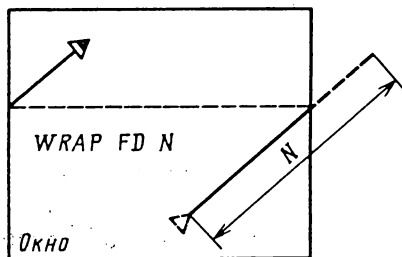


Рис. 3.9. Действие команды "замкнуть края" (WRAP)

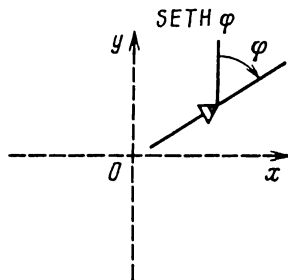
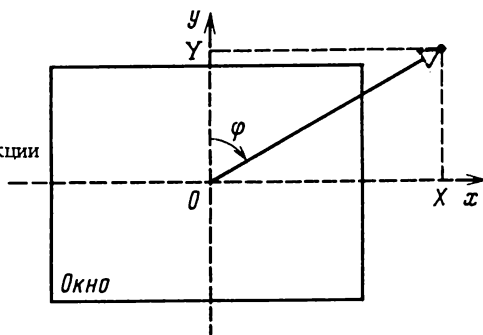


Рис. 3.10. Поворот черепашки на абсолютный угол  $\varphi$  (SETHEADING)

Рис. 1.11. Иллюстрация к действию функции  
TOWARDS



В общем случае световое перо черепашки опускается командой

**PENDOWN**

или

**PD**

В буквальном переводе эта команда означает "перо вниз".

Команда выключить световое перо (перо вверх)

**PENUP**

или

**PU**

Обеспечивает перемещение черепашки без оставления следа.

Имеется также специальная команда "реверсивное световое перо"

**PENREVERSE**

или

**PX**

Если черепашка двигается, то в этом случае она оставляет линию там,

где ее раньше не было и стирает линию там, где она раньше была.

Команда стирания линий

**PENERASE**

или

**PE**

Обеспечивает стирание тех линий, по которым проходит черепашка.

Следует отметить, что команды **PENDOWN**, **PENUP** и **PENREVERSE**

отменяют действие команды **PENERASE**. Соответственно команды

**PENDOWN**, **PENUP** и **PENERASE** отменяют действие команды **PENRE-**

**VERSE**.

Установка цвета (colour) светового пера осуществляется командой

**SETPC C**

где С – код цвета. Коды цвета у разных ПЭВМ могут отличаться. Кодирование цвета ПЭВМ класса IBM описывалось в 3.1. Существуют ПЭВМ, имеющие 4, 8 или 16 цветов, есть отдельные модели с гораздо большим числом цветов. Коды цветов указываются в описаниях ПЭВМ, они обычно те же, что и в Бейсике. Например, у ZX-Spectrum черный цвет имеет код 0, синий 1, красный 2, белый 7 и т.д. (всего 8 цветов).

Версия Лого IBM PC имеет весьма полезный примитив закрашки замкнутых фигур текущим цветом светового пера

## FILL

Для закрашки замкнутой фигуры достаточно поместить внутрь ее черепашку и исполнить команду FILL (рис. 3.12, а). Если черепашка находится вне замкнутой фигуры, закрашивается внешняя часть экрана – до бордюра – рис. 3.12, б. Часть экрана дисплея за пределами окна называется бордюром, рабочее поле, в пределах которого может перемещаться черепашка, обычно называют окном.

Примитив

SETSCRUNCH [X% Y%]

или

SETSCR [X% Y%]

обеспечивает изменение в процентах масштабов изображения по осям x и y. Например, при построении прямоугольника SETSCRUNCH 50 180 уменьшает ширину его до 50 % от первоначальной и высоту увеличивает до 180 % от первоначальной (рис. 3.13). При этом построенные окружности превращаются в эллипсы.

Данные о положении черепашки (вспомним, что она может скрываться) и цвете ее светового пера (даже если оно поднято) принято называть атрибутами.

Примитив (системная переменная Лого)

XCOR

имеет значение координаты X черепашки, а примитив

YCOR

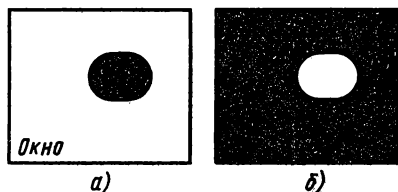
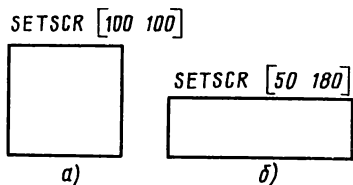


Рис. 3.12. Закрашка замкнутой фигурой (а) и внешней части экрана (б)

Рис. 3.13. Квадрат, построенный черепашкой (а) и его вид при действии команды SETSCRUNCH [50 180] (б)



Выводит значение координаты Y.

Примитив

SHOWNP

Возвращает предикат TRUE (истина), если черепашка есть на экране, и FALSE (ложно), если ее нет.

Примитив

POSITION

Выводит

POS

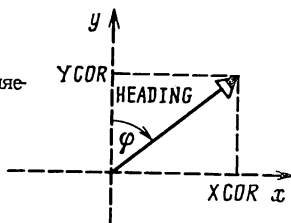
Возвращает растровые координаты [X Y] местоположения черепашки.

Примитив

HEADING

(Направление) выдает угол  $\varphi$  (относительно исходного положения), на который повернута черепашка. Этот угол есть число от 0 до 359. Рис. 3.14 поясняет смысл этих атрибутов.

Рис. 3.14. Данные о положении черепашки, предоставляемые атрибутами XCOR, YCOR, HEADING



Примитив

PENCOLOUR

Выводит

PS

Возвращает число C – код цвета светового пера черепашки.

Примитив

SCRUNCH

возвращает относительные значения сторон (ширины и высоты)  $[X\% \ Y\%]$  и соответственно значения относительной величины шага черепашки при ее перемещении по осям  $x$  и  $y$ .

### 3.3. Задание и вывод графических элементов пользователя (графэм)

Хотя черепашка Лого способна рисовать сложные объекты, сложность рисования их невелика. Кроме того, неопределенность положения черепашки в пределах  $\pm 1$  пикселя затрудняет создание с помощью малых графических объектов строго заданной формы.

Многие современные ПЭВМ имеют специальные средства для создания малых графических элементов (графэм) или образов (спрайтов). Они создаются, так же как и знаки алфавита рабочего языка, матрице знакоместа или нескольких знакомест. Чаще всего используется матрица  $8 \times 8$  пикселей, в отдельных случаях  $16 \times 16$ .

Так, ZX-Spectrum имеет 16 встроенных графэм с кодами их вывода от 128 до 143. Например, графэма с кодом 143 – полностью закрашенный квадрат, графэма с кодом 140 – квадрат с закрашенной нижней половиной. Как и для других знаков алфавита, вывод графэм можно осуществить с помощью примитива PR (см. примеры ниже):

```
?PR CHAR 143
?PR CHAR 140
?
```

Определенная область ОЗУ отводится под графэмы, задаваемые пользователем. У упомянутой ПЭВМ им соответствуют коды вывода от 144 до 164 (всего 21 элемент). До ввода графэм пользователя эти коды соответствуют графэмам в виде больших латинских букв от A до U.

Задание графэмы сводится к составлению ее матрицы и кодированию последней построчно. Графэма может иметь один цвет, общий для всех входящих в нее пикселей. На рис. 3.15 показана матрица графэмы – фигурки человечка. Каждый элемент матрицы кодируется числом 0, если он не закрашен, и 1, если он закрашен. Таким образом, графэма с матрицей  $8 \times 8$  элементов кодируется восемью байтами или восемью десятичными либо шестнадцатеричными числами. Лого обычно оперирует с десятичными кодами в виде чисел

$$D = 2^7 \cdot d_7 + 2^6 \cdot d_6 + \dots + 2^1 \cdot d_1 + 2^0 \cdot d_0,$$

где  $d_0 - d_7$  – двоичные числа, имеющие значение 0 или 1; индекс чисел соответствует их разряду, причем за нулевой разряд принимается последний разряд каждой строки матрицы.

Рис. 3.15. Матрица графэмы с фигуркой человека и коды ее строк

Матрица графэмы	Коды графэмы	
	Двоичные	Десятичные
	00011000	24
	00001000	8
	00111111	63
	01011101	93
	10011101	157
	00010100	20
	00100010	34
	00100001	33

Две приведенные ниже процедуры обеспечивают задание произвольных графэм в ZX-Spectrum:

?

```

TO DEFCHAR :CH :ALIST
IF OR NOT NUMBERP :CH NOT LISTP :ALIST [STOP]
IF OR :CH < 144 :CH > 144 [STOP]
DODEF :ALIST 0
END

TO DODEF :ALIST :COUNT
IF EMPTY :ALIST [STOP]
DEPOSIT 64216 + :CH * 8 + :COUNT FIRST :ALIST
DODEF BUTFIRST :ALIST :COUNT + 1
END

```

Первая процедура DEFCHAR имеет два входных параметра: код вызова графэмы и список :ALIST десятичных кодов строк графэмы. Процедура проверяет правильность задания кода вызова (он лежит в пределах от 144 до 164) и правильность записи списка :ALIST. Если они заданы верно, идет обращение к процедуре DODEF, иначе происходит остановка.

Процедура DODEF обеспечивает занесение кодов строк графэмы в нужную область ОЗУ. Для этого используется описанный далее примитив DEPOSIT.

Графэма на рис. 3.15 задается следующим образом:

```
DEFCHAR 144 [24 8 63 93 157 20 34 33]
```

В квадратных скобках указываются десятичные коды строк графэмы (сверху вниз), перед ними – код вызова графэмы.

После исполнения этой команды команда

SETCUR [12 10] PR CHAR 144

вызовет появление в центре экрана (место указывается командой задания положения курсора SETCUR [12 10]) маленькой фигурки человека.

Скорость вывода заранее составленных графэм намного больше скорости рисования аналогичных фигур непосредственно черепашкой. Поэтому применяя набор быстро выводимых фигур (например человечков в различных позах) можно создавать машинные фильмы, напоминающие мультипликационные. Графэмы полезны также при задании знаков различных алфавитов, отсутствующих у данной ПЭВМ.

Однако построение изображений с помощью графэм имеет один серьезный недостаток – положение графэмы задается с низкой точностью (до точности задания знака в текстовой, а не растровой сетке экрана). Поэтому перемещение графэм всегда носит характер заметных быстрых скачков.

В некоторые версии Лого заложены компромиссные решения, позволяющие любой знак (в том числе и графэму) поместить на то место, где расположена черепашка. При этом координаты знака определяются более точными растровыми координатами.

В версии Лого IBM PC такая возможность реализуется примитивом "установить образ":

SETSHAPE "Цепочка знаков или знак

Примитив

SHAPE

возвращает код первого знака, который использовался при применении примитива SETSHAPE.

Если нужно вернуть черепашке ее "естественный" вид, то используется команда

SETSHAPE "TURTLE

Следующий пример иллюстрирует действие этих команд в процедуре TITLE (титул), выводящей надпись с увеличенным шрифтом последовательно знак за знаком в любое место экрана:

```
?PO "TITLE
TO TITLE :XC :YC :LST
MAKE "NUM 1
PU SETPOS SE :XC :YC
```

```

RT 90 ST
REPEAT COUNT :LST [WRITEWORD ITEM :NUM
: LST MAKE "NUM :NUM + 1]
SETSHAPE "TURTLE
HT PD SETH 0
END

```

```
?TITLE -150 90 "HELLO! ST
```

**H E L L O ! **

Входными параметрами процедуры TITLE являются координаты XС и YС начала надписи и сама надпись – значение переменной входного параметра LST. Эта процедура очень удобна для динамического вывода титульных надписей (итог ее работы показан под листингом первой процедуры).

Еще две процедуры WRITEWORD и DWW создают надписи по сторонам квадрата (рис. 3.16):

Рис. 3.16. Квадрат, образованный словами QWERTY

```

Q Y T R E W Q
W               Y
E               T
R               R
T               E
Y               W
Q W E R T Y Q

```

```

?PO [WRITEWORD DWW]
TO WRITEWORD :WD
IF EMPTY? :WD [FD 5 STOP]
SETSHAPE ASCII FIRST :WD
STAMP FD 12
WRITEWORD BF :WD
END

```

```

TO DWW
CS ST WRITEWORD "QWERTY
LT 90 WRITEWORD "QWERTY
LT 90 WRITEWORD "QWERTY
LT 90 WRITEWORD "QWERTY
HT
END

```

### 3.4. Построение сложных графических объектов

С помощью движущейся и поворачивающейся черепашки можно строить сложные графические объекты. Пример этого будет приведен в гл. 8, здесь же мы остановимся на общих принципах построения сложных объектов.

Лого дает весьма удобный механизм для расчленения сложного алгоритма вычислений или построения графиков на отдельные простые алгоритмы. Концепция структурного программирования предполагает, что сложная задача реализуется по принципу сверху вниз — расчленением ее на отдельные части (структуры). Покажем, как это делается, на примере стилизованной фигурки человека на рис. 3.17.

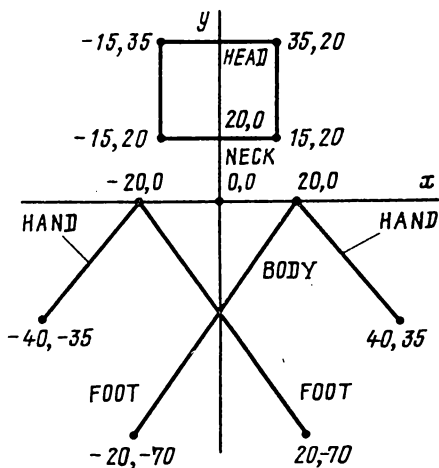


Рис. 3.17. Объект — фигура человека с координатами особых точек

Основная процедура построения фигурки человека MAN должна содержать следующие процедуры: построение туловища BODY, шеи NECK, головы HEAD, рук HANDS и ног FEET. Заключительная процедура COMMENT должна выдавать пояснительную надпись, например "THIS IS A MAN" (это человек). Тексты этих процедур даны ниже

```
?POALL
TO COMMENT
SETCURSOR [5 13]
PRINT [THIS IS A MAN]
END
```

```
TO FEET
PENUP
SETPOS [0 -35]
```

RTHEADING 150  
PENDOWN  
FORWARD 40  
BACK 40  
RIGHT 60  
FORWARD 40  
HIDETURTLE  
END

TO HANDS  
PENUP  
SETPOS [-20 0]  
RTHEADING 210  
PENDOWN  
FD 40  
PENUP  
SETPOS [20 0]  
RTHEADING 150  
PENDOWN  
FD 40  
END

TO HEAD  
SETPOS [-15 20]  
REPEAT 4 [FORWARD 30 RIGHT 90]  
END

TO NECK  
HOME FD 20  
END

TO BODY  
SETPOS [-20 0]  
RTHEADING 90  
REPEAT 3 [FORWARD 40 RIGHT 120]  
END

TO MAN  
CLEARSCREEN  
BODY NECK HEAD HANDS FEET  
COMMENT FULLSCREEN  
END

Процедуры BODY, NECK, HEAD, HANDS и FEET должны реализовать определенные алгоритмы графических построений, доведенные до исчерпывающего и однозначного толкования. Алгоритмы могут быть описаны словесно или представлены в виде графиков.

Выберем словесное описание алгоритмов.

Туловище BODY геометрически есть правильный треугольник. Алгоритм его построения (один из возможных) следующий: вводим черепашку в позицию  $(-20, 0)$ , устанавливаем угол поворота  $90^\circ$  и трижды повторяем перемещение черепашки вперед на 40 шагов и поворот ее на  $360/3 = 120^\circ$ .

Алгоритм построения шеи NECK следующий: выводим черепашку в исходное положение с координатами  $(0, 0)$  и нулевым углом поворота, передвигаем ее вперед на 20 шагов.

Голова HEAD фигурки — квадрат. Его построение можно реализовать следующим алгоритмом: перемещаем черепашку в позицию  $(-15, 20)$ , и, помня, что угол поворота остался нулевым, строим квадрат четырехкратным перемещением черепашки вперед на 30 шагов и последующим поворотом на  $90^\circ$ .

Построение рук HANDS реализуем таким алгоритмом: выводим черепашку с поднятым пером в позицию  $(-20, 0)$ , задаем угол  $210^\circ$ , опускаем перо и передвигаем черепашку вперед на 40 шагов (строится одна рука); перемещаем черепашку, подняв перо в позицию  $(20, 0)$ , устанавливаем угол  $150^\circ$ , опускаем перо и перемещаем черепашку вперед на 40 шагов (строится вторая рука).

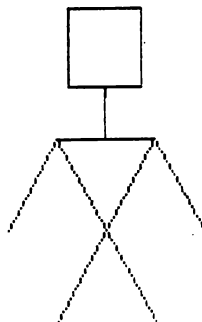
Один из алгоритмов построения ног FEET следующий: поднимаем перо, выводим черепашку в позицию  $(0, -35)$ , задаем угол  $150^\circ$ , опускаем перо и перемещаем черепашку вперед на 40 шагов (строится одна нога); возвращаем черепашку назад на 40 шагов, поворачиваем ее вправо на угол  $60^\circ$ , перемещаем черепашку вперед на 40 шагов (строится вторая нога) и делаем ее невидимой (процедурой FEET заканчиваются построения).

Для вывода над фигуркой человека надписи-комментария THIS IS A MAN можно использовать процедуру COMMENT, выводящую в нужное место курсор и печатающую надпись-комментарий.

Вводя эти процедуры, их можно легко отлаживать отдельно. Например, задав исполнение процедуры BODY, получим изображение туловища фигурки человека. Однако в данном случае последовательность отладки должна соответствовать описанной в главной процедуре MAN, так как некоторые процедуры требуют предварительной установки нужного начального угла поворота черепашки. Разумеется, этот угол можно задавать определенным (как в процедуре BODY)

## THIS IS A MAN

Рис. 3.18. Фигура человека, построенная обращением к процедуре MAN



ной некоторого усложнения процедур. Задав исполнение основной процедуры MAN, получим фигуру человека на рис. 3.18.

При всей наглядности подобной разбивки построений на простейшие части, она вступает в явное противоречие с эффективностью программы в целом. Разумеется, можно механически объединить все процедуры в одну процедуру MAN. Даже это заметно уменьшит общую длину программы, но сделает ее менее наглядной.

Приведенный пример лишь иллюстрирует технику построения сложных фигур. Если эффективность программы является определяющей, то прежде всего следует рассмотреть возможность поиска лучшего алгоритма. Он может базироваться, например, на определенных геометрических закономерностях объекта, нередко теряющихся при детальном разбиении его на составные части.

Так, фигурка на рис. 3.18 может быть построена целым рядом более экономичных (с позиций числа реализующих операций) алгоритмов. К примеру, руки, ноги и часть туловища удобно строить одновременно путем построения двух пересекающихся ломаных, соединяющих точки  $(-40, -35)$ ,  $(-20, 0)$  и  $(20, -70)$  для одной ломаной и  $(40, -35)$ ,  $(20, 0)$  и  $(-20, -70)$  — для другой. Это реализуется следующей процедурой:

```
?PO "MAN1
TO MAN1
CS HOME FD 20 LT 90 FD 15
REPEAT 4 [RT 90 FD 30]
PU SETPOS [20 0] PD FD 40
PU SETPOS [40 -35] PD RT 60
FD 40 RT 60 BK 80
PU SETPOS [-40 -35] PD FD 40
```

LT 60 BK 80 HT SETCURSOR [5 13]  
PR [THIS IS A MAN] FS  
END

Процедура MAN1 обеспечивает построение фигурки, совершенно аналогичной строящейся с помощью процедур MAN, BODY, NECK, HEAD, HANDS, FEET и COMMENT. Процедура MAN1 записана в более компактном виде и занимает значительно меньший объем памяти (более чем вдвое) ПЭВМ. Фигурку человечка она строит также заметно быстрее.

Таким образом, объединение процедур и, главное, оптимизация и укрупнение алгоритмов позволяют значительно повысить эффективность программ, ускорить процесс вычислений, уменьшить затраты памяти ПЭВМ и повысить степень сложности решаемых на них задач.

Приведенные выше примеры показывают, что компромисс между наглядностью программ и эффективностью целиком лежит в сфере представлений пользователя. Для повышения скорости выполнения программ в них можно включать процедуры в машинных кодах. (см. § 4.4).

Лого давно используется для создания видеоэффектов. Достаточно вспомнить часто появляющиеся вставки в научно-популярных телевизионных передачах – раскручивающиеся на глазах спирали, вращающиеся геометрические фигуры и т.д.

На пути совершенствования цветной Лого-графики достигнуты впечатляющие успехи. Так, одна из версий Лого ПЭВМ Atari оперирует сразу четырьмя черепашками и имеет 128 цветов. Еще более интересные возможности Лого ПЭВМ MSX, широко применяемых в нашей системе народного образования. Эта версия позволяет управлять сразу 1 – 30 "черепашками" при 16 цветах [3, 32]. Тут мы вынуждены взять слова "черепашки" в кавычки – по умолчанию первые 10 графических объектов это изображения сердца, грузовика, кошки, собачки, ракеты и т.д. К услугам пользователя редактор графэм, позволяющий строить любые другие объекты в матрице 16 × 16 пикселей. Повышенная скорость вывода графэм открывает возможности создания эффективной динамической графики, например машинных фильмов. Это также расширяет возможности подготовки учебных программ.

