Основы программирования

для

"ЧАЙНИКОВ"™

2-е издание

Begining Programming FOR DUMMIES®

2nd edition

by Wallace Wang

Hungry Minds™

HUNGRY MINDS, INC.

New York, NY ◆ Cleveland, OH * Indianapolis, IN
Chicago, IL * Foster City, CA * San Francisco, CA

Основы программирования для "Чайников"™

2-е издание

Уоллес Вонг

ДИАЛЕКТИКА
Москва • Санкт-Петербург • Киев
2002

ББК 32.973.26-018.2.75
В76

УДК 681.3.07

Компьютерное издательство "Диалектика"

Перевод с английского и редакция *И. Б. Тараброва*

По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Вонг, Уоллес.

В76 Основы программирования для "чайников". : Пер. с англ. — М. : Издательский дом "Вильяме", 2002. — 336 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0197-9 (рус.)

Перед вами одна из самых простых книг, посвященных программированию. Написанная известным автором Уоллесом Вонгом, она позволит вам сделать первые шаги в освоении премудростей написания компьютерных программ. Вы узнаете, что такое язык программирования, и какие языки программирования наиболее популярны на сегодняшний день. Отдельные части книги посвящены использованию языка программирования BASIC, использованию различных структур данных, а также программированию для Internet.

Книга рассчитана на пользователей с начальным уровнем подготовки. Легкий и доступный стиль изложения поможет новичкам как можно быстрее приступить к созданию собственных программ.

ББК 32.973.26-018.2.75

ISBN 5-8459-0197-9 (рус.)
ISBN 0-7645-0835-0 (англ.)

Оглавление

Содержание

Об авторе

После окончания колледжа Уоллес Вонг проработал два года в едва сводящей концы с концами корпорации, которая долго водила своих сотрудников за нос рассказами о производстве оружия, способного в мгновение ока уничтожить все живое на планете Земля. Правда, вскоре он пришел к выводу, что жизнь стоит тратить на что-нибудь поценнее, чем прозябание в корпорации, разглагольствующей о свободе и демократии, но производящей оружие для уничтожения этих самых достижений человечества. С такими мыслями в голове он приобрел свой первый персональный компьютер IBM PC, после чего быстро осознал несовершенство руководств по работе с компьютером и компьютерными программами.

Разобравшись с командами древней операционной системы MS DOS версии **1.25,** Уоллес Вонг решил опубликовать свои заметки в местном компьютерном журнале, чтобы поделиться своими наблюдениями с другими, а также получить деньги не от военно-промышленного комплекса.

После того как читатели дали благосклонные отклики на его статьи, Уоллес стал писать еще, отводя писательской деятельности все больше времени. Впервые в жизни Уоллес стал зарабатывать на том, что не позволяет нажатием одной кнопки уничтожить полмира.

Сегодня этот автор очень удачно продолжает свою карьеру, занявшись написанием и изданием книг на компьютерную тематику. Его основная цель — объяснение простым человеческим языком сложных понятий.

Посвящение

Я посвящаю настоящую книгу всем замечательным людям, которых я повстречал на своем жизненном пути:

Кассандре (моей жене), Джордану (моему сыну), а также Бо, Скрепсу, Таша и Нуиту (нашим котам).

Лили Карни (Lily Carnie), единственному человеку, который всегда видит обе стороны медали.

Всем ребятам, с которыми я познакомился в Riviera Comedy Club (Лас-Вегас). — Стиву Ширрипа (Steve Schirripa), Дону Лернеду (Don Learned), Бобу Зани (Bob Zany), Джерри Беднобу (Gerry Bednob), Брюсу Вегасу (Bruce Vegas).

Патрику Де-Гуиру (Patrick DeGuire), который помогал мне при постановке комедий в Сан-Диего. Также спасибо Лео Фонтену (Leo Fontaine), Данте (Dante), Крису Клобберу (Chris Clobber) и Карен Ронтовски (Karen Rontowski).

Благодарности автора

Если бы не Мэтт Вагнер (Matt Wagner) и Билл Глэдстон (Bill Gladsone), я никогда не написал бы настоящую книгу (хотя это мог сделать бы кто-нибудь другой). Именно по этой причине я совершенно не возражаю против тех 15%, которые они получают от продаж настоящей книги.

Я хочу еще поблагодарить Линду Моррис (Linda Morris), Грега Кроя (Greg Croy), Рева Менгла (Rev Mengle), а также всех остальных редакторов, менеджеров и других сотрудников издательства *Hungry Minds, Inc.*, замечательной компании, в которой приятно работать.

Отдельное спасибо Аллану Вайту (Allen Wyatt) за то, что он проследил, чтобы с книгой было все нормально, а также Кассандре (моей жене) за то, что она спокойно относилась к перемещению нескольких компьютеров по дому и к появлению новых. Как только какой-нибудь компьютер исчезал, ему на смену приходила новая, более совершенная модель, которая обладала большим быстродействием и объемом жесткого диска, но при этом занимала еще больше места.

И напоследок я хотел бы поблагодарить китайские и российские издательства за перевод на соответствующие языки таких моих книг, как *Microsoft Office для "чайников"* и *Visual Basic For Dummies.* Только эти издательства смогли сохранить основную часть моих шуток, адаптировав их соответствующим образом. При переводе моих книг на другие языки мира практически все мои юмористические отступления были опущены.

Введение

Прежде всего, я хотел бы сказать, что управлять компьютером и создавать программы может *любой* человек. Для создания компьютерных программ не нужно обладать невероятным интеллектом или ученой степенью в математических дисциплинах. Вам понадобится только желание в чем-то разобраться и терпение, чтобы не бросить занятия.

Умение писать программы — это такое же умение, как и умение плавать, танцевать или жонглировать. Некоторым людям действительно удается делать это намного лучше, чем другим, но любой человек сможет достичь определенных результатов при должной практике. Именно по этой причине дети становятся асами программирования в раннем возрасте. Дети не обязательно гениальны; они просто склонны познавать новое и не боятся ошибаться.

Если вы когда-нибудь мечтали о написании компьютерных программ, то все остальное зависит от ваших возможностей и желания. Создание программ оказывается очень увлекательной штукой, однако может вызвать и разочарование, а также отнять массу времени. Именно по этой причине издательство *Hungry Minds* приняло решение издать настоящую книгу — чтобы помочь вам научиться составлять компьютерные программы с минимумом усилий и максимумом удовольствия.

Независимо от того, решили вы освоить программирование ради развлечения, для начала новой карьеры или для того, чтобы лучше выполнять свою работу, настоящая книга окажется для вас ценным подспорьем при освоении мира создания компьютерных программ, таящего в себе массу интересного и непознанного.

После того как вы завершите изучение материала настоящей книги, вы сможете выбрать наиболее подходящий для решения определенных задач язык программирования, разобраться с инструментами, часто используемыми программистами, а также создавать программы для личного пользования или для продажи другим.

Прочитав книгу *Основы программирования для "чайников", 2-е издание*, вы получите более подробные сведения об определенном языке программирования, обратившись к таким книгам, как *Visual Basic 6 For Dummies* (Уоллес Вонг), *C For Dummies* (Дэн Гукин, Dan Gookin), *Visual C++6 For Dummies* (Майкл Хаймен, Michael Hyman, и Боб Арнсон, Bob Arnson), *C++ для "чайников"* (Стивен Р. Дэвис, Stephen R. Davis, *Java Programing For Dummies* (Доналд Дж. Кусис, Donald J. Koosis. и Дэвид Кусис, David Koosis), *Window 98 Programing For Dummies* (Стивен Р. Дэвис, Stephen R. Davis), и Ричард Дж. Саймон, Richard J. Simon), *Access Programing For Dummies* (Роб Крамм, Rob Krumm), *Borland C++ Bilder 3 For Dummies* (Джейсон Войкс, Jason Vokes) (все эти книги выпущены издательством *IDG Books Worldwide, Inc.*).

Для кого предназначена настоящая книга

Думаю, эту книгу должны приобрести буквально все, поскольку вам известно, насколько полезна для экономики привычка людей тратить так много денег, как только можно. Однако вам обязательно следует приобрести настоящую книгу, если вы хотите узнать следующее.

- ✓ Как написать компьютерную программу
- ✓ Какие наилучшие языки программирования существуют и как их использовать
- ✓ Как быстрее всего приступить к созданию программы

- ✓ Развитие компьютерных языков программирования
- ✓ Как составлять программы для компьютеров, работающих под управлением Mac OS, Palm, Linux, Windows 95/98/Me/NT/2000 или Pocket PC
- ✓ Стоит ли тратить время на написание программ на Visual Basic, C++, Delphi или другом языке программирования.

Чтобы помочь вам как можно быстрее приступить к написанию компьютерной программы, я рассмотрю написание программ на Liberty BASIC, условно-бесплатном языке профаммирования, который легко загрузить с Web-узла www.libertybasic.com. Воспользовавшись этой книгой и этим языком профаммирования, вы приступите к профаммированию побыстрее или же перейдете к изучению более узконаправленных книг из серии *...для "чайников"*.

Как построена настоящая книга

В настоящей книге я придерживался устоявшихся в книгоиздании традиций, согласно одной из которых книга представляет собой последовательность пронумерованных страниц, следующих одна за другой. Для того чтобы помочь быстрее получить нужную информацию, я разделил материал книги на семь частей, каждая из которых охватывает определенную тему в составлении компьютерных программ, о чем будет рассказано в следующем разделе. Как только вам потребуется помощь, быстро пролистайте настоящую книгу, найдите часть, посвященную интересующей вас теме, и держите книгу под рукой.

Часть I. Создание компьютерной программы

Если вам кажется, что создание компьютерной программы — это нечто чрезвычайно сложное, расслабьтесь. В первой части книги я попытаюсь развенчать мифы о сложности программирования, расскажу о том, как точно работает компьютерная программа, а также расскажу, почему программирование не настолько сложная штука, как считают многие пользователи.

Для того чтобы помочь вам лучше разобраться с программированием, в настоящей части я рассказываю об эволюции языков профаммирования, существовании множества языков программирования, а также основных, как ни удивительно, принципах профаммирования. Вся первая часть поможет вам как можно быстрее приступить к написанию собственных программ.

Часть II. Изучаем программирование на Liberty BASIC

Попытка изучать программирование по книге ничем не лучше изучения дзюдо по древнему трактату. В обоих случаях вы получите неплохие теоретические сведения, но без практического использования вы не сможете их оценить.

Для того чтобы дать вам возможность попрактиковаться в профаммировании, я расскажу, как установить язык программирования Liberty BASIC и использовать его при написании настоящих компьютерных программ. На примере этого языка я продемонстрирую принципы профаммирования; кроме того, вы сможете немедленно увидеть результаты своего труда прямо на собственном компьютере.

Часть III. Дополнительные приемы программирования на Liberty BASIC

Язык программирования Liberty BASIC предоставляет массу дополнительных средств для отображения графики, воспроизведения звука и отладки программ. В на-

стоящей части я расскажу об использовании всех этих инструментов и о принципах написания программ на других языках программирования.

Часть IV. Знакомство со структурами данных

Как и людям, компьютерам необходимо место для хранения информации. Люди хранят информацию в блокнотах, записных книжках, на листах бумаги и т.д. Компьютеры такой возможности лишены.

Вместо этого в компьютерах для хранения информации предназначены *структуры данных*. Структуры данных используются любыми программами, а программисты постоянно придумывают их новые разновидности для различных применений. Поэтому в настоящей части я расскажу вам о том, как компьютерные программы используют структуры данных, а также приведу простые примеры их работы с Liberty BASIC.

Часть V. Алгоритмы: объясните компьютеру, что от него требуется

Алгоритм — это пошаговая инструкция, поясняющая компьютеру, что именно от него требуется. Представьте себе, что алгоритм — это рецепт, которому компьютер должен слепо следовать, не задавая при этом лишних вопросов.

Не существует одного идеального алгоритма, который подходил бы для написания любых компьютерных программ, точно так же как не существует единого рецепта для приготовления всех блюд, известных в мире. Для того чтобы как можно больше упростить написание программ, программисты придумывают простые алгоритмы для решения определенных задач. Об использовании алгоритмов и пойдет речь в этой части.

Часть VI. Программирование для Internet

Internet очень быстро стала неотъемлемой частью компьютерного мира, поэтому в этой части я расскажу вам об основах различных языков программирования для Internet, включая HTML (язык, используемый при создании Web-страниц), JavaScript и Java.

В настоящей части я также расскажу вам о создании невероятных Web-страниц, которые не только замечательно выглядят, но еще и реагируют определенным образом на действия пользователей. Все эти сведения вы сможете использовать при создании собственных Web-страниц и целых Web-узлов.

Часть VII. Великолепные десятки

Для того чтобы еще больше помочь вам при написании компьютерных программ, я разместил в этой части информацию, которая пригодится для повышения вашего уровня программиста.

Именно в этой части я расскажу вам обо всех возможностях, которые открываются перед программистами. Кроме того, вы узнаете, где можно найти и как использовать различные бесплатные, условно-бесплатные и коммерческие языки программирования. В названии многих языков программирования содержатся такие части, как C++ или BASIC, или такие загадочные слова, как LISP, Oberon или Python.

Как работать с настоящей книгой

Многие люди приобретут эту книгу для чтения, хотя найдутся и такие, которые просто украсят ею свои книжные полки. Скорее всего, вы будете использовать эту книгу в качестве справочника, руководства или даже оружия (если запустите ее в человека, который вам совершенно не по душе).

В идеале вы будете читать настоящую книгу, находясь неподалеку от компьютера, Прочтите небольшой отрывок из книги, после чего испытайте свои способности программиста.

Глупые предположения

Я предполагаю, что у вас есть доступ к компьютеру (поскольку попытки освоить программирование окажутся совершенно тщетными, если у вас не будет возможности работать с компьютером). Для того чтобы воспользоваться всеми преимуществами настоящей книги, ваш компьютер должен работать под управлением Windows 95, Windows 98. Windows Me, Windows NT или Windows 2000.

Если вы недостаточно хорошо разобрались с Windows 95, Windows 98, Windows Me, Windows NT или Windows 2000, приобретите книги *Windows 95 для "чайников"*, *Windows 98 для "чайников"* или *Windows Me для "чайников"* (все они написаны Энди Ратбоном и выпущены издательством "Диалектика"). Более подробные сведения о Windows **NT** или Windows 2000 вы найдете в книгах *Windows NT 4.0 For Dummies* или *Windows 2000 Professional для "чайников"* (они написаны Энди Ратбоном совместно с Шерон Кроуфорд).

Пиктограммы, используемые в настоящей *книге*

Пиктограммы указывают на полезные советы, важные сведения, пояснения технических терминов, которые могут ввести вас в замешательство, В настоящей книге используются следующие пиктограммы.

Эта пиктограмма указывает на полезные сведения, которые помогут вам сэкономить время (если вы их запомните, конечно же).

Эта пиктограмма указывает на очень важные сведения, забывать которые не стоит.

Осторожно! Эта пиктограмма указывает на потенциальные проблемы, которых стоит избегать.

Эта пиктограмма указывает на пошаговые пояснения того, как компьютер выполняет инструкции типичной программы.

Эта пиктограмма указывает на полезные сведения, которые желательно знать, но при желании можно и проигнорировать. (Если же вы хотите стать настоящим программистом, вам необходимо напрячь мозг и запомнить как можно больше технической информации, чтобы составить нормальную конкуренцию всем остальным программистам в мире.)

Часть I

Создание компьютерной программы

"А может быть, я лучше подарю тебе богатство?"

В этой части...

Определение того, как именно создается компьютерная программа, может оказаться достаточно запутанным, поэтому вся первая часть книги поможет вам получше познакомиться с восхитительным миром компьютерных программ. Прежде всего вы узнаете, какие функции выполняют программы и как их создают профессиональные программисты.

Затем я расскажу вам о том, что в мире существует огромное множество различных языков программирования, среди которых вы можете выбрать самый подходящий, а также объясню, почему одни языки программирования оказываются намного популярнее других. Вы познакомитесь с инструментами, которые используются программистами при создании, редактировании и распространении программ от начала работы над ними до самого конца.

И наконец, в настоящей части я расскажу вам, о чем стоит побеспокоиться, прежде чем приступать к самостоятельному созданию компьютерных программ. Вы познакомитесь с достоинствами и недостатками различных языков программирования, а также поймете, почему люди могут создавать достаточно серьезные программы, обладая совсем небольшим опытом в данной сфере.

Когда вы завершите изучение настоящей части, вы уже будете представлять себе, как пишется программа, какие действия для этого нужны, а также как превратить восхитительные идеи о программе в настоящий работающий программный продукт, который вы не только сможете использовать самостоятельно, но и продавать или бесплатно предоставлять другим людям. Кто знает? Получив дополнительные сведения, вы, может быть, создадите программу, которая принесет вам целый капитал, *а* это позволит вам создать собственную компанию, занимающуюся написанием программного обеспечения, и заработать миллионы.

Глава 1

Первое знакомство с программированием

В этой главе...

- Что такое программирование
- Как работает компьютерная программа
- Как написать компьютерную программу

Независимо от того, что вы слышали по этому поводу, создавать компьютерные программы совсем несложно. Умение создавать программы — это определенные знания, которыми может овладеть практически любой человек.

Несмотря на то, что компьютеры кажутся очень сложными электронными чудовищами, расслабьтесь. Совсем немногие знают, как именно работают поисковые машины, которые позволяют вам быстро находить необходимую информацию в Internet, а некоторые люди еще не разобрались, как управлять автомобилем. Точно также практически любой может научиться создавать программы, не вдаваясь в подробности о том, как именно работает компьютер.

Зачем нужно уметь программировать

Первый вопрос, который зададите вы (или ваши друзья, сотрудники, родственники): а зачем вообще связываться с программированием? Ответ зависит от того, какие цели вы преследуете, однако самые распространенные ответы приведены ниже.

- ✓ Для удовольствия. Люди учатся кататься на лыжах, танцевать, ухаживать за садом, прыгать с вышки, а также создавать икебану, так как им это нравится и интересно. Они никогда не станут профессионалами в своем хобби, однако это для них очень интересно. Точно так же написание компьютерной программы будет для вас очень интересным; например, вы можете написать простенькую программу, которая отображает на экране монитора ужасное лицо вашего начальника. Более сложные программы могут принести вам миллионы долларов, тогда вам не понадобится больше работать на своего начальника.
- V Для удовлетворения потребностей. Многие люди приступают к изучению программирования, не намереваясь при этом стать профессиональными программистами. Им просто необходи мо написать программу, которая решала бы определенную проблему, так как найти готовую программу не удалось, Например, человеку понадобилась программа для подсчета расходов, а готовой программы под рукой нет. Какими бы ни были ваши интересы, вы всегда сможете написать программу, решающую определенные задачи, и в случае успеха продавать ее другим людям.

- **Ради** новой **или второй карьеры.** Благодаря массовому шествию компьютеров и компьютерных технологий по всему миру вы никогда не будете долго безработным, если умеете писать компьютерные программы. Компании всегда ищут программистов, которые смогли бы написать определенные программы; кроме того, очень часто необходимы программисты, которые смогут внести изменения в уже существующие программы, выполняющие массу разнообразных задач. Если вы знаете, как написать компьютерную программу, вы оказываетесь в намного более выгодном положении и обладаете потенциальной возможностью заработать столько денег, сколько захотите. При этом вы можете не бросать свою существующую работу, а программирование просто позволит вам использовать свои знания.
- **Для испытания своих возможностей.** Многим людям освоение компьютерных технологий кажется чем-то очень страшным или решением сложнейших математических ребусов. Не удивительно, что компьютеры привлекают прежде всего людей с достаточно высоким интеллектом, которым просто нравится создавать новые программы.

Вы проживете замечательную жизнь, создавая компьютерные программы, но вы можете добиться такого же успеха, продавая скрепки для бумаги, исправляя сантехнику или разводя домашних животных. Если вы занимаетесь тем, что не приносит вам радости, никакие деньги в мире не сделают вашу жизнь лучше. Займитесь изучение программирования потому, что вы действительно хотите научиться это делать, а не потому, что решили, что это сделает вас богатым.

tfcuc работает компьютерная программа

Компьютеры не делают ничего такого, что не приказал им делать человек; они так же не самостоятельны, как и среднестатистические подростки. Для того чтобы заставить компьютер сделать что-нибудь полезное, ему необходимо дать очень подробные инструкции.

Дать инструкции компьютеру можно одним из следующих способов.

- Написав программу, которая скажет компьютеру, что он должен делать шаг за шагом, точно так же, как вы действуете при приготовлении блюда по рецепту.
- Приобретя готовую программу, написанную кем-то другим, которая также скажет компьютеру, что он должен делать.

Определенно, чтобы заставить компьютер что-нибудь сделать, вы (или кто-нибудь еше) должны написать программу.

Программа просто указывает компьютеру, что он должен принять определенный тип введенных данных, манипулировать ими, а потом возвратить результат в полезной для человека форме. В табл. 1.1 перечислены наиболее распространенные виды программ, приемлемый ими способ ввода информации, а также результаты их работы.

Таблица 1.1. Входные и выходные данных для различных программ

Вид программы	Входные данные	Что программа делает	Выходные данные
Текстовый процессор	Символы, которые вы вводите с клавиатуры	Форматирование текста; исправление орфографических ошибок	Отображение текста на экране или его печать

Окончание табл. 1.1

Вид программы	Входные данные	Что программа делает	Выходные данные
Игра	Нажатия клавиш или перемещение джойстика	Вычисление перемещения и изменения картинки на экране монитора	Перемещение картинки на экране монитора
Программа для анализа курса акций	Текущие и старые цены акций	Попытки анализировать тенденции изменения курса акций и вычисление их цен на ближайшее время	Предсказание курса акций в ближайшем будущем
Программа для наведения ракет	Текущее месторасположение ракеты и цели	Вычисление того, как совместить месторасположение ракеты и цели	Коррекция траектории ракеты таким образом, чтобы она была постоянно направлена на цель
Программа для оптического распознавания текста (OCR)	Текст, получаемый со сканера	Распознавание формы символов	Преобразование отсканированного текста в текстовый файл, который можно редактировать с помощью текстового редактора
Web-броузер	HTML-коды (HyperText Markup Language — язык разметки гипертекста)	Преобразование HTML-кода в текст и графические изображения	Отображение Web-страницы на экране

Программирование - это решение проблемы

Вообще говоря, программа указывает компьютеру, как решить ту или иную проблему. Поскольку в мире полно проблем, количество программ, которые могут написать люди, бесконечно.

Однако, для того чтобы сообщить компьютеру, как решить одну громадную проблему, обычно вам придется рассказать компьютеру, как решить целый ряд мелких проблем, из которых и состоит большая проблема. Например, если вы хотите создать игру, вам следует решить следующие проблемы.

- ✓ Определите, как должна двигаться картинка (автомобиль, космический корабль, человек) на экране при перемещении пользователем джойстика или другого аналогичного устройства.
- ✓ Определите, каким образом ваш персонаж будет вести себя при столкновениях со стенами, падении в пропасть или сражениях с другими персонажами.
- ✓ Убедитесь в том, что персонаж не совершает неправильных движений, например, не проходит через стены.
- ✓ Нарисуйте пейзаж, окружающий персонаж, и убедитесь, например, в том, что, когда персонаж заходит за дерево, он исчезает из виду.
- ✓ Определите, как будет вести себя персонаж при попадании в него пуль, выпущенных другим персонажем. Определите степень повреждений, изменения характера движения поврежденного персонажа, а также способ отображения повреждений на экране.

Чем проще проблемы, на которые вам удастся разложить свою большую проблему, тем проще вам будет написать программу, управляющую компьютером. Программу, которая имитирует игру в настольный теннис, отображая на экране две ракетки, стол и мячик, написать намного проще, чем программу, имитирующую воздушные сраже-

ния времен Второй мировой, когда придется контролировать перемещение самолетов, бомб, танков и сил противовоздушной обороны.