В Лого-графике ПЭВМ MSX имеется возможность создания особым графических объектов – *спрайтов* (sprite в переводе добрая фея). Это достигнуто вводом приоритетов при отображении накладывающихся друг на друга образов. Спрайт – образ, исчезающий при наложении на него другого образа и возникающий вновь, когда это наложение исчезает. Под спрайтами подразумевают и динамические объекты,

стоящие из нескольких меняющихся графэм. Это позволяет графически создавать, например, облака с меняющейся при движении формой, фигурки людей с забавной походкой, самолеттики, выполняющие фигуры высшего пилотажа, и т.д.

## 4. Общение с "внешним миром" и памятью

### 4.1. Команды вывода и ввода

К основным устройствам вывода информации ПЭВМ относятся дисплей и принтер (рис. 4.1). Вначале рассмотрим вывод информации на экран дисплея. Переключение канала вывода на принтер будет описано несколько позже.

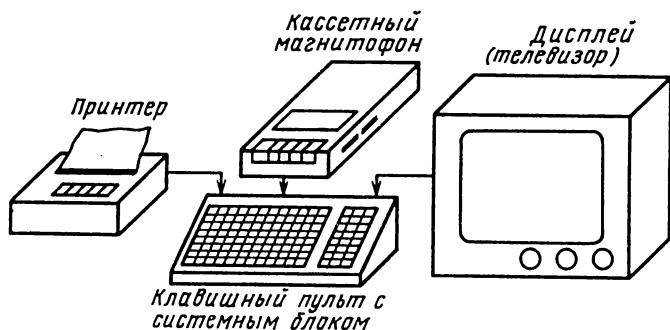


Рис. 4.1. Типовая структура персональной ЭВМ

Команда

```
PRINT      Объект
(PRINT     Объект1 Объект2 ... ОбъектN)
```

или

```
PR         Объект
(PR        Объект1 Объект2 ... ОбъектN)
```

Печатает объект (или объекты) без кавычек или внешних квадратных скобок с переходом к следующей строке. Вернее, выводит объект на экран дисплея.

Другая команда

```
TYPE      Объект
```

или (TYPE Объект1 Объект2 ... ОбъектN)

похожа на PRINT, но отличается тем, что при печати объектов перевод строки отсутствует.

Наконец, примитив

SHOW Объект

печатает, начиная с текущего положения курсора, слово, список или числа, но с квадратными скобками, в которые они заключены.

Различия между этими командами видны из следующих примеров

```
?PR [1 2 3]
1 2 3
?PR 1 PR 2 PR 3
1
2
3
?TYPE 123
123?TYPE [1 2 3]
1 2 3?
?SHOW [1 2 3]
[1 2 3]
?SHOW ["A "B "C]
["A "B "C]
?
```

Эти команды имеются во всех версиях Лого. Версия IBM PC имеет еще одну характерную команду

FORM X N

Эта команда выводит целую часть арифметического выражения X, но с отступом вправо на N позиций:

```
?PR FORM 10/3 10
      3
?PR FORM 100/3 10
      33
?PR FORM 1000/3 10
      333
?PR FORM 1000/3 3
333
?
```

Команды PR, TYPE и SHOW могут использоваться совместно с командой установки курсора SETCUR  $[N_x N_y]$ . Это особенно полезно в графическом режиме работы, так как позволяет накладывать на рисунки и графики различные поясняющие надписи. В некоторых

версиях Лого (например, ZX-Spectrum) перед командами PR, TYPE и SHOW могут стоять команды INK, FLASH, OVER, INVERSE и другие, дающие особые цветовые эффекты (цвет, мигание символов, смешивание с ранее выведенными символами, инверсию изображения и т.д.).

Так, выполнение такого предложения

```
? CS SETCUR [9 10] FLASH PR "TURTLE
```

сначала очищает экран и выводит черепашку в исходное состояние (команда CS), затем переводит курсор в положение, заданное аргументами в команде SETCUR [9 10] (т.е. на 10-е знакоместо 9-й строки), делает мигание надписи (команда FLASH) и, наконец, выводит рядом с черепашкой мигающую надпись TURTLE.

Для переключения канала вывода на принтер используются специальные примитивы. К сожалению, они различны для разных версий Лого. Так, в версии IBM PC для переключения вывода используется слово

```
DRIbble "Имя
```

где "Имя — имя устройства вывода. Так, команда

```
DRIbble "LPT1
```

переключает вывод на первый принтер (возможно подключение нескольких принтеров). Отмена этой команды производится примитивом

```
NODRIbble
```

В ряде версий Лого включение вывода на принтер производится командой

```
PRINton или .PRINter 1
```

Выключение — командой

```
PRINtoFF или .PRINter 0
```

Первая форма используется в версии Лого ZX-Spectrum, вторая — в версии Apple-II. Как отмечалось, печать копии любого изображения графическим принтером выполняется командой COPYSCREEN.

Часто необходимо заставить ПЭВМ выполнять некоторые действия при нажатии на определенную клавишу (клавишу пользователя). Для этого удобно использовать операцию считывания символа нажатой клавиши

```
READCHAR
```

или

RC

Эта операция при нажатии на любую клавишу создает, но не печатает соответствующий символ. Созданный символ можно использовать для задания значений переменных, в условных выражениях и т.д.

Процедура DEMODRAW иллюстрирует функции простейшего графического редактора:

```
?PO "DEMODRAW
TO DEMODRAW
IF RC = 4 [LT 5]
IF RC = 6 [RT 5]
IF RC = 8 [FD 2]
IF RC = 2 [BK 2]
DEMODRAW
END
```

?

При нажатии на клавишу 4 черепашка поворачивается влево на 5°, при нажатии на клавишу 6 – вправо на 5°. Нажатие на клавишу 8 вызывает ее продвижение вперед на 2 шага, а на клавишу 2 – назад на 2 шага. Таким образом можно рисовать замысловатые фигуры.

Лого IBM PC имеет и более сложную конструкцию ввода:

```
READCHARS N
```

или

```
RCS N
```

При исполнении этого примитива ПЭВМ останавливает работу по программе и ждет вывода N символов. После этого работа возобновляется автоматически без нажатия клавиши ENTER (ее нажатием можно прервать ввод, если знаков меньше N).

В следующих примерах этот примитив использован для ввода пяти букв, образующих слово QWERT, а затем восьми цифр (от 1 до 8):

```
?MAKE "X READCHARS 5
?PR :X
QWERT
?MAKE "X RCS 8
?PR :X
12345678
?
```

В обоих случаях введенная строка из N символов присваивается как значение переменной X, а затем выводится на печать.

Примитив

READLIST

или

RL

ведет к останову ПЭВМ и запросу на ввод списка. Он создается нажатием на любые клавиши, кроме клавиши ввода. Последняя фиксирует введенный список. По мере ввода элементов списка они индицируются. При этом действует система редактирования строки. Таким образом, эта команда позволяет вводить список-строку произвольной длины. Ввод завершается нажатием клавиши перевода строки.

В следующем примере примитив RL использован для присвоения переменной X значения символьной строки в виде 16 знаков (десять цифр и шесть букв):

```
?MAKE "X RL
0123456789ABCDEF
?PR :X
0123456789ABCDEF
?
```

Еще один примитив ввода

KEYP

возвращает (но не печатает) значение TRUE (истинно), если нажата допустимая клавиша и допустимая комбинация клавиш. В противном случае выдается значение FALSE (не истинно).

Две процедуры

ROT и CON

```
?PO "ROT
TO ROT :X
IF :X = "R [RT 5]
IF :X = "L [LT 5]
END
```

```
?PO "CON
TO CON
FD 1
IF KEYP [ROT RC]
```

CON  
END

?

при пуске (команда CON) обеспечивают непрерывное движение черепашки. Нажатие клавиши R поворачивает ее вправо на 5°, а L – влево. Таким образом можно рисовать круги, восьмерки, знаки  $\infty$  и т.д. (рис. 4.2).

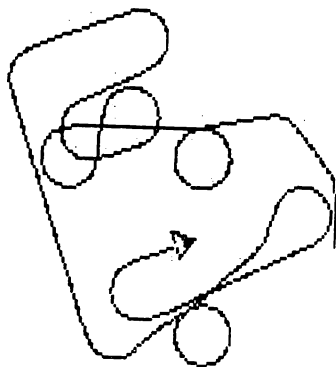


Рис. 4.2. Рисунок, полученный при управлении черепашкой с помощью процедуры CON.

#### 4.2. Работа с накопителями информации

Для записи данных и программ на внешние накопители (см. рис. 4.1) ПЭВМ формирует *файлы* – поименованные упорядоченные массивы данных или инструкций процедур, снабженные заголовком, с указанием типа файла и разбитые на отдельные части (сегменты). При формировании файла ПЭВМ вычисляет специальный код, например в виде суммы кодов всех команд. Этот код также заносится в файл. После считывания файла подобный код вычисляется по кодам, которые реально приняты ПЭВМ с накопителя, и сравнивается с исходным кодом. Если они совпадают, ПЭВМ переходит в режим готовности. В противном случае появляется сигнал ошибки.

Наиболее удобны накопители информации на магнитных дисках, гибких и жестких (имеются не во всех моделях ПЭВМ).

На гибкий диск помещается информация с объемом примерно 0,1 – 1 Мбайта (0,32 у IBM PC). Жесткий диск содержит 5 – 50 Мбайт (также ориентировочно). Время доступа информации у дисковых накопителей – доли-единицы секунд.

Большинство возможностей дисковых накопителей реализуется посредством операционной системы (MS DOS у IBM PC [33]). Однако

некоторые функции реализуются и средствами Лого. Они рассмотрены ниже.

Директива DIR выводит каталог файлов, входящих в директорию (подкаталог) Лого:

```
?DIR
TOOLS      LF      5979   01-01-80   00:04
LOGO       COM     59120   10-01-83   00:00
SAMPLES    LF      2322   10-01-83   00:00
FILE1      LF      1876   10-01-83   00:00
FILE2      LF      6550   10-01-83   00:00
FILE3      LF      5511   10-01-83   00:00
FILE4      LF      2823   10-01-83   00:00
FILE5      LF      3003   10-01-83   00:00
FILE6      LF      2532   10-01-83   00:00
FILE7      LF      2818   10-01-83   00:00
?
```

В данном случае директория содержит интерпретатор Лого (файл с именем LOGO.COM). Директива DIR выводит: название файла, тип файла, его длину (в байтах), дату записи и время (если оно было указано, иначе время дается в виде 00:00).

Запись и считывание с диска осуществляются командами

```
SAVE      "Имя файла [Список процедур]
LOAD      "Имя файла [Список процедур]
```

а стирание файла командой

```
ERASEFILE "Имя файла
```

Запись файла, содержащего изображение на экране дисплея, выполняется командой

```
SAVEPIC   "Имя
```

а считывание такого файла с диска командой

```
LOADPIC   "Имя
```

В ПЭВМ обычно встроено два-три накопителя. Системная переменная — примитив

```
DISK
```

имеет значение данных используемого дискового накопителя.

Некоторые версии Лого имеют команду включения дискового накопителя

## SETDISK N1 N2 N3

где N1 – номер дисководов (если их несколько), N2 – номер щели в дисковом вводе, N3 – номер сектора диска.

У простых ПЭВМ, например ZX-Spectrum, дисковые накопители из-за их дороговизны обычно не применяются. Для записи программ и данных используются бытовые кассетные магнитофоны. На кассету с временем "звучания" 60 – 90 мин. можно поместить до 0,5 – 0,8 Мбайта информации. Сохранность ее выше, чем на дисках. Основным недостатком кассетных магнитофонов как накопителей информации – большое время поиска, записи и считывания (более чем в 100 раз превышающее эти параметры у дисковых накопителей).

Информация при использовании таких накопителей также задается в виде файлов. Если при считывании файла наблюдается сбой, то появляется сигнал ошибки, например "Tape load error" (магнитофон считывает ошибочно). Перед файлом большинство ПЭВМ формирует короткий звуковой сигнал (0,5 – 3 с), который служит для настройки системы автоматического регулирования уровня записи (АРУЗ). Такой системой снабжены современные кассетные магнитофоны, применяемые для записи программ ПЭВМ.

В Лого программа состоит из отдельных процедур. Для их записи на кассетный магнитофон служит примитив SAVE (в переводе – сохранить):

```
SAVE  "Имя файла"  "Имя процедуры"
SAVE  "Имя файла"  [Список имен процедур]
```

Пример:

```
SAVE "BOX" "BOX4"
```

– записывается файл с именем BOX, содержащий одну процедуру BOX4,

```
SAVE "BOXS" [BOX NBOX]
```

– записывается файл с именем BOXS, содержащий две процедуры, которые указаны в списке, заключенном в квадратные скобки.

Часто при вводе этих команд ПЭВМ предупреждает пользователя о необходимости включения лентопротяжного механизма кассетного магнитофона. Оно должно предшествовать исполнению команды SAVE.

У некоторых версий Лого, например ZX-Spectrum, имеются дополнительные команды SAVED (запись данных-текстов), BSAVE (запись кодов в виде байтов) и SAVEALL (запись одновременно всех процедур и данных). Правила применения их аналогичны описанным. За исключением того, что примитив

## SAVEALL "Имя файла

не содержит указаний на то, что записывается (в этом нет нужды, так как он ведет к записи всех процедур и определенных переменных). Примитив SAVEALL в ряде случаев очень удобен. С его помощью можно записать текущее состояние ПЭВМ, даже если вы прервали работу с Лого-программой где-то на середине.

Используется также примитив

## SAVESCR "Имя файла

Он обеспечивает запись на ленту экранного файла, т.е. всего изображения, которое сформировано Лого-программой в графическом режиме (за исключением служебных строк).

Считывание программ и данных с кассетного магнитофона осуществляется командой

```
LOAD "Имя файла" "Имя процедуры"
LOAD "Имя файла" [Список имен процедур]
```

Примеры:

```
LOAD "BOX" "BOX4
```

считывается процедура BOX4 в виде файла BOX;

```
LOAD "BOXS [BOX NBOX]
```

– считываются процедуры BOX и NBOX в виде файла BOXS.

Встречаются также примитивы LOADD, BLOAD, LOADSCR.

Смысл их тот же, что указан для процедур записи. Вместо примитива LOADALL обычно используется примитив

```
LOAD "Имя файла
```

без указания имени или списка процедур. В этом случае ПЭВМ считывает все процедуры и значения переменных, которые были записаны командой SAVEALL.

При считывании файлов с магнитной ленты ПЭВМ сообщает пользователю (надписями на экране дисплея): имя файла, тип файла, имя вводимых с файла процедур.

У ZX-Spectrum, помимо записи файлов на обычный кассетный магнитофон возможна запись на специальный магнитофон-микродрайв, обладающий ускоренным поиском файлов и высокой скоростью их записи и считывания. Для включения микродрайва используется команда SETDRIVE. По команде CATALOG выводится сообщение об имени всех процедур, записанных на данную кассету. Команда ERASE-FILE стирает файл с указанным далее именем или списком имен.

### 4.3. Пауза, звук и управление роботом

В Лого IBM PC пауза в вычислениях задается примитивом

PAUSE

а примитив

CO

(от continue – продолжение) сбрасывает паузу и вызывает продолжение работы.

В других версиях Лого для создания паузы на заданное время используется примитив

WAIT N

который создает паузу в N долей секунды (1/50 или 1/60). Заметим, что для создания паузы можно использовать и примитив цикла REPEAT, заставив ПЭВМ N раз выполнять какое-либо действие. Сколько долей секунды занимает каждый цикл, легко проверить экспериментально.

Примитив TONE (тон) в версии Лого IBM PC создает звуковой сигнал с частотой F длительностью T

TONE F T

Длительность T задается в единицах машинного времени ( $\approx 0,05$  с).

У ряда версий Лого для задания звука используется примитив SOUND

Примитив

SOUND Длительность Высота

создает звук заданной длительности (в секундах) и высоты.

К некоторым ПЭВМ, например ZX-Spectrum, можно подключать робот, который будет дублировать перемещение Лого черепашки. Запуск внешнего робота выполняется командой

STARTROBOT

При этом Лого ищет в памяти двоичный файл управления роботом. Если он не найден, Лого пытается загрузить файл с внешнего магнитного накопителя, например кассетного магнитофона (см. выше).

Команда

STOPROBOT

отменяет действие предшествующей команды и снимает управление с внешнего робота.

#### 4.4. Операции с памятью

В Лого-системе ОЗУ ПЭВМ хранит (помимо интерпретатора, защищенного от стирания) введенные пользователем процедуры (программы) и значения обозначенных переменных. В ходе отладки программ необходимо выводить содержимое памяти на экран дисплея (на печать). Для этого служит ряд команд, указанных ниже.

Команда

PO "Имя процедуры  
или

PO [Список имен процедур]

обеспечивает вывод на индикацию (печать) листинга одной процедуры или ряда процедур, указанных в списке. Форма листинга много раз приводилась, и не стоит сопровождать эту команду еще одним таким примером. Отметим лишь, что процедуры разделяются пустой строкой. Действие этой команды также не раз представлялось.

Команда

PONS

выдает имена и значения всех переменных:

```
?MAKE "A 123
?MAKE "B [1 2 3 4]
?MAKE "C [ONE TWO FREE]
?PONS
C IS [ONE TWO FREE]
B IS [1 2 3 4]
A IS 123
?
```

Команда

POPS

выводит на индикацию (печать) листинги всех процедур, разделяя их пустой строкой (см. пример в §3.4).

Команда

POALL

выдает вначале листинги всех процедур, а затем имена и значения всех переменных (в форме, аналогичной приведенной для команды PONS).

Отметим, что эти листинги получаются в режиме непосредственного счета. При этом вывод завершается появлением знака ? и ПЭВМ готова к дальнейшей работе.

Команда

POTS

обеспечивает вывод имен всех имеющихся в ОЗУ процедур с указанием их входных данных, например в виде:

```
?POTS
TO SENTEN :NOUNS :RHYMES
TO ORDER.NOUNS
TO PICK.RHYME
TO POETRY
TO ADD.RHYME
TO ADD.NOUNS
TO ASK
TO INSTRUС
TO INIT.NOUNS.RHYMES
TO START
?
```

Процедуры и значения переменных могут быть уничтожены с помощью ряда команд. Это необходимо для очистки памяти от ставших ненужными процедур и данных или для высвобождения памяти под ввод процедур и данных.

Команда

ERALL

(от слов erase all – стереть все) обеспечивает полную очистку памяти от всех процедур и данных. Они уничтожаются. Однако программа-интерпретатор Лого сохраняется.

Избирательное стирание процедуры или процедур из списка обеспечивает команда

```
ERASE "Имя процедуры
ERASE [Список имен процедур]
```

Слово ERASE может заменяться сокращением ER.

Команда

ERN "Имя переменной

или

ERN [Список имен переменных]

стирает переменную с указанным именем (например ERN "А стирает переменную :А) или переменные, указанные в списке (например, ERN [B C D] стирает переменные :B :C :D). Не следует путать стирание переменной с ее обнулением. Стирание означает, что переменная перестает существовать как объект.

Команда

ERPS

стирает все процедуры (но сохраняет все переменные), а команда

ERNS

стирает все переменные (но сохраняет все процедуры).

Емкость свободной памяти в Лого принято указывать в особых единицах – узлах (nodes)<sup>1</sup>. Она распределяется между переменными и вводимыми процедурами. Часть памяти расходуется на временное хранение значений переменных, листингов процедур в редакторе и другие системные нужды. Поэтому возникает необходимость контролировать объем свободной памяти. Контроль осуществляется примитивом

NODES

который выдает число свободных узлов. Очистка памяти от "мусора" с объединением всех введенных процедур осуществляется с помощью примитива

RECYCLE

При этом процедуры и значения глобальных переменных сохраняются.

Перед введением команд в машинных кодах (байтах) в ОЗУ ПЭВМ необходимо зарезервировать место для них. Примитив

.RESERVE N

обеспечивает резервирование места под N байтов. Они располагаются в ОЗУ с начального адреса  $A_0$  и кончаются конечным  $A_0 + N$ . В действительности фиксируется конечный адрес (большой), причем он зависит от типа ПЭВМ и организации структуры памяти.

Примитив

.RESERVED

возвращает список  $[A_0 \ A_0 + N]$ , в котором  $A_0$  указывает начальный, а

---

<sup>1</sup> Один узел равен 5 байт.

$A_0 + N$  – конечный адрес ячеек ОЗУ, зарезервированных под машинные коды.

Пример:

.RESERVE 1000	Резервируется 1000 ячеек ОЗУ.
PR .RESERVED	Распечатка списка зарезервированных
64024 65024 [ $A_0$ $A_0 + N$ ]	ячеек ОЗУ.

Иногда в Лого-программы полезно включать фрагменты программ, записанные в десятичных кодах. Они заносятся прямо в ОЗУ ПЭВМ обычно в виде десятичных кодов. Для этого служит команда

.DEPOSIT Адрес Код

Считывание содержимого ячейки с заданным адресом выполняет функция

.EXAMINE Адрес

Пример:

```
? .DEPOSIT 65000 56
? .EXAMINE 65000
56
?
```

При работе с Лого возможно выполнение программ в машинных кодах, хранящихся в ОЗУ. Для этого используется команда

CALL Адрес

Следует предостеречь пользователя от необдуманного применения команд .DEPOSIT и CALL. Оно может ввести систему с Лого в неопределенное состояние. Например, ПЭВМ может "зависнуть", т.е. отказаться от выполнения любой операции.

Система Лого, как и любая другая система программирования интерпретирующего типа, является довольно медленной. Введение команд в машинных кодах позволяет заметно повысить скорость счета – практически в десятки раз. К сожалению, за это приходится платить дорогой ценой – потерей наглядности программ и сложностью программирования в машинных кодах, которое доступно пользователям-профессионалам.

#### *4.5. Определение и переопределение процедур*

Как отмечалось, процедуры пользователя создаются с помощью примитивов TO и END. Однако часто возникает необходимость в создании копий процедур с новыми именами, переименовании про-

цедур, выявлении тождественности их имен с именами примитивов и т.д.

Примитив копирования

COPYDEF "Новое имя "Старое имя

создает процедуру с новым именем, в остальном тождественную процедуре со старым именем. Если стереть процедуру со старым именем, можно таким образом переименовать процедуру.

Так, если есть процедура FACTORIAL, то команды

```
COPYDEF "FAC "FACTORIAL
ER      "FACTORIAL
```

заменяют в этой процедуре имя FACTORIAL на имя FAC (более короткое).

Примитив

DEFINE "Имя [Текст процедуры]

создает в режиме непосредственного счета (а не в редакторе) процедуру с указанным именем на основе ее текста. При этом примитивы TO и END вставляются автоматически, а после ввода команды появляется знак ? без сообщения об определении процедуры.

Пример: команда

```
DEFINE "B [[S] REPEAT 4 [FD :S RT 90]]
```

создает процедуру с именем B, строящую квадрат с заданной стороной (значением переменной S).

Если затем дать команды ? B 40 <ENTER>, будет построен квадрат со стороной в 40 пикселей. Команда ? TS очищает экран, а команда ? PO "B выводит листинг полученной только что процедуры в его естественном виде:

```
TO B :S
REPEAT 4 [FD :S RT 90]
END
```

Текст любой процедуры (без объявления TO, имени и конца END) можно сформировать в виде текста с помощью примитива-функции

TEXT "Имя процедуры

Например, выполнение команды

PR TEXT "B

выдает текст только что созданной процедуры B в виде

[S] [REPEAT 4 [FD :X RT 90]]

Имена процедур и переменных в Лого не должны совпадать с названиями примитивов. В сомнительных случаях нужно выявлять тождественность имен. Поскольку различные версии Лого могут иметь несколько различных набор примитивов, полезной является команда

## .CONTENTS

Она дает список всех примитивов, присущих данной версии Лого. (У ряда версий Лого ту же функцию выполняет команда .PRIMITIVES.)

Другая функция

PRIMITIVE "Имя

позволяет уточнить, является ли указанное имя примитивом или нет. Если да, то функция возвращает предикат TRUE, если нет — предикат FALSE.

Примеры:

PR PRIMITIVE "END

TRUE

PR PRIMITIVE "BOX

FALSE

Следовательно, слово END нельзя использовать в качестве имени (это примитив), а слово BOX можно (это не примитив).

Иногда полезно объединение (пакетирование) нескольких процедур в одну. Для этого служит (не у всех версий Лого) команда

PACKAGE "Имя пакета [Список процедур]

Пример: если есть процедуры А и В, то команда

PACKAGE "С [А В]

создает процедуру С в виде объединенных в пакет процедур А и В.

Команда

PKGALL "Имя пакета

объединяет в пакет с указанным именем все процедуры, имеющиеся в памяти ПЭВМ.

Примитив

BURY "Имя пакета

стирает пакет с указанным именем, т.е. уничтожает все процедуры пакета и его имя. Однако в ОЗУ ПЭВМ пакет все же сохраняется. Он

может быть вызван и восстановлен командой

UNBURY "Имя пакета

После этого пакет можно использовать обычным образом.

Примитив

.CONTENTS

в версии Лого ZX-Spectrum возвращает текст в виде листинга всех определенных объектов, исключая переменные, процедуры и примитивы. Таким образом, его действие совсем иное, чем в версии Лого IBM PC.

Пример: пусть в ПЭВМ с очищенной памятью введена процедура

```
TO SF
PR "START
MAKE "X "FINISH
PR :X
END
```

Дав команду

PR .CONTENTS

получаем текст FINISH X START SF

Из него исключены примитивы TO, PR, MAKE и END и переменная :X. Обратите внимание, что определенные объекты перечислены в обратном порядке (от конца до начала) и что функция этого примитива резко отлична от описанной для версии Лого IBM PC.

#### 4.6. Работа с редактором

Язык Лого представляет пользователю обширные возможности по редактированию данных и текстов процедур, вводимых в ПЭВМ, а также по управлению ПЭВМ.

В режиме непосредственных вычислений действует редактор текущей строки. Это означает, что до нажатия клавиши перевода строки (ENTER) имеется возможность изменения ее текста путем забоя или исправления неправильно введенных символов. Обычно для этого используется возможность перемещения вдоль строки курсора. Нажатие клавиши перевода строки вводит измененный текст строки и ведет к ее исполнению.

В IBM PC имеется ряд специальных функциональных клавиш с надписями вида FN, где N = 1, 2, ..., 10. С их помощью можно управлять различными режимами экрана. Так, нажатие клавиши F1 устанавливает текстовый режим (TEXTSCREEN), F4 дает полную очистку экрана

(FULSCREEN), F2 дает смешанный режим (MIXEDSCREEN), F3 устанавливает 80 знаков в текстовой строке и т.д. В зависимости от реализации Лого эти функции могут несколько меняться, однако, воспользовавшись подсказками ПЭВМ, их легко уточнить.

Особенно удобен для редактирования процедур экранный редактор Лого. Он вызывается (в режиме непосредственных вычислений) вводом команды

EDIT

или

ED

При этом ПЭВМ переходит в режим индикации текстовой информации. Признаком перехода в режим редактирования являются исчезновение знаков ? и > и появление в верхнем левом углу курсора. При указанном вводе редактора текст соответствует последнему взаимодействию пользователя с ПЭВМ (например, текст последней процедуры, бывшей ранее на редактировании). Если взаимодействия не было, выводится слово TO – приглашение к заданию процедур.

В редакторе можно записывать и редактировать как одну, так и несколько процедур. Редактирование сводится к перемещению курсора специальными клавишами (например, ←, ↑, ↓ и →) влево, вверх, вниз и вправо. При этом на место курсора можно вставить любой новый знак. С помощью клавиши забоя можно устранить знак, помеченный курсором. Редактор обычно имеет еще две возможности: выход из него без внесения исправлений в текст процедуры и выход с внесением исправлений.

В IBM PC для облегчения работы с редактором используются подсказки, выводимые в нижние служебные строки экрана. У других ПЭВМ функции редактора подробно описываются в технической документации. Рассмотрим их для Apple-II и ZX-Spectrum.

У Apple-II для операций редактирования и управления используется одновременное нажатие клавиши CTRL и некоторых других клавиш:

- |          |   |
|----------|---|
| CTRL – G | – немедленно останавливает редактирование или выполнение процедуры, дает возврат в Лого и печать приглашения – знака (?), |
| CTRL – W | – временно останавливает работу (нажатие любой клавиши ведет к продолжению работы),                                       |
| CTRL – Z | – приостанавливает выполнение вычислений, действует как команда RAUSE,  |

CTRL – S	– освобождает верхние 20 строк под графику режим SPLITSCREEN),
CTRL – L	– освобождает весь экран под графику (режим FULLSCREEN),
CTRL – T	– освобождает весь экран под текст (режим TEXTSCREEN),
CTRL – C	– включает режим редактирования,
CTRL – M	– действует аналогично клавише RETURN (возврат строки).

Ниже отмечены функции редактирования ПЭВМ Apple-II, связанные с курсором:

CTRL – A	– устанавливает курсор в начало линии,
CTRL – B	– возвращает курсор на один знак,
CTRL – I	– стирает знак, отмеченный курсором,
CTRL – E	– устанавливает курсор в конец линии,
CTRL – F	– перемещает курсор вперед на один знак,
CTRL – H	– действует аналогично клавише ← (левая стрелка),
CTRL – K	– стирает помеченный курсором знак,
CTRL – N	– перемещает курсор вниз на одну строку,
CTRL – O	– открывает новую строку с позиции курсора,
CTRL – P	– перемещает курсор вверх на одну строку,
CTRL – Q	– позволяет включать новые знаки в строку,
CTRL – V	– действует аналогично клавише → (правая стрелка),
CTRL – V	– перемещает курсор вниз текста одной страницы,
CTRL – Y	– вставляет новый знак на место стираемого командой CTRL – K,
ESC >	– помещает курсор в конец текстовой страницы (буфера) редактора,
ESC <	– помещает курсор в начало текстовой страницы редактора.

У ZX-Spectrum ввод операций редактирования связан с использованием префиксных клавиш CAPS SHIFT (сокращенно CAPS) и SYMBOL SHIFT (сокращенно SYS). Операции вводятся нажатием одной из этих клавиш с другой клавишей. Одновременное нажатие клавиш CAPS и SYS устанавливает моду (MODE) – режим работы клавиатуры.

CAPS - 5	- перемещает курсор на один знак влево,
CAPS - 6	- перемещает курсор на одну строку вниз,
CAPS - 7	- перемещает курсор на одну строку вверх,
CAPS - 8	- перемещает курсор на один знак вправо,
CAPS - 0	- удаляет знак слева от курсора,
E MODE CAPS - 5	- передвигает курсор к началу строки,
E MODE CAPS - 6	- перемещает курсор к концу экрана,
E MODE CAPS - 7	- перемещает курсор к началу экрана,
E MODE CAPS - 8	- перемещает курсор к концу строки,
E MODE B	- перемещает курсор к началу текста,
E MODE E	- перемещает курсор к концу текста,
E MODE Y	- удаляет строку, начиная с позиции курсора и далее с запоминанием ее,
E MODE R	- вводит хранимую строку, начиная с позиции курсора,
E MODE S	- останавливает "прокрутку" текста (нажатие любой клавиши восстанавливает ее),
E MODE P	- передвигает курсор на предыдущую страницу,
E MODE N	- передвигает курсор на следующую страницу,
E MODE C	- выводит систему Лого из редактирования с определением всех измененных процедур и переходом в режим непосредственных вычислений,
CAPS BREAK/SPACE	- выдает отказ от редактирования с выходом из редактора в режим непосредственных вычислений.

Во всех версиях Лого на редактирование можно вызвать заранее объявленную процедуру, используя команду

ED "Имя процедуры

или ряд процедур, указанных в списке

ED [Список имен процедур]

Примитив

EDNS

служит для редактирования переменных – изменения их имен и значений.

Например, зададим переменные

```
Make "A "OWERT
```

```
MAKE "C 123
```

Теперь введем команду ENDS. Получим сообщение

```
MAKE "C 123
```

```
MAKE "A "QWERT
```

Используя средства редактирования, изменим текст:

```
MAKE "C 123
```

```
MAKE "A "START
```

Выведем ПЭВМ из редактора с внесением исправлений. Если теперь ввести команду вывода переменных PONS, получим:

```
MAKE "B 123
```

```
MAKE "C 123
```

```
MAKE "A "START
```

Из этого примера видно, что редактор сохраняет имена и значения тех переменных, у которых изменено имя – в данном случае это переменная C.

## **5. Расширение Лого и реализация численных методов**

### *5.1. Вычисление дополнительных функций*

Как отмечалось, Лого – язык расширяемый, и в него можно включить дополнительные команды в виде внешних процедур. Ниже описан ряд таких процедур, расширяющих вычислительные возможности Лого (версия IBM PC [28, 30]).

Процедура SUMLIST с входным параметром – листом числовых данных обеспечивает вычисление суммы всех чисел в листе:

```
?PO "SUMLIST
```

```
TO SUMLIST :N
```

```
IF (BL :N) = [] [OP FIRST :N]
```

```
OP (FIRST :N) + (SUMLIST BF :N)
```

```
END
```

```
?PR SUMLIST [2 3 4 5]
```

```
14
```

```
?
```

Это рекурсивная процедура. При каждом обращении к самой себе процедура выделяет первый элемент листа и суммирует с текущей суммой. При этом первый элемент листа удаляется. Сумма выдается, как только лист становится пустым, что и служит сигналом к окончанию работы с процедурой.

Процедура AVR с входным параметром – листом числовых данных – обеспечивает вычисление среднего значения всех данных:

```
?PO "AVR
TO AVR :N
OP (SUMLIST :N) / COUNT :N
END
```

```
?PR AVR [2 3 4 5]
3.5
?
```

Это подчиненная процедура: в ней используется определенная ранее процедура .SUMLIST. Полученная сумма делится на число данных в листе, выполняемое командой COUNT :N.

Процедура ABS с входным параметром X преобразует X любого знака в абсолютное значение X:

```
?PO "ABS
TO ABS :X
IF :X < 0 [OP -:X] [OP :X]
END
```

```
?PR ABS 123
123
?PR ABS -123
123
?
```

Процедура просто выдает X, если  $X \geq 0$ , и меняет знак X, если  $X < 0$ .

Процедура MIN с двумя входными параметрами X и Y (числами) оставляет меньшее из этих двух чисел:

```
?PO "MIN
TO MIN :X :Y
IF :X < :Y [OP :X]
OP :Y
END
```

```
?PR MIN 123 456
123
?
```

Процедура MAX оставляет максимальное из двух чисел:

```
?PO "MAX
TO MAX :X :Y
IF :X > :Y [OP :X]
OP :Y
END
```

```
?PR MAX 123 456
456
?
```

Работа процедур MIN и MAX достаточно очевидна.

Процедура LOG с входным параметром X вычисляет десятичный логарифм числа X по формуле  $\log X = \ln X / \ln 10$ :

```
?PO "LOG
TO LOG :X
IF :X < 0 [OP [Error: argument X < 0]]
OP ( LN :X ) / ( LN 10 )
END
```

```
?PR LOG 1000
3
?PR LOG -2
Error: argument X < 0
?
```

Поскольку отрицательные числа не имеют логарифма, при  $X < 0$  задан вывод краткого напоминания об этом:

Error :argument X < 0 (Ошибка : аргумент X < 0).

Функция TAN описывалась в гл. 1. Приводим ее вновь для полноты библиотеки процедур. Эта функция вычисляет  $\operatorname{tg} x = (\sin x) / (\cos x)$ :

```
?PO "TAN
TO TAN :X
OP ( SIN :X ) / ( COS :X )
END
```

```
?PR TAN 30
0.5773502692
?
```

Лого IBM PC содержит функцию ARCTAN, но не имеет функций ARCSIN и ARCCOS. Процедура ARCSIN вычисляет арксинус по формуле

$$\arcsin x = \arctg(x/\sqrt{1-x^2}),$$

если  $x < 1$ . Иначе выдается сообщение

Error: argument X > = 1 (Ошибка: аргумент X > = 1).

Листинг процедуры и контрольный пример:

```
?PO "ARCSIN
TO ARCSIN :X
IF ABS :X < 1 [OP ARCTAN (:X / (SQRT (1 - :X * :X)))
OP [Error: argument X > = 1] STOP
END
```

```
?PR ARCSIN .5
30
?
```

Процедура ARCCOS вычисляет арккосинус X по формуле

$$\arccos x = 90 - \arctg(x/\sqrt{1-x^2}),$$

если  $x < 1$ :

```
?PO "ARCCOS
TO ARCCOS :X
IF ABS :X < 1 [OP 90 - ARCTAN (:X / SQRT (1 - :X * :X))]
OP [Error: argument X > = 1]
END
```

```
?PR ARCCOS .5
60
?PR ARCCOS 2
Error: argument X > = 1
?
```

Гиперболический синус вычисляется по формулам

$$E = \exp(x) \text{ и } \operatorname{sh} x = (E - 1/E)/2$$

с помощью процедуры HSIN:

```
?PO "HSIN
TO HSIN :X
```

```

LOCAL "E MAKE "E EXP :X
OP (:E - 1 / :E) / 2
END

```

```

?PR HSIN 1
1.175201194
?

```

В процедуре используется локальная переменная  $E = \exp(x)$ , введенная для ускорения вычислений. При вычислении  $\operatorname{sh}x$  по обычной формуле  $\operatorname{sh}x = (e^x - 1/e^x)/2$  пришлось бы вычислять  $e^x$  дважды.

Гиперболический косинус вычисляется по формулам

$$E = \exp(x) \text{ и } \operatorname{ch}(x) = (E + 1/E)/2$$

с помощью процедуры HCOS:

```

?PO "HCOS
TO HCOS :X
LOCAL "E MAKE "E EXP :X
OP (:E + 1 / :E) / 2
END

```

```

?PR HCOS 1
1.543080635
?

```

Гиперболический тангенс вычисляется по формулам

$$E = \exp(x) \text{ и } \operatorname{th}x = (E - 1/E)/(E + 1/E)$$

с помощью процедуры HTAN:

```

?PO "HTAN
TO HTAN :X
LOCAL "E MAKE "E EXP :X
OP (:E - 1 / :E) / (:E + 1 / :E)
END

```

```

?PR HTAN 1
0.761594156
?

```

Для вычисления обратного гиперболического синуса используется формула

$$\operatorname{arch} x = \ln(x + \sqrt{1 + x^2})$$

и процедура ANSIN:

```
?PO "AHSIN
TO AHSIN :X
OP LN (:X + SQRT (1 + :X * :X))
END
```

```
?PR AHSIN 1
0.8813735868
?
```

Обратный гиперболический косинус вычисляется по формуле

$$\operatorname{arch} x = \ln(x + \sqrt{x^2 - 1}), \text{ если } x > 1.$$

Для этого используется процедура AHCOS:

```
?PO "AHCOS
TO AHCOS :X
IF :X > 1 [OP LN (:X + SQRT (:X * :X - 1))]
OP [Error: argument X < = 1]
END
```

```
?PR AHCOS 2
1.316957897
?PR AHCOS .5
Error: argument X < = 1
?
```

Для вычисления обратного гиперболического тангенса используется формула

$$\operatorname{arth} x = \ln \sqrt{(x+1)(1-x)},$$

справедливая при  $-1 < x < 1$ , и процедура АНТАН:

```
?PO "ANTAN
TO ANTAN :X
IF :X > 1 [OP [Error: argument X > 1]]
IF :X < -1 [OP [Error: argument X < -1]]
OP LN SQRT ((:X + 1) / (1 - :X))
END
```

```
?PR ANTAN .5
```

```

0.5493061446
?PR AHTAN 2
Error: argument X > 1
?PR AHTAN -2
Error: argument X < -1
?

```

Язык Бейсик содержит специальный оператор задания функций пользователя DEFFN и оператор исполнения такой функции FN. Можно создать процедуры с этими именами, используя средства Лого:

```

?PO [DEFFN FN]
TO DEFFN
TYPE [F ( :X ) =] MAKE "F RL
END

```

```

TO FN :X
OP RUN :F
END

```

```

?DEFFN
F ( :X ) =(SIN :X) / :X
?PR FN 10
1.736481777E-0002
?PR FN 30
1.666666667E-0002
?PR (SIN 10) / 10
1.736481777E-0002
?

```

Процедура DEFFN печатает запрос

$F(X) =$

и ожидает ввода функции, записанной по синтаксическим правилам Лого. Введенный лист функции присваивается как значение переменной F.

Процедура FN с входным параметром X реализует операции OP RUN :F, т.е. вывод результата исполнения листа инструкций – значения переменной F. В контрольном примере показано задание функции  $F(X) = (\sin X)/X$  и вычисление ее значений.

Заметим, что этот очень полезный прием (в виде ввода функции по запросу с пульта, отсутствующий в Бейсике IBM PC) позволяет строить универсальные программы, например вычисления определенных

интегралов с вводимым подынтегральным выражением, графиков любых функций и т.д. Это повышает диалоговые возможности программ.

Еще одна полезная процедура CHARTEST

```
?PO "CHARTEST
TO CHARTEST
MAKE "I 0
REPEAT 256 [TYPE :I TYPE CHAR :I TYPE CHAR 32 MAKE "I :I +
END
```

выводит коды и соответствующие им символы клавиатуры (в стандарте ASCII).

## 5.2. Преобразование чисел по основанию

Лого использует только обычные десятичные числа. Однако с помощью нескольких простых процедур в нем можно обеспечить преобразование чисел с различным основанием, в том числе шестнадцатеричных чисел с особой формой представления (см. ниже). Листинги этих процедур:

```
?PO [HEXTODEC DECTOHEX N.TO.C C.TO.N AN
YBASE.TO.DEC DEC.TO.ANYBASE CONVERT]
TO HEXTODEC :N
OP CONVERT :N 16 10
END
```

```
TO DECTOHEX :N
OP CONVERT :N 10 16
END
```

```
TO N.TO.C :N
IF :N < 10 [OP :N]
OP CHAR 55 + :N
END
```

```
TO C.TO.N :N
IF NUMBERP :N [OP :N]
OP (ASCII :N) - 55
END
```

```
TO ANYBASE.TO.DEC :N :BASE :POWER
IF EMPTY P :N [OP 0]
OP ( :POWER * C.TO.N LAST :N ) + ANYBAS
```

```
E.TO.DEC BL :N :BASE :POWER * :BASE
END
```

```
TO DEC.TO.ANYBASE :N :BASE
IF :N < :BASE [OP N.TO.C :N]
OP WORD DEC.TO.ANYBASE INT QUOTIENT :N
:BASE :BASE N.TO.C REMAINDER :N :BASE
END
```

```
TO CONVERT :N :FRBASE :TOBASE
OP DEC.TO.ANYBASE ANYBASE.TO.DEC :N :FR
BASE 1 :TOBASE
END
```

Функции этих процедур следующие:

HEXTODEC :N	– преобразует шестнадцатеричное число в десятичное,
DECTOHEX :N	– преобразует десятичное число в шестнадцатеричное,
N . TO . C :N	– преобразует число N в код C,
C . TO . N :C	– преобразует код C в число N,
ANYBASE . TO . DES :N	– преобразует число с любым обоснованием в десятичное число,
DES . TO . ANYBASE :N	– преобразует десятичное число в число с любым основанием,
CONVERT :N :FBASE :TOBASE	– преобразует число N с основанием FBASE в число с основанием TOBASE.