Программирование совсем несложно; оно просто отнимает много времени

На самом деле программирование совсем несложно и не является чем-то загадочным и сверхъестественным. Если вы в состоянии написать пошаговые инструкции, которые позволят человеку найти ваш дом, вы сможете написать и компьютерную программу.

Самое сложное в программировании — определение небольших проблем, образующих проблему, которую вам необходимо решить. Так как компьютеры абсолютно глупы, вам придется рассказать им, как выполнять любые действия.

Если вам необходимо рассказать другу, как пройти к вашему дому, вы даете ему приблизительно следующие указания.

1. Иди на юг до шоссе X.
2. Дойди до перекрестка с улицей Y.
3. Поверни направо и иди до освещенной части улицы.
4. Подойди ко второй двери по левой стороне улицы.

Конечно же, если вы попытаетесь дать подобные указания компьютеру, компьютер "растеряется" и попросит у вас дополнительную информацию.

1. Откуда необходимо начинать движение к шоссе X?
2. Как найти перекресток с улицей Y?
3. Что такое освещенная часть улицы?
4. Что делать после того, как я подойду ко второй двери по левой стороне улицы? Припарковать автомобиль? Затрубить в горн? Ворваться в гараж?

Вам необходимо объяснять компьютеру, как выполнять *любые* действия, что оказывается намного сложнее, чем давать указания маленьким детям. Пока компьютер не получит от вас все необходимые подробные инструкции, он просто не сможет ничего делать.

Некоторые программы никогда не работают

Потратив несколько лет на написание программ, многие приходят к выводу, что часто начать с начала оказывается намного проще (и дешевле), чем понять, почему существующая программа не хочет выполнять определенные действия.

Например, в середине 1980-х правительство Соединенных Штатов Америки высказало восхитительную идею о создании самоуправляемой противовоздушной системы *Sergeant York*. Назначение этой системы было очень просто — обнаружение вражеского самолета или ракеты и его последующее уничтожение.

К сожалению, программа, предназначенная для управления системой Sergeant York, так никогда и не заработала должным образом. Потратив миллионы долларов на написание и многократное переписывание программы, ее тестирование и последующее переписывание, программисты решили, что наконец-то заставили программу работать как следует.

Для того чтобы отметить свои достижения, компания, которая отвечала за создание системы Sergeant York, организовала демонстрацию для генералов и высших должностных лиц Пентагона. Компания расположила систему Sergeant York в центре поля, рассадила генералов и высших должностных лиц поблизости от нее, после чего запустила беспилотный летающий аппарат, чтобы продемонстрировать возможности системы Sergeant York по уничтожению вражеских летательных аппаратов.

Однако вместо того, чтобы направить пушки на летательный аппарат, система Sergeant York направила их на трибуну, на которой находились генералы и высшие должностные лица.

Не стоит и говорить, с какой паникой генералы и высшие должностные лица начали покидать линию огня. К счастью, система Sergeant York не стала применять оружие, однако Пентагон немедленно закрыл проект Sergeant York.

Поэтому, если вы когда-нибудь приступите к написанию компьютерной программы и решите бросить ее, прежде чем полностью закончите работу над ней, хорошенько подумайте о последствиях.

Что нужно для успешного написания компьютерных программ

Если вы считаете, что создавать программу интереснее, чем ее использовать, у вас есть все необходимое для того, чтобы создавать компьютерные программы. Если вы хотите изучить написание компьютерных программ, вам необходимы три следующих качества.

- ✓ Стремление. Если вы чего-то очень сильно хотите, вы обязательно это получите (но если вы совершите что-то противозаконное, вы рискуете провести немало времени в тюрьме). Если вы хотите научиться программировать, ваше желание обязательно вам поможет, независимо от того, сколько препятствий окажется у вас на пути.
- ✓ Любознательность. Здоровая доза любознательности может подогревать ваше стремление к экспериментированию и дальнейшему совершенствованию навыков программирования даже после прочтения настоящей книги. Благодаря любопытству изучение программирования окажется для вас менее скучным и более интересным. А если вам интересно, вы обязательно изучите и запомните больше сведений, чем любой абсолютно незаинтересованный в этом человек (например, ваш начальник).
- ✓ Воображение. Создание компьютерных программ — это навык, но воображение поможет сделать этот навык более совершенным и направленным. Обладающий изрядной долей воображения начинающий программист всегда будет создавать намного более интересные и полезные программы, чем замечательный программист без воображения. Если вы не знаете, что же делать со своими навыками программирования, ваш талант просто погибнет без воображения.

Стремление, любознательность и воображение — вот три самых важных качества, которыми должен обладать каждый программист. Если вы обладаете ими, вам стоит беспокоиться только о мелочах: какой язык программирования изучать (например, C++), что там с математикой и т.д.

Изучение основ программирования может (особенно вначале) показаться просто невыполнимой задачей, однако не стоит слишком беспокоиться по этому поводу. Вы просто вспомните 1960-е годы, когда целые команды очень высокооплачиваемых профессиональных программистов стали виновниками возникновения Проблемы 2000, так как не учли возможность того, что написанные ими программы все еще будут использоваться в 2000 году. Понять, как пишутся компьютерные программы, относительно просто; гораздо сложнее написать программу, которая бы решала определенные. действительно необходимые проблемы.

Глава 2

Кое-что о языках программирования

В этой главе...

- Языки программирования
- Различия между языками программирования
- Какой язык выбрать

Программирование — это не более чем написание пошаговых инструкций, с помощью которых вы "говорите" компьютеру, что он должен делать. И так как компьютер не слишком сообразителен, он нуждается во множестве точных пошаговых инструкций.

Компьютер не понимает язык, используемый людьми для общения (русский, английский, французский, испанский или любой другой). Из-за этого люди вынуждены писать инструкции на понятном для компьютера языке. Поэтому мы можем ввести термин *язык программирования*.

Набор инструкций, которыи "говорит" компьютеру, что он должен делать, называется *исходным кодом*. С его помощью определяется, как именно работает программа.

Зачем столько языков программирования

Среди многих языков программирования вы всегда сможете найти именно тот язык, который подходит для решения данной задачи. При появлении нового типа проблем люди создают новые языки.

Конечно, на самом деле компьютер понимает только один язык, состоящий из нулей и единиц, который называется *машинным языком*. Обычно программа, написанная на машинном языке, выглядит приблизительно так:

```
0010  1010  0001  1101
OOH   1100  1010  1111
0101  OHO   1101  0101
1101  1111  0010  1001
```

Машинный язык обладает следующими основными недостатками.

- ✓ Человек часто ошибается при вводе цифр. Тогда компьютер не сможет правильно выполнить инструкции.
- ✓ Для написания программ на машинном языке необходимо много времени (еще больше времени требуется на то, чтобы понять, что же именно компьютер должен сделать).

Поэтому только некоторые используют для программирования машинный язык. Для того чтобы упростить написание программ, программисты придумали простой язык программирования, который называется *язык ассемблера*.

Вся прелесть языка ассемблера

С помощью языка ассемблера вы сможете намного быстрее создавать программы. чем при использовании машинного языка. Если в машинном языке для написания программ используются нули и единицы, в языке ассемблера вы встретитесь с легко запоминаемыми сочетаниями букв (такими как JMP, MOV и ADD), которые соответствуют определенным инструкциям, написанным на машинном языке.

С помощью данных сокращений вы не только быстро и легко напишете программу. Они также позволяют изменять программу в дальнейшем. Программа, написанная на языке ассемблера, выглядит подобным образом:

```
title Kap Program
; This program displays "Take a nap!" on the screen dosseg
.model small
.stack 100h
.data
my_message db 'Take a nap!',0dh,0ah,'$'
.code
main proc
  mov ax,data
  mov ds,ax
  mov ah,9
mov  dx, offset  my_message
  int  21h
  mov  ax,4C00h
  int  21h
  main endp
end main
```

Большинство программистов не могут написать правильно работающую программу с первого раза. Так что возможность изменения программ очень полезна. Для того чтобы в будущем изменить программу, добавив в нее новые функции, вы должны понимать, как эта программа работает.

Программисты создали языка ассемблера только для своего собственного удобства. Компьютер же не представляет, как прочитать или выполнить инструкцию, написанную на языке ассемблера.

Из-за того что компьютер не может прочитать инструкции, написанные на языке ассемблера, программисты создали специальные программы, переводящие язык ассемблера на машинный язык. Эти программы называются *ассемблерами.*

Язык ассемблера обладает следующими двумя отличиями от машинного языка,

- ✓ Программы, написанные на языке ассемблера, намного проще читать.
- ✓ Программы, написанные на языке ассемблера, намного проще писать (и изменять).

И конечно, язык ассемблера обладает определенными недостатками.

- ✓ Программы, написанные на языке ассемблера, очень медленно запускаются и занимают много места (как на физическом диске, так и в памяти) в отличие от подобных программ, написанных на машинном языке.
- ✓ Вы не можете просто перенести (или говоря на жаргоне программистов, *импортировать)* программу, написанную на языке ассемблера, с одного компьютера на другой.
- ✓ Написание программ на языке ассемблера — довольно скучное дело, требующее больших затрат времени. Вот почему некоторые люди уже предпочитают не писать программы на языке ассемблера.

Вообще говоря, чем проще разобраться в языке программирования, тем больше и медленнее будут создаваемые с его помощью программы. Все программисты хотят создавать программы, простые в написании, быстро работающие и занимающие минимум дискового пространства.

Язык программирования С

Написание программ на языке ассемблера часто требует больших затрат сил и времени. При этом их достаточно сложно изменить и невозможно перенести с одного компьютера на другой. Для того чтобы разрешить эту ситуацию, программисты создали много языков программирования, например COBOL и FORTRAN. (О преимуществах и недостатках данных языков программирования вы узнаете в подразделе "Языки программирования высокого уровня".)

Некоторые программисты понимали, что необходим язык, который обеспечил бы доступ к аппаратному обеспечению (что не позволяет сделать язык ассемблера), При этом программы, написанные на данном языке, должны быть простыми для чтения, написания и изменения (таковы языки COBOL и FORTRAN). В конечном итоге был создан язык программирования, названный просто С.

Для создания языка программирования С программисты использовали уже существующий язык программирования В (при этом языка программирования А вообще никогда не существовало).

Программисты старались упростить процесс создания программ. Поэтому программа, написанная на языке программирования С, понятна многим. Это демонстрирует следующий пример:

```
main()
{
  printf ("Take a nap!");
}
```

Данная программа, написанная на языке программирования С, позволяет отобразить на экране сообщение Take a nap! (Отдохни немного!). Для отображения этой же надписи в предыдущих подразделах настоящей главы использовалась программа, написанная на языке ассемблера. Сравнивая обе программы, можно сказать, что программа, написанная на языке С, намного короче и ее проще читать, чем аналогичную программу, написанную на языке ассемблера.

При использовании программ, написанных на языке ассемблера, программисты жертвуют скоростью и дисковым пространством. Программы, написанные на языке С, запускаются медленнее, чем аналогичные программы, написанные на языке ассемблера. Это объясняется тем, что, в отличие от языка С, язык ассемблера тесно связан с естественным языком компьютера (машинным языком). Таким образом, перед тем как окончательно перевести программу, написанную на языке С, на машинный язык, необходимо перевести ее на язык ассемблера. Однако исходный код программы, написанной на языке С, проще исправить, чем исходный код, написанный на языке ассемблера (и намного проще, чем прочитать, написать и изменить исходный код аналогичной программы, написанной на машинном языке).

При использовании языка программирования С вы ощутите следующие преимущества.

- ✓ Созданные с его помощью программы легче читать и писать.
- V С помощью программ, созданных на языке программирования С, вы получите доступ ко всем элементам компьютера, чего нельзя сделать при работе с программами, созданными на языке ассемблера.

✓ Вы легко можете переносить программы с одного компьютера на другой. Программы, созданные на языке программирования С, можно запустить на других компьютерах без дополнительной доработки. Это основное отличие данных программ от программ, созданных на языке ассемблера и на машинном языке.

Третье преимущество покажется несколько странным, так что я постараюсь пояснить его. Компьютеры понимают язык программирования С нисколько не лучше, чем язык ассемблера. (Хорошо известно, что компьютеры не понимают очень многое, что им должны объяснять программисты.) Если вы напишете профамму с помощью языка программирования С, ваш компьютер не будет иметь ни малейшего представления о том, как прочитать ваши инструкции.

Для того чтобы компьютер читал и понимал инструкции, написанные на языке программирования С, вы должны перевести эту программу на машинный язык. Программисты создали специальные программы, называемые *компилирующими программами,* или просто *компиляторами,* которые выполняют этот перевод. С помощью подобной программы вы переведете на машинный язык профамму, написанную на языке профаммирования С.

При переводе с одного человеческого языка на другой легкий текст легче и переводить. Перевод детской книги с французского языка на японский будет намного проще, чем перевод сложных диссертаций по математике в основном потому, что в детской книге используются простые слова, в то время как в диссертации вы встречаете очень много терминов. Так и перевод профаммы, написанной на языке С, происходит намного сложнее, чем перевод профаммы, написанной на языке ассемблера.

Есть только один способ запуска программы, написанной на языке профаммирования С, на другом компьютере. На данном компьютере необходимо записать соответствующую компилирующую программу. Так как язык профаммирования С — простой язык, записать компилирующие профаммы на других компьютерах не представляет никакого труда, особенно если вы сравните их с аналогичными профаммами, написанными на других языках программирования, таких как Ada или LISP.

Компилирующие профаммы для языка профаммирования С довольно просты в написании, поэтому вы обнаружите их на многих компьютерах. Теоретически вы можете написать профамму на языке профаммирования С для компьютеров Macintosh, скопировать ее на компьютер, работающий под управлением Windows 2000, перекомпилировать ее и запустить. При этом вы оставите ее без изменения или совсем немного измените ее.

Несмотря на то, что теоретически для запуска профамм, написанных на языке программирования С, на нескольких компьютерах не требуется переделки профамм, в действительности вы должны будете сделать это. Однако изменение профамм, написанных на языке профаммирования С, проходит намного проще, чем изменение программ, написанных на языке ассемблера или на машинном языке.

Благодаря своей мощности и возможности переноса на другой компьютер, язык программирования С быстро завоевал популярность во всем мире, Большинство профамм написано на языке профаммирования С. Современные программы теперь пишутся на улучшенном варианте языка С — на языке C++. Некоторые великолепные (или очень плохие) профаммы, написанные на языках профаммирования С и C++, входят в состав таких операционных систем, как Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, UNIX и Linux, а также в такие программы, как Quicken, Netscape Navigator и Microsoft Word.

Несмотря на популярность языка профаммирования С, он обладает некоторыми недостатками, о которых нельзя не сказать.

- ✓ Программы, созданные на языке программирования С, достаточно велики и медленно работают, как и их аналоги, написанные на языке ассемблера или на машинном языке.
- ✓ Язык программирования С предоставляет программистам доступ ко всем элементам компьютера, включая возможность управления оперативной памятью компьютера. К сожалению, эта возможность может сыграть с вами дурную шутку. Если вы неправильно напишете программу, она может случайно повредить оперативную память компьютера, что приведет к поломке всего компьютера.

Предпринимая отчаянные попытки сделать программы, написанные на языке С, более надежными, программисты создали три языка программирования, похожих на язык С, а именно C++, *Java* и C#. Все три языка *объектно-ориентированные,* что позволяет программистам создавать маленькие программы, которые можно многократно использовать и изменять. При использовании программ, созданных на языках программирования Java и C#, исключено повреждение памяти, как это было при работе с языком программирования С.

Языки программирования высокого уровня

Поскольку программы, написанные на языке ассемблера и на машинном языке, были достаточно сложными и плохо работали, программисты создали языки программирования, все более приближающиеся к человеческому языку — FORTRAN, COBOL, Pascal и Ada. Создавая языки программирования, похожие на обычный человеческий язык, программисты надеялись, что появятся программы, которые будет легко писать и изменять.

Одним из первых появился язык программирования *FORTRAN* (название которого образовано от слов FORmula TRANslator). FORTRAN специально создан для выполнения математических расчетов. Другим языком программирования высокого уровня является *COBOL* (название образовано от слов COmmon Business-Oriented Language), созданный для обработки коммерческих данных. Так как каждый язык программирования предназначался для определенных целей, программисты не могли использовать языки FORTRAN и COBOL для написания видеоигр, операционных систем или программ для работы с текстом (попытайтесь сделать это, если сильно захотите).

Программирование до сих пор остается сложным для большинства людей, поэтому компьютерные гении создали языки программирования Pascal и BASIC, которые использовались для обучения людей программированию. Язык программирования *BASIC* (название образовано от слов Beginner's All-purpose Symbolic Instruction Code) создан для того, чтобы показать новичкам, как именно надо программировать. Начинающие программисты приступали к изучению программ, написанных на языке программирования С. Однако из-за сложности этого языка многие люди теряли уверенность в своих силах.

Основное преимущество языка BASIC очевидно. Для того чтобы отобразить на экране сообщение Take a nap! (Отдохни немного!), необходимо ввести только одну следующую команду:

```
PRINT "Take a nap!"
```

По сравнению с аналогичным исходным кодом программ, написанных на языке ассемблера или на машинном языке, исходный код программ, написанных на языке BASIC, позволяет вам сконцентрировать свое внимание именно на том, что необходимо сделать, а не на написании команд.

Язык программирования *Pascal* (язык назван в честь французского философа Блеза Паскаля, Blaise Pascal) также предназначен для того, чтобы помочь начинающим программистам понять, как надо программировать. Основное отличие между языками BASIC и Pascal состоит в том, что язык программирования Pascal позволяет создавать хорошо струк-

турированные программы, легкие для понимания. Вы легко сможете прочитать их, а также изменить в будущем. Программа для отображения на экране фразы Take a nap! (Отдохни немного!), написанная на языке Pascal, выглядит следующим образом:

```
Program Message (Input, Output);
Begin
  Writeln ('Take a nap!');
End.
```

В отличие от программы, написанной на языке программирования Pascal, программы, написанные на языке программирования BASIC, намного проще. Это позволяет быстрее писать программы, однако прочитать и понять большую программу будет достаточно сложно. Язык Pascal более структурированный, что позволяет составить схему программы еще до того, как вы начнете ее писать. Так вы поступаете, планируя свой летний отпуск. Это может занять некоторое время, но после такого планирования ваша программа и ваш отпуск будут лучше организованы. При этом вы не запутаетесь в программе, а приехав в Париж, вы не будете в полночь бродить по улицам из-за того, что забыли забронировать номер в гостинице. С другой стороны, язык программирования BASIC позволяет вам незамедлительно приступить к написанию программы.

Язык программирования BASIC настолько популярен, что программисты попытались объединить структурированность языка Pascal и простоту языка BASIC для создания различных диалектов языка BASIC. Liberty BASIC (о котором будет рассказано в главе 5) служит примером структурированной версии языка BASIC.

Как обычно, языки программирования высокого уровня, такие как Pascal, BASIC, FORTRAN, Ada и COBOL, обладают рядом недостатков, о которых я расскажу ниже.

- ✓ Программы, написанные на языках программирования высокого уровня, большие и медленно работают по сравнению с аналогичными программами, написанными на языке программирования C, языке ассемблера или на машинном языке.
- ✓ Языки программирования высокого уровня не обеспечивают доступ ко всем элементам компьютера, как это делает язык программирования C, язык ассемблера или машинный язык. Написание на этих языках таких программ, как операционные системы или утилиты для работы с дисками (например, набор утилит Norton Utilities), намного сложнее (но не невозможно).
- ✓ Языки программирования высокого уровня больше похожи на человеческий язык. Таким образом, написание компилирующих программ для таких языков — достаточно сложная задача. Если на вашем компьютере не установлена компилирующая программа для используемого вами языка программирования высокого уровня (например, для языка программирования Ada), вы не сможете написать работающую на вашем компьютере программу.

Язык программирования *Ada* — это сложный язык программирования высокого уровня. Компилирующие программы для языка программирования Ada для различных компьютеров занимают достаточно много времени. Можно найти очень немного компилирующих программ для языка программирования Ada, поэтому программисты редко пишут программы на этом языке. Язык программирования Ada никогда не станет таким популярным, как языки C, COBOL, FORTRAN, BASIC или Pascal.

Конечно, любой программист, использующий языки программирования высокого уровня Pascal, BASIC, FORTRAN, Ada и COBOL, приведет кучу преимуществ данных языков над языком программирования С, языком ассемблера или машинным языком. Ниже перечислены некоторые преимущества языков программирования высокого уровня.

- ✓ С помощью языков программирования высокого уровня вы сможете писать программы намного быстрее, чем на языке ассемблера или на машинном языке. (Можно написать программу на языке программирования С, которая будет выполнять то же, что и программа, написанная на языке программирования высокого уровня.)
- ✓ Изучение языков программирования высокого уровня требует меньше времени, чем изучение машинного языка, языка ассемблера или языка С.
- ✓ Поскольку языки программирования высокого уровня не предоставляют доступ ко всем элементам компьютера, вы не сможете написать программу, которая приведет к поломке компьютера.
- ✓ Чтение программ на языках высокого уровня и их изменение намного проще, чем в аналогичных программах, написанных на языке программирования С, языке ассемблера или на машинном языке.
- ✓ Программы, написанные на языках программирования высокого уровня, можно запускать на различных компьютерах. Написав программу на языке программирования высокого уровня, вы (теоретически) легко перенесете ее на любой компьютер.

Языки программирования для быстрой разработки приложений RAD

Основная часть языков программирования создана в те времена, когда компьютеры не могли отображать ничего, кроме текста. Они не умели отображать графику, указатели мыши или окна.

Из-за того что на экране компьютера можно отобразить только текст, языки программирования C++, BASIC и Pascal используют простые команды. Это видно из следующего примера:

```
PRINT "This sentence appears on-screen."
```

Как только на экране появились окна, полосы прокрутки и панели задач, программисты захотели создавать программы, в которых были бы все эти элементы. Для того чтобы удовлетворить эти требования, разработчики языков стали создавать диалекты уже существующих языков, которые они окрестили *языками программирования для быстрой разработки приложений RAD*.

Языки RAD позволяют создавать программы, в которых есть всевозможные элементы (включая интерфейс пользователя). На рис. 2.1 показан интерфейс Visual Basic.

Есть три основных языка RAD: *Visual BASIC* (созданный на основе языка BASIC), *Delphi* (созданный на основе языка Pascal) и C++ *Builder* (созданный на основе языка C++).

Языки RAD обладают следующими преимуществами.

- ✓ Благодаря графическому интерфейсу написание программ стало намного быстрее.
- ✓ Языки RAD упрощают сознание пользовательских интерфейсов. Теперь можно заниматься самой программой, а не ее внешним видом. Раньше, до языков RAD, приходилось писать инструкции для создания пользо-

вательского интерфейса, а только затем приступать к созданию самой программы. Это приводило к потере времени и ошибкам в программе.

- ✓ Благодаря тому, что языки RAD созданы на основе языков программирования высокого уровня (BASIC, Pascal и C++), зная эти языки, вы без труда сможете программировать на языках RAD.

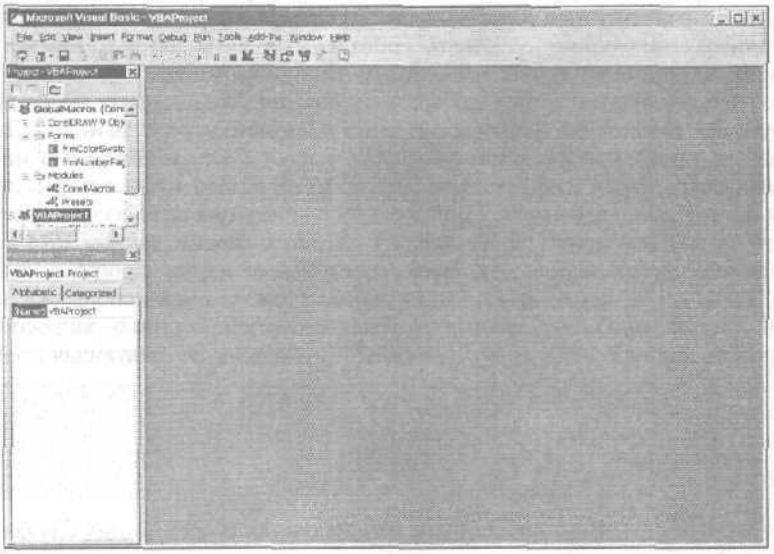

Рис. 2.1. Visual Basic позволяет создавать интерфейс пользователя с помощью простых команд

Как ни странно, языки RAD также имеют ряд недостатков. Это не должно вас удивлять, просто в мире нет совершенства.