Заинтересованный читатель уже должен быть готов к анализу этих сравнительно простых процедур. Чтобы упростить их понимание отметим, что разряд шестнадцатеричного числа HEX имеет следующее представление в десятичной форме DEC:

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Машинные коды цифр 0 – 9 есть числа 48 – 57, а коды букв A – F есть числа 65 – 70.

Примеры на преобразование чисел:

```
?PR HEXTODEC 64
100
?PR HEXTODEC "ABCDEF
11259375
```

```

?PR DECTONEX 100
64
?PR DECTONEX 11259375
ABCDEF
?PR CONVERT 1000 10 2
1111101000
?PR CONVERT 1111101000 2 10
1000
?
```

Первые четыре примера даны на взаимное преобразование шестнадцатеричных и десятичных чисел. Два последних примера иллюстрируют преобразование десятичного числа 1000 в двоичное число и наоборот. Шестнадцатеричными числами часто указывают адреса ячеек ОЗУ и содержащиеся в них коды.

### 5.3. Вычисление значений факториала и полинома

Процедуры вычисления факториала FACTORIAL и значений полинома POL наглядно иллюстрируют полезность рекурсии и применения примитива организации цикла REPEAT с заданным числом повторений.

Факториал  $x!$  по определению есть

$$\begin{aligned}
 x! &= 1 \text{ при } x = 0, \\
 x! &= 1 \cdot 2 \cdot \dots \cdot x \text{ при целых } x.
 \end{aligned}
 \tag{5.1}$$

Не целые числа и числа отрицательные факториала не имеют.

С вычислением факториала связаны немалые трудности. Значения  $x!$  весьма быстро возрастают с ростом  $x$ . Например, на Бейсике IBM PC нельзя вычислять  $x!$  при  $x > 34$ , так как уже при  $x = 35$   $x! \approx 1 \cdot 10^{40}$  и выходит за допустимые верхние пределы для чисел. Вторая проблема связана с тем, что при многих приложениях теории чисел факториал надо вычислять со всеми верными знаками. Для этого на Бейсике разработаны довольно сложные и медленно считающие программы, основанные на замене арифметических операций символьными.

Лого IBM PC легко обходит все эти трудности. Рекурсивная процедура FACTORIAL вычисляет  $x!$  по формулам (5.1) и оценивает допустимые для вычисления факториала типы чисел:

```

?PO "FACTORIAL
TO FACTORIAL :X
IF NOT (INT :X) = :X [OP [Must be aninteger] STOP]
IF :X < 0 [OP [Negative don't work]]
IF MEMBERP :X [0 1] [OP 1]
```

```
OP :X * FACTORIAL :X - 1
END
```

```
?PR FACTORIAL 10
3628800
?PR FACTORIAL 40
8.159152844E+0047
?SETPRECISION 50
?PR FACTORIAL 40
8159152832478977343456112695961158942720000000000
?
```

Под листингом процедуры даны контрольные примеры на вычисление 10! и 40!. Видно, что при вычислении 40! Лого стал выдавать результат в экспоненциальной форме вывода числа (с порядком +47). При этом точны лишь 10 знаков результата. В последнем примере команда

```
SETPRECISION 50
```

позволяет получить до 50 верных чисел результата. При этом значение 40! получено со всеми верными 48 знаками!

Значение полинома (степенного многочлена)

$$P(x) = A_0 + A_1 x + A_2 x^2 + \dots + A_{N-1} x^{N-1} + A_N x^N$$

удобно вычислять по так называемой схеме Горнера:

$$P(x) = (\dots (A_N x + A_{N-1}) x + A_{N-2}) x + \dots + A_1) x + A_0.$$

Эта схема заменяет сложные операции возведения  $x$  в степень более простыми операциями умножения и сложения.

Из схемы Горнера видно, что коэффициенты полинома  $A_0, A_1, \dots, A_N$  используются поодиночке, начиная с конца. Это позволяет наметить следующий алгоритм вычислений. Зададим  $A_0, A_1, \dots, A_N$  как объекты списка – значения переменной  $A$ . Введем вспомогательную переменную  $AP$  со значением  $A$ , зададим  $P = 0$  и найдем  $AP$  как число элементов списка за вычетом 1. Организовав цикл из  $N$  вычислений, реализуем схему Горнера: начиная с  $I = N$  вычисляем  $A_I N$  и прибавляем  $A_{I-1}$ , суммируем результат со значением переменной  $P$ . В каждом цикле используем последний элемент списка  $AP$  и затем удаляем его. По выходе из цикла прибавляем к  $P$  оставшийся объект (это  $A_0$ ) и получаем  $P(x)$ .

Этот алгоритм и реализован в процедуре POL:

```
?PO "POL
TO POL :X
```

```

MAKE "AP :A
MAKE "P 0
MAKE "N (COUNT :A) - 1
REPEAT :N [MAKE "P (:P + LAST :AP) * :X MAKE "AP BL :AP]
OP :P + FIRST :AP
END

```

```

?MAKE "A [1 1 2 3 4]
?PR POL 1
11
?PR POL 2.45
203.6934
?

```

Перед использованием данной процедуры (см. контрольный пример) с помощью примитива MAKE переменной A присваивается значение списка коэффициентов полинома  $A_i$ . В данном случае вычисляется

$$P(x) = 1 + x + 2x^2 + 3x^3 + 4x^4$$

для  $x = 1$  и  $x = 2,45$ .

#### 5.4. Квадратичная интерполяция

Часто некоторая зависимость  $y(x)$  бывает представлена в виде таблицы значений  $x_i$  и  $y_i$  (вспомним, например, таблицы логарифмов). Эти значения называются узловыми, а точки  $(x_i, y_i)$  — узлами.

Интерполяцией функции, заданной узловыми значениями  $x_i$  и  $y_i$ , называют определение  $y$  по заданному  $x$  в любой точке между узлами (рис. 5.1). Для этого реальную зависимость  $y(x)$  заменяют полиномом. Ограничимся квадратичной интерполяцией, при которой  $y(x)$  полином второй степени. Такая интерполяция широко применяется при обработке данных таблиц.

Допустим, интерполируется зависимость  $y_i(x_i)$ . Она задается узлами: левым YL, центральным YO и правым YR. Им соответствуют

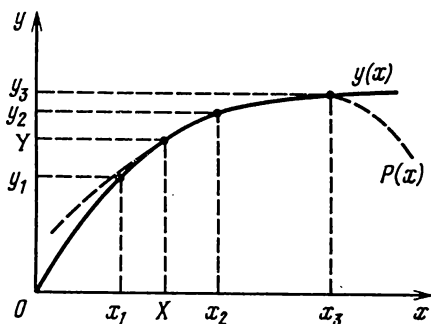


Рис. 5.1. Интерполяция функции, заданной узловыми точками  $(X_i, Y_i)$  с помощью полинома

абсциссы  $XO - H$ ,  $XO$  и  $XO + H$ , где  $H$  – шаг интерполяции. Интерполяция ведется с помощью выражений:

$$P = (X - XO)/H,$$

$$Y(X) = (P^2 - P)YL/2 + (1 - P^2)YO + (P^2 - P)YR/2,$$

где  $x = X - XO$  – значение абсциссы, для которого вычисляется  $Y(X)$ .

Квадратичная интерполяция осуществляется с помощью процедуры INT2, имеющей пять входных параметров: YL, YO, YR, XO и H. Листинг процедуры и примеры ее применения:

```
?PO "INT2
TO INT2 :YL :YO :YR :XO :H
TYPE [INPUT X =] MAKE "X RL
MAKE "P ((FIRST :X) - :XO) / :H
MAKE "P2 :P * :P
TYPE [Y ( X ) =]
PR. (:P2 - :P) * :YL / 2 + (1 - :P2) * :YO + (:P2 + :P) * :YR / 2
INT2 :YL :YO :YR :XO :H
END

?INT2 .22389 .11036 .0025 2.2 .2
INPUT X =2.1
Y ( X ) =0.16641625
INPUT X =2.3
Y ( X ) =0.05572125
INPUT X =
```

В этой процедуре реализован присущий Бейсику диалог. После указания имени процедуры INT2 и списка входных параметров процедура выдает запрос

INPUT X = (введите X =)

и ждет ввода  $x$ . После ввода  $x$  результат выдается с кратким комментарием и новым запросом, т.е. в виде

Y(X) = Число  
INPUT X =

Это позволяет неоднократно вычислять  $y(x)$  по вводимым  $x$ . Выйти из такого цикла можно с помощью клавиши Break (прерывание) или более "хитрым" способом – ввести вместо числа  $x$  букву (счет прерывается по ошибке).

## 5.5. Решение нелинейных уравнений

В математике имеется множество нелинейных уравнений вида

$$F(x) = 0, \tag{5.2}$$

которые невозможно решить в аналитическом виде. К ним относятся, например, трансцендентные уравнения. Для решения таких уравнений

приходится использовать различные численные методы [5 – 10].

Рассмотрим достаточно простой и в то же время весьма надежный численный метод решения уравнения (5.2) – метод деления пополам или метод дихотомии.

При этом методе задается отрезок  $[a, b]$ , так что  $x \in [a, b]$ . Границы  $a$  и  $b$  выбираются так, чтобы корень уравнения заведомо попал в этот отрезок (говорят, что корень локализуется). Метод дихотомии заключается в повторяющемся делении интервала  $[a, b]$  пополам. Если  $F(a) > 0$ , то метод реализуется следующим алгоритмом. Находим  $x = (a + b)/2$  и значение  $F(x)$ . Если  $F(x) > 0$ , полагаем  $a = x$ , если нет, то  $b = x$ . Проверяем условие  $(b - a) > \varepsilon$ , где  $\varepsilon$  – малое число, задающее погрешность результата. Если это условие выполняется, повторяем весь цикл вычислений. Если нет, выводим  $x$  как корень и  $F(x)$  для контроля на равенство 0.

Если в начале вычислений  $F(a) < 0$ , то нужно задать  $F(x) = -F(x)$ . Это позволит использовать без изменений описанный алгоритм. Метод дихотомии реализуется процедурой MD, входными параметрами которой являются пределы  $a = :A$  и  $b = :B$ , значение  $\varepsilon = :EPS$  и пусковой коэффициент  $k = :K$ .

Параметр  $:K$  задается при начальном обращении к процедуре равным 0. При этом вычисляется  $F(a)$  при  $x = a$ . Если  $F(a) > 0$ , то параметру  $:K$  внутри процедуры придается значение +1, а если  $F(a) < 0$ , то –1. В дальнейшем используется значение  $F(x)$ , умноженное на  $:K$ , что обеспечивает поиск корня в интервале  $[a, b]$  как при  $F(a) < 0$ , так и при  $F(a) > 0$ .

Для применения процедуры MD нужно задать процедуру F, вычисляющую и выводящую значение  $F(x)$ . Представим листинг процедур MD и F и результаты решения уравнения

$$F(x) = x^3 + x^2 - x - 1 = ((x + 1)x - 1)x - 1 :$$

```
?PO [MD F]
TO MD :A :B :EPS :K
IF :K = 0 [MAKE "X :A IF F > 0 [MAKE "K 1] [MAKE "K -1]]
MAKE "X (:A + :B) / 2
IF F * :K > 0 [MAKE "A :X] [MAKE "B :X]
IF :B - :A > :EPS [MD :A :B :EPS :K] [TYPE [X =] PR :X TYPE [F ( X ) =] PR F ST
OP]
END

TO F
OP ((:X + 1) * :X - 1) * :X - 1
END

?MD 0 3 1E-7 0
X =1.000000033
F ( X ) =0.000000132
?
```

Результатом работы процедуры MDM является приближенное значение  $\bar{x}$ , при котором  $F(\bar{x}) \approx 0$ . Реальное значение  $F(\bar{x})$  при  $x = \bar{x}$ , называемое невязкой, также выводится на индикацию.

Процедура MDM практически не содержит диалога. Приведем альтернативный вариант процедур с довольно развитым диалогом:

```
?PO [MDM MDM1]
TO MDM :K
IF :K = 0 [MDM1 IF (RUN :F) > 0 [MAKE "K 1] [MAKE "K -1]]
MAKE "X (:A + :B) / 2
IF :K * (RUN :F) > 0 [MAKE "A :X] [MAKE "B :X]
IF :B - :A > :EPS [MDM :K] [TYPE [X =] PR :X TYPE [F ( X ) =] PR RUN :F STOP]
END

TO MDM1
TYPE [INPUT A =] MAKE "A FIRST RL
TYPE [INPUT B =] MAKE "B FIRST RL
TYPE [INPUT EPS =] MAKE "EPS FIRST RL
TYPE [INPUT F ( :X ) =] MAKE "F RL
MAKE "X :A
END

?MDM 0
INPUT A =0
INPUT B =3
INPUT EPS =.1E-7
INPUT F ( :X ) =((:X + 1)*:X - 1)*:X - 1 ;
X =1.000000006
F ( X ) =0.000000024
?
```

Здесь процедура MDM содержит единственный входной параметр K, который задается равным 0. Если K = 0, то идет обращение к процедуре организации диалога MDM1. Она запрашивает значения  $A = a$ ,  $B = b$ ,  $EPS = \varepsilon$  и  $F(x)$ , т.е. просит ввести  $F(x)$  с пульта. Как отмечалось (§ 5.1), такой прием делает программы универсальными. После этого процедура MDM реализует описанный выше алгоритм метода дихотомии. Вычисление  $F(x)$  производится командой RUN :F, где значение :F есть лист инструкции, введенный по запросу F(:X) = .

Результаты вычислений по этой программе совпадают с полученными несколько ранее. Отметим, что если нужно вычислять несколько корней или несколько значений одного корня (для разных  $a$ ,  $b$  и  $\varepsilon$ ), то лучше пользоваться первым вариантом программы, так как в этом случае отпадает необходимость заново вводить выражение для  $F(x)$ .

Имеется множество других методов нахождения корня уравнения  $F(x) = 0$ , локализованного в начальном интервале  $[a, b]$ . Описание этих методов можно найти в [5 – 10]. Однако метод дихотомии является одним из наиболее простых и надежных. Он обеспечивает гарантированную сходимость к корню, локализованному в интервале  $[a, b]$ , и слабую зависимость сходимости от гладкости функции.

## 5.6. Численное дифференцирование и интегрирование

Численное дифференцирование (вычисление производных) обычно выполняется с помощью специальных формул, позволяющих вычислять производные функции  $y(x)$ , которая задана таблично или аналитически. Численному дифференцированию функций, приближенно заданных таблицей, присущи большие погрешности. Однако вычисление производных аналитически заданных гладких функций вполне возможно с практически приемлемой точностью. Одно из изящных применений численного дифференцирования заключается в построении графика производной  $y'(x)$  функции  $y(x)$ .

Численное дифференцирование аналитически заданной функции  $y(x)$  может выполняться по следующему алгоритму.

1. Задаемся значением  $x = :X$  и шагом  $h = :H$ .
2. Вычисляем значения  $(x - 2h)$ ,  $(x - h)$ ,  $(x + h)$  и  $(x + 2h)$ .
3. Для этих значений находим ординаты функции  $y(x - 2h)$ ,  $y(x - h)$ ,  $y(x + h)$  и  $y(x + 2h)$ .
4. Вычисляем производную

$$y'(x) = (y(x - 2h) - 8y(x - h) + 8y(x + h) - y(x + 2h)) / (12h).$$

Этот алгоритм реализован процедурой DIF с входными параметрами X и H. Эта процедура использует подпроцедуру Y с входным параметром X, которая вычисляет значение  $y(x)$  по заданному  $x$ . Листинги процедур и контрольный пример на вычисление производной функции  $y = \sqrt{x}$ :

```
?PO [DIF Y]
TO DIF :X :H
OP ((Y (:X - 2 * :H)) - 8 * (Y (:X - :H)) + 8 * (Y (:X + :H)) - Y (:X + 2 * :H)) / 12 / :H
END

TO Y :X
OP SQRT :X
END

?PR DIF 2 .1
0.3535529075
?PR DIF 2 .01
0.3535534
?
```

Численное интегрирование обычно сводится к вычислению значения определенного интеграла

$$I = \int_a^b f(x) dx.$$

Для этого могут использоваться различные методы [6 – 8].

Ограничимся вычислением  $I$  с помощью наиболее распространенного метода Симпсона, сочетающего умеренную сложность с довольно высокой точностью вычислений. При нем интервал  $[a, b]$  интегрирования разбивается на  $m$  четных участков, на каждом из которых  $f(x)$  аппроксимируется квадратичной параболой. Значение интеграла вычисляется по составной формуле Симпсона

$$I = \frac{h}{3} [f(a) + f(b) + 2 \sum_{n=1}^{\frac{m}{2}-1} f(a - 2nh) + 4 \sum_{n=1}^{\frac{m}{2}} f(a - h + 2nh)].$$

От  $m$  зависит точность интегрирования. Для гладких  $f(x)$  обычно  $m = 8 - 24$ . При этом удастся получить 6–7 верных знаков результата. Вычисление определенного интеграла аналитически заданной функции выполняется с помощью процедуры IS с тремя входными параметрами (пределами интегрирования  $a = :A$ ,  $b = :B$  и числом разбиений  $m = :M$ ). Совместно с процедурой IS должна использоваться процедура FI с входным параметром X, обеспечивающая вычисление подынтегральной функции  $f(x)$ . Приведем листинги этих процедур и контрольный пример на вычисление определенного интеграла

$$I = \int_0^1 \sqrt{2x+1} dx.$$

```
?PO [IS FI]
TO IS :A :B :M
MAKE "H (:B - :A) / :M
MAKE "M2 :M / 2
MAKE "N 0 MAKE "I1 0
REPEAT :M2 - 1 [MAKE "N :N + 1 MAKE "I1 :I1 + FI :A + 2 * :N * :H]
MAKE "I2 0 MAKE "N 0
REPEAT :M2 [MAKE "N :N + 1 MAKE "I2 :I2 + FI :A - :H + 2 * :N * :H]
OP :H * ((FI :A) + (FI :B) + 2 * :I1 + 4 * :I2) / 3
END

TO FI :X
OP SQRT (2 * :X + 1)
END

?PR IS 0 1 16
1.39871724
?
```

Отметим, что точное значение интеграла  $I = 1,3987175$ , т.е. верны 6 знаков результата после разделительной точки. Точность интегрирования растет с ростом  $m$ , но соответственно растет и время интегрирования.

## 5.7. Решение дифференциальных уравнений

### Обыкновенное дифференциальное уравнение

$$y' = dy/dx = f(x, y)$$

обычно решается методом Рунге–Кутты при заданных начальных значениях  $x = x_0$  и  $y = y_0 = y(x_0)$ . Решение заключается в нахождении зависимости  $y(x)$ .

При решении (5.3) методом Рунге – Кутта значения  $x$  меняются от значения  $x_0$  до конечного значения  $x_F$  с шагом  $h$ . При этом на каждом шаге находятся:

$$\begin{aligned} K_1 &= hf(x_i, y_i); & K_2 &= hf(x_i + h/2, y_i + K_1/2); \\ K_3 &= hf(x_i + h/2, y_i + K_2/2); & K_4 &= hf(x_i + h, y_i + K_3); \\ Y_{i+1} &= y_i + (K_1 + 2K_2 + 2K_3 + K_4)/6. \end{aligned}$$

Решение (5.3) по приведенным выше формулам выполняется с помощью процедуры RK с четырьмя входными параметрами:  $x_0 = :X$ ,  $y_0 = :Y$ ,  $h = :H$  и  $x_F = :XF$ . В этой процедуре используется подпроцедура FXY с входными параметрами  $x = :X$  и  $y = :Y$ , вычисляющая  $f(x, y)$  при заданных  $x$  и  $y$ . Листинги этих процедур и результаты решения дифференциального уравнения

$$y' = dy/dx = xy/2$$

представлены ниже:

```
?PO [RK FXY]
TO RK :X :Y :H :XF
MAKE "K1 :H * FXY :X :Y
MAKE "X :X + :H / 2
MAKE "K2 :H * FXY :X :Y + :K1 / 2
MAKE "K3 :H * FXY :X :Y + :K2 / 2
MAKE "X :X + :H / 2
MAKE "K4 :H * FXY :X :Y + :K3
MAKE "Y :Y + (:K1 + :K2 + :K2 + :K3 + :K3 + :K4) / 6
TYPE [X =] TYPE :X TYPE [; Y =] PR :Y
IF :X < :XF [RK :X :Y :H :XF]
END

TO FXY :X :Y
OP :X * :Y / 2
END

?RK 0 1 .2 1
X =0.2; Y =1.010050167
X =0.4; Y =1.04081077
X =0.6; Y =1.094174266
```

X = 0.8; Y = 1.173510814  
 X = 1; Y = 1.284025256  
 ?

Подобным образом могут решаться и системы из N дифференциальных уравнений. Однако из-за отсутствия у Лого массивов индексированных переменных решение систем уравнений при использовании Лого более сложно, чем на Бейсике или Фортране, хотя и вполне возможно.

### 5.8. Статистическая обработка массива данных

Пусть имеется массив данных – чисел  $x_i$ . Стандартный статистический анализ одномерного массива чисел (или сокращенно одномерная статистика) заключается в нахождении ряда статистических параметров, перечисленных ниже.

Начальные моменты K-го порядка ( $K = 1, 2, 3$  и  $4$ ):

$$SK = \frac{1}{N} \sum_{i=1}^N x_i^K.$$

С их помощью без обновления ввода  $x_i$  можно найти центральные моменты:

$$\begin{aligned} M_2 &= S_2 - S_1^2, \\ M_3 &= S_3 - 3 \cdot S_1 \cdot S_2 + 2 \cdot S_1^3, \\ M_4 &= S_4 - 4 \cdot S_1 \cdot S_3 + 6 \cdot S_1^2 \cdot S_2 - 3 \cdot S_1^4. \end{aligned}$$

Начальный момент  $S_1$  характеризует наиболее вероятное (среднее) значение  $x_i$ . Дисперсия  $D = M_2$  характеризует вероятную степень отклонения  $x_i$  от среднего значения (заметим, что  $\delta = \sqrt{D}$  есть средне-квадратическая погрешность). Так называемая несмещенная дисперсия для нормального распределения вычисляется по формуле

$$DO = M_2 \cdot N / (N - 1).$$

Коэффициент асимметрии

$$A = M_3 / (M_2)^{3/2}$$

характеризует характер скошенности функции плотности распределения  $P(x)$  [6 – 10]. А на остроту пика  $P(x)$ , в сравнении с нормальным распределением, указывает значение коэффициента эксцесса

$$E = (M_4 / M_2^2) - 3.$$

Заметим, что для нормального распределения  $M_4 / M_2^2 = 3$ , а  $E = 0$ .

Кроме того, полезно вычислить вспомогательные коэффициенты

$$A3 = \sqrt{\frac{6(N-1)}{(N+1)(N+3)}},$$

$$A4 = \sqrt{\frac{24N(N-2)(N-3)}{(N-1)^2(N+3)(N+5)}}.$$

Если А и Е в 2 – 3 раза меньше А3 и А4 соответственно, то закон распределения  $P(x)$  можно принять за нормальный (при нем точные значения А и Е равны 0).

Все эти вычисления реализуются с помощью двух процедур STAT1 и WORK:

```

TO STAT1 :XI
MAKE "N COUNT :XI IF :N < 3 [PR [N < 3] STOP]
MAKE "S1 0 MAKE "S2 0 MAKE "S3 0 MAKE "S4 0
MAKE "A 0 REPEAT :N [WORK]
MAKE "S1 :S1 / :N MAKE "S2 :S2 / :N MAKE "S3 :S3 / :N MAKE "S4 :S4
                                                    :N
MAKE "M2 :S2 - :S1 * :S1
MAKE "M3 :S3 - 3 * :S1 * :S2 + 2 * :S1 * :S1 * :S1
MAKE "M4 :S4 - 4 * :S1 * :S3 + 6 * :S1 * :S1 * :S2
MAKE "M4 :M4 - 3 * (POWER :S1 4)
MAKE "D :M2 MAKE "D0 :D * :N / (:N - 1)
MAKE "A :M3 / (SQRT (:M2 * :M2 * :M2))
MAKE "E :M4 / :M2 / :M2 - 3
MAKE "A3 SQRT (6 * (:N - 1) / (:N + 1) / (:N + 3))
MAKE "A4 24 * :N * (:N - 2) * (:N - 3) / (:N - 1) / (:N - 1)
MAKE "A4 SQRT (:A4 / (:N + 3) / (:N + 5))
PONS
END

TO WORK
MAKE "A :A + 1 MAKE "E ITEM :A :XI MAKE "S1 :S1 + :E
MAKE "S2 :S2 + :E * :E MAKE "S3 :S3 + POWER :E 3
MAKE "S4 :S4 + POWER :E 4
END

```

?

Процедура STAT1 имеет один входной параметр – лист с входными данными. Она вычисляет число данных N (если  $N < 3$  вычисления прерываются) и обнуляет переменные сумм S1, S2, S3 и S4 и переменную счетчика А. Затем идет обращение к подпроцедуре WORK (работа) в цикле, повторяющемся N раз. Процедура WORK при каждом исполнении увеличивает А на 1 и формирует суммы  $x_K^k$  (где  $K = 1, 2, 3$  и 4). После этого процедура STAT1 выполняет оставшиеся вычисления.

В процедуре STAT1 использован оригинальный прием, резко упрощающий программу. В ней использованы переменные, имена которых адекватны обозначениям результатов вычислений. В конце список значений этих переменных выводится всего одной командой PONS.

Вот так выглядит вычисление статистических параметров массива из 10 чисел:

```

?STAT1 [9 8 10 9 11 12 10 10 9 11]
A4 IS 0.9224435939
A3 IS 0.6145098678
E IS -0.753921039
D0 IS 1.433333333
D IS 1.29
M4 IS 3.7377
M3 IS 0.288
S4 IS 10379.7
S3 IS 1008.9
S2 IS 99.3
S1 IS 9.9
XI IS [9 8 10 9 11 12 10 10 9 11]
N IS 10
M2 IS 1.29
A IS 0.1965657838
?

```

### 5.9. Спектральный анализ

Из курса высшей математики известно, что любое периодическое колебание можно разложить на отдельные гармонические составляющие – гармоники. Спектральный анализ заключается в получении амплитуд и фаз этих гармоник на частоте  $f = kf_1$ , где  $k$  – номер гармоники, а  $f_1$  – частота первой гармоники.

Спектральный анализ можно распространить на более общий случай финитных, т.е. непериодических функций  $y(t)$ , которые полностью определены на конечном отрезке времени  $[0, t_0]$ . В этом случае функция на любой частоте  $f$  характеризуется комплексной спектральной плотностью  $S(f) = S_c(f) + jS_s(f)$ , причем

$$S_c(f) = \int_0^{t_0} y(t) \cos(2\pi ft) dt, \quad S_s(f) = \int_0^{t_0} y(t) \sin(2\pi ft) dt.$$

Если считать, что  $y(t)$  задается дискретными значениями  $y_i$ , то

$$\frac{S_c}{\Delta t} = \frac{a_n N}{2} \approx \sum_{i=0}^{N-1} y_i \cos(2\pi f i \Delta t), \quad (5.4)$$

$$\frac{S_s}{\Delta t} = \frac{b_n N}{2} \approx \sum_{i=0}^{N-1} y_i \sin(2\pi f i \Delta t), \quad (5.5)$$

где  $a_n$  и  $b_n$  – коэффициенты Фурье для периодической  $y(t)$  с периодом  $t_0 = T$ .

Кроме того, спектральная плотность и фазовый сдвиг на частоте  $f$  равны:

$$S(f) = \sqrt{S_c(f)^2 + S_s(f)^2}, \quad \varphi(f) = -\arctg(S_c(f)/S_s(f)).$$

Численный спектральный анализ на Лого ниже реализуется с помощью трех процедур. Первая процедура SI служит для диалогового ввода номера первого ненулевого отсчета IS, номера последнего ненулевого отсчета IF, общего числа участков разбиения  $T - N$ , времени  $T$  и списка ненулевых отсчетов  $y_{IS}, y_{IS+1}, \dots, y_{IF}$ . Такой способ ввода (без нулевых отсчетов) экономит память ПЭВМ и сокращает время счета, особенно если на значительном интервале времени  $y(t) = 0$  (т.е.  $y(t)$  есть импульс).

Процедура SI имеет следующий листинг:

```
?PO [SI S WORK1]
TO SI
TYPE [INPUT Istart =] MAKE "IS FIRST RL
TYPE [INPUT Ifinish =] MAKE "IF FIRST RL
TYPE [INPUT N =] MAKE "N FIRST RL
TYPE [INPUT T =] MAKE "T FIRST RL
TYPE [INPUT Y [ ] =] MAKE "Y RL
MAKE "DN :IF - :IS + 1
MAKE "DT :T / :N
S
END
```

В конце этой процедуры вычисляется число ненулевых отсчетов  $DN = IF - IS + 1$  и время  $\Delta t = T/N$ , после чего идет обращение к следующей процедуре S.

Процедура S выполняет следующие операции: очищает узлы памяти, задает ввод частоты  $f$ , вычисляет (5.4) и (5.5), используя только ненулевые отсчеты  $y_i$ , и вычисляет  $S(f)$  и  $\varphi(f) = \text{DEG}$ . Вычисление сумм (5.4) и (5.5) осуществляется обращением к подпроцедуре WORK1. Листинги процедуры S и подпроцедуры WORK1 представлены ниже:

```
TO S
RECYCLE TYPE [INPUT f =] MAKE "F FIRST RL
MAKE "M 360 * :F * :DT MAKE "J 0 MAKE "S 0 MAKE "
REPEAT :DN [WORK1]
TYPE [Ss / dt =] PR :C TYPE [Sc / dt =] PR :S
MAKE "SI SQRT (:S * :S + :C * :C)
TYPE [S ( f ) =] PR :SI * :DT
TYPE [DEG ( f ) =] PR -90 + ARCTAN (:C / :S)
S
END
```

# TO WORK1

```
MAKE "J :J + 1 MAKE "I :IS + :J MAKE "F :M * :I
MAKE "YI ITEM :J :Y MAKE "S :S + :YI * SIN :F
MAKE "C :C + :YI * COS :F
END
```

В процедуре S углы переведены из радиан в градусы. Процедура S рекурсивная, т.е. после вычислений следует возврат к ее началу, запрос на ввод частоты  $f$  и возможность повторения вычислений для новых  $f$ . Таким образом реализуется последовательный спектральный анализ.

Для проверки процедур спектрального анализа вычислим спектральную плотность и фазовый сдвиг прямоугольного импульса с длительностью 1 мкс, периодом  $T = 4$  мкс при  $N = 32$ , задании  $y_0 = y_7 = 1$  и остальными нулевыми отсчетами. Процедура спектрального анализа следующая:

```
?SI
INPUT Istart =0
INPUT Ifinish =7
INPUT N =32
INPUT T =4E-6
INPUT Y [ ] =1 1 1 1 1 1 1 1
INPUT f =250000
Ss / dt =4.576585193
Sc / dt =5.576585193
S ( f ) = 9.017641949E-0007
DEG ( f ) =-50.625
INPUT f =500000
Ss / dt =-0.9999999997
Sc / dt =5.027339493
S ( f ) = 6.407288621E-0007
DEG ( f ) =-101.25
INPUT f =
```

Приведенные примеры наглядно показывают, что мнение о "детской" принадлежности Лого ошибочно и на этом языке можно успешно реализовать многие достаточно сложные научно-технические, математические и учебные расчеты. При этом программная реализация получается простой и наглядной и сочетается с превосходными графическими возможностями Лого, расширенное описание которых дано в следующей главе.

## 6. Внешние процедуры Лого-графики

### 6.1. Построение простых геометрических фигур

Язык Лого открывает почти неограниченные возможности для создания разнообразных процедур машинной графики [13-32]. Прежде всего расширим набор возможностей Лого-графики, включив в него несколько простых и часто необходимых процедур. Начнем с построения отрезков прямой.

Отрезок прямой может задаваться различными способами. Из них наиболее удобны два. Первый способ заключается в задании координат  $(X_1, Y_1)$  начальной точки отрезка и выводе в эту точку черепашки с поднятым световым пером с помощью примитива SETPOS (рис. 6.1). Затем световое перо опускается и черепашка переводится в конечную точку с координатами  $(X_2, Y_2)$ . При своем перемещении черепашка оставляет след в виде отрезка прямой, соединяющей точки  $(X_1, Y_1)$  и  $(X_2, Y_2)$ . В конце перемещения световое перо черепашки вновь поднимается.

Этот способ реализуется процедурой LINE, входными параметрами которой являются координаты  $(X_1, Y_1)$  и  $(X_2, Y_2)$  начальной и конечной точек отрезка:

```
?PO "LINE  
TO LINE :X1 :Y1 :X2 :Y2  
PU SETPOS SE :X1 :Y1  
PD SETPOS SE :X2 :Y2  
END
```

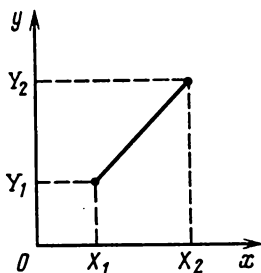


Рис. 6.1. Построение отрезка прямой с концевыми точками  $(X_1, Y_1)$  и  $(X_2, Y_2)$

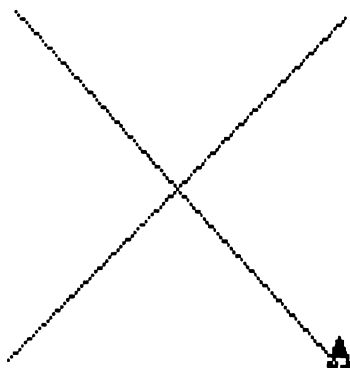


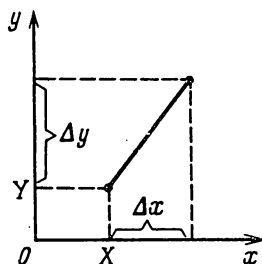
Рис. 6.2. Вид отрезков прямых, построенных с помощью процедуры LINE

На рис. 6.2 показано построение двух пересекающихся прямых с помощью описанной процедуры. Для этого было задано исполнение команд:

```
?LINE -60 -60 60 60
?LINE -60 60 60 -60
```

Другой распространенный способ построения отрезка прямой заключается в его "протяжке" от исходной точки с текущими координатами  $(X, Y)$  до новой точки с координатами  $(X + \Delta X, Y + \Delta Y)$  (рис. 6.3). Координаты  $(X, Y)$  удобно не задавать, а определять как координаты последней построенной точки. Для этого у Лого можно использовать функции `XCOR` и `YCOR`, вырабатывающие значения этих координат. Данный способ удобен тем, что позволяет тянуть ломанные линии заданием всего двух параметров  $\Delta X$  и  $\Delta Y$  каждого их отрезка.

Рис. 6.3. Построение отрезка прямой по заданной начальной точке  $(x, y)$  и приращениям координат  $\Delta x$  и  $\Delta y$



Последний способ реализуется процедурой `DRAW :DX :DY` с выходными параметрами `:DX = ΔX` и `:DY = ΔY`:

```
?PO "DRAW
TO DRAW :DX :DY
PD SETPOS SE XCOR + :DX YCOR + :DY
PU
END
```

На рис. 6.4 показано построение треугольника путем тройного применения процедуры `DRAW`:

```
?DRAW -50 50
?DRAW 100 0
?DRAW -50 -50
```



Рис. 6.4. Вид ломанной линии — треугольника, построенной с помощью процедуры `DRAW`

Как показывают приведенные примеры, с помощью процедур LINE и DRAW можно строить различные изображения, синтезируемые из отрезков прямых. Однако для построения некоторых фигур целесообразно иметь соответствующие процедуры. Примером может служить процедура построения прямоугольника BOX, описанная в § 1.5.

Специфика Лого-графики заключается в отсутствии примитивов построения дуг, окружностей и эллипсов. Однако эти фигуры легко строить, используя перемещения черепашки с одновременным ее поворотом. Ограничимся разработкой процедуры построения окружностей CIRCLE с заданным положением центра (X, Y) и радиусом R.

Алгоритм построения следующий. Вычисляем шаг перемещения черепашки  $D = 2\pi R/36$ , который необходим для построения 36-угольника, заменяющего окружность. Из-за дискретности раstra экрана такой многоугольник будет выглядеть практически как окружность. Вначале построения нужно ввести черепашку в точку с координатами (X - R, Y). Затем, опустив световое перо 36 раз, повторяем операции: поворот черепашки на  $5^\circ$  вправо, продвижение на D шагов вперед, снова поворот на  $5^\circ$ . В результате черепашка совершит поворот на  $360^\circ$  и, двигаясь мелкими шажками, опишет окружность. Двойной поворот по  $5^\circ$  вместо одного на  $10^\circ$  обеспечивает повышенную точность задания центра окружности.

Процедура CIRCLE имеет три входных параметра – координаты центра X и Y и радиус R:

```
?PO "CIRCLE
TO CIRCLE :X :Y :R
MAKE "D 2 * PI * :R / 36
PU SETPOS SE :X - :R :Y PD
REPEAT 36 [RT 5 FD :D RT 5] PU
END
```

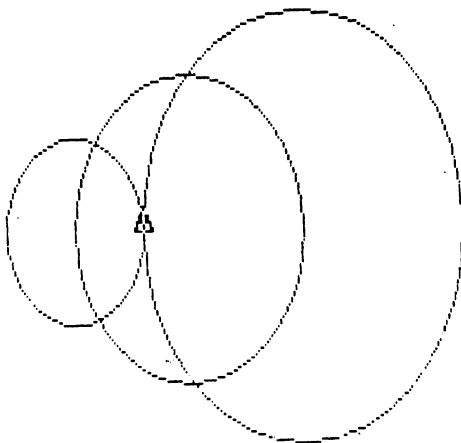
Исполним следующие команды:

```
?CIRCLE -50 0 30
?CIRCLE 0 0 50
?CIRCLE 50 0 70
```

В результате получим изображение трех окружностей (рис. 6.5).

Внимательный читатель, конечно, обратил внимание, что фигуры на рис. 6.5 – эллипсы, а не окружности. Однако на экране дисплея они выглядят как окружности. Дело в том, что графический принтер, с

Рис. 6.5. Три окружности, построенные с помощью процедуры CIRCLE



помощью которого получен рис. 6.5 (да и все остальные рисунки в книге), несколько вытягивает изображение по вертикали.

Следует отметить, что, используя команду

`SETSCR [MxMy]`

можно установить разные масштабы по осям  $x - M_x$  и  $y - M_y$ . Это может скорректировать описанные выше искажения. И напротив, меняя  $M_x$  и  $M_y$ , можно всегда превратить окружность в эллипс.

А что будет, если число 36 после примитива REPEAT сделать равным  $36/3 = 18$ ? Нетрудно догадаться, что будет построена половина окружности, т.е. дуга. Таким образом, процедуру CIRCLE легко превратить в процедуру построения дуг. Предоставим реализовать эту возможность читателю.

## 6.2. Фигуры вращения

Способность черепашки вращаться вокруг своей оси резко упрощает построение фигур, получаемых вращением объектов. Наблюдение за построением таких фигур способствует развитию геометрических представлений и имеет явную практическую пользу – фигуры вращения широко применяются в технике и в изобразительном искусстве.

Любопытные картинки можно получить, вращая даже простейшие объекты, например отрезок прямой. Процедура GD с входным параметром – углом  $A$  – просто вращает отрезок прямой с длиной в 80 растровых единиц вокруг одного конца:

```
?PO "GD
TO GD :A
```

```

HT REPEAT 360 / :A [SETPOS [0 0] FD 80
RT :A]
END

```

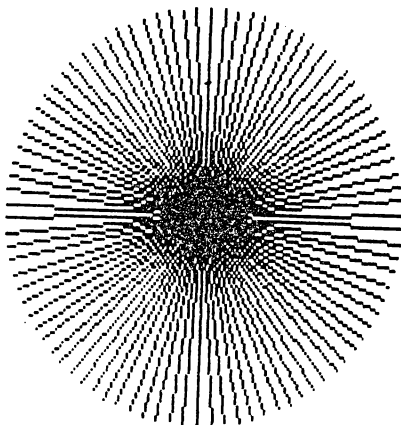


Рис. 6.6. Фигура, полученная вращением отрезка прямой вокруг одного из концов

На рис. 6.6 показан рисунок, полученный при обращении GD 4. Из-за дискретности раstra и возникновения эффекта "муара" картинка выглядит гораздо сложнее и причудливее, чем это можно было ожидать.