- ✓ Программы, написанные на языках RAD, нельзя переносить с одного компьютера на другой. Например, язык Visual Basic работает только с Windows 95, Windows 98, Windows Me, Windows NT и Windows 2000. Написанные в Visual Basic программы не будут работать на компьютерах Macintosh, с операционной системой Linux или с другими. Для того чтобы они заработали, их придется существенно изменить.
- ✓ Программы, созданные на языках RAD, больше и работают намного медленнее, чем аналогичные программы, написанные на языках программирования C++, BASIC или Pascal. С помощью языков RAD создание программ ускорилось. Но пришлось пожертвовать скоростью и размером программы.

Языки программирования баз данных

Языки программирования, такие как C++, BASIC или Pascal, создавались как языки на все случаи жизни. С их помощью можно написать программу авиасимулятора, бухгалтерскую программу, программу для речевого ввода текста или текстовый процессор.

Однако чаше всего компьютер используется для хранения различной информации: имен, адресов, телефонных номеров, разных документов, а также выполненной работы. Компьютер хранит такую информацию в базе данных.

Успешное проведение почти любой деловой сделки зависит от информации, хранящейся в базе данных: сведения о заказчиках, их финансовое состояние, штат сотрудников.

Из-за сложностей в работе с базами данных основная часть пользователей не пользуется ими. Для того чтобы упростить использование баз данных, почти все базы содержат в себе язык программирования.

При работе с базой данных, в которой используется язык программирования, можно создавать собственные базы данных. Лучше всего то, что языки программирования баз данных позволяют создавать собственные базы данных намного быстрее, чем с помощью универсальных языков программирования, таких как C++ или BASIC. При использовании языка программирования базы данных необходимо написать инструкции только для управления информацией в базе данных. Для универсальных языков программирования, например языка C++, необходимо написать инструкцию для сохранения информации. Затем необходимо написать инструкцию для управления данной информацией. При этом тратится в два раза больше времени.

Основная часть популярных программ для работы с базами данных *dBASE, FileMaker, FoxPro* и *Microsoft Access* используют свои собственные языки программирования. Для управления большим объемом данных программы базы данных используют язык, называемый *SQL* (Structured Query Language — Язык структурированных запросов). Для того чтобы отобразить надпись Take a nap!, в dBASE используется следующая программа:

```
row = 15
column = 15
clear
@ row, column SAY "Take a nap!"
```

Языки программирования баз данных используются в следующих случаях.

- ✓ С помощью языка программирования базы данных можно написать программу для сохранения определенной информации намного быстрее, чем с помощью универсальных языков программирования, таких как C++ или Pascal.
- ✓ Создавать базы данных очень выгодно. Научившись создавать собственные базы данных, вы не будете тратить деньги на их приобретение.

Конечно, языки программирования баз данных нельзя использовать во всех случаях жизни. Они обладают несколькими серьезными ограничениями.

- ✓ Программы базы данных часто неразрывно связаны с определенным компьютером. Например, если вы создадите собственную базу данных, используя программу FileMaker, она будет работать только на компьютере, на котором запущена программа FileMaker. Из-за того, что в настоящее время программа FileMaker работает только с Windows и Macintosh, ее нельзя запустить на компьютерах, работающих под управлением Linux.
- ✓ Языки программирования базы данных удобны для создания собственных баз данных. Однако для создания других программ, таких как игры, текстовый процессор, различные утилиты (например, антивирусная утилита), можно использовать другие способы. Языки программирования базы данных не всегда подходят для этих целей.

Языки программирования для создания сценариев

Написание собственных программ позволяет вам чувствовать себя независимым, однако это занимает много времени и сил. Предположим, что вы решили написать текстовый процессор специально для написания сценариев.

Если вы будете использовать универсальные языки программирования C++ или Pascal, сначала придется написать инструкции для создания простого текстового процессора, затем написать инструкции, которые добавят в текстовый процессор параметры, необходимые для создания и форматирования сценариев.

Вместо того чтобы делать это все самому, лучше воспользоваться программами, в которых есть свои собственные языки для создания сценариев. Вам не надо создавать текстовый процессор с нуля, достаточно просто купить уже существующую программу (например, WordPad, WordPerfect или Microsoft Word) и затем воспользоваться языком программирования для создания сценариев для текстовых процессоров. Это сделает текстовый процессор подходящим для выполнения поставленной задачи (в нашем случае это написание и форматирование сценариев). Язык для создания сценариев позволяет бросить все силы на выполнение задания и не отвлекаться на ненужные мелочи.

Основная часть программ, созданных компанией *Microsoft*— Word, Excel, PowerPoint и Access, — включают в себя язык для создания сценариев, называемый *Visual Basic for Applications (VBA)*, который немного похож на Visual Basic. В операционной системе для компьютеров Macintosh также есть язык программирования для создания сценариев, называемый *Apple Script*. Так что написание программ на этих компьютерах также можно автоматизировать. В следующем примере показано, как с помощью программы AppleScript отобразить на экране сообщение Take a nap! (Отдохни немного!):

```
On DisplayMessage()
  display dialog "Take a nap!" (buttons  {"OK"}
end DisplayMessage
DisplayMessage()
```

Языки программирования для создания сценариев удобно использовать по следующим причинам.

- ✓ Подобные языки позволяют изменять уже существующие программы, такие как текстовый процессор или программа для создания электронных таблиц. Благодаря этому можно создавать более сложные программы, затратив достаточно мало времени.
- ✓ Языки программирования для создания сценариев намного легче, чем более мощные универсальные языки программирования (например, C++). Чем быстрее вы разберетесь с языком, тем быстрее начнете писать программы.

Однако, перед тем как сломя голову бросаться изучать языки программирования для создания сценариев, обратите внимание на следующие проблемы.

- ✓ Языки программирования для создания сценариев связаны с определенной программой. Если вы создадите собственный текстовый процессор, воспользовавшись таким языком, он будет работать только на компьютере, на которых запущен данный текстовый процессор. Так что, создав свой собственный Microsoft Word, вы сможете воспользоваться им только на компьютерах, на которых может запускаться Microsoft Word (на компьютерах, работающих под управлением Windows, и на компьютерах Macintosh).
- ✓ Вам будет очень сложно продать свою программу. Ведь для того, чтобы воспользоваться вашей программой, пользователи должны приобрести или уже иметь программу (текстовый процессор, программа для создания электронных таблиц и т.д.), на основе которой вы создали свою. Если вы создадите программу на основе WordPerfect, пользователи, использующие программу Microsoft Word, не смогут воспользоваться вашей программой.

> ✓ Языки программирования для создания сценариев не обеспечивают все возможности, предоставляемые универсальными языками программирования (например, C++). Поэтому используйте языки программирования для создания сценариев только в том случае, если вас удовлетворяют их скромные возможности.

Языки программирования для создания Web-страниц

На заре распространения Internet люди общались с помощью обычных текстовых документов, в которых не было графики, анимированных рисунков и разноцветных текстов, к которым мы привыкли, глядя на современные Web-страницы. И хотя люди все равно читают текстовые документы, иногда просто сложно разобраться в них, если для поиска необходимой информации приходится прокручивать текст.

Для того чтобы избежать этих проблем и привлекательнее оформить текст, программисты создали язык *HTML* (HyperText Markup Language). Благодаря ему на Web-страницах появились графические элементы.

Код HTML говорит броузеру, как правильно отобразить страницу. Поэтому всякий раз при использовании броузера для отображения Web-страницы он автоматически преобразует код HTML (рис. 2.2) в великолепные картинки, показанные на рис. 2.3.

В определенный момент времени пользователям надоело смотреть на Web-страницы, которые напоминали доску объявлений, размещенную на экране монитора. Для того чтобы Web-страница смогла "общаться" с пользователем (при игре, заполнении различных форм и т.д.), программисты придумали специальные языки программирования для создания Web-страниц: *Java, JavaScript* и *VBScript.*

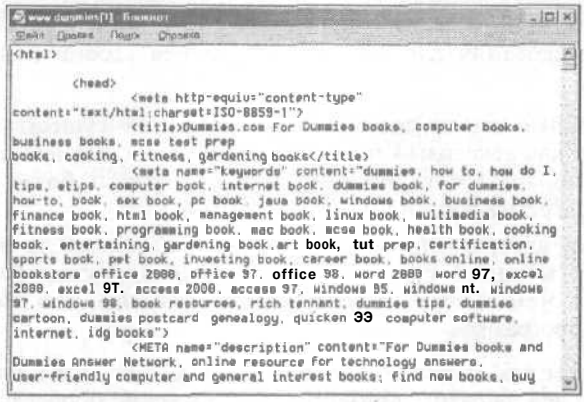

Рис. 2.2. Код HTML очень трудно прочитать, а еще труднее понять

Язык Java позволяет создавать два типа программ — автономные приложения (игры или текстовые процессоры) и небольшие аплеты, запускаемые на Web-странице. В следующем примере показано, как выглядит приложение, написанное на языке Java, с помощью которого можно отобразить на экране фразу Take *a* nap! (Отдохни немного!):

```
Public class DisplayMessage {
  public static void main (String args[]) {
    system.out.println("Take a nap!");
  }
}
```

Рис. 2.3. Ход HTML, показанный на рис. 2.2, определяет, как выглядит эта страница

Языки программирования для создания Web-страниц позволяют создавать Web-узлы, которые больше напоминают игры, а не отсканированное с бумаги изображение. Такие интерактивные Web-страницы можно разместить на Web-узле для того, чтобы пользователи повторно вернулись на этот узел.

Языки программирования для создания Web-страниц обеспечивают следующие преимущества.

- ✓ Можно создавать Web-страницы, выглядящие очень занимательно, что заставит пользователей чаще обращаться к данной странице.
- ✓ Языки программирования для создания Web-страниц достаточно просто выучить. Созданные с их помощью Web-страницы будут доступны всем пользователям Internet.

При использовании языков программирования для создания Web-страниц вы можете столкнуться со следующими проблемами.

- ✓ Не все броузеры поддерживают все особенности языков программирования для создания Web-страниц, таких как JavaScript и VBScript. В результате пользователи старых броузеров не смогут запустить программы, написанные вами с помощью языков программирования для создания Web-страниц.
- ✓ Пользователи, которые подключаются к Internet на низкой скорости (используя модемы со скоростью передачи данных не более 28,8 Кбит/с), будут испытывать трудности с запуском программ, созданных на языках программирования для создания Web-страниц (например, VBScript). Данные программы будут запускаться очень медленно, что отпугнет посетителей вашей Web-страницы.
- ✓ Созданные с помощью таких языков (например, с помощью языка Java) программы смогут запустить только пользователи, имеющие доступ к Internet. Если вы думаете, что ваша программа будет нужна еще кому-либо, кроме вас, не пользуйтесь языками программирования для создания Web-страниц.

Какой же язык изучать?

Самого лучшего языка не существует. Если вы собираетесь стать профессионалом в написании программ, вам необходимо изучить один из языков программирования высокого уровня (наиболее популярен язык программирования C++), а также один из языков программирования баз данных (например, SQL). Изучив язык программирования C++, вы не ошибетесь.

Зная этот язык, вы всегда сможете найти работу в любой компании, занимающейся программированием.

Несмотря на большую популярность языка программирования C++, часто используются и другие языки. На многих устаревших компьютерах до сих пор работают программы, написанные на языке программирования COBOL. Поэтому нужны программисты, которые умеют усовершенствовать данные программы, а также писать новые. Очень часто крупные компании выплачивают таким программистам высокую заработную плату.

Если вы собираетесь работать самостоятельно, предпочтительнее всего научиться создавать собственные программы для баз данных. Для этого вам понадобится изучить такие языки программирования, как SQL или VBA, которые используются в программе Microsoft Access. Для того чтобы создавать Web-страницы, необходимо знать HTML, а также немного знать Java, JavaScript, VBScript и другие языки программирования для Internet. Самым нужным будет тот язык программирования, который позволит решить поставленные перед вами задачи легко и быстро. Это может быть язык программирования C++, BASIC, Java, SQL или язык ассемблера.

Для того чтобы познакомиться с тем, как работают различные языки программирования, посетите Web-узел `www.latech.edu/~acm/HelloWorld.shtml`. Здесь вы найдете несколько примеров программ, написанных на различных языках программирования. С помощью всех этих программ на экране монитора отображается надпись "Hello World!".

Глава 3

Как написать программу

В этой главе...

- Проектирование программы
- Некоторые технические детали
- Выбор языка программирования
- Как должна работать программа
- Жизненный цикл типичной профаммы

Хотя вы можете, сидя перед компьютером, приступить к написанию профаммы прямо сейчас, обойдясь безо всякого планирования, результат будет таким же плачевным, как и при приготовлении бисквита смешением инфедиентов в произвольной пропорции.

Вы можете написать очень простую программу, отображающую на экране фотографию любимого кота, без предварительной подготовки, но для какого-нибудь проекта посложнее без предварительной подготовки вам не обойтись. После того как вы убедитесь в том, что точно знаете, что именно должна выполнять профамма и что должно отображаться на экране, приступайте к написанию собственно профаммы, выполняющей определенное задание.

Прежде чем писсипь программу

Если вы определили структуру программы, прежде чем приступили к ее разработке, вы не потратите время впустую на написание профаммы, которая не работает или работает неправильно, не решая поставленную перед ней задачу. Планируя, вы значительно увеличите вероятность того, что ваша профамма будет работать правильно и справится с поставленной перед ней задачей.

При проектировании профаммы вам следует обращать внимание на следующих три вопроса.

- ✓ Пользователь. Кто будет использовать вашу профамму?
- ✓ Целевой компьютер. Какой компьютер необходим для запуска вашей программы? Это должен быть компьютер, работающий под управлением Windows 95/98/Me/NT/2000 или Mac OS, компьютер Amiga, мэйнфрейм, карманный компьютер, работающий под управлением Palm OS или PocketPC, или настоящий суперкомпьютер?
- ✓ Вы. Вы планируете написать программу самостоятельно или собираетесь воспользоваться помощью других? Если вы хотите обратиться за помощью, кто будет отвечать за написание той или иной части программы?

Пользователи программы

Если единственным пользователем вашей профаммы будете вы сами, вам просто создать профамму, которая будет выглядеть и работать только так, как хотите именно вы, поскольку вы точно знаете все нюансы ее работы. Но если вы хотите предостав-

лять или продавать программу другим, вы обязательно должны знать, кто будет пользователем вашей программы.

Знание предпочтений потенциальных пользователей вашей программы оказывается очень критичным. Если по какой-то причине ваша программа придется не по душе пользователям, они вряд ли будут ее использовать. То, как именно работает программа, очень часто оказывается неважным.

Подойдя к проектированию программ с учетом особенностей потенциальных пользователей программы, вы значительно увеличите шансы программы на успех среди пользователей, а значит, вы получаете возможность заработать на продаже копий программы.

Даже если вы напишете безупречно работающую программу, пользователи все равно будут ее игнорировать, если им не понравится интерфейс программы, им непонятно, как заставить программу выполнить то или иное действие, она работает не так, как программа, к которой они уже привыкли, в интерфейсе программы используются неудачные цветовые комбинации и т.д. Основная задача программиста — написать программу, которая будет максимально соответствовать пожеланиям ее пользователей, независимо от того, насколько нелогичными они могут выглядеть. (Пожелания, а не пользователи, конечно же.)

Целевой компьютер

Определив, кто же потенциальный пользователь вашей программы, необходимо определить, какой компьютер потребуется этому пользователю для запуска вашей программы и работы с ней. Тип компьютер в значительной мере определяет то, какой язык программирования вы будете использовать для написания программы, с каким аппаратным обеспечением придется иметь дело вашей программе, а также каков максимальный объем вашей программы.

Например, если вы пишете программу, которая будет запускаться на компьютере Macintosh, программа может рассчитывать на такие преимущества, как отличные звуковая и графическая подсистемы, большой жесткий диск и значительный объем памяти. Однако, если вам необходимо, чтобы программа работала на карманном компьютере Palm, вам придется переписать ее практически с нуля, так как возможности такого компьютера при воспроизведении звука и отображении графики и обьем оперативной памяти значительно скромнее.

Если вы можете спокойно скопировать свою программу с компьютера одного типа на компьютер другого типа вообще безо всяких или с совсем небольшими изменениями, вашу программу можно считать *переносимой*. Подобные характеристики программы в значительной мере зависят от того, какой язык использовался для ее написания. Именно по этой причине такие языки программирования, как C и C++, используются гораздо чаще, чем другие.

Переносимость и кроссплатформенность

Вместо того чтобы использовать компьютеры одного типа, многие программисты стараются создавать такие программы, которые смогут работать на многих типах компьютеров, например компьютерах, работающих под управлением Mac OS и Windows 95/98/Me/NT/2000. Любая профамма, которая работает на компьютерах двух типов, считается *кроссплатформенной*. Например, профамма Microsoft Word кроссплатформенная, поскольку вы можете приобрести ее версию как для Mac GS, так и для Windows.

Программа, которая способна работать на компьютерах нескольких типов, значительно расширяет круг ваших потенциальных заказчиков, однако и приводит к увеличению числа потенциальных проблем, с которыми вы рискуете столкнуться. Например, вам придется

предоставлять техническую поддержку пользователям каждой версии вашей профаммы или пытаться заставить все версии программы работать совершенно одинаково, несмотря на то, что им приходится работать под управлением различных операционных систем и на компьютерах с совершенно различными возможностями.

Например, в свое время существовали версии профаммы WordPerfect для MS DOS, Windows, Macintosh, Amiga и Atari ST. Это приводило к необходимости нанимать очень много программистов для написания каждой версии текстового процессора; помимо того, было необходимо обеспечивать поддержку пользователей всех этих версий. Не стоит и говорить, что все это приводило к огромным расходам компании. Разработка многочисленных версий профаммы и оказание технической поддержки значительно сказывались на прибылях компании, что привело к прекращению разработки и оказания технической поддержки версий WordPerfect для таких платформ, как Amiga и Atari ST.

Ваш собственный уровень программирования

При проектировании любой программы вам стоит не забывать о собственном уровне подготовки программиста. Вы можете вынашивать просто гениальные идеи, но, если вы начинающий программист с совсем небольшим опытом, на написание программы может уйти уйма времени (если вы не бросите это раньше, отчаявшись).

Ваш уровень подготовки и опыт написания программ определяют, какому именно языку программирования вы отдадите предпочтение. Опытные программисты практически всегда пишут программы на C или C++. Однако новичкам придется или потратить массу времени на изучение этих языков программирования, прежде чем они смогут писать программы, или же им использовать более простой язык программирования, например BASIC.

Некоторые новички тратят время на изучение сложных языков, таких как C или C++, после чего приступают к написанию программ. Другие предпочитают более легкий подход и выбирают простой язык программирования, например Visual Basic, чтобы немедленно приступать к созданию (и продаже) программ. Не бойтесь начинать с изучения сложного языка программирования, но не стоит и избегать языка попроще. Основная ваша цель — написать программу, которую вы сможете или использовать сами, или продавать другим.

Не забывайте о льготах

Вместо того чтобы изучать программирование самостоятельно, многие люди предпочитают нанимать других для написания программ. Однако будьте осторожны! Многие программисты — свободные художники живут по принципу "заплатите много, а как закончу работу — хоть потоп".

Вот как это происходит. Вы нанимаете кого-нибудь для написания необходимой программы; программист получает деньги. Затем этот человек пишет программу, которая работает не совсем так, как вы того хотели. Вместо того чтобы потерять деньги, уже вложенные в разработку программы, вы платите программисту дополнительную сумму, после чего он пишет новый вариант программы, который все равно работает несколько не так, как должно было бы быть.

И вы оказываетесь в затруднительном положении. Будете ли вы продолжать платить деньги программисту, который никогда не выполнит свою работу, или вы смиритесь с потерей средств. Еще хуже то, что вы не сможете нанять нового программиста для продолжения работы над программой, так как ее исходный код находится у первого программиста, ее автора, а значит, никто другой не сможет внести в нее изменения. Единственный способ внести изменения в программу — снова нанять автора профаммы и платить ему и т.д.

Многие программисты пишут свои программы на таком простом языке программирования, как Visual Basic, а затем нанимают программистов по-опытнее, чтобы те переписали программы на более сложном языке программирования, таком как C или C++, благодаря чему программа работает быстрее и эффективнее.

Написание программы: технические подробности

Совсем немногие люди создают программы в один прием. Основная часть программ эволюционирует со временем. Процесс ввода команд требует очень много времени и при этом часто возникают проблемы, поэтому программисты пытаются избегать действительного написания программы до тех пор, пока не будут абсолютно уверены в том, что они знают, что делают.

Создание прототипов

Для того чтобы убедиться в том, что они не потратят месяцы (или даже годы) на создание программы, которая не будет работать должным образом, программисты очень часто начинают с создания *прототипа* программы. Точно так же, как архитекторы начинают с создания картонных или пластиковых моделей небоскребов, прежде чем приступать к реальному строительству, программисты начинают с макета (прототипа) будущей программы.

Прототип программы обычно уже содержит необходимый интерфейс пользователя, такой как выпадающие меню и диалоговые окна. Прототип выглядит как настоящая программа, однако выбор команд меню не приводит ни к каким результатам. Основная идея создания прототипа — показать, как будет выглядеть программа и как она будет действовать, обойдясь без написания действительных команд, которые она выполняет при работе.

Получив прототип программы в том виде, в котором программист себе его представлял, он может продолжить свою работу, используя прототип при создании окончательного варианта программы.

Многие программисты используют Visual Basic, поскольку этот язык программирования позволяет очень легко и быстро создавать прототипы программ. Воспользовавшись Visual Basic для создания прототипа, который продемонстрирует работу интерфейса пользователя вашей программы, смело приступайте к добавлению команд и превращению прототипа в настоящую полноценную программу.

Выбор языка программирования

Определив прототип, который должным образом проиллюстрирует внешний вид программы, на следующем шаге вы выбираете наиболее подходящий язык программирования.

Вы можете написать любую программу на любом языке программирования. Хитрость в том, что некоторые языки программирования позволяют создавать определенные типы программ намного проще, чем другие.

Выбор того или иного языка программирования так же сильно разграничивает людей, как и принадлежность к определенной религии или политической партии. Хотя вы никогда не сможете найти "совершенно идеальный" язык программирования,

подходящий для любой ситуации, рассмотрите несколько языков программирования. Ведь если ваша программа работает, абсолютно никого не будет интересовать, какой язык программирования вы использовали при ее создании. В следующих подразделах я рассматриваю потенциальные причины использования при написании программ того или иного языка программирования.

C/C++

Язык программирования C или C++ может похвастаться серьезной поддержкой, независимо от используемых типа компьютера и операционной системы, поэтому вы никогда не ошибетесь, если выберете при написании программ один из этих языков программирования. Основные преимущества языков программирования C и C++ как средств для создания программ перечислены ниже.

- ✓ **Эффективность.** Язык программирования C/C++ позволяет создавать небольшие и быстро выполняемые программы по сравнению с другими языками программирования (за исключением языка ассемблера или машинных кодов).
- ✓ **Переносимость.** Если вы написали программу на C/C++, вы сможете легко скопировать ее на другой компьютер и запустить ее, внеся в нее некоторые (как правило, незначительные) изменения. Таким образом, вы можете написать одну программу, немного откорректировать ее для разных платформ и получить намного более широкий круг потенциальных заказчиков.
- ✓ **Большое число программистов.** Благодаря тому, что язык программирования C/C++ известен очень многим программистам, у вас никогда не возникнет проблем с поиском программиста для внесения изменений в программу при необходимости в дальнейшем.