Еще одна процедура LGD с входным параметром – углом A

```

?PO "LGD
TO LGD :A
CS WINDOW HT
MAKE "N 180 / :A + 1
REPEAT :N [PU SETPOS [-159 1] PD FD 400
  RT :A]
SETH 180
REPEAT :N [PU SETPOS [160 1] PD FD 400
  RT :A]
SETH 90
REPEAT :N [PU SETPOS [1 100] PD FD 300
  RT :A]
SETH 270
REPEAT :N [PU SETPOS [1 -99] PD FD 300
  RT :A]
FS
END

```

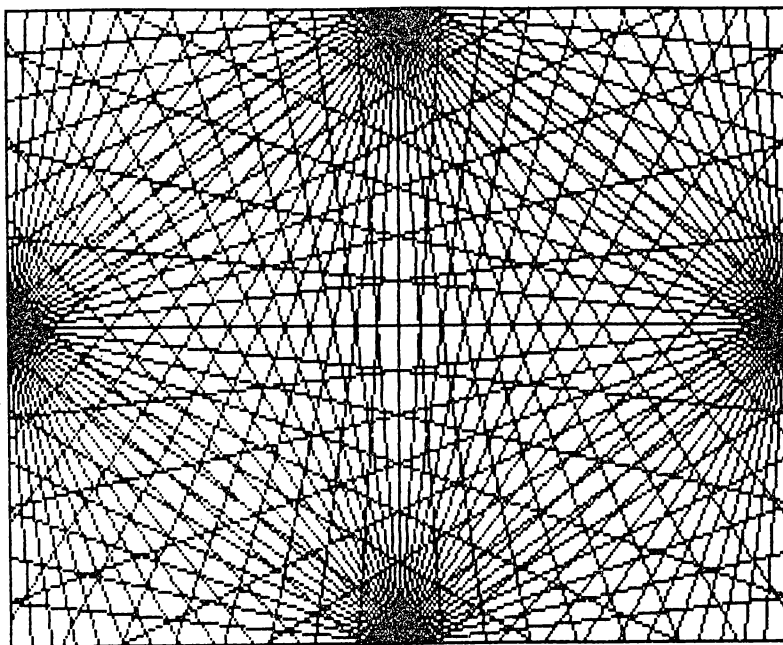


Рис. 6.7. Рисунок, полученный вращением четырех отрезков прямых вокруг конца, зафиксированного в середине каждой стороны окна экрана

строит более сложную фигуру вращением четырех прямых. Конец каждой прямой зафиксирован в середине соответствующей стороны экрана. Каждая прямая вращается вокруг этого конца в пределах  $180^\circ$  с шагом  $A$ . Команда WINDOW открывает возможность перемещения черепашки и за пределами окна (для этого выбирается большая длина отрезка), но видимы лишь ее перемещения в пределах окна. Рис. 6.7 показывает узор, полученный с помощью команды LGD 5.

Процедура BOXN с входными параметрами – числом сторон многоугольника  $N$ , длиной стороны  $L$  и числом многоугольников  $M$  – строит  $M$  правильных многоугольников, поворачивающихся на угол  $360/M$  вокруг своей начальной вершины:

```
?PO "BOXN
TO BOXN :N :L :M
LOCAL "N1 LOCAL "M1
MAKE "N1 360 / :N
MAKE "M1 360 / :M
```

```

REPEAT :M [REPEAT :N [FD :L RT :N1] RT
:M1 ]
END

```

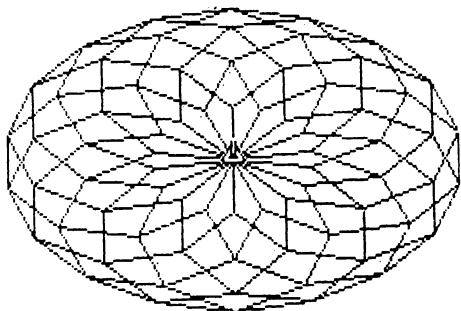


Рис. 6.8. Фигура, полученная вращением 18 девятиугольников сжатием масштаба по оси  $y$

На рис. 6.8 показано построение 18 девятиугольников со стороной  $L = 40$ , полученное обращением `BOXN 9 40 18`, после сжатия вдвое (командой `SETSCR`) масштаба по оси  $y$ . Не правда ли, удивительно, что столь сложная фигура строится с помощью небольшой процедуры?

Процедуры `PATTERN` и `SC`

```

?PO [PATTERN SC]
TO PATTERN
REPEAT 6 [SC RT 60]
END

```

```

TO SC
REPEAT 36 [FD 5 RT 10]
REPEAT 4 [FD 56 RT 90]
END

```

строят еще одну сложную фигуру вращения (рис. 6.9). Процедура `SC` строит окружность и квадрат, а процедура `PATTERN` обеспечивает шестикратное обращение к процедуре `SC` с поворотом каждой фигуры на  $60^\circ$ .

Процедуры `FIGROT` и `FIGROT1` позволяют строить  $M$   $N$ -угольников, каждый из которых вращается вокруг своего центра, поворачиваясь на угол  $DQ$ . Исходный многоугольник вписан в невидимую окружность радиуса  $R$ , затем радиус  $R$  рекуррентно меняется как  $R = KR$ , где  $K$  — множитель. Таким образом, в данном случае имеет место комбинация поворота  $N$ -угольника с изменением его размеров.

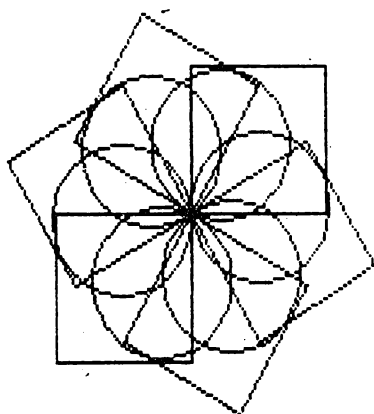


Рис. 6.9. Фигура, полученная вращением окружности и квадрата

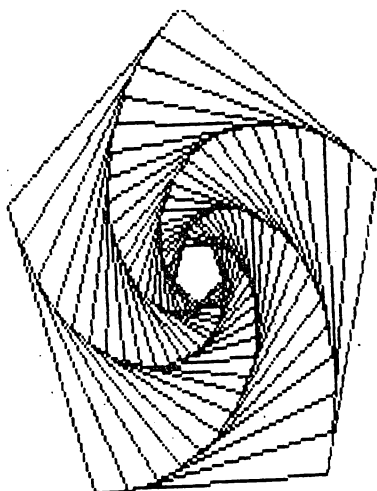


Рис. 6.10. Фигура, полученная с помощью процедуры FIGROT

Листинги процедур FIGROT и FIGROT1:

```
?PO [FIGROT FIGROT1]
TO FIGROT :N :M :DQ :R :K
CS MAKE "Q 360 / :M
REPEAT :M [FIGROT1 :N :R RT :DQ MAKE "R :R * :K] HT
END
```

```
TO FIGROT1 :N :R
PU LT 180 / :N FD :R
MAKE "Q 360 / :N
MAKE "Q1 180 / :N RT 90 + :Q1
MAKE "A :R * 2 * SIN :Q1
PD REPEAT :N [FD :A RT :Q] PU
LT 90 - :Q1 + :Q BK :R RT :Q1
END
```

На рис. 6.10 показана фигура, полученная при задании команды FIGROT 5 23 8.1 10 1 1.

### 6.3. Магия графической рекурсии

Рекурсия – мощный аппарат программирования. В этом мы уже не раз убеждались при решении чисто вычислительных задач. Но особенно велика познавательная и практическая ценность рекурсивных процедур графики. Придумыванием геометрических фигур, получающихся рекурсивно, занимались многие выдающиеся математики глубокой древности [12]. Лого идеально приспособлен для изучения графических рекурсивных процедур – подлинной магии современной машинной графики и программирования.

Описывая рекурсивные процедуры, будем рассматривать какой-либо конкретный пример. Однако многие рекурсивные процедуры порождают огромное число вариантов рисунков при различном сочетании входных параметров. Может не хватить жизни на просмотр всех этих вариантов. Но даже несколько часов, проведенных за этой увлекательной работой, заметно способствуют развитию математической смекалки и пониманию того, какие математические закономерности заложены в окружающих нас объектах (снежинках, узорах мороза, кристаллах и т.д.). Многие из этих узоров (например, звезды и кресты) легли в основу государственной символики многих стран. Это лучше всего говорит о том, что рекурсия издревле интересовала людей.

Одна из простейших рекурсивных графических процедур POLY :

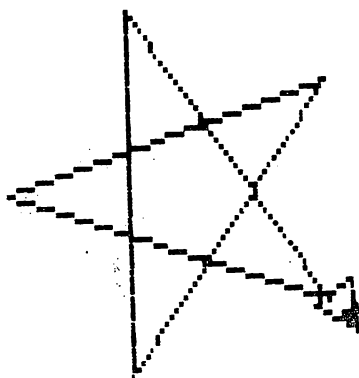
```
?PO "POLY
TO POLY :S :A
FD :S RT :A
POLY :S :A
END
```

заставляет черепашку бесконечно передвигаться вперед на N шагов, а затем поворачиваться вправо на угол A. Этого вполне достаточно, чтобы наблюдать огромное число замысловатых фигур. На рис. 6.11 показана одна из них – звездочка. Она получена вводом команды POLY 70 144. Команда POLY 60 120 будет чертить треугольник, а POLY 50 60 – шестиугольник. Вообще, если  $N = 360/A$  при  $A \leq 120^\circ$  целое число, то процедура POLY вычерчивает правильный N-угольник.

Линейчатая спираль создается рекурсивной процедурой SPI

```
?PO "SPI
TO SPI :L :S :A
FD :S RT :A
SPI :L :S + :L :A
END
```

Рис. 6.11. Звезда, полученная с помощью рекурсивной процедуры POLY



У этой процедуры три входных параметра: начальное перемещение  $S$ , приращение перемещения  $L$  и угол поворота черепашки  $A$ . Процедура заканчивается обращением к самой себе, но шаг  $S$  каждый раз увеличивается на  $L$ . В результате формируется кусочно-линейная раскручивающаяся спираль.

Картинка на рис. 6.12 получена прерыванием построения, начатого командой SPI 1 4 88. Вследствие того, что длина каждой линии постепенно растет, спираль раскручивается. На ее вид большое влияние оказывает угол  $A$ . В данном случае он чуть меньше  $90^\circ$  ( $A = 88^\circ$ ).

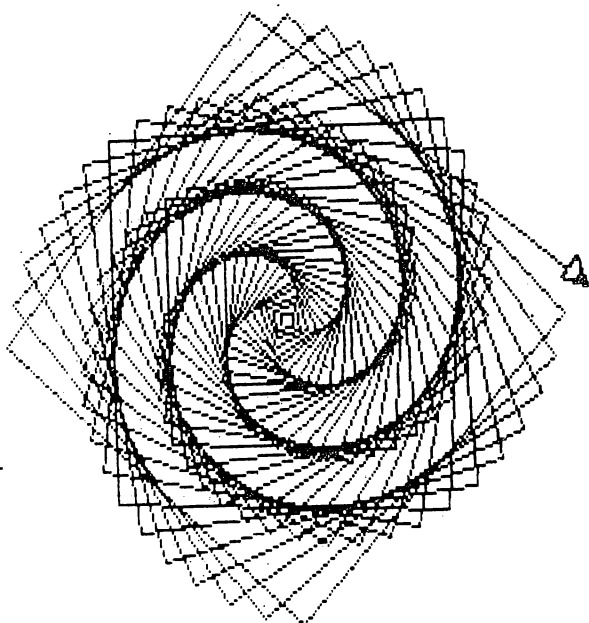


Рис. 6.12. "Квадратная" раскручивающаяся спираль, полученная с помощью рекурсивной процедуры SPI

Поэтому виток спирали близок к квадрату. На каждом витке "квадрат" поворачивается на  $(90 - 88)^\circ = 2^\circ$ , что ведет к характерному спиралеобразному движению каждой из четырех его вершин. Это дает столь причудливый вид "спирали". Взяв подходящие значения  $H$ ,  $S$  и  $A$ , с помощью этой процедуры можно строить и спирали обычного вида. Попробуйте это!

Две процедуры NP и NPOLY, из которых вторая рекурсивная

```
?PO [NP NPOLY]
TO NP :S :N :D
IF :S < 10 [PR [S IS TOO SMALL!] STOP]
IF :N < 4 [PR [NEED AT LEAST 4 SIDES!] STOP]
IF :D < 2 [PR [DEPTH TOO SMALL!] STOP]
PU SETPOS [-40 -10] PD
NPOLY :S :N :D HT
END

TO NPOLY :S :N :D
IF :S < 10 [STOP]
REPEAT :N [NPOLY :S / :D :N :D FD :S RT 360 / :N]
END
```

строят большой  $N$ -угольник ( $N \geq 4$ ) со стороной  $S$ , а в углы его помещают малые  $N$ -угольники со стороной  $D$ . Фигура на рис. 6.13 получена при подаче команды NP 50 8 3.

Процедура RESPI

```
?PO "RESPI
TO RESPI :S :A :M
MAKE "C 1
REPEAT :M [FD :S * :C RT :A MAKE "C :C
+ 1]
RESPI :S :A :M
END
```

строит  $M$  отрезков кусочно-линейной спирали с начальной длиной отрезка  $S$  и ее поворотом на угол  $A$ . При каждом построении  $S$  умножается на множитель  $C$  (вначале  $C = 1$ ), а затем  $C$  увеличивается на 1. В результате спираль раскручивается. После построения отрезков все повторяется вновь за счет рекурсивного обращения процедуры к самой себе. Однако при этом черепашка оказывается на новом месте.

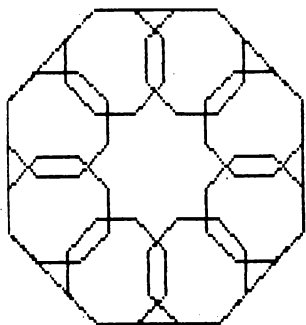


Рис. 6.13. Фигура, полученная исполнением  
рекурсивной процедуры NP

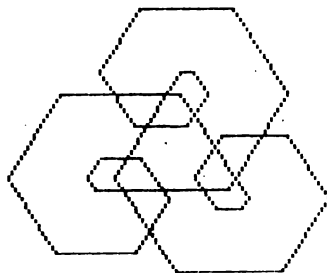


Рис. 6.14. Фигура, полученная исполнением  
рекурсивной процедуры RESPI

Параметры S, A и N можно подобрать так, что будет рисоваться замкнутая линия. Рис. 6.14 получен следующим обращением к процедуре: RESPI 5 60 10. Процедура RESPI также позволяет строить множество забавных и поучительных фигур.

Предоставляем читателю возможность разобраться в еще двух процедурах построения кусочно-линейных спиралей RESPI1 и SUBSPI1:

```
?PO [RESPI1 SUBSPI]
TO RESPI1 :S :A :M :L
MAKE "C 1 REPEAT :M [SUBSPI]
RESPI1 :S :A :M :L
END

TO SUBSPI
FD :S * :C
IF MEMBERP :C :L [LT :A] [RT :A]
MAKE "C :C + 1
END
```

Из множества создаваемых ими фигур на рис. 6.15 показана одна, полученная вводом команды RESPI 1 2 65 9 [1 2 3 4].

Процедуры CYCLO и CYCLO1 строят звезды из отрезков синусоиды:

```
?PO [CYCLO CYCLO1]
TO CYCLO :X :Y :K :A :S
PU SETPOS SE :X :Y PD
```

```
CYCLO1 :K :A :S .
END
```

```
TO CYCLO1 :K :A :S
FD :S * SIN (:K * HEADING) RT :A
CYCLO1 :K :A :S
END
```

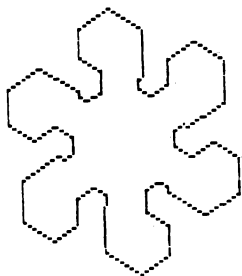


Рис. 6.15. Фигура, полученная исполнением процедуры RESPII

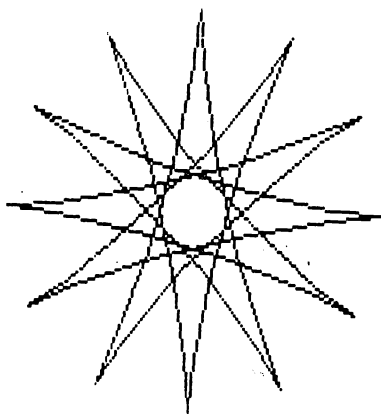


Рис. 6.16. Звезда из отрезков синусоиды, построенная с помощью процедуры CYCLO.

Входные параметры ведущей процедуры CYCLO: координаты центра X и Y, множитель K, угол поворота A и шаг S. На рис. 6.16 показана одна из звезд, полученная исполнением команды CYCLO 0 -80 6 5 40.

Сложные комбинации передвижения черепашки с поворотом обеспечивает процедура TREE

```
?PO "TREE
TO TREE :N
IF :N < 2 [STOP]
FD :N RT 15
TREE (3 * :N / 4)
LT 30 TREE (3 * :N / 4)
RT 15 BK :N
END
```

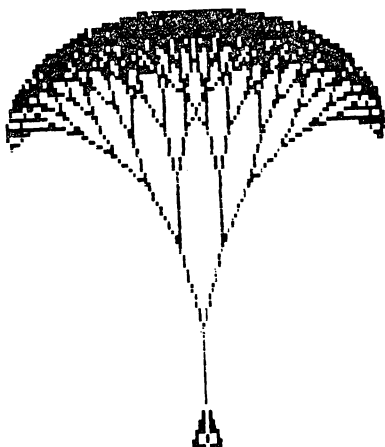


Рис. 6.17. "Парашютик" с древовидным куполом, построенный исполнением процедуры TREE.

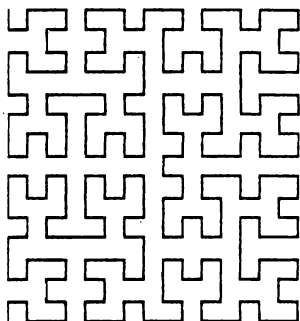


Рис. 6.18. Один из лабиринтов Гильберта, построенный рекурсивной процедурой HILBERT.

"Парашютик" (или "дерево"), построенный исполнением команды TREE 30, показан на рис. 6.17.

Последняя из процедур HILBERT с рекурсивной процедурой HBERT строит лабиринты Гильберта, алгоритм построения которых был предложен знаменитым древним математиком:

```
?PO [HILBERT HBERT]
TO HILBERT :S :L :I
IF :L < 1 [PR [L TOO SMALL] STOP]
PU SETPOS [60 -60] PD
HBERT :S :L :I
END
```

```
TO HBERT :S :L :I
IF :L = 0 [STOP]
LT :I * 90
HBERT :S (:L - 1) (-1 * :I)
FD :S RT :I * 90
HBERT :S (:L - 1) :I
FD :S
HBERT :S (:L - 1) :I
```

```

RT :I * 90 FD :S
HBERT :S (:L - 1) (-1 * :I)
LT :I * 90
END

```

Построение таких фигур – любимое занятие опытных программистов. Лого позволяет создавать наиболее простые из программ, создающих фигуры Гильберта. На рис. 6.18 показана одна из множества таких фигур, полученная вводом команды HILBERT 8 4 1.

#### 6.4. Построение графиков функций

При выполнении научных и учебных расчетов часто необходимо строить графики произвольных функций на фоне координатных осей с делениями и оцифровкой их [11]. И в этом случае программная реализация решения этой задачи на Лого оказывается довольно простой.

Рассмотрим две процедуры C4 и GF:

```

?PO [C4 GF]
TO C4 :XM :YM
PU HOME SETPOS SE -140 2 PD
REPEAT 4 [BK 2 RT 90 FD 70 LT 90 FD 2]
PU SETPOS SE 2 -90 RT 90 PD
REPEAT 4 [BK 2 LT 90 FD 45 RT 90 FD 2]
PU SETCURSOR [13 1] TYPE -:XM
SETCURSOR [13 37] TYPE :XM
SETCURSOR' [2 19] TYPE :YM
SETCURSOR [22 19] TYPE -:YM
END

```

```

TO GF :XM :YM :N
TYPE [F ( X ) =] MAKE "F RL
TYPE [INPUT C4 ( Y / N ) =] MAKE "Z RL
SETTEXT 1
IF :Z = [Y] [C4 :XM :YM]
IF :F = :N [CT STOP]
MAKE "X -:XM MAKE "M 140 / :XM
MAKE "L 90 / :YM MAKE "H 2 * :XM / :N
SETPOS SE -140 :L * (RUN :F)

```

Процедура C4 строит координатную систему с центром в середине экрана. Она имеет два входных параметра ХМ и УМ. Значения  $\pm$  ХМ проставляются по краям оси  $x$ , а значения  $\pm$  УМ – по краям оси  $y$ .

Процедура GF строит график произвольной функции. Вначале она запрашивает аналитическое выражение для функции  $F(:X)$ , затем следует запрос INPUT C4 (Y/N) – ввод C4 (да/нет). Ответ Y (да) приводит к обращению к процедуре C4, т.е. построению координатной системы. Далее процедура GF строит график введенной функции.

Пусть нужно построить график в виде суммы двух волновых процессов, описываемых соотношением

$$F(x) = \sin(500x) + 0,5 \cos(600x).$$

Не следует удивляться большим значениям аргумента, нужно вспомнить, что Лого-функции  $\sin$  и  $\cos$  вычисляются при аргументе, заданном не в радианах, а в градусах. Допустим, что  $x$  должен меняться от  $-4$  до  $+4$ , а  $y$  от  $-2$  до  $+2$ . Поскольку  $F(x)$  быстроосциллирующая функция, нужно взять достаточно большое число точек  $N$ . Возьмем  $N = 200$ .

Тогда обращение к процедуре GF будет иметь вид

?GF 4 2 200

В ответ ПЭВМ дает запрос

F(:X) =

т.е. требует ввода функции. После ввода имеем

$$F(:X) = (\text{SIN } 500 * :X) + 0.5 * (\cos 600 * :X)$$

Ответив на запрос

INPUT C4 (Y/N) = Y

получим график введенной функции.

Однако возможности процедуры на этом не кончаются. Имеется возможность наложения на построенный график графика новой функции. Для этого достаточно повторить обращение к процедуре GF, а в ответ на запрос

F(:X) =

ввести новую функцию. Допустим, мы задали  $F(X) = \sin(100X)$ , т.е.  $F(:X) = \sin 100 * :X$

В итоге получим графики этих двух функций (рис. 6.19) и приглашение к вводу новой функции.