Программисты создают коммерческие программы на C/C++, но многие программисты при создании собственных программ выбирают другие языки программирования, поскольку им не нравятся следующие характеристики C/C++.

- ✓ **Сложность в освоении.** Язык программирования C/C++ — один из самых сложных языков программирования для освоения. К тому времени как вы полностью изучите C/C++, вы уже давно завершили бы написание программы, воспользовавшись для ее создания другим языком программирования.
- ✓ **Сложность для прочтения и понимания.** Программисты гораздо чаще вносят изменения в готовые программы, чем создают новые. В связи с природой языка C/C++ порой чрезвычайно сложно разобраться с текстом готовой программы, чтобы внести в нее необходимые изменения.
- ✓ **Сложность.** Язык программирования C/C++ предоставляет вам мощь и гибкость при манипулировании памятью и непосредственным доступом к оборудованию компьютера. Это не только повышает вероятность того, что ваша программа сделает нечто такое, чего вы совсем не предполагали (например, займет всю доступную память компьютера, что не позволит нормально работать другим программам), но и увеличивает время, которое вы потратите на отладку программы, пока не убедитесь в ее корректном функционировании. (Об ошибках компьютерных программ я подробно поговорю в главе 15, "Отладка программ".)

Программисты пишут коммерческие программы для Windows и Macintosh на C/C++, так как эти языки программирования обеспечивают скорость, эффективность и переносимость. Если эти характеристики программы для вас очень важны, ваш единственный выбор — язык программирования C/C++.

Visual Basic

Visual Basic — один из самых популярных языков программирования для написания программ (после C/C++). Несмотря на то, что серьезные программисты отдают предпочтение C/C++, многие программисты используют Visual Basic по целому ряду причин.

- ✓ **Простота в изучении. Вы** можете изучить Visual Basic и приступить к написанию программ намного быстрее, чем при выборе любого другого языка программирования. Чем быстрее вы изучите язык программирования, тем быстрее вы сможете создавать (и продавать) программы.
- ✓ **Возможность быстрого создания прототипов.** Используя Visual Basic, вы очень быстро создадите прототипы. После этого вам просто превратить прототип в настоящую работающую программу. Другие языки программирования намного сложнее при создании прототипов, а значит, вам лучше создать прототип, а затем создавать полноценную работающую программу с нуля.

Однако профессиональные программисты часто называют Visual **Basic** "игрушечным" языком программирования по целому ряду причин.

- ✓ **Медленное выполнение программ.** Программы, написанные на Visual Basic, обычно работают намного медленнее, чем программы, написанные на других языка программирования, например C/C++. Если скорость выполнения программы для вас очень важна, на языке программирования Visual Basic останавливаться не стоит.
- ✓ **Неэффективность.** Программы, написанные на Visual Basic, обычно занимают огромное дисковое пространство, даже если речь идет о совсем небольшой программе. Если вам необходима программа, которая будет занимать как можно меньше места на диске, на языке программирования Visual Basic останавливаться не стоит.
- ✓ **Негибкость.** Язык программирования Visual Basic потому и прост в изучении, что скрывает от вас многие технические подробности взаимодействия программы с компьютером. В то же время он не дает вам полного контроля над компьютером, что значительно снижает возможности программы.
- ✓ **Ограниченная переносимость.** Visual Basic работает только на компьютерах, работающих под управлением Windows 95/98/Me/NT/2000 или PocketPC, следовательно, программы, написанные с помощью этого языка программирования, на компьютерах других типов работать не будут. Попытка копирования и запуска программы Visual **Basic** на компьютере Macintosh окажется достаточно проблематичной, что вряд ли увенчается успехом, поэтому вам лучше потратить время на создание программы с нуля на C/C++.

И профессионалы, и новички используют Visual Basic для создания коммерческих программ и программ для личных нужд. Если вам необходимо быстро написать программу, но нет времени на изучение C/C++, используйте Visual Basic.

Delphi

Язык программирования Delphi очень похож на Visual Basic, в качестве базового языка используется Pascal. Хотя Delphi не так популярен, как Visual Basic или C/C++, у этого языка программирования есть круг стойких поклонников. Можно сказать, что в Delphi объединены все лучшие (и худшие) черты Visual Basic и C/C++. Для использования Delphi есть целый ряд причин.

- ✓ **Простата в изучении.** Несмотря на то, что Delphi не настолько прост в изучении, как Visual Basic, его все равно намного проще освоить, чем C/C++, поэтому можно приступать к созданию программ намного раньше.
- ✓ **Быстрое создание прототипов.** Delphi позволяет создавать прототипы практически так же быстро, как и Visual Basic. После этого вы сможете превратить прототип в работающую программу, как и при использовании Visual Basic.
- ✓ **Мощность.** Delphi обеспечивает практически такую же мошь и гибкость, как и C/C++, обходясь при этом без основных недостатков последних. Программы, написанные на Delphi, практически так же быстры и малы, как и программы, написанные на C/C++. Delphi ограждает вас от сложностей взаимодействия программы и компьютера (в отличие от C/C++), поэтому вероятность того, что вы напишете программу, которая "съест" всю доступную память компьютера, достаточно мала.

Несмотря на то, что программисты часто считают Delphi удачнее, чем Visual Basic, и практически таким же мощным, как и C/C++, у этого языка программирования есть целый ряд недостатков.

- ✓ **Меньшая популярность.** Delphi базируется на языке программирования Pascal, который гораздо менее популярен, чем C/C++. Поэтому поиск программиста, который сможет при необходимости внести изменения в программу, написанную на Delphi, весьма проблематичен.
- ✓ **Ограниченные возможности.** Delphi ограждает вас от сложностей взаимодействия программы и компьютера, а значит, ограничивает ваши возможности контроля за работой компьютера. Если вам необходим полный контроль над оборудованием компьютера, без C/C++ вам не обойтись,
- ✓ **Ограниченная переносимость.** Delphi работает только на компьютерах, работающих под управлением Windows 95/98/Me/NT/2000 или Linux, следовательно, программы, написанные с помощью этого языка программирования, на компьютерах других типов работать не будут. Попытка копирования и запуска программы Delphi на компьютере Macintosh окажется достаточно проблематичной, что вряд ли увенчается успехом, поэтому вам лучше потратить время на создание программы с нуля на C/C++.

Опытные программисты часто способны на Delphi создать программы, которые будут работать быстрее, чем программы, написанные начинающим программистом на C/C++. Однако практически любые профаммы на Delhi работают гораздо быстрее своих аналогов, написанных на Visual Basic. Поэтому программисты часто используют Delphi, если им не нужно обеспечить переносимость программ между компьютерами различных типов.

Java

Java — это относительно новый язык программирования, который быстро стал популярным. Используя Java, вы сможете создавать не только полноценные профаммы, но и мини-программы (называемые *аплетами*), предназначенные для выполнения в

Internet. Несмотря на молодость этого языка программирования, многие программисты отдают ему предпочтение по целому ряду причин.

- **Феноменальная переносимость.** Java — это совершенно переносимый язык программирования, поэтому любая написанная с его помощью программа (по крайней мере, теоретически) может работать под управлением любой операционной системы (Windows, Macintosh или Linux) без каких-либо изменений.
- **Более надежный,** чем C/C++. Java унаследовал все преимущества C/C++, избежав при этом его основных недостатков. Программы Java деликатнее подходят к использованию памяти компьютера и реже приводят к зависанию компьютера, чем аналогичные программы, написанные на C++.
- **Похожесть на** C/C++. Поскольку Java очень похож на C/C++, любой программист, знающий C/C++, может быстро приступить к созданию программ на Java.

Язык Java относительно нов, поэтому компания *Sun Microsystems* (создавшая этот язык программирования) продолжает его совершенствовать. Несмотря на все достоинства, языку программирования Java свойственны и некоторые недостатки.

- **Медленный и менее эффективный.** Программы, написанные на Java, обычно работают медленнее, чем аналогичные программы, написанные на C/C++, однако эта ситуация постепенно улучшается по мере появления более эффективных компиляторов Java.
- **Сложность в изучении.** Java очень напоминает C/C++, но сложен в изучении. Если вы хотите как можно быстрее приступить к написанию программ, выбирайте Visual Basic или даже Delphi.
- **Необходимость тестирования.** Теоретически, программы, написанные на Java, способны работать на любых компьютерах безо всяких изменений. В действительности вы должны проверять свои программы на Java на различных типах компьютеров, чтобы убедиться в их корректном функционировании. Из-за этого вам придется потратить больше времени на тестирование, хотя вы могли бы потратить это время на дальнейшее совершенствование программы.

Язык программирования Java все еще развивается, поэтому многие компании заняли выжидательную позицию, а не сразу приступили к написанию программ на Java. Однако, если самым важным критерием для вас является переносимость программы, вам стоит предпочесть Java таким языкам программирования, как Visual Basic и Delphi — и даже C/C++.

Специализированные языки программирования

В качестве альтернативы использованию языков программирования общего назначения, таких как C/C++, Visual Basic, Delphi и Java, стоит подумать об использовании специализированного языка программирования, например LISP и Prolog; они интенсивно используются в программах искусственного интеллекта. Кроме того, существуют специализированные языки программирования для быстрого создания баз данных, такие как Clarion и PowerBuilder.

Если вы создаете программу, которой при работе нужно обращаться к базе данных, использование готовой СУБД (например, Microsoft Access) и использование ее встроенного языка написания сценариев для создания программы окажется намного проще, чем специализированного языка программирования.

В зависимости от выбранного приложения специализированный язык программирования полезен по целому ряду причин.

- **Быстрая разработка.** Специализированный язык программирования предназначен для решения узкого круга проблем. Написание программы с помощью специализированного языка программирования позволит намного быстрее получить результат, чем при использовании другого языка программирования общего назначения, в том числе и Visual Basic.
- **Простота.** В отличие от языков программирования общего назначения, специализированные языки программирования позволяют намного проще создавать программы определенного типа. Если вы найдете подходящий язык программирования, вы завершите работу намного быстрее, чем при использовании C/C++ или Visual Basic.

Прежде чем вы выберете тот или иной специализированный язык программирования, не забывайте о существовании следующих проблем,

- **Ограниченная переносимость.** Многие специализированные языки программирования не позволяют создавать программы, которые выполнялись бы на различных компьютерах без внесения значительных изменений. В некоторых случаях запуск программы, написанной на специализированном языке программирования, на компьютере другого типа вообще невозможен.
- **Медленное выполнение.** Программы, написанные на специализированном языке программирования, обычно выполняются намного медленнее, чем аналогичные программы, написанные с помощью языка общего назначения, например C/C++.
- **Ограниченная распространенность.** Специализированные языки программирования известны намного меньшему кругу специалистов, чем языки общего назначения, что значительно усложняет внесение изменений в программу при возникновении такой необходимости (а если честно, делает это практически невозможным).
- **Сложность распространения.** Если вы написали программу, используя специализированный язык программирования, например dBASE или FileMaker, распространение программы значительно усложняется, а иногда оказывается и полностью невозможным. Например, если вы написали программу, используя Microsoft Access, вам придется распространять вместе с программой дополнительный компонент, приобретаемый отдельно. В результате программа будет работать медленнее и занимать больше места на диске.

В зависимости от поставленной перед вами задачи специализированные языки программирования помогут вам написать программу намного быстрее и проще, чем при использовании любого языка программирования, включая Visual Basic. Следите только за тем, чтобы преимущества специализированного языка программирования преобладали над его недостатками.

Если вы написали программу с помощью специализированного языка программирования, она полностью от него зависит. Всегда помните о том, что возможности программы ограничены возможностями языка программирования, использованного при ее создании.

Использование нескольких языков программирования

Вместо того чтобы писать целую программу с помощью одного языка программирования (например, C++), некоторые компиляторы способны преобразовать исходный код в специальный файл, который называется *объектным файлом.* Основное назначение объектных файлов состоит в том, что один программист может использовать C++, второй — язык ассемблера, а третий — Pascal Каждый из них пишет свою часть программы на своем любимом языке программирования и сохраняет ее в виде отдельного объектного файла. Затем все эти объектные файлы связываются и образуют одну большую программу. Программа, которая отвечает за связывание объектных файлов, называется *компоновщиком,* или *редактором связей.*

В мире Microsoft Windows возможность написания программы на нескольких языках программирования связана с применением динамически подключаемых библиотек DLL, т.е. специальных программ. При создании трех файлов DLL могут использоваться, например, языки C, Java и COBOL. Затем четвертый программист пишет программу на Visual Basic, который обеспечивает собственно интерфейс пользователя, а команды берутся из библиотек **DLL.**

Третий способ состоит в написании программы с использованием любимого языка (например, Pascal), после чего пишется инструкция на языке ассемблера в качестве части программы. (Только имейте в виду, что переключаться между разными языками в рамках одной программы позволяют далеко не все компиляторы.) Различные языки программирования позволяют вам пользоваться преимуществами каждого из них, сведя недостатки к минимуму.

Как должна работать программа

Выбрав подходящий язык программирования, не спешите приступать к вводу программ. Точно так же, как программисты создают прототипы интерфейса пользователя будущей программы, они создают и прототипы (макеты) инструкций, которые четко описывают, как должна работать программа. Подобные макеты инструкций называются *псевдокодами.*

Если вам необходимо написать программу, которая будет управлять полетом ядерной ракеты, направляя ее к какому-нибудь городу для уничтожения любых признаков жизни в радиусе 100 км, например, псевдокод вашей программы будет выглядеть следующим образом.

```
1. Получить координаты цели.
2. Получить текущие координаты ракеты.
3. Вычислить траекторию ракеты для попадания в цель.
4. Взорвать ядерную боеголовку.
```

Используя псевдокод, вы сможете быстрее выявить оплошности в логической последовательности действий еще до написания программы, так как такие недочеты очень легко скрываются за сложным синтаксисом команд языка программирования.

В предыдущем примере вы можете увидеть, что каждая инструкция псевдокода требует дополнительных уточнений, прежде чем вы сможете приступать к написанию программы. Вы не можете просто дать компьютеру команду "Получить координаты цели", поскольку он сразу "поинтересуется" у вас, а откуда он получит эти самые координаты. Поэтому вам придется переписать псевдокод следующим образом.

```
1. Получить координаты цели.
   а) Попросить технический персонал ввести координаты цели.
   б) Убедиться в том, что указанные координаты цели имеют допустимые
      значения.
   в) Сохранить координаты цели в памяти.
2. Получить текущие координаты ракеты.
```

```
3.Вычислить траекторию ракеты для попадания в цель.
4.Взорвать ядерную боеголовку.
```

Однако вы сможете еще детализировать инструкции, переписав их следующим образом.

```
1.Получить координаты цели
   а) Попросить технический персонал ввести координаты цели.
   б) Убедиться в том, что указанные координаты цели имеют допустимые
      значения.
   Сохранить координаты цели в памяти.
      • Убедиться в том, что координаты указаны полностью.
      • Убедиться в том, что указанные координаты достижимы для ракеты.
      • Убедиться в том, что указанные координаты не приведут к поражению
        ракетой дружественных территорий.
2.Получить текущие координаты ракеты.
3.Вычислить траекторию ракеты для попадания в цель.
4.Взорвать ядерную боеголовку.
```

Когда программисты определили общие задачи, с которыми должна справляться программа, и детализировали каждый пункт программы, они говорят, что завершили нисходящее проектирование программы. Другими словами, они начали с самого верха (сформулировав общие задачи, поставленные перед программой), после чего спустились вниз, конкретизируя каждую из задач, до тех пор, пока не описали каждый шаг, который предпримет компьютер.

Написание псевдокода требует больших затрат времени. Однако единственной альтернативой будет немедленное написание программы безо всякого планирования, что аналогично тому, что вы сядете в автомобиль и будете ехать постоянно на север, а затем искренне удивитесь, куда подевался легкий южный ветерок.

Псевдокод — это очень полезный инструмент для описания общей структуры программы, что позволит вам проще определить, какие данные потребуются программе при работе. Основная идея состоит в том, чтобы использовать понятный человеческий язык для описания пошаговых инструкций, которые должен выполнять компьютер, пекле чего использовать псевдокод при написании программы на любом языке программирования (C/C++, FORTRAN, Pascal, Java и т.д.), который для вас предпочтительнее.

Жизненный цикл типичной программы

Некоторые программы программисты пишут, выпускают и оставляют без своего внимания. Однако большинство программ все-таки проходят через различные циклы, в течение которых они постоянно обновляются, пока не перестанут быть полезными. (Именно по этой причине многие люди каждые несколько лет приобретают новую версию текстового процессора, хотя алфавит остается без изменений на протяжении столетий.)

Вообще, каждая программа проходит через цикл разработки (в течение которого вы создаете ее и продаете), цикл сопровождения (когда вы как можно быстрее исправляете обнаруженные в работе программы ошибки), а также цикл обновления (когда вы добавляете в программу новые средства для того, чтобы продать одну вещь еще раз).

Цикл разработки

Каждая программа начинается с пустого экрана перед глазами программиста. Во время разработки вы проводите программу от идеи до действительно работающей программы. Во время разработки вы выполняете следующие шаги.

1. Формулируется общая идея программы.
2. Принимается решение о потенциальных пользователях программы.
3. Принимается решение о типе компьютера, на котором программа будет выполняться.
4. Выбирается один из возможных языков программирования.
5. Программа проектируется с использованием псевдокода или другого инструмента для определения ее общей структуры.
6. Пишется программа.
7. Программа тестируется.
8. Этот шаг также называется альфа-тестированием.
9. Исправляются любые обнаруженные во время альфа-тестирования ошибки.
10. Повторяйте пп. 7-8 как можно чаще.
11. Передайте копии программы другим людям для ее тестирования,
12. Этот шаг также называется бета-тестированием.
13. Исправляются любые обнаруженные во время бета-тестирования ошибки.
14. Повторяйте пп. 9-10 как можно чаше.
15. Программа выпускается, после чего вы гордитесь тем, что она работает так, как вы и обещали.

Цикл сопровождения

Многие программисты скорее предпочитают писать новые программы, чем сопровождать и изменять написанные раньше. Однако число программ, которые программист сможет написать за год, намного меньше, чем количество уже существующих программ, поэтому в определенный момент своей жизни вам придется сопровождать или дополнять программу, написанную раньше вами или кем-то другим.

Ниже приведен приблизительный список действий, который вам придется выполнить для поддержания существующей программы.

1. Просмотрите все отчеты об обнаруженных ошибках, чтобы определить, какая именно часть программа отвечает за их появление.
2. Исправьте ошибки.
3. Проверьте работу программу, чтобы убедиться в том, что все обнаруженные ошибки исправлены, а новые не появились.
4. Исправьте все ошибки, обнаруженные во время проверки.
5. Повторяйте пп. 1—4 для исправления всех ошибок, о которых вы получаете сообщения.
6. Из-за природы программного обеспечения вы можете выполнять эти действия на протяжении нескольких лет.
7. Выпустите исправление для программы, которое пользователи смогут добавить к существующей версии программы для внесения исправлений.

Цикл обновления

Компании не зарабатывают деньги на том, что выпускают "заплатки" для программ и делают их более стабильными. Компании зарабатывают деньги на том, что продают новые версии программ, снабжая их новыми функциями и параметрами, которые большинству пользователей просто не нужны.

Однако, в связи с тем что изменения в программу вносятся для того, чтобы программа пользовалась преимуществами нового оборудования или операционной системы, вам порой придется добавлять в написанную раньше программу новые функции. Цикл обновления программы обычно сопровождается следующими действиями.

1. Определите, какие новые функции вы хотите добавить в свою программу.
2. Спланируйте, как именно должна работать новая функция (используя псевдокод или другие средства).
3. Внесите в программу необходимые изменения.
4. Проверьте работоспособность новой функции (на стадии альфа-тестирования) и убедитесь, что функция работает должным образом и не приводит к возникновению новых проблем.
5. Исправьте любые проблемы, выявленные при альфа-тестировании.
6. Передайте несколько копий программы другим людям для бета-тестирования.
7. Исправьте любые проблемы, выявленные при бета-тестировании.
8. Повторяйте пп. 1–7 для каждой функции, которую решили добавить в программу.
9. Выпустите новую версию программы и ожидайте поступления сообщений об ошибках, которые не позволяют программе работать должным образом, чтобы вы могли снова начать цикл сопровождения программы.

Несмотря на любые университетские курсы и громкие должности наподобие "самый главный программист", программирование до сих пор остается больше искусством, чем наукой. Написание, изменение и обновление программного обеспечения не требует высокого интеллекта или ученой степени в области математики, а требует творческих способностей, способности принимать решения и воображения. Вы можете писать программы любым способом, однако во избежание проблем вам следует подходить к написанию программы как можно методичнее.

Глава 4

Инструменты настоящего программиста

В этой главе...

- Написание программы в окне редактора
- Использование компилятора и интерпретатора
- Отлавливаем "блохи" с помощью отладчика
- Написание файла справки
- Написание установочной программы

Для того чтобы упростить написание компьютерной программы, программисты создали целый ряд инструментов. Как молоток, отвертка или ножовка, каждый инструмент программирования создается с определенной целью. После того как вы узнаете, для чего предназначен тот или иной инструмент, вы сможете немедленно приступать к написанию программы.

Для написания программы вам потребуются два очень важных инструмента.

- *Редактор* (чтобы вы могли написать инструкции для компилятора).
- *Компилятор*, который преобразует ваши инструкции в машинные коды, чтобы компьютер смог понять, что же вы от него хотите. Вместо того чтобы использовать компилятор, во многих языках программирования используется *интерпретатор*. Основное различие между ними состоит в том, что интерпретатор преобразует инструкции в машинные коды при каждом запуске программы, а компилятор делает это всего один раз, после чего сохраняет полученный результат в виде выполняемого файла с расширением .exe.

При написании программы вы можете воспользоваться следующими дополнительными инструментами.

- *Отладчик* (эта программа позволяет находить проблемы и ошибки в работе программы).
- Программа для создания *файлов справки* (позволит вашей программе отображать справочные сведения на экране, а вас избавит от необходимости снабжать каждую программу руководством пользователя на бумаге).
- *Программа установки* (установит и настроит работу вашей программы на компьютере пользователя).

Если вы приобрели определенный язык программирования, такой как Visual Basic, Delphi или Visual C++, вы обычно получаете вместе с ним редактор, компилятор, отладчик, а иногда и программу установки.

Написание программы в окне редактора

При написании программы вы должны вводить инструкции в текстовом файле. Хотя для создания текстового файла вы можете использовать текстовый процессор, такая программа предоставляет вам массу возможностей для форматирования текста, которые при написании программы совершенно не нужны.

Тестовый файл (файл ANSI) — это файл, содержащий только символы, которые вы вводите с клавиатуры. Формат ANSI поддерживается всеми компьютерами. ANSI — это аббревиатура от American National Standards Institute (Национальный институт стандартизации США).

Программа состоит из одной или нескольких инструкций, которые говорят компьютеру, что делать. Инструкции, образующие программу, называются ее *исходным кодом*.

Чтобы не связываться с текстовыми процессорами, программисты создали специальные программы для написания, редактирования и печати исходного кода программы. Ни один программист не написал программу с первого раза без единой ошибки; большую часть рабочего времени программист проводит за внесением исправлений в исходный код программы. Поэтому программа, предназначенная для написания, редактирования и печати исходного кода программ, называется *редактором*.

Linux и движение открытого кода

На заре компьютерной эры программисты свободно создавали исходные коды своих программ и обменивались ими. Идея состояла в том, что если над программой будут добровольно работать много людей, шансы на то, что программа будет работать надежно, возрастают.

После этого программисты решили, что за свой нелегкий труд было бы неплохо получать деньги, и прекратили свободно распространять исходный код своих программ, благодаря чему индустрия программного обеспечения приняла современный вид (с высокими ценами, ненадежными программами и отвратительно написанными руководствами пользователя).

Однако сегодня идея обмена исходными кодами программ (известного как *движение открытого кода*) вновь витает в воздухе благодаря появлению операционной системы Linux. Пользователи снова могут получать исходный код программы без платы за него.

Доступность исходного кода программы позволяет вам при необходимости вносить в программу определенные изменения или (что вероятнее) нанимать для этого стороннего программиста. В любом случае вы не находитесь в полной зависимости от компаний, которые, не предоставляя исходных кодов программ, заставляют нас платить за обновления, которые не исправляют обнаруженные ошибки, или предлагают новые функции, которые вам не нужны.

Существует два вида редакторов — *редакторы командной строки* и *оконные редакторы*. Старые компьютеры, обладающие ограниченным объемом памяти и дискового пространства, работали только с редакторами командной строки, которые могли обрабатывать только по одной строке за раз. Например, в состав MS DOS входил редактор командной строки, который назывался EDLIN.EXE. Многие программисты используются оконные редакторы, которые позволяют вносить изменения в большое количество инструкций, отображаемых на экране. Редакторы командной строки устарели, однако в некоторых ситуациях вам придется обращаться к их услугам.

Редактор во многом напоминает текстовый процессор, однако при этом предлагает специальные средства, значительно упрощающие работу программиста, например, автоматическое форматирование исходного кода, сочетания клавиш для быстрого внесения изменений или всплывающие окна справочной системы при наборе команд. Пример такого окна приведен на рис. 4.1. Если вам необходимо написать или изменить исходный код программы, без редактора вам не обойтись.

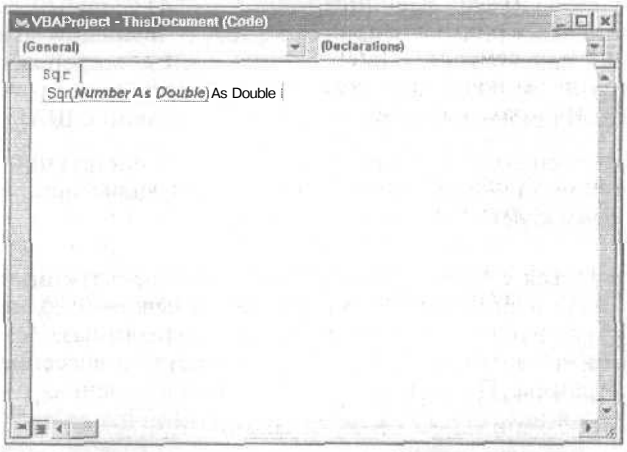

Рис. 4.1. Редактор Visual Basic предлагает вам всплывающее окно справочной системы, в котором отображается правильный синтаксис команды

Использование компилятора и интерпретатора

Вот вы ввели инструкции в окне редактора, используя язык программирования C++ или Java. Что же делать дальше? Компьютер не имеет никакого представления о том, что означают введенные вами команды. Компьютеры понимают только машинные коды, поэтому вам необходима специальная программа, которая преобразовала бы исходный код программы (инструкции, написанные на C++ или Java) в машинные коды,

Для преобразования исходного кода программы в машинные коды вы можете воспользоваться программой одного из двух типов.

- ✓ Компилятор
- ✓ Интерпретатор

Компиляторы

Компилятор преобразует исходный код программы как одно целое, после чего сохраняет полученные машинные коды, т.е. полный эквивалент исходного кода программы, в виде *выполняемого файла*. Этот процесс во многом напоминает работу переводчика, который переводит целый роман с испанского языка на арабский.

Процесс преобразования исходного кода программы в машинные коды с помощью компилятора называется *компиляцией* программы.

Скомпилировав программу, вы можете спокойно распространять выполняемый вариант своей программы, не предоставляя при этом ее исходный код. Все коммерческие программные продукты (такие как Microsoft PowerPoint и Lotus 1-2-3) скомпилированы.

После того как вы скомпилировали программу, вам больше не нужно использовать компилятор (если только вы не решите внести изменения в исходный код программы).

Компиляторы создают машинный код, предназначенный для определенного типа микропроцессоров, например PowerPC (используемый в компьютерах Macintosh) или семейства Intel Pentium (или его клонов, например, AMD Athlon). Если вы написали программу на BASIC и планируете использовать ее на компьютерах Macintosh и Windows, вам придется откомпилировать ее дважды — один раз для Macintosh и один раз для Windows.

Не все компиляторы обладают одинаковыми возможностями, хотя все они способны преобразовать исходный код программы в машинные код. Например, один и тот же исходный код на C++ одним компилятором может быть превращен в программу большого объема, работающую быстрее, а вторым — в небольшую программу, но работающую медленнее.

Интерпретаторы

Второй, менее популярный способ преобразования исходного кода в машинные коды состоит в использовании *компилятора*. Интерпретатор последовательно преобразует в машинный код каждую строку исходного кода программы. Этот процесс очень напоминает перевод речи, записанной на английском языке, на русский по отдельным приложениям.

В отличие от компиляторов, интерпретатор после преобразования исходного кода программы в машинные коды сохраняет полученный результат не в виде выполняемого файла, а только в памяти компьютера. Как только вы выключите компьютер, версия программы на машинных кодах улетучивается. Для того чтобы снова запустить программу, вам придется еще раз воспользоваться интерпретатором для преобразования исходного кода программы в машинные коды.

Если кто-нибудь захочет запустить вашу программу, ему понадобятся и интерпретатор, и исходный код программы- Исходный код программы позволяет любому человеку понять, как же вы писали свою программу (а значит, позволяет вносить в нее изменения без вашего разрешения), поэтому при создании коммерческих программ интерпретаторы практически никогда не используются.

Интерпретаторы используются многими языками для разработки Web-страниц, такими как JavaScript и VBScript. Поскольку Web-страницы просматриваются с помощью компьютеров различным типов, вы не можете скомпилировать программы, написанные на JavaScript или VBScript, в машинные коды. Вместо этого для выполения программы используется интерпретатор.

Раньше, когда компьютеры были достаточно маломощными и им часто не хватало памяти, интерпретаторы пользовались популярностью, поскольку обеспечивали достаточно неплохую обратную связь. Как только вы вводили какую-нибудь команду, интерпретатор немедленно сообщал вам о том, будет ли она работать, и даже выводил результат ее применения. При использовании интерпретатора вы могли писать и тестировать программу одновременно. Теперь, когда компьютеры стали намного быстрее, программисты нашли использование компиляторов более удобным, чем интерпретаторов.

P-код: объединение компилятора и интерпретатора

Необходимость заставить программу работать на нескольких типах компьютеров очень часто превращалась в источник постоянной головной боли. Например, программы для Macintosh и Windows используют выпадающие меню и диалоговые окна, Вам необходимо написать один набор команд для создания выпадающих меню на Macintosh и второй набор команд — для создания выпадающих меню на Windows.

Поскольку одна программа практически никогда не будет работать на компьютерах различных типов без внесения в нее значительных изменений, программисты объединили средств компилятора с интерпретатором для создания так называемого *р-кода.*

Вместо того чтобы компилировать исходный код программы непосредственно в машинные коды, р-код преобразует исходный код программы в специальный промежуточный формат файлов. Для того чтобы запустить программу, скомпилированную в р-код, вы используете интерпретатор. Такой двущаговый процесс позволяет запускать программу на любом компьютере, на котором установлен интерпретатор р-кода.

Среди языков программирования, использующих п-код, наиболее популярен Java. После компиляции программы на Java в р-код вы можете скопировать этот код на компьютер, работающий под управлением Mac OS, Windows или Linux. Если на компьютере установлен интерпретатор р-кода, вы сможете запустить программу, не внося в нее никаких изменений.

Лучше всего то, что вы можете запускать скомпилированный р-код без исходного кода программы, т.е. вы можете предоставлять его другим, не передавая при этом исходный код программы.

Если вам интересно, то язык Liberty BASIC, о котором я еще расскажу в настоящей книге, сохраняет инструкции BASIC в виде отдельного файла, в котором используется р-код. Если вы распространяете любую программу, созданную с помощью Liberty BASIC, вам также придется распространять и специальную программу выполнения, которая позволит запускать полученный с помощью Liberty BASIC п-код на другом компьютере.

Следует отметить, что у р-кода есть свои недостатки. Программы, написанные с использованием р-кода, обычно работают намного медленнее, чем программы, откомпилированные непосредственно в машинные коды. Несмотря на то, что программы с использованием р-кода могут выполняться без исходного кода, с помощью которого они были созданы, вы всегда сможете их *декомпилировать.*

Декомпиляция р-кода позволяет восстановить исходный код, кроторый программист использовал при создании программы. Поэтому, если вы написали программу на Java и откомпилировали ее в р-код, ваш конкурент может декомпилировать его и увидеть исходный код программы на Java. Таким образом, он получает возможность просто-напросто украсть вашу программу.

Можно декомпилировать любую программу, в том числе и скомпилированную в виде машинных кодов. Однако, в отличие от декомпиляции программ в виде р-кода, декомпиляция программы в виде машинных кодов никогда не позволит восстановить исходный код программы на языке высокого уровня, использованный программистом при ее создании. Если вы скомпилировали программу в машинные коды, ее исходный код мог быть на C++, COBOL. FORTRAN, BASIC, Ada, LISP, Pascal или любом другом языке программирования. Так как декомпилятор не имеет никакого представления о том, на каком языке изначально была написана программа, он сможет декомпилировать вариант программы в машинных кодах в ее эквивалент на языке ассемблера. Получив вариант программы в виде команд языка ассемблера, вы можете внести любые изменения в полученный исходный код. Декомпиляция позволяет достаточно легко похищать идеи, реализованные другими программистами.

Что же выбрать?

Если вы хотите писать программы для продажи, используйте компилятор, который защитит исходный код программы. Если вы хотите написать программу, которая будет выполняться на Web-странице, используйте интерпретатор или p-код. Если вы хотите, чтобы программа выполнялась на компьютерах различных типов, вам стоит остановить свой выбор на p-коде. Чтобы еще больше защитить свои права на программу, используйте несколько компиляторов для модификации программы для ее запуска на компьютерах различных типов.

Язык, который вы выберете для написания программы, определяет, сможете вы использовать компилятор, интерпретатор или p-код. Например, вы очень часто будете преобразовывать программы, написанные на Java, в p-код, хотя можно скомпилировать их и непосредственно в p-код. С другой стороны, программы, написанные на C++, обычно компилируются и очень редко интерпретируются или преобразуются в p-код.

Отлавливаем "блохи" с помощью отладчика

Ни одна компьютерная программа не работает на 100% правильно, именно это объясняет, почему компьютеры часто зависают, теряют сведения о забронированных билетах на самолет или просто временами выкидывают всякие фокусы. С точки зрения математики написание программы, которая гарантированно работала бы без ошибок все время, просто невозможно, так как никто не проверит, как она ведет себя вместе со всевозможными комбинациями программного и аппаратного обеспечения.

Проблема, которая приводит к некорректной работе программы, часто называется *багом* (от англ. bug — насекомое, жучок).

Очень давно в компьютерах использовались механические реле и вакуумные лампы, а не печатные платы и микропроцессоры. Однажды один компьютер наотрез отказался работать должным образом. Ученые начали искать источник проблемы; ведь компьютер должен работать. Каково же было их удивление, когда они обнаружили насекомое, которое попало в одно из реле и не позволяло ему закрываться, что и приводило к некорректной работе компьютера. После этого все проблемы, возникающие в работе компьютера, англоязычные пользователи начали называть *багами*, несмотря на то, что настоящие жучки давно не имеют ничего общего с проблемами в работе компьютера.

Поскольку написать программу, которая на 100% работала бы без проблем, просто нельзя, даже операционные системы (например, Windows 2000) содержат определенные ошибки. Для того чтобы преобразовать исходный код программы в машинный код, вы должны использовать компилятор или интерпретатор, т.е. еще одну программу, в которой тоже возможны ошибки. Кроме того, ошибки могут быть и в вашей программе. При таком количестве источников потенциальных ошибок вас вряд ли удивит, что ошибки так же заполняют компьютерные программы, как тараканы населяют дешевые гостиничные номера.

Несмотря на то, что у вас практически нет возможности исправить ошибки в программах, написанных другими программистами (разве что просто не приобретать

такие программы), вы значительно сократите их число в собственной программе, воспользовавшись *отладчиком*. *Отладчик* — это специальная программа (которая, к сожалению, тоже может содержать ошибки), которая позволяет выявлять ошибки в написанной вами программе.

Отладчики предлагают несколько способов обнаружения ошибок в программе.

- ✓ **Пошаговое выполнение.** Отладчик запускает вашу программу — одну команду за другой, поэтому вы всегда можете определить, в какой именно строке содержатся ошибки. Этот процесс во многом напоминает прочтение написанных инструкций, которые вы выполняете, когда первый раз ищете дорогу к дому знакомого. Последовательно выполняя такие инструкции, вы легко сможете определить, где именно повернули не в ту сторону.
- ✓ **Контрольные точки.** Вместо того чтобы выполнять всю программу пошагово, вы можете определить *контрольную точку,* которая указывает на место, с которого пошагово будет выполняться только определенная часть программы. Поэтому, если вы потерялись, вы можете пропустить те инструкции, которые точно выполнили правильно, и выполнить только те из них, в правильном выполнении которых не уверены. Точно также используются контрольные точки для пошагового выполнения только части программы.
- ✓ **Наблюдение.** Наблюдение позволяет вам видеть, как программа сохраняет данные в памяти, и определить, какие именно это данные. Если ваша программа сохраняет неправильные данные (например, имя вместо номера телефона), вы точно сможете определить, с какой частью программы связана ошибка. На рис. 4.2 приведен пример работы отладчика, который выделил определенную инструкцию в программе, а также отобразил окно со значением определенного параметра. Каждый раз, когда вы будете исследовать строку программы, отладчик всегда подскажет вам, как определенная строка влияет на параметр, за которым ведется наблюдение. Как только вы заметите изменение параметра, отладчик немедленно покажет вам, какая строка программы привела к этому изменению.

Отладчик на самом деле показывает вам то, как именно компьютер намерен интерпретировать команды программы. Он показывает вам, какие инструкции необходимо подкорректировать, чтобы при выполнении программы не возникли ошибки. Конечно же, если вы исправили одну ошибку, это может привести к проявлению нескольких новых. Именно поэтому написание программы, полностью лишенной ошибок, практически невозможно.

Написание файла справки

Ни у кого не возникает проблем с использованием дверного звонка, тостера или микроволновой печи, но люди все еще испытывают сложности, осваивая работу с компьютерами и видеомагнитофонами.

Проблема с видеомагнитофонами состоит в том, что порой достаточно сложно разобраться с назначением тех или иных элементов управления, просто глядя на них. Точно так же, компьютерные программы достаточно сложны, чтобы сразу разобраться с ними, взглянув на интерфейс. Если вы сможете написать программу, которая будет действительно проста в использовании, люди обязательно станут с ней работать.

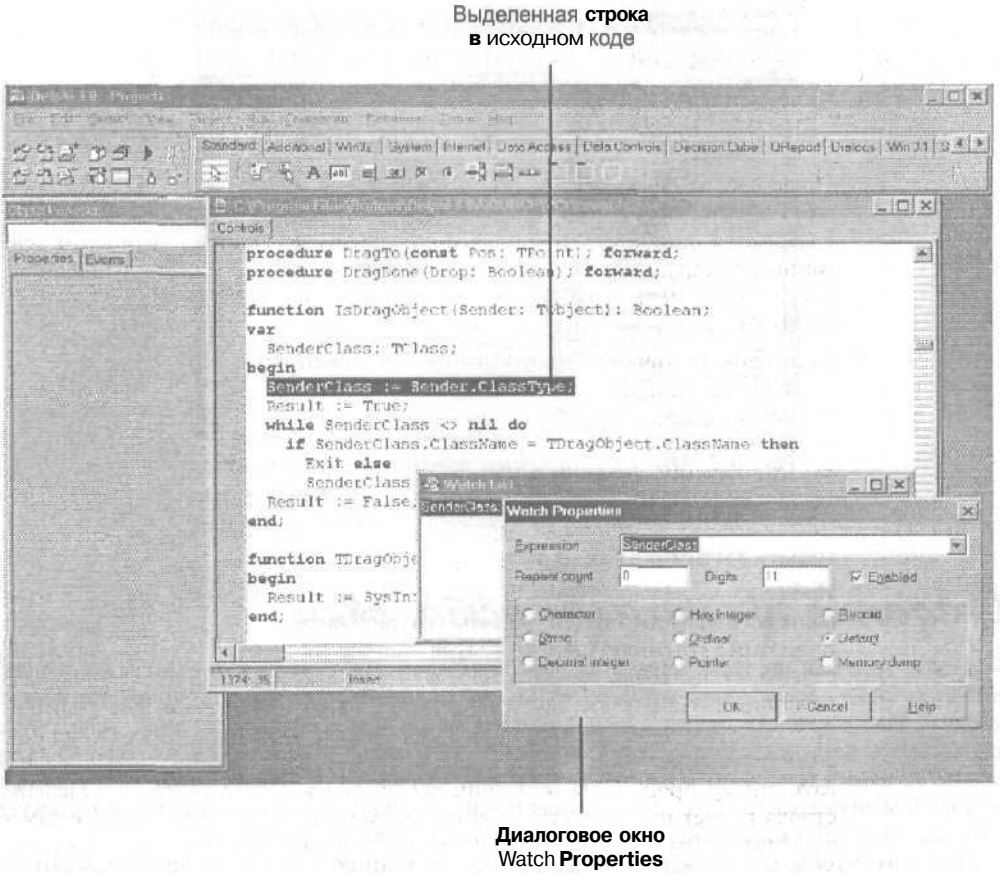

Рис. 4.2. В одном окне отображается исходный код программы, а во втором — параметры, за которыми вы наблюдаете

Компьютерные программы все еще в большинстве случаев разрабатываются программистами для других программистов, поэтому компьютеры остаются загадкой для среднестатистического пользователя. Для того чтобы помочь несчастному пользователю, большинство программ содержат справочную систему.

Файл справки обеспечивает отображение на экране инструкций и поясняющих сведений. Теоретически, если у пользователя возникли проблемы при работе с программой, он должен обратиться к справке и найти там все необходимые пояснения. На рис. 4.3 приведен пример программы, которая пытается всячески помочь пользователю при выполнении его работы.

Несмотря на то, что файлы справки не способны полностью заменить проектирование программ таким образом, чтобы они были понятны с первого взгляда, большая часть программ включает справочную систему. Если вы хотите создать современную программу, без включения в ее состав файла справки вам не обойтись.

Для создания файла справки вы можете воспользоваться специальной профаммои, которая в значительной мере автоматизирует этот процесс.

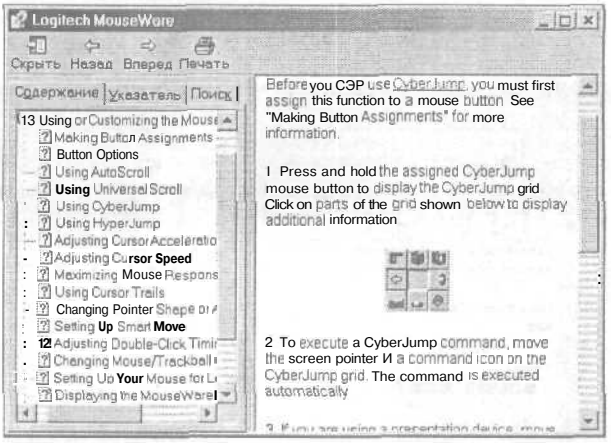

Рис. 4.3. Многие программы предлагают пользователям замечательную справочную систему, в которой пользователь найдет ответы на любые свои вопросы

Создание программы установки

После того как вы написали свою программу, протестировали и отладили ее, а также написали файл справки, вам остается только предоставить или продать ее другим пользователям. Вы можете скопировать свое творение на дискету или компакт-диск, таким образом заставив покупателей вручную копировать программу на жесткий диск, однако такой подход вызывает проблемы. Например, пользователи могут не совсем правильно скопировать файлы программы. Заставив пользователей вручную копировать файлы, вы рискуете добиться того, что они вообще не захотят использовать вашу программу.

Для того чтобы как можно больше упростить копирование файлов программы на жесткие диски компьютеров пользователей, большинство коммерческих программ поставляются вместе со специальной программой установки. Пользователи запускают эту программу, которая автоматически копирует все необходимые файлы в подходящее место на жестком диске. Благодаря использованию программ установки разработчики программного обеспечения могут быть уверенными в том, что их продукты установятся правильно.

Поэтому последним шагом в распространении программы другим пользователем будет использование специальной программы установки, которая сгруппирует все файлы программы в несколько архивных файлов, а затем распакует их на жесткий диск другого компьютера.

Программы установки предлагают следующие средства для распространения программ.

- **Сжатие файлов.** Многие программы достаточно велики, что означает невозможность их записи на одну дискету. Вместо того чтобы заставлять пользователей работать с целым набором дискет или даже компакт-дисков при установке программы, программа установки сожмет файлы программы таким образом, чтобы они уместились на минимальном количестве дискет или компакт-дисков.
- **Отображение графических изображений и воспроизведение звука.** Установка программы обычно занимает несколько минут, в течение которых компьютер копирует файлы с дискеты или компакт-диска на жесткий диск. Вместо того чтобы заставлять пользователя смотреть на пустой экран, программа

установки может отображать на экране различные сообщения, чтобы сделать процесс установки поинтереснее. Другие программы установки отображают различные диаграммы, свидетельствующие о степени установки программы. Таким образом, пользователь всегда знает, сколько ему осталось ждать до завершения установки программы.

- ✓ **Упрощение процесса копирования.** Еще важнее то, что программа установки значительно упрощает копирование файлов программы на жесткий диск пользователя, а это делает установку более быстрой и надежной.

Первое впечатление о программе пользователи получают во время ее установки, поэтому программа установки должна выглядеть и выполнять все необходимые операции как можно профессиональнее.

Конечно же, вы твердо должны быть уверены в том, что профамма выполняет все операции по установке программного обеспечения корректно, в противном случае впечатление пользователей от вашей программы будет крайне негативным.

Часть II

Изучаем программирование на Liberty BASIC

В этой части

То, какой язык программирования вы используете (C/C++, BASIC, Prolog, Java, COBOL, Perl, Ada и т.д.), не играет никакой роли: все компьютерные программы создаются в соответствии с несколькими общими принципами. Если вы знаете, как написать программу на одном языке программирования (например, BASIC), вы быстро убедитесь в том, что изучить второй или третий язык программирования (например, C/C++ или Java) намного проще.

Для того чтобы вы могли попрактиковаться в программировании, в настоящей книге подробно описана работа с языком программирования Liberty BASIC, который является настоящим компилятором для Windows, позволяющим писать и компилировать готовые для личного использования или продажи программы. Просто загрузите Liberty BASIC из Internet, наберите текст нескольких программ и посмотрите, как они будут работать.

В этой части вы найдете не только текст программ на Liberty BASIC, но и их аналоги на других языках программирования (C/C++, Pascal или Java), которые я привел для того, чтобы познакомить вас и с другими языками. Вы сможете сравнить тексты программ, чтобы увидеть, как различные языки программирования решают одни и те же задачи, хотя соответствующие команды могут выглядеть совершенно по-иному.

Глава 5

Знакомьтесь: язык программирования Liberty BASIC

В этой главе...

- Зачем изучать Liberty BASIC
- Установка Liberty BASIC на компьютере
- Написание программ на Liberty BASIC
- Использование редактора Liberty BASIC
- Использование справочной системы Liberty BASIC
- Завершение работы с Liberty BASIC

Лучший способ изучить что-нибудь — попробывать использовать. Если вы хотите научиться программировать, вам следует как можно быстрее начать писать программы на своем компьютере.