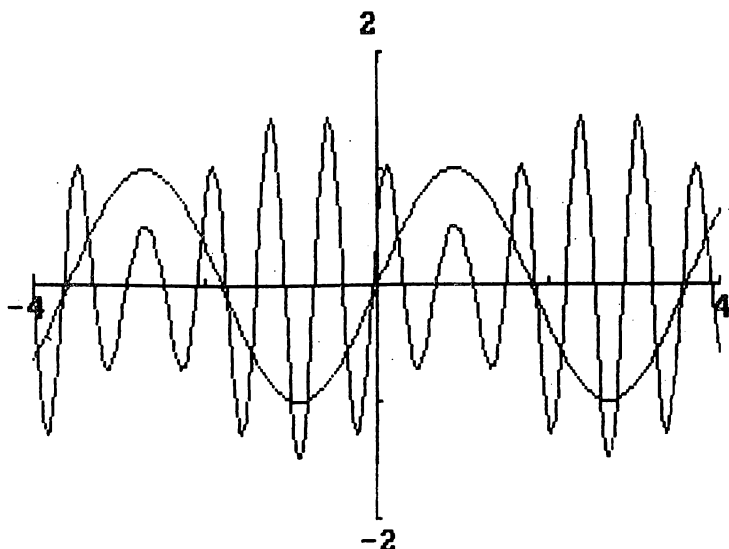


Рис. 6.19. График двух функций на фоне координатных осей, построенный процедурой GF

### 6.5. Кривые в полярной системе координат

В полярной системе координат любая точка функции  $F(\rho, Q)$  задается значением угла  $Q$  (от  $0$  до  $360^\circ$  для одного цикла построения) и зависимостью длины радиуса-вектора  $\rho$  от  $Q$  (рис. 6.20).

Лого имеет удобный аппарат для построения графиков функций в полярной системе координат по точкам. Алгоритм построения каждой точки функции следующий. Вначале командой HOME черепашка отправляется в исходное состояние. Затем командой SETH :  $Q$  она поворачивается на угол  $Q$ , а командой FD перемещается на расстояние, пропорциональное длине  $\rho(Q)$ . Это расстояние задается в растровых координатах. Определив их как значения переменных XCOR и YCOR, с помощью команды DOT можно построить точку. Угол  $Q$  меняется от  $0$  до  $360^\circ$ . Число точек задается как  $360/N$ , где  $N$  – коэффициент дробления. Целесообразно совместить построение графика функции с построением координатных осей.

При этом алгоритме кривые строятся в Лого-системе координат (см. рис. 2.1, а). Между тем часто желательно перейти к обычной системе координат (см. рис. 2.1, б), при которой радиус-вектор для  $Q = 0$  сливается с осью  $x$ . Кроме того, часто нужно строить кривые не по отдельным точкам, а соединяя их отрезками прямых. При этом можно

заметно уменьшить число опорных точек и увеличить скорость построения графиков.

Указанные возможности легко реализуются, если координаты  $x$  и  $y$  в декартовой системе координат выразить через  $\rho$  и  $Q$  в полярной системе координат:

$$x = \rho(Q) \cos Q, \quad y = \rho(Q) \sin Q.$$

Эти соображения положены в основу процедуры GFPOL с подпроцедурой GFPOL1:

```
?PO [GFPOL GFPOL1]
TO GFPOL :XYM :N :K
TYPE [Ro ( :Q ) =] MAKE "RO RL
HOME PD LT 90 FD 85 BK 170
HOME FD 85 BK 170 MAKE "M 85 / :XYM
MAKE "L 360 / :N MAKE "Q 0 PU
SETPOS SE (RUN :RO) * :M 0 PD
REPEAT :L [GFPOL1] PU
END
```

```
TO GFPOL1
MAKE "Q :Q + :N MAKE "R :M * RUN :RO
MAKE "X :R * COS :Q MAKE "Y :R * SIN ( :
Q * :K)
SETPOS SE :X :Y
END
```

Процедура GFPOL имеет входные параметры: масштабный множитель XYM, количество точек-узлов N и множитель аргумента Q.

На рис. 6.21 показан результат выполнения процедуры GFPOL при следующем обращении к ней:

GFPOL	1	6	1
RO (:Q)	=	0.8	
GFPOL	1	6	3
RO (:Q)	=	0.8	

В первом случае формируется окружность – фигура Лиссажу для равенства аргументов у  $\cos Q$  и  $\sin(KQ)$ . Во втором случае получается фигура Лиссажу для кратности частот  $K = 3$ .

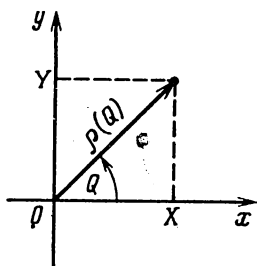


Рис. 6.20. Полярная система координат

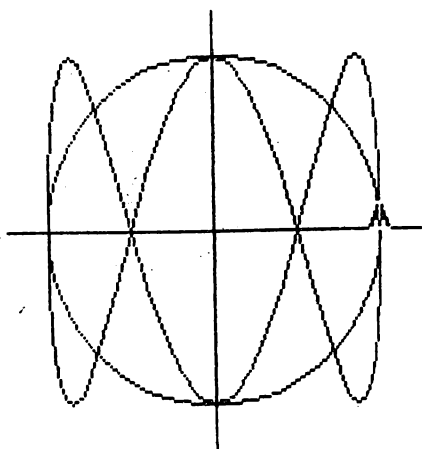


Рис. 6.21. Две кривые, построенные в полярной системе координат с помощью процедуры GFPOI

При произвольной кратности частот (в том числе дробной) для построения фигур Лиссажу можно использовать специальную процедуру LISSAJOU :A :B :C :D :I с подпроцедурой LISS :A :B :

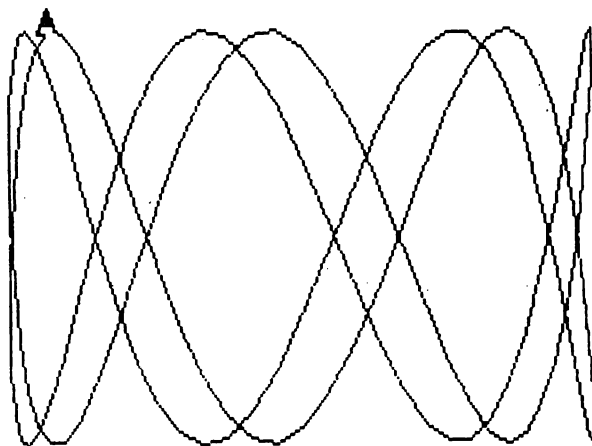
```
?PO [LISSAJOU LISS]
TO LISSAJOU :A :B :C :D :INC
ST CS PU LISS 0
END

TO LISS :Q
MAKE "X 120 * COS (:A * :Q + :B)
MAKE "Y 80 * COS (:C * :Q + :D)
SETPOS SE :X :Y PD LISS :Q + :INC
END
```

Здесь входные параметры :A и :B задают аргумент и фазу первой косинусоиды: :C и :D — то же для второй косинусоиды, параметр :I задает шаг изменения угла. Процедура LISSAJOU задает начальное значение угла :Q равным нулю и запускает рекурсивную процедуру LISS. На рис. 6.22 показан результат, полученный при следующем обращении к процедуре:

```
?LISSAJOU 5 45 17.5 0 1
```

Рис. 6.22. Фигура Лиссажу, построенная с помощью процедуры LISSAJOU



Фигуры Лиссажу обычно наблюдают с помощью электронного осциллографа, к входам  $x$  и  $y$  которого подключают два генератора синусоидальных сигналов. Если их частоты некрatны, то фигуры Лиссажу вращаются, что затрудняет их наблюдение и анализ (остановка фигур указывает на кратность частот). Как видно из приведенных примеров, ПЭВМ прекрасно моделирует процесс построения фигур Лиссажу и позволяет наблюдать его в развитии – в виде движения черепашки. При этом геометрические особенности фигур Лиссажу выявляются наиболее полно.

#### 6.6. Построение линейных объемных и цветных гистограмм

Гистограммой обычно называют ряд вертикальных или горизонтальных столбиков, высота которых характеризует ординату  $y_i$  у некоторой зависимости. Чаще всего это зависимость, получаемая в результате статистической обработки данных, например потребление электроэнергии по месяцам, выпуск какой-либо продукции по годам и т.д. Гистограмма характеризуется числом столбцов и высотой каждого из них (ее удобно выражать в растровых единицах экрана дисплея).

Гистограммы могут быть простыми или объемными. В простых гистограммах каждый столбец отображается в виде прямоугольника. Он может быть раскрашен или заштрихован каким-либо цветом. Читатель может легко составить программу для построения таких гистограмм.

В заключение рассмотрим построение более совершенных и эстетичных трехмерных цветных гистограмм. Для задания основных цветов в Лого IBM PC приходится изменять номер цветовой палитры N (с помощью команды SETPAL N, где N = 0 или 1). Удобно ввести специальные процедуры, которые вырабатывают код с нужного цвета и заодно переключают номер цветовой палитры.

Ниже приведены листинги шести таких процедур:

```
?PO [WHITE MAGENTA CYAN BROWN RED GREEN  
]
```

```
TO WHITE  
SETPAL 1  
OP 3  
END
```

```
TO MAGENTA  
SETPAL 1  
OP 2  
END
```

```
TO CYAN  
SETPAL 1  
OP 1  
END
```

```
TO BROWN  
SETPAL 0,  
OP 3  
END
```

```
TO RED  
SETPAL 0  
OP 2  
END
```

```
TO GREEN  
SETPAL 0  
OP 1  
END
```

Эти процедуры при использовании команд управления цветом (например, SETPC) выдают коды C и устанавливают нужную палитру для

следующих цветов :WHITE – белый, MAGENTA – пурпурный, CYAN – голубой, BROWN – коричневый, RED – красный и GREEN – зеленый.

Для построения трехмерных цветовых гистограмм используются три процедуры 3DGIST, GWORK и 3DCOL:

```
?PO [3DGIST GWORK 3DCOL]
TO 3DGIST :YI :CI :S
CS PU HT CLEAN
MAKE "N COUNT :YI
MAKE "DX INT 300 / :N
MAKE "W :DX - :S
MAKE "X -150 MAKE "Y -100
REPEAT :N [GWORK]
CT SETCURSOR [0 14]
TYPE [3D GISTOGRAMMS]
FS
END
```

```
TO GWORK
MAKE "C FIRST :CI
IF :C = "WHITE [SETPC WHITE]
IF :C = "MAGENTA [SETPC MAGENTA]
IF :C = "CYAN [SETPC CYAN]
IF :C = "BROWN [SETPC BROWN]
IF :C = "RED [SETPC RED]
IF :C = "GREEN [SETPC GREEN]
MAKE "CI BF :CI
3DCOL :X :Y :W FIRST :YI
MAKE "YI BF :YI
MAKE "X :X + :DX
END
```

```
TO 3DCOL :X :Y :W :H
MAKE "XC :X
REPEAT :W [PU SETPOS SE :XC :Y PD FD :H
MAKE "XC :XC + 1]
SETPOS SE :X (:Y + :H) RT 45
PD FD :S RT 45 FD :W
RT 135 FD :S
```

```

SETH 45 SETPOS SE :X + :W :Y
FD :S LT 45 FD :H SETH 0 PU
END

```

Входными параметрами основной процедуры **3DGIST** являются: лист высот столбцов в растровых единицах **YI**, лист с наименованиями цветов **CI** и параметр, характеризующий расстояние между столбцами **S**. Процедура вычисляет число столбцов и делит ось **x** так, чтобы столбцы располагались на ней равномерно с отделением друг от друга. Затем **N** раз используется процедура **GWORK**.

Процедура **GWORK** идентифицирует и устанавливает цвет каждого столбца и обращается к процедуре построения каждого столбца **3DCOL**.

Процедура **3DCOL** имеет три параметра – координаты **X**, **Y** опорного угла столбца, его ширину **W** и высоту **H**. Закрашенная передняя часть столбца строится вычерчиванием **W** вертикальных линий с высотой **H**. Таким образом, оператор закрашки **FILL** в данном случае не используется.

Рис. 6.23 получен вводом следующей команды:

```

?3DGIST [40 90 130 180 120 60 ] [ WHITE
MAGENTA CYAN BROWN RED GREEN ] 20

```

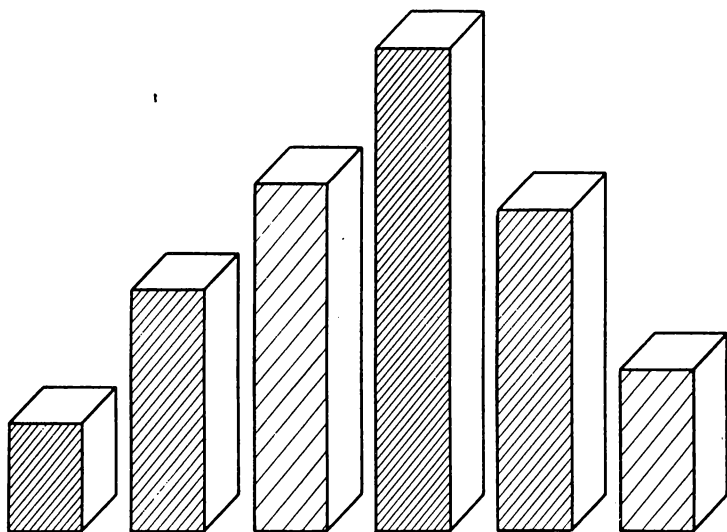


Рис. 6.23. Цветная (цвета выглядят как оттенки яркости) трехмерная гистограмма

Интересно отметить два момента. Рис. 6.24 выполнен принтером, не имеющим цветной печати. Однако можно заметить, что столбцы имеют различную яркость. Это связано с тем, что принтер ПЭВМ IBM PC дает разную штриховку изображения в зависимости от установленного цвета.

Второе обстоятельство проявляется при наблюдении гистограмм на экране дисплея. При первом построении цвет столбцов отвечает заданному. Однако в окончательном виде столбцы будут иметь только три основных цвета, соответствующих последней из используемых палитр (нельзя использовать одновременно обе палитры). Это ограничение специфично для IBM PC и версии Лого для нее.

## Приложение

### Словарь процедур Лого

Ниже представлено около 250 слов-имен встроенных в Лого процедур (примитивов). Словарь составлен по наиболее массовым и известным версиям Лого IBM PC, Apple-II и ZX-Spectrum. Слова, характерные для определенной версии, помечены указанием об этом. Слова без указаний имеются во всех версиях. Словарь пригоден для разбора и других версий Лого, хотя в этом случае возможны отличия в функциях и синтаксисе.

#### Арифметические операции и функции

- ARCCOS X — вычисляет и выдает значение  $\arccos x$  в виде угла  $\varphi$  (здесь и далее  $\varphi$  в градусах) (ZX-Spectrum),
- ARCCOT X — вычисляет и выдает значение  $\operatorname{arccot} x$  в виде угла  $\varphi$  (ZX-Spectrum),
- ARCSIN X — вычисляет и выдает значение  $\arcsin x$  в виде угла  $\varphi$  (ZX-Spectrum),
- ARCTAN X — вычисляет и выдает значение  $\operatorname{arctg} x$  в виде угла  $\varphi$ ,
- COSINE  $\varphi$  или  $\cos \varphi$  — вычисляет и выдает значение  $\cos \varphi$ ,
- COTANGENT  $\varphi$  или  $\cot \varphi$  — вычисляет и выдает значение  $\cot \varphi$  (ZX-Spectrum),
- DIFFERENCE A B — вычисляет разность  $(a - b)$  (IBM PC),
- DIV A B — вычисляет и выдает частное, полученное делением  $a$  на  $b$  (ZX-Spectrum),
- INT X — выдает целую часть  $x$  с отбрасыванием дробной части,
- PI — выдает число  $\pi$  (IBM PC),
- EXP X — вычисляет  $e^x = \operatorname{EXP}(x)$  (IBM PC),
- LN I X — вычисляет  $\ln x$  (IBM PC),
- POWER A B — вычисляет  $a^b$  (ПЭВМ IBM PC),
- PRODUCT AB — вычисляет и выдает значение произведения  $a \cdot b$ ,
- (PRODUCT AB ... Z) — вычисляет и выдает произведение  $a \cdot b \dots z$ ,
- QUOTIENT AB — вычисляет и выдает частное от деления  $a$  на  $b$  (ZX-Spectrum),
- RANDOM N — выдает случайное число с равномерным распределением в интервале  $[0, n - 1]$ , причем  $n$  должно быть целым положительным числом,

REMAINDER AB — вычисляет и выдает остаток от деления  $a$  на  $b$ ,  
 RERANDOM — инициализирует генератор случайных чисел, обеспечивая  
 точность их серий,  
 ROUND N — вычисляет и выдает целое число, полученное округлением  $n$  до  
 ближайшего меньшего числа,  
 SINE  $\varphi$  или SIN  $\varphi$  — вычисляет и выдает значение  $\sin \varphi$ ,  
 SQRT  $x$  — вычисляет и выдает значение квадратного корня  $\sqrt{x}$  для  $x \geq 0$ ,  
 SUM AB — вычисляет и выдает сумму  $a + b$ ,  
 (SUM  $a b \dots z$ ) — вычисляет и выдает сумму  $a + b + \dots + z$ ,  
 TANGENT  $\varphi$  или TAN  $\varphi$  — вычисляет и выдает  $\tan \varphi$  (ZX-Spectrum).

### Инфиксные операции

$A + B$  — вычисляет и выдает сумму  $a + b$ ,  
 $A - B$  — вычисляет и выдает разность  $a - b$ ,  
 $- A$  — придает  $a$  обратный знак,  
 $A * B$  — вычисляет и выдает произведение  $a \cdot b$ ,  
 $A / B$  — вычисляет и выдает частное  $a / b$ .

### Операции сравнения

$A > B$  — дает значение TRUE, если  $a > b$ , и FALSE в противном случае,  
 $A < B$  — дает значение TRUE, если  $a < b$ , и FALSE в противном случае,  
 $A = B$  — дает значение TRUE, если  $a = b$ , и FALSE в противном случае  
 (операция эквивалентна EQUALP).

### Операции с переменными

LOCAL "Имя или  
 (LOCAL "Имя 1 ... "Имя N) — локализация указанных именами переменных  
 внутри процедуры (IBM PC, Apple-II),  
 MAKE "Имя Объект — определяет переменную с заданным именем и присваивает  
 ей значение объекта,  
 NAME "Имя Объект — порождает переменную с указанным именем и значением  
 объекта,  
 NAMER Объект — дает TRUE, если объект имеет значение, и FALSE если его нет,  
 THING "Имя — дает содержимое имени (например THING "X" то же, что :X).

### Логические операции и функции

AND Предикат1 Предикат2 или  
 (AND Предикат1 ... ПредикатN) — дает TRUE, если все предикаты  
 TRUE, иначе дает FALSE,  
 EQUALP Предикат1 Предикат2 или  
 (EQUALP Предикат1 ... ПредикатN) — дает TRUE, если все предикаты эквивалентны,  
 иначе дает FALSE,

NOT Предикат — дает TRUE, если предикат1 AND ... AND предикатN не истинен, иначе дает FALSE.  
OR Предикат1 Предикат2 или (OR Предикат1 ... ПредикатN) — дает TRUE, если хотя бы один предикат истинен, иначе дает FALSE.

### Обработка слов и списков

ASCII Литера — дает код ASCII указанной литеры,  
BUTFIRST Объект или BF Объект — создает указанный объект в указанном месте его первым элементом,  
BUTLAST Объект или BL Объект — создает указанный объект в указанном месте его последним элементом,  
CATH "Имя" Лист инструкций — создает лист инструкций с заданным именем,  
CHAR N — выдает литеру с кодом N,  
COUNT Объект — дает число элементов в указанном объекте,  
EMPTYP Объект — дает значение TRUE, если объект пуст, и FALSE, если он заполнен,  
EQUALP Объект1 Объект2 — дает значение TRUE, если объекты 1 и 2 эквивалентны, и FALSE, если они не эквивалентны,  
ERROR — выдает сигнал ошибки,  
FIRST Объект — выдает первый элемент списка,  
FPOT "Объект" [Список] — создает новый список, полученный из указанного с включением в его начало объекта,  
GROP "Имя" "Признак" — выдает объект с указанным именем и признаком, если он порожден примитивом PPROP (IBM PC, Apple-II).  
ITEM [Список] — выделяет из списка n-й объект,  
LAST Объект — выделяет последний элемент списка (или последнюю литеру, если объект — слово),  
LIST Объект1 Объект2 — создает список из двух объектов,  
(LIST Объект1 Объект2 ... ОбъектN) — создает список из N объектов ( $N \geq 2$ ),  
LISTP Объект — дает TRUE, если объект список, и FALSE, если нет,  
LPUT "Объект" [Список] — создает новый список, полученный из старого с включением в его конец указанного объекта,  
MEMBER "Объект" [Список] — дает значение TRUE, если объект является элементом списка, и FALSE — если нет,  
NUMBERP Объект — дает значение TRUE, если указанный объект число, и FALSE, если он не число,  
PLIST "Имя" — выделяет признак листа, ассоциированный с указанным именем (IBM PC, Apple-II),  
PPROP "Имя" "Признак" Объект — придает объекту с указанным именем признак (IBM PC, Apple-II),  
REMPROP "Имя" "Признак" — ликвидирует объект с указанным именем и признаком — см. PPROP (IBM PC, Apple-II),

SENTENCE Объект1 Объект2 или SE Объект1 Объект2 — выдает список из двух объектов,

(SENTENCE Объект1 Объект2 ... ОбъектN) или (SE Объект1 Объект2 ... ОбъектN) — выдает список из указанных объектов,

THROW "Имя — сравнивает имя с CATH — именем и дает сигнал ошибки, если имена не идентичны,

WORD "Слово1 "Слово2 — выдает слово, составленное из двух входных параметров (слов, но не списка),

(WORD "Слово1 "Слово2 ... "СловоN) — выдает слово, составленное из N входных параметров (слов),

WORDP Объект — дает значение TRUE, если указанный объект слово, и FALSE, если он не слово.