Начните изучать программирование с освоения одного из сотни языков программирования, ориентированных на новичков: Pascal, LOGO или SmallTalk. Однако самым популярным языком программирования для начинающих все еше остается BASIC. Этот язык достаточно прост, чтобы помочь вам разобраться в основных концепциях программирования, но при этом обладает всеми необходимыми средствами, чтобы позволить вам создавать коммерческие программы.

Чтобы помочь вам в изучении программирования, я советую начать с Liberty BASIC, языка программирования, обладающего всеми необходимыми возможностями, но дистрибутив которого занимает меньше 2 Мбайт. Все, что вам необходимо для работы с Liberty BASIC, — это компьютер, работающий под управлением Windows 95/98/Me/NT/2000. Основная часть примеров программ, которые приведены в настоящей книге, написаны именно на этом языке программирования.

Зачем изучать Liberty BASIC

Если у вас возник серьезный интерес к изучению программирования, вы наверняка зададите вопрос: а почему не начать с изучения C/C++? Конечно, вы можете начать изучать программирование с любого языка программирования, но Liberty BASIC обладает целым рядом преимуществ.

Liberty BASIC бесплатен (почти)

Liberty BASIC — это условно-бесплатная программа, которую вы можете бесплатно использовать до тех пор, пока не решите, стоит за нее платить или нет. Таким образом, вы получаете возможность начать изучать программирования, не платя за коммерческие продукты, такие как Visual Basic или Visual C++.

Liberty BASIC очень прост

Liberty BASIC позволит вам изучить основы программирования, дав настоящий опыт создания компьютерных программ. Другие языки программирования, C++ или Java, потребуют от вас изучать достаточно сложные дополнительные темы, такие как указатели, объектно-ориентированный подход, а также выделение памяти. Чтобы не связываться с такими сложностями, по крайней мере, на первых порах, начните с изучения Liberty BASIC.

После того как вы освоите основы программирования на Liberty BASIC, вам будет легче изучить более серьезный язык программирования, такой как C++ или Java.

Liberty BASIC работает в среде Windows

Когда компьютеры работали под управлением MS DOS, компания *Microsoft* выпускала бесплатный интерпретатор BASIC, который назывался QBASIC. Хотя вы все еще можете использовать QBASIC при работе с Windows (например, с Windows 98, но не Windows 2000), многие пользователи ожидали появления версии BASIC для Windows, которая бы позволила им изучать основы программирования.

Поскольку у компании *Microsoft* не было никаких планов по выпуску Windows-версии QBASIC, единственным вариантом остался Liberty BASIC. Liberty BASIC позволяет не только писать программы на компьютерах, работающих под управлением Microsoft Windows, но и создавать настоящие программы, которые вы сможете продавать или использовать самостоятельно.

Если вы заплатите за полную версию Liberty BASIC, вы сможете создавать настоящие программы для Windows и продавать их другим. Условно-бесплатная версия Liberty BASIC позволяет только писать программы.

Установка Liberty BASIC

Для того чтобы установить Liberty BASIC, вам необходимо загрузить его из Internet с Web-узла `www.libertybasic.com`, распаковать полученный файл, запустить программу `intall.exe`, после чего следовать указаниям на экране. Имейте в виду, что новые версии программ выходят достаточно часто, поэтому возможности или внешний вид окон версии Liberty BASIC, которая рассматривается в настоящей книге, несколько отличаются от версии Liberty BASIC, загруженной вами при изучении материала книги.

Загруженный вами файл является сжатым ZIP-файлом. Для его распаковки вам необходима специальная утилита, например WinZip, которую можно загрузить с Web-узла компании *Nico Mak Computing* (`www.winzip.` com).

Для того чтобы установить версию Liberty BASIC на своем компьютере, выполните следующее.

1. Запустите утилиту-архиватор, например WinZip.
2. Загрузите сжатый файл Liberty BASIC, который вы взяли из Internet и сохранили на жестком диске компьютера.
3. Разархивируйте файлы Liberty BASIC в отдельную папку.
4. Для этого вам понадобится создать соответствующую папку.
5. Выберите команду Выполнить из меню Пуск.
6. На экране появится диалоговое окно Запуск программы.

7. Щелкните на кнопке Обзор.
8. На экране появится диалоговое окно Обзор.
9. Щелкните на значке файла `install.exe` в папке, в которой вы разархивировали файлы Liberty BASIC при выполнении п. 3, после чего щелкните на кнопке Открыть.
10. На экране еще раз появится диалоговое окно Обзор.
11. Щелкните на кнопке OK.
12. Следуйте дальнейшим инструкциям на экране для установки Liberty BASIC на вашем компьютере.

Запуск Liberty BASIC

После того как вы установили Liberty BASIC, программа создала собственную папку и добавила пункт Liberty BASIC в меню Пуск. Для того чтобы запустить Liberty BASIC, выполните следующее.

1. Щелкните на кнопке Пуск на панели инструментов Windows.
2. Раскроется меню Пуск.
3. **Выберите команду** Liberty BASIC⇨Liberty **BASIC.**
4. Запустится программа Liberty BASIC.

Для того чтобы упростить и ускорить запуск программы Liberty BASIC, поместите ее ярлык на рабочий стол Windows.

Для того чтобы поместить ярлык программы на рабочий стол Windows, выполните следующее.

1. Щелкните правой кнопкой мыши на рабочем столе Windows.
2. На экране появится контекстное меню.
3. Выберите команду Создать⇨Ярлык.
4. На экране появится диалоговое окно Создание ярлыка.
5. Щелкните на кнопке Обзор.
6. На экране появится диалоговое окно Обзор.
7. Найдите выполняемый файл программы Liberty BASIC (`Liberty.exe`), выделите его и щелкните на кнопке Открыть.
8. Вам понадобится просмотреть содержимое нескольких дисков и папок, прежде чем вы найдете этот файл. После того как вы щелкнете на кнопке Открыть, на экране еще раз появится диалоговое окно Создание ярлыка.
9. Щелкните на кнопке Далее.
10. На экране появится диалоговое окно Выбор названия программы.
11. Укажите имя, например Liberty BASIC, в текстовом поле Укажите название ярлыка, после чего щелкните на кнопке Готово.
12. Значок программы Liberty BASIC появится на рабочем столе. Когда вы захотите запустить программу Liberty BASIC в следующий раз, вам будет достаточно дважды щелкнуть на ее значке.

Ваша первая программа на Liberty BASIC

Для написания, редактирования и запуска программ на BASIC предназначен редактор Liberty BASIC. Для того чтобы оценить всю мощь Liberty BASIC, введите следующие строки в окне реестра:

```
PRINT "This BASIC program mimics a really bad boss."
PRINT
PRINT "What is your name?"
INPUT Name$
PRINT "Hello " +Name$+". You're fired!. Have a nice day"
END
```

Liberty BASIC, как и множество версий BASIC, абсолютно не чувствителен к реестру символов, которыми вы вводите команды, т.е. ему все равно, используете вы прописные или строчные буквы. Однако многие программисты предпочитают использовать все строчные буквы при вводе команд BASIC.

В отличие от текстового процессора, редактор Liberty BASIC не разбивает текст по строкам в рамках окна, поэтому, если вы вводите текст в одну строку, он просто будет выходить за рамки окна.

Эта программа приказывает компьютеру выполнять следующие действия.

1. **Первая строка отображает на экране сообщение** This BASIC program mimics a really bad boss. (Эта программа имитирует работу действительно нехорошего начальника).
2. Вторая строка приводит к простому пропуску одной строки на экране.
3. Первая строка отображает на экране сообщение what is your name? (Введите свое имя).
4. Четвертая строка ожидает, пока пользователь введет сведения. После того как пользователь нажмет клавишу <Enter>, программа сохранит введенные сведения в памяти в качестве значения переменной Name$.
5. Пятая строка отображает на экране сообщение Hello (привет), затем введенное имя пользователя, а затем — You're fired!. Have a nice day. (Ты уволен. Всего хорошего). Знаки "+" указывают программе на необходимость добавления к надписи значения переменной Name$.
6. Шестая строка указывает компьютеру на то, что программа завершена.

Выполнение программы на **Liberty BASIC**

Завершив написание программы на BASIC, воспользуйтесь комбинацией клавиш <Shift+F5> или выберите команду Run⇨Run (Выполнение⇨Выполнить) из строки меню Liberty BASIC. Результат запуска и выполнения программы, написанной на Liberty BASIC, так же как и окно самой программы, показаны на рис. 5.1.

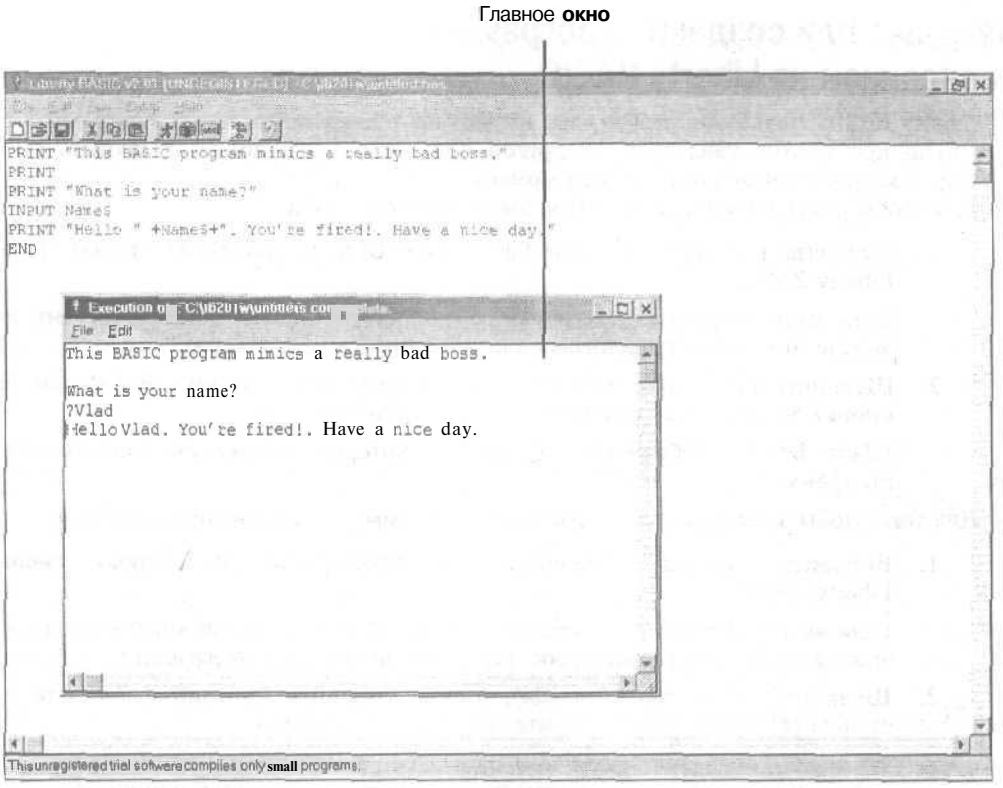

Рис. 5.1. Запуск вашей первой программы, написанной на Liberty BASIC

После того как вы запустите программу, Liberty BASIC отобразит текст в окне, которое называется *основным*. После этого вы можете использовать команды, доступные из строки меню основного окна, для сохранения или печати результатов выполнения программы.

Сохранение программы, написанной на Liberty BASIC

Хотя вы и можете вводить текст программы при каждом запуске Liberty BASIC, гораздо проще сохранять программу на жестком диске. После этого вы можете просто загрузить программу и выполнить ее или даже внести в нее необходимые изменения.

Для того чтобы сохранить программу, выполните следующее.

1. Выберите команду File⇨Save (Файл⇨Сохранить) из строки меню Liberty BASIC.
2. На экране появится диалоговое окно Save As (Сохранить как).
3. Введите имя файла в текстовом поле Filename (Имя файла).
4. При сохранении файла программы вы можете указать любую папку.
5. Щелкните на кнопке OK.

Liberty BASIC автоматически добавляет расширение .bas ко всем сохраняемым вами файлам программ. Если у вас нет на то веских оснований, не изменяйте это расширение, чтобы всегда легко идентифицировать программы, написанные на BASIC.

Загрузка или создание программы, написанной на Liberty BASIC

Liberty BASIC позволяет отображать на экране только текст одной программы за раз. Если необходимо увидеть текст другой программы, вам придется закрыть программу, с которой работаете в данный момент.

Для чтобы создать новую программу, выполните следующее.

1. Выберите команду File⇨New File (Файл⇨Создать файл) из строки меню Liberty BASIC.

   Если вы не сохранили программу, с которой работаете в данный момент, на экране отобразится диалоговое окно с запросом о подтверждении.

2. Щелкните на кнопке Yes (Да), чтобы сохранить текущий файл, или на кнопке No (Нет), если не хотите сохранять изменения.

   Liberty BASIC отобразит пустое окно, в котором вы сможете вводить текст программы.

Для того чтобы загрузить уже сохраненную программу, выполните следующее.

1. Выберите команду File⇨Open (Файл⇨Открыть) из строки меню Liberty BASIC.

   Если вы не сохранили программу, с которой работаете в данный момент, на экране отобразится диалоговое окно с запросом о подтверждении.

2. Щелкните на кнопке Yes (Да), чтобы сохранить текущий файл, или на кнопке No (Нет), если не хотите сохранять изменения.

   На экране появится диалоговое окно Open a BAS File (Открыть файл). Для нахождения нужного файла вам понадобится сменить диск или папку.

3. Найдите необходимый файл, щелкните на его значке, после чего щелкните на кнопке OK.

Текст выбранной программы появится на экране.

Использование комбинаций клавиш

Редактор Liberty BASIC работает как простой текстовый процессор. Для перемещения по тексту программы можно использовать мышь и клавиатуру. В табл. 5.1 перечислены комбинации клавиш и результаты их использования, которые доступны в программе Liberty BASIC.

Таблица 5.1. Комбинации клавиш редактора Liberty BASIC

Комбинация клавиш	**Результат** применения
<Home>	Перемещение курсора к началу строки
<End>	Перемещение курсора к концу строки
<Shift+Стрелка>	Выделение текста в направлении стрелки
<Delete>	Удаление символа, на который наведен курсор, или всей выделенной части текста
<Backspace>	Удаление символа, расположенного слева от курсора

Окончание табл. 5.1

Комбинация клавиш	Результат применения
<Ctrl+F>	Поиск и замена текста в программе
<Ctrl+A>	Выделение всего текста программы
<Ctrl+Z>	Отмена выполнения последней команды

Использование справочной системы Liberty BASIC

Как и большинство других программ для Windows, Liberty BASIC обладает неплохой справочной системой. Для того чтобы получить доступ к этой справочной системе, выполните следующее.

1. Выберите команду Help⇨Contents (Справка⇨Содержание) из строки меню Liberty BASIC.

   На экране отобразится окно справочной системы, в котором будут перечислены все доступные темы.

2. Щелкните на интересующей вас теме.
3. В зависимости от выбранной темы, на экране отобразится та или иная информация о работе с Liberty BASIC.
4. Закройте окно справки, как только получите все необходимые сведения.

Завершение работы с Liberty BASIC

Очевидно, что вам необходимо завершить работу с Liberty BASIC, чтобы выполнить на компьютере какую-нибудь другую работу. Для того чтобы завершить работу с Liberty BASIC, выполните следующее.

1. Выберите команду File⇨Exit Liberty BASIC (Файл⇨Завершить работу с Liberty BASIC) из строки меню.

   Если вы не сохранили программу, с которой работаете в данный момент, на экране отобразится диалоговое окно с запросом о подтверждении.

2. Щелкните на кнопке Yes (Да), чтобы сохранить текущий файл, или на кнопке No (Нет), если не хотите сохранять изменения.

   На экране отобразится еще одно диалоговое окно с запросом о подтверждении завершения работы с программой Liberty BASIC.

3. Щелкните на кнопке Yes (Да), чтобы завершить работу с Liberty BASIC (или на кнопке No (Нет), если хотите продолжить работу).

   Работа с Liberty BASIC будет успешно завершена.

Глава 6

Обработка ввода и вывода

В этой главе...

> Работа с ввод ом -вывод ом: старый способ
> Работа с ввод ом -вывод ом: современный способ
> Печать данных

Каждая программа получает данные *(ввод),* каким-то образом их обрабатывает, после чего выдает их в каком-то формате *(вывод).*

Раньше программисты передавали данные компьютеру, используя много различных способов — от переключения различных тумблеров до использования бумажных лент, перфокарт, телетайпов и, наконец, клавиатур и мониторов. После того как компьютер проведет с введенными данными определенные операции, он выведет полученные результаты или на бумаге, или на экране монитора.

Несмотря на то, что сегодня при работе с компьютерами используются всевозможные раскрывающиеся меню, диалоговые окна, кнопки, а также полосы прокрутки и многое другое, большинство современных языков программирования все еще связаны с прошлым, когда программы ожидали от пользователя ввода определенной информации. После этого программа выполняла определенные действия с этой информацией и выводила результаты в следующей строке.

Именно по этой причине многие языки программирования, BASIC и даже C/C++, содержат встроенные команды для считывания и отображения данных в строке на экране. Естественно, что подобные программы выглядят достаточно примитивно, особенно при сравнении с хорошо написанными современными программами, однако не стоит немедленно делать какие-то выводы. Поскольку вы только приступили к изучению основ программирования, вам будет намного полезнее понять, как программы принимают вводимые вами сведения и как выводят полученные результаты, а не как выглядят окна программ на экране.

Работа с вводом-выводом: старый способ

Под вводом данных понимается получение программой данных от какого-нибудь внешнего источника. Несколько потенциальных источников данных перечислены ниже.

- ✓ Информация, которую пользователь вводит с помощью клавиатуры.
- ✓ Перемещение указателя мыши.
- ✓ Данные, раньше сохраненные в файле (например, файле текстового процессора).
- ✓ Данные, поступившие в компьютер из внешнего источника (например, Web-страница, полученная с помощью модема, или изображение, полученное с помощью сканера)

Под выводом понимается возвращение программой информации, с которой предварительно были проведены определенные манипуляции. Ниже перечислено несколько различных способов вывода информации.

- ✓ Отображение данных на экране (текст, изображение, видео)
- ✓ Печать данных на бумаге с помощью принтера
- ✓ Воспроизведение звука с помощью звуковой подсистемы компьютера.

При программировании на языке BASIC простейший способ вывода информации — использование команды PRINT, которая позволяет отобразить на экране любой текст, заключенный в кавычки. В следующем примере команда PRINT используется для отображения на экране строки What are you looking at? (На что же ты смотришь?), как показано на рис. 6.1:

```
PRINT "What are you looking at?"
END
```

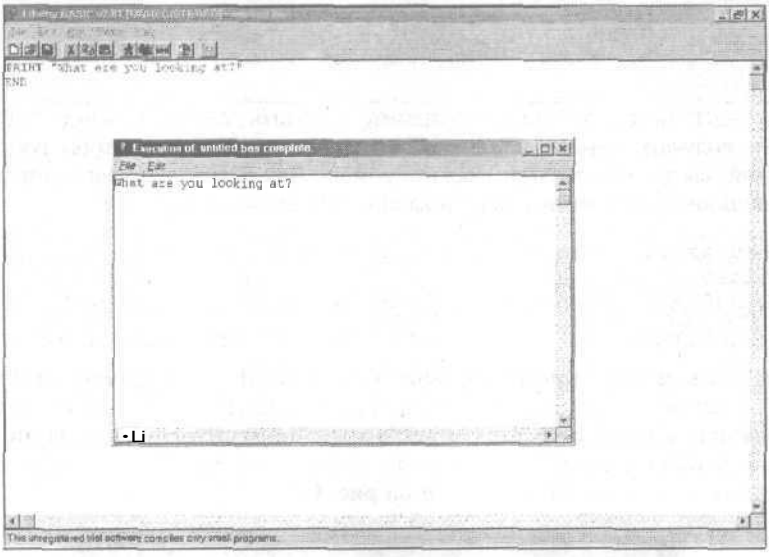

Рис. 6.1. Результат выполнения команды PRINT приведен в основном окне

Команда PRINT в языке Liberty BASIC отображает текст в *основном* окне. Если вы не хотите, чтобы информация отображалась на экране таким образом, воспользуйтесь командой NOMAINWIN. Когда вы начнете создавать программы, в которых будет использоваться графический интерфейс пользователя (подробности в главе 14, "Создание интерфейса пользователя"), вам не понадобится использовать основное окно для представления текста на экране.

Эквивалентная программа на C

В отличие от BASIC, другие языки программирования заставляют вас заключать инструкции между специальных слов или символов, выступающих в качестве скобок, ограничивающих начало и конец программы. Например, при использовании языка программирования C/C++ самая простая программа, которую вы сможете написать, состоит из слова main, за которым следуют скобки, а также еще одной пары фигурных скобок, как показано в следующем примере:

```
main (,)
{
}
```

Если вы пишете инструкции программы, используя C/C++, вы должны заключать их в эти странные фигурные скобки. Просто ради того, чтобы продемонстрировать вам, насколько по-разному выглядят программы в разных языках программирования, я приведу пример программы на C, которая позволяет пользователю вводить имя и отображает эти сведения на экране, добавляя фразу That sounds like the name of a moron! (Звучит довольно-таки глупо!):

```
main ( )
.{
char rayboss [15]
printf ("What is the name of your boss?" \n) ;
scanf ("%s", myboss);
printf ("%s", myboss) ;
printf ("?,That sounds like the name of a moron!");
}
```

Команда PRINT позволяет вашей программе выводить данные; команда INPUT позволяет программе получать данные, введенные с клавиатуры. Для того чтобы воспользоваться этой командой, введите ее, а затем укажите символ или фразу, которая будет представлять введенные пользователем данные, как показано в следующем примере:

```
PRINT "What is the name of your boss?"
INPUT Myboss$
PRINT Myboss$ + "? That sounds like the name of a moron!"
END :
```

Если *вы* введете этот пример в окне Liberty **BASIC,** программа отобразит текст What is the name of your boss? (Как зовут вашего начальника?). **Если введете имя и нажмете** клавишу **<Enter>, программа отобразит** введенное **вами имя, после чего добавит к нему строку** That sounds like the name of a moron! (Звучит довольно-таки глупо!), **как показано на рис. 6.2.**

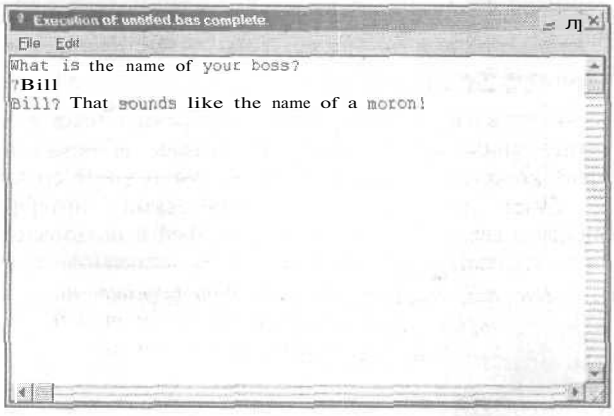

Рис. 6.2. Команда INPUT позволяет получать данные с клавиатуры

Не стоит беспокоиться о технических аспектах использования команд PRINT и INPUT, так как они представляют старый способ получения и отображения информации на экране. В Windows и других современных операционных системах программы предлагают более удобные способы получения и отображения информации благодаря использованию графического интерфейса пользователя (Graphical User Interface — GUI).

Работа с вводом-выводом: современный способ

Большинство современных компьютеров используют графический интерфейс пользователя при отображении данных. Такие команды, как PRINT и INPUT, использовались на старых компьютерах, с которыми больше никто не работает, а современные языки программирования предлагают специальные команды для создания и отображения данных в окнах и диалоговых окнах.

Каждый язык программирования предлагает различные виды команд для создания и отображения данных в окнах, диалоговых окнах и меню. Команды, о которых мы говорим в настоящей главе, уникальны для Liberty BASIC и не срабатывают в других версиях языка программирования BASIC.

Ввод данных

Проще всего вводить данные, предложив пользователю это сделать с помощью специального диалогового окна. Для того чтобы создать это диалоговое окно, вам придется воспользоваться командой PROMPT, за которой следует текст, который должен отображаться (например, просьба ввести имя или число), а затем переменная, которой введенная информация будет присвоена, как показано на следующем примере:

```
PROMPT "What is the name of your boss?"; name$
```

В результате выполнения этой команды на экране отобразится диалоговое окно, показанное на рис. 6.3.

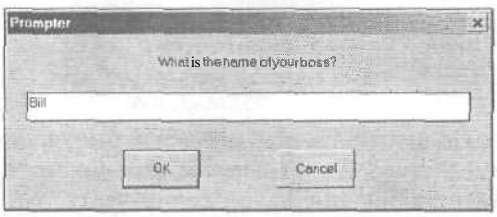

Рис. 6.3. Команда PROMPT отображает на экране диалоговое окно, в котором пользователь может ввести данные

Отображение вывода

Простейший способ отображения данных на экране — использование диалогового окна Notice (Уведомление). Для создания этого диалогового окна вам придется воспользоваться командой NOTICE, за которой следует текст, который должен отображаться. Приведенная ниже программа использует команду NOTICE для отображения определенной фразы вместе с информацией, введенной пользователем. Внешний вид диалогового окна Notice показан на рис. 6.4.

```
NOMAINWIN -
I PROMPT "What is the name of your boss?"; name$
NOTICE name$ + "? That sounds like the name of a moron!"
END
```

Рис. 6.4. Команда NOTICE используется для отображения одноименного диалогового окна

Печать данных

Один из наиболее популярных способов получения данных — их печать, т.е. создание *твердой копии*. Для выполнения подобных действий в языке Liberty BASIC используются две команды — LPRINT и DUMP, как показано в следующем примере:

```
LPRINT "Save the dolphins! Kill the tuna!"
i DUMP
' END
```

Команда LPRINT отправляет данные на принтер, установленный по умолчанию. Команда DUMP приказывает принтеру немедленно приступить к работе.

Если хотите, вы можете пропустить команду DUMP, однако команда LPRINT тогда не сразу приступит к печати.

Команда LPRINT начинает печать в верхнем левом углу страницы. Если вы хотите, чтобы данные были напечатаны в другом месте страницы, вам необходимо добавить несколько строк, чтобы изменить положение надписи на странице.

Для того чтобы изменить положение надписи по вертикали, воспользуйтесь еще раз собственно командой LPRINT, как показано ниже:

```
LPRINT
LPRINT "Save the dolphins! Kill the tuna!" -
  DUMP
END
```

В этом примере команда LPRINT **просто** напечатала **пустую строку, а вторая — строку** Save the dolphins! Kill the tuna! (Сохраним дельфинов. Наловим тунца!)

Для того чтобы изменить положение по горизонтали, воспользуйтесь командой SPACES(x), где буква x представляет собой число пробелов, которые вы хотите вставить. Например, если вы хотите вставить перед строкой текста пять пробелов, воспользуйтесь командой SPACE$(5), как показано в следующем примере:

```
LPRINT "Save the dolphins! Kill the tuna!"
LPRINT SPACE$(5); "Save the dolphins! Kill the tuna!"
S DUMP
END
```

В результате выполнения этой программы на бумаге будет напечатано следующее:

```
Save the dolphins! Kill the tuna!
```

При использовании команды SPACES (x) вам необходима точка с запятой, за которыми должна следовать помешенная в кавычки строка, которую нужно напечатать.

В настоящей главе мы кратко рассмотрели ввод и вывод информации. В главе 14 я подробно расскажу вам о создании интерфейса пользователя с использованием окон, диалоговых окон и меню.

Глава 7

Переменные, константы и комментарии

В этой главе...

- Использование переменных
- Создание и использование констант
- Добавление комментариев к тексту программы

Переменные — это одни из самых важных компонентов программирования, поскольку они позволяют программам сохранять данные и манипулировать ими, чтобы компьютер мог определенным образом реагировать на различные условия.

Помимо сохранения данных в виде переменных, программы также используют такие штуки, как *константы* и *комментарии*. Константы — это фиксированные значения, которые могут понадобиться программе при работе, а комментарии — это пояснения, используемые программистами для объяснения того, как работает программа.

Хотя идея использования переменных, констант и комментариев покажется вам настоящей загадкой, не беспокойтесь. Вы разберетесь в их назначении, как только приступите к созданию собственных программ.

Использование переменных

Если вы сообщите свое имя компьютеру, а он не сможет запомнить введенные вами данные, смело считайте его бесполезной железякой. Для того чтобы сделать компьютеры хоть чуть-чуть полезными, программы должны приказать компьютеру сохранять все получаемые данные для использования в дальнейшем.

После того как вы снабдите программу определенными сведениями, профамма сохраняет их в памяти. Так как информация может приходить в программу из нескольких источников, программа должна дать каждым сведениям определенное имя, чтобы как-то упорядочить их в своей памяти. Подобные имена называются *переменными*, так как сведения, которые это имя будут представлять, изменяются. Переменные обычно используются:

- ✓ для сохранения введенных данных;
- ✓ для сохранения результатов вычислений, работающих с введенными данными или результатами предыдущих вычислений.

Переменная временно сохраняет любую полученную информацию, а профамма использует эту информацию в будущем. Хотя вы можете присваивать переменным практически любые имена, лучше всего давать им такие имена, которые максимально точно соответствовали бы типу сохраненных в виде этих переменных сведений. Например, переменной, предназначенной для сохранения номера телефона, можно дать имя `PhoneNumber` или что-нибудь настолько же описательное.

Хотя вы можете присваивать переменным максимально описательные имена, не стоит забывать о том, что каждый язык программирования налагает определенные ограничения на создание имен переменных. Например, для языка программирования Liberty BASIC характерны следующие правила создания имен переменных.

- ✓ Первым символом в имени переменной обязательно должна быть буква, например *A* или *S,* В частности нельзя создать переменную 7me.
- ✓ Имена переменных могут содержать точки, например My.Boss. Однако вы не сможете разделять части имени переменной пробелами: My Boss.
- ✓ Имена переменных чувствительны к регистру используемых символов, т. е. строчные и прописные буквы различаются. Например, Phonenumber и PHONENUMBER — это имена двух различных переменных с точки зрения языка программирования Liberty BASIC.
- ✓ В качестве имен переменных нельзя использовать зарезервированные ключевые слова Liberty BASIC, например END и PRINT.

Если язык программирования различает строчные и прописные символы, программисты называют его *чувствительным к регистру* (case-sensitive). Например, языки программирования C/C++, Java и C# чувствительны к регистру, а языки программирования Pascal и большинство версий BASIC — нет. (Liberty BASIC — это одно из исключений, чувствительных к регистру.)

В приведенном ниже примере используются две переменные. Одна переменная (Salary) используется для сохранения данных, представляющих заработок пользователя, а вторая (TaxOwed) — результатов определенных вычислений:

```
NOMAINWIN
Salary = 25000
TaxOwed =Salary.*.95
NOTICE "This is how much tax you owe = $";
       TaxOwed
END
```

Эта программа приказывает компьютеру выполнить следующее.

1. Первая строка приказывает Liberty BASIC не отображать основное окно.
2. Вторая строка приказывает компьютеру присвоить переменной Salary значение 25000.
3. Третья строка приказывает компьютеру выполнить следующее: умножить значение переменной Salary на 0,95, после чего присвоить полученный результат вычислений переменной TaxOwed.
4. Четвертая строка приказывает компьютеру выполнить следующее: отобразить диалоговое окно, а в нем — сообщение This is how much tax you owe (После удержания налогов вы получаете), **за которым будет указано значение переменной** TaxOwed.
5. Пятая строка сообщает компьютеру о завершении программы.

Присвоение переменной определенного значения

В Liberty BASIC переменные не содержат никакой информации до тех пор, пока вы не присвоите им определенное значение. Так как основным назначением переменной является сохранение данных, вы можете присвоить переменной значение одним из трех способов.

- ✓ Присвоение переменной фиксированного значения.
- ✓ Присвоение переменной результата вычислений.
- ✓ Использование команды PROMPT для получения данных от пользователя, после чего эти данные используются в качестве значения переменной.

Присвоение переменной фиксированного значения

Простейший способ присвоить переменной определенное значение — использовать знак равенства между именем переменной и этим значением. Этот метод позволяет присвоить переменной числовое или строковое значение, как показано в следующем примере:

```
CEOSalary = 9000000
Message2Workers$ = "So I make more than I'm worth. So what?"
```

В первой строке вы присваиваете переменной CEOSalary значение **9000000. Во второй строке вы присваиваете переменной** Message2Workers$ **значение** So I make more than I'm worth. So what? (Да, я получаю больше, чем зарабатываю. Ну и что?).

Если вы назначите переменной строковое значение, имя переменной должно заканчиваться знаком $. Если вы назначите переменной числовое значение, знака $ в имени переменной не должно быть.

Пока вы не присвоите переменной какое-то значение, она будет равна или нулю (0) или пустой строке (“”).

Присвоение переменной результата вычислений

Переменные представляют числа или строки, которые могут изменяться, поэтому переменным в качестве значений присваивают результаты вычислений. Если вы хотите сохранить в виде переменной числовой параметр, просто присвойте ей следующее выражение:

```
DumpestPersonIQ = 6
BossIQ = DumpestPersonIQ/3
```

В этом примере Liberty BASIC создает переменную BossIQ. Затем программа делит на 3 значение переменной DumpestPersonlQ. И только после этого присваивает результат переменной BossIQ.

Если вам необходимо присвоить переменной текстовое значение, обратите внимание на следующий пример:

```
Cry$ = "I want to eat"
NewCry$ — Cry$ + "food that is really bad for me!"
```

В этом примере в первой строке переменной Cry$ **присваивается значение** I want to eat (Я хочу есть). **Вторая строка создает новую переменную** NewCry$. **Затем она объединяет значение переменной** Cry$ **со строкой** food that is really bad for me! (пищу, которая для меня чрезвычайно вредна!).

Таким образом, переменная NewCry$ **представляет строку** I want to eat food that is really bad for me! (Я хочу есть пищу, которая для меня чрезвычайно вредна!).

Для того чтобы увидеть, как Liberty BASIC назначает переменным числовые и строковые значения, введите текст следующей программы:

```
NOMAINWIN
Parents = 2
Whacks = 20
MotherAxWhacks" = Parents * Whacks
FatherAxWhacks = MotherAxWhacks + 1
FirstName$ = "Lizzie"
LastName$ = "Borden"
FullName$ = FirstName$ + LastName$
NOTICE FullName$ + " had an ax, gave her mother " +chr$(13)+
str$(MotherAxWhacks)+_
 "wracks. When she saw what"_
+ chr$(13)+ "  she had done, gave_
 her father " + str$(FatherAxWhacks) +"."
END
```

Эта программа приказывает компьютеру выполнить следующее.

1. Первая строка указывает Libety BASIC, что отображать основное окно не следует.
2. Вторая строка создает переменную Parents и присваивает ей значение 2.
3. Третья строка создает переменную Whacks и присваивает ей значение 20.
4. Четвертая строка создает переменную MotherAxWhacks. Затем она умножает значение переменной `Parents` (равное 2) на значение переменной whacks (равное 20). Затем полученное значение (40) присваиваетсяпеременнойMotherAxWhacks.
5. Пятая строка создает переменную FatherAxWhacks. Затем она добавляет 1 к значению переменной MotherAxWhacks (равному 40). Затем полученное значение (41) присваивается переменной `FatherAxWhacks`.
6. Шестая строка создает переменную `FirstName$` и присваивает ей значение `Lizzie`. (Обратите внимание, что в имени переменной `FirstName` есть знак $, следовательно, этой переменной можно присваивать только строковые значения.)
7. Седьмая строка создает переменную `LastName$` и присваивает ей значение Borden.
8. Восьмая строка создает переменную `FullName$` и присваивает ей объединенное значение переменных `FirstName$` и `LastName$`. Таким образом, значение переменной `FullName$` равно `Lizzie` Borden. (Обратите внимание на необходимость пробелов, так как в противном случае мы получили бы LizzieBorden.)
9. **Девятая строка создает диалоговое окно Notice, в котором отображается значение переменной** `FullName$` **(равное** Lizzie Borden), **а также строка** had an ax, gave her mother (держа топор, нанесла матери). **Обратите внимание на использование в конце строки знака подчеркивания, который указывает Liberty BASIC на продолжение команды со следующей строки.**
10. **Десятая строка добавляет символ перехода на новую строку (ему соответствует ASCII-код 13), за которым следует значение переменной** MotherAxWhacks **(равное 40) и строка** `wracks.` when she saw what (`ударов.` Когда она поняла, что).

11. Одиннадцатая строка добавляет символ перехода на новую строку (которому соответствует ASCII-код 13), за которым следует строка she had done, gave her father (она сделала, нанесла отцу).

12. Двенадцатая строка указывает на необходимость отобразить значение переменной FatherAxWhacks (равное 41).

13. Тринадцатая строка сообщает Liberty BASIC об окончании программы.

Присвоение переменной значения, полученного с помощью команды PROMPT

Команда PROMPT отображает на экране диалоговое окно, в котором пользователю предлагается ввести определенные данные — либо числовые, либо строковые. Ниже приведен пример использования команды PROMPT для запроса у пользователя ввода значения параметра. Как только пользователь введет значение (например, число 49) в диалоговом окне, компьютер сохранит это число в качестве значения переменной Salary. Затем программа использует переменную TaxOwed для сохранения результатов вычислений:

```
NOMAINWIN
PROMPT "How much money did you make last
      year?"; Salary
TaxOwed = Salary * .95
NOTICE "This is how much tax you owe = $";
     . TaxOwed
END
```

Эта программа приказывает компьютеру сделать следующее.

1. Первая строка указывает Liberty BASIC, что отображать основное окно не следует,
2. Вторая строка отображает диалоговое окно с сообщением HOW much money did you make last year? (Сколько вы заработали за прошлый год?). После того как пользователь введет какое-то число, оно будет присвоено переменной Salary.
3. Третья строка умножает значение переменной Salary на 0,95. Результат сохраняется в качестве значения переменной TaxOwed.
4. Четвертая строка отображает диалоговое окно Notice с сообщением This is how much tax you owe = $ (На руки вы получили = $); **затем указывается значение переменной** TaxOwed.
5. **Пятая строка сообщает Liberty BASIC об окончании программы.**

Эквивалентная программа на С

Чтобы вы не подумали, что программы всегда выглядят так, как при использовании Liberty BASIC, я приведу вам эквивалент программы из раздела "Присвоение переменной значения, полученного с помощью команды PROMPT", написанный на С:

```
main  ()
Я
 float salary, taxowed;
printf  ("How much money did you make last year?");
scanf  ("%f", &salary);
```

```
taxowed = salary * .95
printf ("This is how much tax you owe = $%8.2f");
}
```

Если вы хотите, чтобы пользователь ввел строковое значение (например, имя) в диалоговом окне, вам следует добавить знак $ в конце имени переменной, предназначенной для сохранения этой строки, например YourName$. Знак $ указывает Liberty BASIC на то, что переменная может сохранять только строку, например имя, почтовый индекс или название улицы.

Пример использования диалогового окна для сохранения строкового значения приведен ниже.

```
NOMAINWIN
PRIMPT "What is your name?"; YourName$
Message$ = YourName$ + ", you deserve a raise!"
NOTICE Message$
END
```

Эта программа приказывает компьютеру сделать следующее.

1. Первая строка указывает Liberty BASIC, что отображать основное окно не следует.
2. Вторая строка отображает диалоговое окно с сообщением what is your name? (Как вас зовут?). После того как пользователь введет какое-то число, оно будет присвоено переменной YourName$.
3. Третья строка **добавляет фразу** , you deserve a raise! (, вы заслужили повышение по службе!) **к** значению **переменной** YourName$. **Полученный результат сохраняется в качестве значения переменной** Messages.
4. Четвертая строка отображает диалоговое окно Notice с сообщением, соответствующим **значению переменной** Message$.
5. Пятая строка сообщает Liberty BASIC об окончании программы.

Если пользователь ввел число при выполнении п. 2, например 45. Liberty BASIC воспримет это число как строку 45.

Объявление переменных

Переменные позволяют программам сохранять данные и манипулировать ими. Поэтому объявления всех переменных, которые будут использоваться программой, а также указание типа данных, для хранения которых они будут использоваться, окажется полезным, когда необходимо получить четкое представление о том, как же работает программа.

В отличие от большинства языков программирования, BASIC позволяет создавать и использовать переменные в любом месте программы. Хотя это полезно при написании программы, вам будет очень сложно разобраться в структуре программы в дальнейшем, если понадобится внести в нее какие-то изменения.

Давайте рассмотрим еще раз пример программы из раздела "Присвоение переменной результата вычислений". Сколько переменных используется в этой программе? Если вы не можете быстро дать ответ на этот вопрос, вам придется потратить на это время, просмотрев текст всей программы, строка за строкой, прежде чем вы сможете

ответить. (В программе используется семь переменных: Parents, Whacks, MotherAxWhacks, FatherAxWhacks, LastName$, FirstNameS И FullName$.)

Для того чтобы позволить вам (или кому-то еще) быстро разобраться во всех переменных в программе, многие языки программирования, такие как C/C++ и Pascal, заставляют вас объявлять все используемые в программе переменные в самом ее начале. Объявление переменных в начале программы преследует две цели.

- ✓ Для определения *имен* всех переменных, используемых в программе.
- ✓ Для определения *общего числа* переменных, используемых в программе.

Зная общее число переменных в программе, вы лучше разберетесь в ее работе, поскольку можете определить, где именно программа хранит данные.

Liberty BASIC поддерживает исходный диалект языка программирования BASIC, которому недостает конструкций современных языков программирования, которые появились в более поздних версиях BASIC, например объявления переменных и константы.

Для того чтобы объявить переменную в некоторых версиях BASIC (например, Visual Basic, но не Liberty BASIC), вам следует использовать команду DIM, как показано ниже:

```
DIM Eyeballs
```

Эта команда приказала компьютеру создать переменную Eyeballs. Вы можете сразу объявить несколько переменных, разделяя их имена запятыми, как показано ниже:

```
DIM Eyeballs, Bullets, Logs
```

Эта команда приказала компьютеру создать переменные Eyeballs, Bullets, Logs.

Теперь, если вы перепишете текст программы, объявив в ее начале все используемые переменные, вы сможете легко определять, сколько же переменных в программе. Как видно из следующего пример для QBAS1C (диалекта языка BASIC, очень похожего на Liberty BASIC), программа прежде всего объявляет все необходимые переменные, благодаря чему вы легко определите, сколько же переменных используется в программе:

```
DIM Parents, Whacks, MotherAxWhacks, FatherAxWhacks, LastName$,
      FirstName$, FullName$
Parents = 2
Whacks = 20
MotherAxWhacks = Parents * Whacks
FatherAxWhacks = MotherAxWhacks + 1
FirstName$ = "Lizzie"
LastName$ = "Borden"
FullName$ = FirstName$ + LastName$
PRINT FullName$ + " had an ax, gave her mother ";
      MotherAxWhacks;
PRINT " wracks. When she saw what she had done, gave her";
PRINT " father ";FatherAxWhacks
END
```

В этом примере вы быстро увидете, что в программе три переменные используются для сохранения строковых значений (LastName$, FirstName$ и FullName$), а еще четыре переменные используются для сохранения числовых значений (Parents, Whacks, MotherAxWhacks, FatherAxWhacks).

Эквивалент программы на Java

Язык программирования Java очень похож на C/C++, поэтому если вы знакомы с C/C++, у вас не возникнет проблем с изучением Java. Для того чтобы дать вам небольшое представление о том, как выглядит программа, написанная на Java, я печатаю текст программы, которая выполняет то же самое, что и приведенная выше программа, написанная на QBASIC, но переписанная на Java:

```
public class TrivialApplication
  {
 public static void main (String args[])
  {
 int parents, whacks, motheraxwhacks, fatheraxwhacks;
 String lastname, firstname, fullname;
 parents = 2;
 whacks — 20;
 motheraxwhacks = parents * whacks;
 fatheraxwhacks - motheraxwhacks + 1;
 firstname = "Lizzie";
 lastname = "Borden";
 fullname = FirstName$ + LastName;
System.out.println(fullname 4 " had an ax, gave her
       mother " +motheraxwhacks);
System.out.println("wracks. When she saw what she had done,
       gave her");
System.out.println(" father " +FatherAxWhacks); .
        }
}
```

Если вы запустите эту программу, написанную на Java, она поведет себя точно так же, как ее эквивалент, написанный на QBASIC. Однако определение числа переменных в данном случае оказывается гораздо более простой задачей.

Определение типа данных, которые хранит переменная

В дополнение к возможности определения переменных и указания их имен в самом начале программы, многие языки программирования (но не Liberty BASIC) также заставляют вас указывать тип данных, которые хранит переменная. Определение типа данных преследует две цели.

- ✓ Определяется тип данных, которые хранит переменная. Если вы четко укажете, какой тип данных хранит переменная, вы (или другой программист) сможете намного быстрее понять, где именно программа сохраняет те или иные данные.
- ✓ Предотвращается присвоение переменной данных недопустимого для нее типа, что позволяет избежать многих ошибок при выполнении программы.

Для того чтобы вы могли увидеть, как объявляются переменные в различных языках программирования и указываются типы данных, которые они содержат, я привожу вам пример объявления переменной IQ для хранения целых значений типа integer, написанный на языке C:

```
main ()
{
  int  IQ
}
```

На языке программирования Pascal этот же пример будет выглядеть следующим образом:

```
Program Main;
```

```
Var
  IQ: integer;
```

А на языке программирования BASIC, например Visual Basic, это же объявление переменной будет выглядеть так:

```
DIM IQ AS INTEGER
```

Объявление переменных используется не только для удобства компьютера; оно используется для удобства программиста, который должен прочитать, разобраться и внести необходимые изменения в текст программы в дальнейшем.

Использование констант

Значения переменных при выполнении программы могут изменяться; именно поэтому они называются *переменными*. Однако порой вам понадобится фиксированное значение, которое будет использоваться во всей программе. Обратите внимание на приведенный ниже пример. Можете ли вы быстро определить назначение числа 0,1975?

```
Balance = 43090
OurProfit = Balance * .1975
PRINT "Pay this amount, or we'll get you = ";
     Balance
PRINT "Today's current loan sharking interest
      rate =";  .1975
END
```

Быстрый взгляд на приведенный только что текст программы не позволит вам определить назначение числа 0,1975. Так как назначение некоторых чисел не всегда очевидно без дополнительных объяснений, программисты используют *константы*. Благодаря этому вы можете использовать достаточно описательные имена для представления фиксированных значений.

Liberty BASIC построен на исходном диалекте языка программирования BASIC, который не поддерживает константы. Однако другие версии BASIC (например, Visual Basic и некоторые другие) лишены таких недостатков.

Для того чтобы продемонстрировать вам использование констант, я приведу текст программы, написанной на языке программирования Pascal:

```
Program UnderstandingConstants;
Const
  InterestRate = 0.1975;
 Var
  OurProfit : real
  Balance : real
Begin
   Balance := 43090
 7 OurProfit := Balance * InterestRate;
   Balance := Balance + OurProfit;
   Writeln ('Pay this amount, or we'll get you = ' , Balance:6:2);
   Writeln('Today's current loan sharking interest rate =',
InterestRate:1:4);
End.
```

Если вы запустите эту программу, на экране отобразится следующее:

```
[Pay this amount, or we'll get you - 51600,28
Today's current loan sharking interest rate = 0,1975
```

В приведенной выше программе значение 0,1975 используется в программе дважды (в строках 9 и 12). Если интересующий вас размер процентной ставки изменится с 0,1975 до 0,2486, а в вашей программе константы не используются, придется долго разбираться в тексте программы и повсеместно изменять значение 1975 на 0,2486. В небольших программах, например такой, как показано выше, это не представляет особого труда. Однако в больших программах внесение подобных изменений приводит к большим затратам времени и появлению ошибок.