### Условные выражения, передача управления

BYE— выход из Лого в системный монитор или базовый язык — Бейсик (ZX-Spectrum),

END — завершает задание процедуры переводом ПЭВМ в режим прямых вычислений,

GO "Имя — дает безусловный переход к метке с указанным именем (IBM PC, Apple-II),

IF Предикат [Список инструкций 1] [Список инструкций 2] — если предикат дает TRUE, обеспечивается выполнение инструкций списка 1, иначе — списка 2 с последующим переходом к новой строке,

IFFALSE [Список инструкций] или IFF [Список инструкций] — выполняет список инструкций, если TEST дает FALSE (IBM PC, Apple-II),

IFTRUE [Список инструкций] или IFT [Список инструкций] — выполняет список инструкций, если TEST дает TRUE (IBM PC, Apple-II),

LABEL "Имя — создает метку — см. примитив GO,

OUTPUT или OP — придает результатам функции выходных параметров процедуры,

REPEAT N [Список инструкций] — обеспечивает выполнение N раз инструкций списка (N > 0, у дробных N дробная часть отбрасывается),

RUN [Список инструкций] — выполняет список инструкций как строку программы, например PR RUN [2 + 3 \* 5] дает 17,

STOP — обеспечивает остановку данной процедуры с возвратом к вызывающей процедуре,

TOPLEVEL — прекращает выполнение текущей команды с возвратом управлений на верхний уровень,

.DOS — возвращает управление в ДОС (IBM PC).

### Управление экраном и курсором

BACKGROUND или BG — выдает код цвета основы (экрана),

BRIGHT N — задает включение (N = 1) и выключение (N = 0) повышенной яркости

окна экрана (ПЭВМ ZX-Spectrum),

CLEAN — стирает изображение, оставляя черепашку на месте,

CLEARTEXT или CT — стирает текст (в графическом режиме только в служебных строках),

CLEARSCREEN или CS — стирает экран в графическом режиме (IBM PC),

COPYSCREEN — задает печать принтером полной копии изображения в окне экрана,

CURSOR — дает в виде списка  $[N_x N_y]$  номер столбца  $N_x$  и номер строки  $N_y$  текущего положения курсора,

FLASH — включает режим мигания изображения (ZX-Spectrum),

FULLSCREEN или FS — открывает весь экран для графики (IBM PC),

INVERSE — включает режим инверсии (обмен между цветами линии и страницы) (ZX-Spectrum),

MIXEDSCREEN или MS — устанавливает смешанный режим отображения информации (IBM PC),

NORMAL — возвращает изображение к нормальному режиму,

OVER — включает режим покрытия одного изображения другим,

PALLETE или PAL — выдает номер цветовой палитры (ПЭВМ IBM PC),

SETCURSOR  $[N_x N_y]$  или SETCUR  $[N_x N_y]$  — устанавливает курсор на знакоместо в строке  $N_x$  и столбце  $N_y$ ,

SETTC  $[C_3 C_0]$  — устанавливает цвет знака с кодом  $C_3$  и цвет основы с кодом  $C_0$  (IBM PC),

SETTEXT N — создает текстовое окно внизу экрана на N строк (IBM PC),

SETPAL P — устанавливает цветовую палитру с номером P (IBM PC),

SETWIDTH — устанавливает число строк  $N = 40$  или  $80$  в строке (IBM PC),

TEXTSCREEN или TS — стирает текст с экрана, курсор устанавливается в верхнем углу экрана,

.SETSCREEN M — устанавливает режим (моду M) работы экрана (IBM PC),

.SCREEN — выдает моду M режима работы экрана (IBM PC).

### Управление черепашкой

BACK N — или BK N — задает перемещение черепашки на N шагов назад (N задается в растровых единицах — пикселях),

DOT [X Y] — ставит точку на экране с координатами (X, Y) в пикселях, черепашка остается на месте,

FENCE — запрещает черепашке перемещаться за пределами установленной рабочей части экрана,

FILL — закраска замкнутой поверхности цветом светлого пера (IBM PC),

FORWARD N или FD N — задает перемещение черепашки на N шагов вперед,

HEADING — дает значение угла  $\varphi$  (от 0 до 359°), на который повернута черепашка в данный момент,

HIDETURTLE или HT — прячет черепашку, т.е. делает ее невидимой,

HOME — возвращает черепашку в исходное состояние в центре экрана,

LEFT  $\varphi$  или LT  $\varphi$  — вызывает вращение черепашки на  $\varphi$  градусов влево,

PENCOLOUR или PC — выдает номер цвета светового пера черепашки,  
 PENDOWN или PD — опускает вниз световое перо черепашки, т.е. приводит к тому, что движущаяся черепашка оставляет след,  
 PENERASE или PE — обеспечивает стирание линий, по которым перемещается световое перо,  
 RENVERSE или PX — включает реверсивный режим работы светового пера черепашки,  
 PENUP или PU — поднимает световое перо, что делает след от перемещения черепашки невидимым,  
 POSITION или POS — выдает координаты  $[X\ Y]$  текущего положения черепашки,  
 RIGHT  $\phi$  или RT  $\phi$  — обеспечивает вращение черепашки на  $\phi$  градусов вправо,  
 SCRUNCH — дает соотношение сторон  $[N_x\ N_y]$  рабочей части экрана,  
 SETBG C — устанавливает цвет с фона рабочей части экрана,  
 SETBORDER C или SETBR C — устанавливает цвет с бордюра (окантровки) экрана,  
 SETHEADING  $\phi$  или SH  $\phi$  — задает угол поворота  $\phi$  черепашки,  
 SETPC C — устанавливает цвет с светового пера,  
 SETPOS  $[X\ Y]$  — обеспечивает перемещение черепашки в позицию с координатами  $(X, Y)$ ,  
 SETSCRUNCH  $[N_x\ N_y]$  или SETSCR  $[N_x\ N_y]$  — устанавливает соотношение сторон по осям  $x$  и  $y$  (нормально  $n_x = n_y = 100$ ),  
 SETSHAPE "Цепочка знаков — выводит с места черепашки цепочку знаков (IBM PC),  
 SETSHAPE "TURTLE — восстанавливает обычный вид черепашки (IBM PC),  
 SETX N — перемещает черепашку по оси X на N шагов,  
 SETY N — перемещает черепашку по оси Y на N шагов,  
 SHOWNP — вырабатывает значение TRUE (истинно), если черепашка видна на экране, и FALSE (ложно), если она не видна,  
 SHOWTURTLE или ST — вызывает появление черепашки на экране,  
 TOWARD  $[\tilde{X}\ \tilde{Y}]$  — выдает угол  $\phi$  (в градусах), на который должна быть повернута черепашка, чтобы ее головка смотрела в точку с координатами  $(X, Y)$ ,  
 WINDOW — делает рабочую часть экрана окном и разрешает черепашке двигаться за его пределами (видимы движения ее лишь в пределах окна),  
 WRAP — замыкает края рабочей части экрана,  
 XCOR — дает значение координаты X (в пикселях) текущего положения черепашки,  
 YCOR — дает значение координаты Y (в пикселях) текущего положения черепашки.

### Общение с внешними устройствами

CO — обеспечивает продолжение работы после паузы (IBM PC),  
 DRIBBLE "Имя — включает внешние устройства, например  
 DRIBBLE "LPT1 включает принтер (IBM PC),  
 FORM X N — печатает целую часть X с отступом последней цифры на N позиций вправо (IBM PC),  
 KEYP — вырабатывает TRUE, если нажата допустимая клавиша, или допустимая комбинация клавиш,

PAUSE — останавливает вычисления (пауза), возобновление см. СО (IBM PC),  
 PRINT Объект или PR Объект — печатает объекта с переводом строки,  
 PRINTOFF — выключает принтер (ZX-Spectrum),  
 PRINTON — включает принтер (ZX-Spectrum),  
 READCHAR или RC — вырабатывает значение литеры нажимаемой клавиши, но не печатает ее,  
 READLIST или RL — останавливает работу, создает список, заданный пользователем и фиксируемый нажатием клавиши перевода строк,  
 SHOW Объект — печатает объект, заданный в качестве ввода,  
 SOUND T H — создает звуковой сигнал длительностью звучания T и высотой тона H (ZX-Spectrum, Apple-II),  
 STARTROBOT — пускает робот, повторяющий действие черепашки (ПЭВМ ZX-Spectrum),  
 STOPROBOT — останавливает робот (ZX-Spectrum),  
 TONE F T — создает звуковой сигнал с частотой F и длительностью T (IBM PC),  
 TYPE Объект — печатает объект без перевода строк,  
 WAIT N — создает паузу в вычислениях на время N машинных интервалов,  
 .PRINT N — включает (N = 1) и выключает (N = 0) принтер (Apple-II).

#### Работа с внешними накопителями

DIR — выдает каталог файлов диска (IBM PC),  
 CATALOG — выдает каталог файлов диска или микродрайва (ZX-Spectrum), Apple-II),  
 DISK — выдает данные о диске,  
 ERASEFILE "Имя" — стирает файл с указанным именем (IBM PC, Apple-II),  
 LOAD "Имя" [Список процедур] — считывает файл с указанным именем,  
 LOAD "Имя" — считывает файл данных,  
 LOADPIC "Имя" — считывает файл с копией изображения экрана с указанным именем (ПЭВМ IBM PC),  
 LOADSCR "Имя" — считывает экранный файл (ПЭВМ ZX-Spectrum),  
 SAVE "Имя файла" "Имя процедуры" — записывает файл с поименованной процедурой, присвоением файлу имени (имена до 7 знаков) и записью значений переменных,  
 SAVE "Имя файла" [Имя1 Имя2 ... ИмяN] — записывает файл с процедурами, имеющими указанные в списке имена, с присвоением ему имени и записью значений переменных,  
 SAVEALL "Имя файла" — записывает все процедуры и значения всех переменных (ZX-Spectrum),  
 SAVED "Имя файла" "Имя переменной" — записывает файл со значением указанной переменной и присваивает ему имя (ZX-Spectrum),  
 SAVED "Имя файла" [Имя1 Имя2 ... ИмяN] — записывает файл со значениями переменных, имена которых указаны в списке, и присваивает файлу имя (ZX-Spectrum),  
 SAVESCR "Имя" — записывает экранный файл с изображением и присваивает файлу имя,

SETDISK N1 N2 N3 — выбирает дисковод с номером N1, дорожкой N2 и сектором N3,  
 SETDRIVE — включает микродрайв (ZX-Spectrum),  
 .SETSERIAL — включает последовательный интерфейс (ZX-Spectrum),  
 .SERIALIN N — задает ввод через порт N последовательного интерфейса (ZX-Spectrum).

### Операции с памятью

ERALL — стирает все процедуры, снимает объявления переменных,  
 ERASE "Имя или ER. "Имя — стирает процедуры с указанным после кавычек именем,  
 ERASE [Имя1 Имя2 ... ИмяN] или ER [Имя1 Имя2 ... ИмяN] — стирает процедуры с именами, указанными в списке,  
 ERN "Имя — снимает объявление (стирание) переменной с указанным именем,  
 ERN [Имя1 Имя2 ... ИмяN] — снимает объявления переменных с именами, указанными в списке,  
 ERNS — снимает объявления всех переменных (процедуры сохраняются),  
 ERPS — стирает все процедуры (значения переменных сохраняются),  
 NODES — выдает число свободных узлов памяти,  
 PO "Имя — выдает на печать листинг процедуры с указанным именем,  
 PO [Имя1 Имя2 ... ИмяN] — выдает на печать листинг процедур с указанными в списке именами,  
 PONS — дает распечатку имен и значений всех переменных,  
 POPS — дает листинг всех процедур,  
 POTS — выдает каталог всех процедур,  
 RECYCLE — очищает память с сохранением процедур и значений переменных,  
 .CALL — исполняет команду в машинных кодах с адреса  $A_0$ ,  
 .DEPOSIT A C — заносит код C по адресу A,  
 .EXAMINE A — выводит код C из ячейки ОЗУ с адресом A,  
 .RESERVE N — резервирует в ОЗУ N байтов,  
 .RESERVE D — создает список  $[A_0 \ A_0 + N]$ , где  $A_0$  — начальный адрес,  $A_0 + N$  — конечный адрес.

### Определение и переопределение процедур

BURY "Имя — стирает пакет процедур с указанным именем — см. PACKAGE и PKGALL (IBM PC, Apple-II),  
 COPYDEF "Имя1 "Имя2 — создает копию процедуры с именем 2 и присваивает ей имя 1,  
 DEFINE "Имя Текст — создает процедуру с указанным текстом и присваивает ей указанное имя,  
 END — завершает задание процедуры,

**PACKAGE** "Имя Список процедур — объединяют в пакет с указанным именем процедуры списка (IBM PC, Apple-II),

**PKGALL** "Имя — объединяет в пакет все процедуры (IBM PC, Apple II),

**PRIMITIVEP** "Имя. — дает TRUE, если указанное имя совпадает с именем какого-либо примитива и FALSE, если совпадения нет,

**TEXT** "Имя — выдает листинг процедуры (без входа в редактор) с указанным именем,

**TEXT** [Имя1 Имя2 ... ИмяN] — выдает листинги процедур (без входа в редактор) с указанными в списке именами,

**UNBURY** "Имя — восстановление стертого BURY пакета с заданным именем (IBM PC, Apple-II),

**.CONTENTS** — выдает состав словаря (IBM PC) или листинг определенных объектов с исключением переменных, процедур и примитив (ZX-Spectrum),

**.PRIMITIVES** — выдает список всех примитивов (ZX-Spectrum).

### Редактирование и определение процедур

**EDIT** или **ED** — дает переход в состояние редактирования с выводом на экран последнего состояния взаимодействия с редактором,

**EDIT** "Имя или **ED** "Имя — дает переход в состояние редактирования с выводом листинга процедуры с указанным именем,

**EDIT** [Имя1 Имя2 ... ИмяN] или **ED** [Имя1 Имя2 ... ИмяN] — дает переход в состояние редактирования с выводом листинга процедур с указанными в списке именами,

**EDNS** "Имя — вызывает на редактирование переменную с указанным именем,

**EDNS** [Имя1 Имя2 ... ИмяN] — вызывает на редактирование переменные с именами, указанными в списке,

**END** — закрывает процедуру,

**TO** — открывает процедуру (после TO указывается имя процедуры и входные параметры).

### Список литературы

1. Ершов А.П. Как учить программированию//Микропроцессорные средства и системы. — 1986. — №1. — С. 91.
2. Современный компьютер: Пер. с англ./Под ред. В. М. Курочкина — М.: Мир, 1986. —212 с.
3. Иванов А. Г., Карпова А. В. Перспективы применения языка Лого в обучении//ЭВМ массового применения. — М.: Наука, 1987.
4. Черемных С. В., Гиглавы А. В., Поляк Ю. Е. От микропроцессоров к персональным ЭВМ. — М.: Радио и связь, 1988. — 288 с.
5. Дьяконов В. П. Применение персональных ЭВМ и программирование на языке Бейсик. — М.: Радио и связь, 1989. — 288 с.

6. Дьяконов В. П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. — М.: Наука, 1987. — 240 с.
7. Дьяконов В. П. Справочник по расчетам на микрокалькуляторах. — 3-е изд. — М.: Наука, 1989. — 464 с.
8. Вычислительная математика/Н. И. Данилина, Н. С. Дубровская, О. П. Кваша, Г. Л. Смирнов. — М.: Высшая школа, 1985. — 472 с.
9. Бронштейн И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся вузов. — М.: Наука, 1980. — 976 с.
10. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. — М.: Наука, 1973. — 832 с.
11. Рыбасенко В. Д., Рыбасенко И. Д. Элементарные функции. Формулы, таблицы, графики. — М.: Наука, 1987. — 416 с.
12. Игнатьев Е. И. В царстве смекалки. — М.: Наука, 1987. — 176 с.
13. Field G. Logo for the Sinclair Spectrum. — London: Macmilan, 1985. — 16 p.
14. Regourd J. P., Testard-Vaillant F.X. Le LOGO en pratique. — Lagny: Ed. du. P.S.I., 1985. — 136 p.
15. Menzel K. LOGO in 100 beispieien. — Stuttgart: Teubner, 1985. — 234 p.
16. Bitter G. G., Watson N. R. Apple LOGO Primer. — Reston: Prentice Hall, 1983. — 206 p.
17. TRS-80 LOGO Comp. — Worth: Tandy, 1983. — 70 p.
18. Wild J., Zyskowski M. Shaping LOGO on your Apple. — Beaverton: Dilithium Press, 1984. — 138 p.
19. Allen Z. R., Davic R. E., Johnson J. F. Thinking About [Sup([TIG])] LOGO: A Graphic Book at Computing With Ideas. — N. Y.: Holt Rinehart and Winston, 1984. — 236 p.
20. Simms F. Ny Fractions using LOGO//Analog Comput. — 1986. — N 44. — p. 61.
21. Lambert R. LOGO (and more) for the Commodore 64. — Beaverton: Dilithium Press, 1984. — 117 p.
22. Linderholm O. Microsoft LOGO//Austral. Person. Comput. — 1986. — Vol. 7, N 5. — p. 83.
23. Haigh R. W., Radford L. E. LOGO for Apple Computer: A Self-Teaching Guide. — N. Y.: Willey, 1984. — 316 p.
24. Martin D., Martin J. A. 88 Apple Logo Programs. — Howard: W. Sams Co. — 422 p.
25. Field G. Logo on the BBC Computer and Acorn Electron. — London: Macmilan, 1985. — 127 p.
26. Lothe H. U. Problemlösgen und Programmieren mit LOGO. — Stuttgart: Teubner, 1984. — 165 S.
27. Sinclair ZX-Spectrum LOGO-2. Programming Reference Manual//Sinclair Research Ltd. — 1984. — 79 p.
28. Ross P. Logo Programming for the IBM PC//Addison Wesley. — 1985. — 258 p.
29. Berentes D. Apple (R). Logo: A complete illustrated Handbook. — N. Y.: Willey, 1985. — 406 p.
30. Tobias J., Short J., Burrowes S. et al. Beyond Mindstorms: Teaching With IBM LOGO. — N. Y.: Reinchart and Winston, 1985. — 431 p.
31. Solomon C., Minsky M., Harvey B. Logo Works. Challenging Programms in Logo//McGraw-Hill Book Comp. — 1986. — 388 p.
32. MSX-LOGO Reference Manual//Logo Computer System Inc. — 1985. — 142 p.
33. Хаузер Д., Хирт Дж., Хоукинс Б. Операционная система MS DOS. — М.: Финансы и статистика, 1987. — 168 с.

Предисловие .....	3
<b>1. Краткое знакомство с языком программирования Лого .....</b>	<b>5</b>
1.1. Что такое Лого? .....	5
1.2. Загрузка Лого в персональную ЭВМ .....	8
1.3. Первые шаги .....	10
1.4. Черепашка Лого .....	13
1.5. Процедуры и их редактирование .....	15
<b>2. Вычислительные и логические возможности Лого .....</b>	<b>20</b>
2.1. Алфавит Лого, его объекты, слова и списки .....	20
2.2. Примитивы, процедуры, входные параметры и переменные .....	22
2.3. Арифметические и логические операции и функции .....	27
2.4. Обработка и модификация слов и списков .....	34
2.5. Условные выражения и передача управления .....	43
<b>3. Графические возможности Лого .....</b>	<b>48</b>
3.1. Команды управления экраном и цветом .....	48
3.2. Управление черепашкой и ее световым пером .....	52
3.3. Задание и вывод графических элементов пользователя (графэм) .....	60
3.4. Построение сложных графических объектов .....	64
<b>4. Общение с "внешним миром" и памятью .....</b>	<b>69</b>
4.1. Команды вывода и ввода .....	69
4.2. Работа с накопителями информации .....	74
4.3. Пауза, звук и управление роботом .....	78
4.4. Операции с памятью .....	79
4.5. Определение и переопределение процедур .....	82
4.6. Работа с редактором .....	85
<b>5. Расширение Лого и реализация численных методов .....</b>	<b>89</b>
5.1. Вычисление дополнительных функций .....	89
5.2. Преобразование чисел по основанию .....	96
5.3. Вычисление значений факториала и полинома .....	98
5.4. Квадратичная интерполяция .....	100
5.5. Решение нелинейных уравнений .....	101
5.6. Численное дифференцирование и интегрирование .....	104
5.7. Решение дифференциальных уравнений .....	106
5.8. Статистическая обработка массива данных .....	107
5.9. Спектральный анализ .....	109
<b>6. Внешние процедуры Лого-графики .....</b>	<b>112</b>
6.1. Построение простых геометрических фигур .....	112
6.2. Фигуры вращения .....	115
6.3. Магик графической рекурсии .....	120
6.4. Построение графиков функций .....	126
6.5. Кривые в полярной системе координат .....	128
6.6. Построение линейных объемных и пятых гистограмм .....	131
Приложение .....	136
Список литературы .....	144

2 р. 20 к.



Радио и связь