Поэтому, чтобы сэкономить время и избежать появления ошибок, программисты стараются использовать константы. При использовании констант, как показано в приведенном выше примере программы, написанной на языке программирования Pascal, вам достаточно всего один раз изменять значение константы `InterestRate` с 0,1975 на 0,2486 (в строке 3). Компьютер автоматически вставит новые значения в строки 9 и 12, в которых также используется константа InterestRate.

Константы обеспечивают следующие преимущества.

- ✓ Они ставят числовые или строковые значения в соответствие описательным именам.
- ✓ Изменение значения константы автоматически вносит изменение во всю программу целиком, причем быстро и без новых ошибок.

Добавление комментариев к тексту программы

Если вы написали небольшую программу, любой человек сможет достаточно быстро понять, как она работает. Однако, если вы написали достаточно большую программу, понять ее назначение намного сложнее не только другим пользователям, но и вам, если только не изучать ее построчно.

Для того чтобы облегчить понимание (а значит, и поддержание) программы (а, как известно, программисты достаточно ленивы), каждый язык программирования позволяет добавлять к исходному коду программы комментарии. Комментарии позволяют сохранять вместе с исходным кодом программы следующие сведения.

- ✓ Кто автор программы
- ✓ Дата создания и внесения последних изменений
- ✓ Назначение программы
- ✓ Как программа работает
- ✓ Каким образом программа получает, сохраняет и выводит данные
- ✓ Любые проблемы, обнаруженные при работе программы

При использовании Liberty BASIC вы можете добавлять комментарии одним из двух способов.

- ✓ С помощью команды `REM` (сокращение от REMark — замечание)
- V С помощью знака апострофа (')

В следующем примере иллюстрируется использование команды `REM` и апострофа для добавления комментариев в текст программы. Несмотря на то, что вы можете ис-

пользовать в программе обе разновидности комментариев, для удобства рекомендуется использовать только одну из них.

```
'Создана 29 марта 199Э года
'Написана John Doe
'Эта программа отображает на экране
'сообщение, предназначенное потенциальным
'подражателям, которые приходят со своими
'собственными идеями, а не крадут мои
REM Эта программа отображает на экране
REM сообщение, предназначенное потенциальным
REM подражателям, которые приходят со своими
REM собственными идеями, а не крадут мои
NOMAINWIN 'Предотвращает отображение на экране основного окна
NOTICE "Не крадите идеи, изложенные в настоящей книге"
END 'Это последняя строка а программе
```

Поскольку комментарии используются только для удобства людей, компьютер видит *эту* программу в следующем виде:

```
NOMAINWIN
NOTICE "Не крадите идеи, изложенные в настоящей книге"
END
```

Апостроф лучше подходит для создания комментариев, потому что позволяет добавлять комментарии как часть существующей строки или как отдельную строку. Команда REM позволяет добавлять комментарии только отдельной строкой.

Комментарии существуют только для вашего удобства. Компьютер просто игнорирует все комментарии, с которыми сталкивается в программе. Поэтому убедитесь в том, что ваши комментарии полезны, но не слишком многословны; в противном случае они скорее будут отвлекать, а не направлять.

Комментарии можно использовать для того, чтобы приказать Liberty BASIC временно игнорировать определенные строки кода программы. Вместо того чтобы целиком удалить строку, проверить работоспособность программы, а затем снова набрать текст удаленной строки, вы можете просто закомментировать ее, как показано ниже:

```
NOMAINWIN
'A = SQR((B*B)+ (C+C))
END
```

Если вы запустите эту программу, Liberty BASIC увидит только следующее:

```
NOMAINWIN
END
```

Для восстановления закомментированной строки вам достаточно удалить апостроф, как показано ниже:

```
NOMAINWIN
A = SQR((B*B)+ (C+C))
END
```

Глава 8

Забавы с числами и строками

В этой главе...

- Выполнение математических операций
- Использование встроенных математических функций Liberty BASIC
- Использование строк для хранения данных
- Преобразование строк в числа

К компьютерной программе самое главное — возможность манипулировать любыми данными, которые она получает, и давать ответ в такой форме, чтобы это нравилось пользователям и они хотели платить вам деньги за использование этой программы. Программы оперируют с данными двух типов — *числами* и *строками*.

К распространенным видам программ, манипулирующих числами, относятся электронные таблицы, программы бухгалтерского учета и даже видеоигры (так как им необходимо вычислять пути перемещения всех этих автомобилей, драконов, самолетов и т.д., движением которых вы управляете). К распространенным видам программ, манипулирующих строками, относятся электронные таблицы (они сохраняют, сортируют и упорядочивают такие данные, как имена), текстовые процессоры и программы для машинного перевода. .

Сложение, вычитание, деление и умножение

Четыре основных способа манипулирования числами — сложение, вычитание, деление и умножение. Используя эти четыре математические операции, вы сможете создать любую самую сложную математическую формулу.

Для сложения, вычитания, деления и умножения двух чисел (или двух переменных, представляющих числа) вы можете использовать символы, перечисленные в табл. 8.1.

Таблица 8.1. Математические операции

Математическая операция	Используемый символ	Пример	Результат
Сложение	+	2+5	7
Вычитание	-	77-43	34
Деление	/	20/4	5
Умножение	*	4*7	28
Возведение в степень	^	2^3	8

Знак деления (/) обычно находится на клавиатуре в двух местах — на той же клавише, что и знак вопроса (?), и на цифровой клавиатуре. Символ возведения в степень (^) располагается на клавише <6> основной клавиатуры. Знак вычитания также можно использовать для обозначения отрицательных чисел, например −34,5 или −90.

Вы наверняка уже знаете, как работают операции сложения, вычитания, деления и умножения, но вы меньше знакомы с операцией возведения в степень.

Операция возведения в степень означает простое умножение числа на само себя указанное число раз. Например, формула 4^3 указывает Liberty BASIC на необходимость умножить число 4 на само себя три раза. Результат: 4^3=4*4*4=64.

Использование переменных

Любое математическое вычисление (сложение, вычитание, деление и умножение) приводит к получению одного значения, которое вы можете присвоить переменной:

```
TaxYouOwe = 125000 * 1.01
```

Вместо того чтобы при создании математических формул прибегать к определенным числам (например, 125000*1,01), вы можете использовать переменные, как показано ниже:

```
PROMPT "How much money did you make last
      year"; NetIncome
TaxYouOwe = NetIncome * 1.01
NOTICE "You owe this much in taxes = ";
      TaxYouOwe
; END
```

Для того чтобы упростить понимание математических вычислений, всегда используйте переменные или константы, а не настоящие числа. Например, в предыдущем примере вы вряд ли сможете определить, что такое число 1,01. Вместо числа используйте в формуле константу, как показано ниже:

```
TaxRate = 1.01
PROMPT "How much money did you make last
      year; NetIncome
TaxYouOwe = NetIncome * TaxRate
NOTICE "You owe this much in taxes = ";
      TaxYouOwe
; END
```

Используя константу (в нашем примере константа TaxRate представляет число 1,01), вы сможете легко понять, что означает число 1,01 и зачем оно в математической формуле.

Переменные в формулах используются одним из следующих двух способов.

- ✓ Представление чисел в математической формуле
- ✓ Сохранение результатов вычислений, полученных в результате применения математической формулы

Будьте осторожны при присвоении имен переменным. Если вы допустите ошибку при указании имени переменной или перепутаете символы верхнего или нижнего регистра, Liberty BASIC решит, что вы создали новую переменную, и назначит ей в качестве значения нуль или пустую строку. Если ваша программа работает не так, как нужно, убедитесь в том, что имена всех переменных указаны правильно и используются с одним и тем же количеством символов верхнего и нижнего регистра.

Что такое приоритет операций

Простые формулы, такие как NetIncome*TaxRate, понять совсем несложно; вы просто перемножаете два числа, каждое из которых представляется определенной переменной. Однако вы можете создавать и более сложные математические формулы, комбинируя операции сложения, вычитания, деления или умножения, как показано ниже:

```
TaxYouOwe — PastTaxes + NetIncome * TaxRate
```

Если значение переменной NetIncome **равно 50 000, значение переменной** TaxRate **равно 1,01, а значение переменной** PastTaxes **— 25 000, формула будет выглядеть следующим образом:**

```
TaxYouOwe = 2500 + 5000 * 1.01
```

Поэтому теперь интересно, как происходит вычисление: Liberty BASIC сначала сложит 2500 и 50 000, а затем умножит полученное значение на 1,01 (в результате будет число 53025} или же он сначала умножит 50 000 на 1,01, только после чего добавит 2500 (в результате будет число 53 000).

Результат совместного выполнения операций сложения, вычитания, деления и умножения в одной формуле вызывает у некоторых недоумение, поэтому в языках программирования используется такое понятие, как *приоритет операций*, что позволяет компьютеру определить, какие математические операции необходимо выполнять в первую очередь. Liberty BASIC выполняет математические операции в следующем порядке, начиная с самого приоритетного, т.е. сверху вниз.

- ✓ Возведение в степень (^)
- ✓ Умножение (*) и (^); деление (/)
- ✓ Сложение (+) и вычитание (—)

Прежде чем выполнять следующую программу, попытайтесь определить, каким образом компьютер будет вычислять результат:

```
MyMoney = 3 +4 ^ 5 - 8 / 5 * 7
I PRINT MyMoney
END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. **Первая строка приказывает компьютеру создать переменную** MyMoney.

   Поскольку компьютер начинает с выполнения операции возведения в степень, Liberty BASIC выполнит операцию 4^5, что даст результат 1024. Формула будет выглядеть следующим образом:

   ```
   MyMoney = 3 + 1024 - 8 / 5 * 7
   ```

   Затем компьютер выполнит операции умножения и деления. Так как эти операции имеют одинаковый приоритет, компьютер начнет с выполнения первой операции умножения или деления, в направлении слева направо. Сначала будет выполнена операция 8/5 (результат — 1,6), а затем операция умножения на 7. В результате формула примет следующий вид;

   ```
   MyMoney = 3 + 1024 - 11.2
   ```

   И наконец, компьютер выполнит все операции сложения и вычитания, в направлении слева направо. Сначала будет выполнена операция 3+1024 (результат — 1027); затем из полученного результата вычитается число 11,2. В результате получен следующий ответ:

```
MyMoney = 1015.8
```

2. **Вторая строка прикажет компьютеру напечатать значение переменной MyMoney, равное 1015,8.**

3. **Третья строка сообщит компьютеру о завершении программы.**

Компьютер всегда выполняет математические операции в направлении слева направо, если все операторы обладают одинаковым приоритетом, например умножение и деление или сложение и вычисление.

Использование скобок

Попытка запомнить приоритет различных математических операций часто приводит к недоразумениям. Еще хуже, что этот самый приоритет может привести к тому, что компьютер будет выполнять вычисления совсем не в том порядке, в котором вы предполагали. Предположим, вы написали следующую программу:

```
BigValue= 3 + 4 ^5
PRINT BigValue
 END
```

В данном случае компьютер сначала выполнит операцию возведения в степень (4^5=1024), затем добавит число 3, что приведет к результату 1027.

А если вы хотите, чтобы компьютер сначала сложил числа 3 и 4, и только затем выполнил возведение в степень? В этом случае вам необходимо использовать скобки для того, чтобы сообщить о ваших намерениях компьютеру, что показано ниже:

```
BigValue= • (3 + 4) ^5
PRINT BigValue
END.
```

В этом случае сначала складываются числа 3 и 4, а затем формула принимает вид BigValue = 7^5, что приводит к результату 16 807.

Как только компьютер видит какое-нибудь выражение, заключенное в скобки, он обязательно начинает с вычисления этого выражения. Только после этого он начинает выполнять математические операции согласно их приоритету в направлении слева направо, пока не подсчитает всю формулу.

Используйте скобки только для одного математического оператора, например (3+4). Хотя вы и можете заключать в скобки несколько математических операций, например (3+4^5), это сделает использование скобок бессмысленным, так как все равно будет непонятно, какие операции компьютеру выполняет первыми. Конечно же, вы можете использовать несколько пар скобок, чтобы создать достаточно сложные формулы, как показано ниже:

```
EasyTaxCode = ((3 + 4) ^5 / 3 - 8) / 5 * -7
```

Если вы не будете использовать скобки, Liberty BASIC получит совершенно иной результат.

Скобки также полезны при создании отрицательных чисел. В большинстве языков программирования для того, чтобы показать, что число отрицательное, достаточно поставить перед ним знак "минус", как показано ниже:

```
X = -87.09
```

Но если вы хотите создать отрицательное число в Liberty BASIC, вам необходимо выполнить несколько иную операцию, или умножив его на (−1) или отняв его от нуля, как показано ниже:

```
;X = (0 - 87.09)
```

Или же вы можете поступить следующим образом:

```
X = (-1 * 87.09)
```

Использование встроенных математических функций Liberty BASIC

Комбинируя математические операторы, вы можете создавать практически любые математические формулы. Однако создание некоторых математических формул оказывается достаточно сложной задачей; для упрощения этой задачи Liberty BASIC (так же как большинство других языков программирования) предлагает вам целый набор встроенных математических функций, перечисленных в табл. 8.2.

Таблица *8.2. Встроенные математические функции языка Liberty BASIC*

Функция	Результат
ABS(X)	Возвращает абсолютное значение числа X
ACS(X)	Возвращает арккосинус числа X
ASN(X)	Возвращает арксинус числа X
ATN(X)	Возвращает арктангенс числа X
COS(X)	Возвращает косинус числа X
EXP(X)	Возвращает результат возведения экспоненты в степень X
INT(X)	Возвращает максимальное целое число, меньшее или равное числу X
LOG(X)	Возвращает натуральный логарифм числа X (имейте в виду, что число X должно быть положительным и не равным нулю)
SIN(X)	Возвращает синус числа X
TAN(X)	Возвращает тангенс числа X

Если вы не знаете такие термины, как арксинус и логарифм, вам наверняка не придется их использовать. Все, что вам нужно запомнить, — это то, что все языки программирования, и Liberty BASIC в том числе, предлагают встроенные математические функции, которые вы можете при необходимости использовать.

Многие языки программирования содержат встроенную математическую функцию SQR(X), которая позволяет вычислять квадратные корни. Хотя эта функция в Liberty BASIC еще отсутствует, вы можете извлекать квадратные корни, возводя число в степень 0,5, как показано в следующем примере:

```
X = 9 ^ 0.5
```

В этом примере квадратный корень извлекается из числа 9, что дает результат 3.

Для того чтобы увидеть, как работают математические функции Liberty BASUC, запустите следующую программу и вводите различные числа от 0 до 1:

```
PROMPT "Type in a number"; AnyNumber
PRINT "The ABS value ="; ABS(AnyNumber)
PRINT "The ACS value = "; ACS(AnyNumber)
PRINT "The ASN value = "; ASN(AnyNumber)
PRINT "The ATN value = "; ATN(AnyNumber)
```

```
PRINT "The COS value = "; COS(AnyNumber)
PRINT "The INT value = "; INT(AnyNumber)
PRINT "The LOG value = "; LOG(ABS(AnyNumber))
PRINT "The SIN value = "; SIN(AnyNumber)
PRINT "The SQR value = "; (ABS(AnyNumber) ^0.5)
PRINT "The TAN value = "; TAN(AnyNumber)
PRINT
END
```

При вычислении квадратного корня или логарифма вы можете использовать только целые числа.

Функции арксинуса и арккосинуса воспринимают числа только от 0 до 1,0. Если вы укажете число из другого диапазона, функция не сработает и Liberty BASIC отобразит сообщение об ошибке.

Манипулирование строками

Помимо манипулирования цифрами, компьютеры также могут манипулировать и строками. Строка — это последовательность любых символов, которые вы вводите с клавиатуры, включая буквы, символы (#, & и +), а также числа.

В Liberty BASIC строкой считается любая последовательность символов, заключенных в кавычки, как показано ниже:

```
PRINT "Everything enclosed in quotation marks"
PRINT "is a string, including the numbers
    below: "
PRINT "12 - 9 * 8"
PRINT "You can even mix letters and numbers
    like this"
PRINT "I made $4,500 last month and still
    feel broke. "
END
```

В предыдущем примере формула 72 = 9*8 на самом деле является строкой, хотя и содержит цифры, потому что все, заключенное в кавычки, воспринимается языком программирования Liberty BASIC как строка.

Объявление переменных как строк

Как и в случае **цифр, вы** можете **использовать строки непосредственно** в тексте программы:

```
PRINT "Print me."
PRINT 54
END
```

Как работает со строками язык программирования C/C++

В отличие от BASIC **(и многих** других **языков программирования, в том** числе и Pascal и Java), язык **программирования C/C++ не** поддерживает **строковые типы данных. В программах C/C++ используется более примитивный тип данных, такой как** *символьный* (обозначается как char).

Символьный тип данных позволяет содержать только один символ (например, буква, символ, цифра), поэтому для манипулирования строками в C/C++ приходится использовать *массивы* символов. (Не переживайте. Достаточно подробно о массивах мы поговорим в главе 16, "Сохранение информации в массивах". Сейчас вам необходимо просто запомнить, что программы, написанные на C/C++, работают со строками несколько иным образом, чем программы, написанные на BASIC.)

Просто для того, чтобы вы получили представление о том, как программы, написанные на C/C++, обрабатывают строки, я приведут небольшой пример. В этом примере программа определяет переменную myrelative как массив, способный содержать до 10 символов:

```
main ()
{
  char myrelative[10];
  printf ("Type a name of a male relative you hate.\n";
        scanf ("%s", &myrelative);
        printf ("%s", &myrelative);
        printf ("says he doesn`t like you either!"0;
}
```

Как и в случае цифр, вы можете сохранять строки в качестве переменных, чтобы использовать одну строку в программе несколько раз, но при этом вам не нужно каждый раз ее набирать. Таким образом, ваша программа может получать строку, сохранять ее в качестве значения переменной, а затем манипулировать ею для получения полезного результата, например отображения на экране сообщения, адресованного пользователю.

При создании переменной необходимо сообщать Liberty BASIC, что вы создаете переменную именно для хранения строк. Говоря техническим языком, вы создаете переменную для хранения данных *строкового типа*. Для того чтобы создать переменную для хранения строковых данных, в Liberty BASIC используется такой простой прием, как добавление знака доллара {$) в конец имени переменной, как показано в следующем примере:

```
StringVariable$ = "This variable can hold
                  "only strings."
```

Если вы забыли объявить переменную как переменную строкового типа, но попытаетесь присвоить ей строковое значение, Liberty BASIC отобразит сообщение об ошибке и прекратит выполнение программы. Для проверки того, что вы сохраняете данные в переменных необходимого типа, компиляторы, в частности Liberty BASIC, пытаются проверить, написали ли вы программу без ошибок.

Объединение строк

В отличие от работы с цифрами, вы не можете отнимать, делить или перемножать строки. Но вы можете их складывать (эта операция называется *объединением*). Для того чтобы объединить две строки, вам необходимо использовать знак "плюс", как показано в следующем примере:

```
PROMPT "What is your name?"; MyNanme&
PRINT "Hello, " + MyName$ + ". Isn't it time
      to take"
PRINT  "an early lunch from your job now?"
 END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру отобразить на экране сообщение What is your name? (Как вас зовут?). Введенные пользователем сведения сохраняются в виде переменной MyName$.
2. Вторая строка приказывает компьютеру создать одну большую строку, которая будет использовать значение переменной MyName$. Например, если значение MyNameS равно Tasha, на экране будет выведено сообщение Tasha. Isn't it time to take (Tasha. Разве сейчас время).
3. Третья строка приведет к отображению на экране сообщения an early lunch from your job now? (делать столь ранний обеденный перерыв).
4. Четвертая строка сообщит компьютеру о завершении программы.

Если вы объединяете строки, убедитесь в том, что вы вставили пробел между строками, чтобы они не налезли друг на друга (напримертак). Обратите внимание на пробел между запятой после слова Hello и кавычкой в последнем примере.

Использование функций Liberty BASIC для fradb/fot co строками

Если вы можете только сохранять и объединять строки, Liberty BASIC покажется очень скучным. Именно по этой причине в состав Liberty BASIC входит целый набор встроенных функций для всевозможного манипулирования и упорядочения строк,

Использование верхнего и нижнего регистров

Если в строках содержатся символы, они отображаются одним из трех способов.

- ✓ только строчными буквами, как здесь
- ✓ ТОЛЬКО ПРОПИСНЫМИ БУКВАМИ, КАК ЗДЕСЬ
- ✓ Смесь ПРОПИСНЫХ (верхний регистр) и строчных (нижний регистр) букв, Как Здесь

Для преобразования всей строки целиком к нижнему регистру используйте специальную функцию LOWERS. Для преобразования всей строки целиком к верхнему регистру используйте специальную функцию UPPER$.

Обе функции берут всю строку целиком и преобразуют ее к определенному регистру, как показано ниже:

```
UPPER$("hello") ' HELLO
LOWER$("GOOD-BYE") ' good-bye
```

Запустите следующую программу, чтобы увидеть, как работают обе функции:

```
PROMPT "What would you like to convert?";
       ThisString$
PRINT "This is what happened when you use
      LOWER$:"
PRINT LOWER$(ThisString$)
```

```
PRINT
PRINT "This is what happened when you use
      UPPER$:"
PRINT UPPER$(ThisString$)
PRINT
END
```

Если вы запустите эту программу и введете строку I Don't Like Anyone Who Copies My Work (Мне не нравятся все те, кто копируют мою работу), результат работы будет следующим:

```
This is what happened when you use LOWER$:
i don't like anyone who copies ray work

This is what happened when you use UPPER$:
I DON'T LIKE ANYONE WHO COPIES MY WORK
```

Обе функции, LOWER$ и UPPERS, работают только с буквами. Они не применяются к символам ($, % или @) и числам.

Определение длины строки

Если вам необходимо манипулировать строками, вам понадобится определить длину строки. Для того чтобы определить количество символов в строке, воспользуйтесь функцией LEN, как показано ниже:

```
LEN("Greetings from Mars!")
```

В этом примере длина строки равна 20 символам, включая два пробела и восклицательный знак. Для того чтобы увидеть функцию LEN в действии, выполните следующую программу:

```
PROMPT "Type a string:"; CountMe$
TotalLength = LEN(CountV\Me$)
PRINT "The total number of characters in your string is:"
PRINT TotalLenght
END
```

Поэтому, если вы запустите программу и введете строку Beware of copycat publishers, результат будет следующим:

```
The total number of characters in your string is:
28
```

При определении общей длины строки функция LEN удаляет все пробелы до и после любых видимых символов, как показано в табл. 8.3.

Таблица 8.3. Как функция LEN подсчитывает пробелы в строке

Строка	Длина строки
" Hello!"	6 символов (удаляются пять пробелов в начале строки)
"What are you looking at?	24 символа (удаляются пять пробелов в конце строки)
"Boo! Go away!"	13 символов (учитываются пробелы между словами)

Обрезание строки

Строки содержат пробелы до и после видимых символов, поэтому вам понадобится удалить эти пробелы, прежде чем определять длину строки.

К счастью, в состав Liberty BASIC входит специальная функция для удаления лишних пробелов:

```
TRIM$("   Hello, there!") ' "Hello, there!"
TRIM$("Good-bye!    ")  ' "Good-bye"
```

Для того чтобы увидеть, как функция TRIM$ обрезает все лишние пробелы, запустите следующую программу:

```
Astring$ = " Hello, there!"
PRINT "The original length of the string = ";
      LEN(Astring$)
TempString$ = TRIM$(Astring$)
PRINT TempString$
PRINT "The new length is now = ";
      LEN(TempString$)
END
```

В результате выполнения программы на экране отобразится следующее:

```
 The original length of the string = 14
I Hello, there!
j The new length is now = 13
```

Добавление пробелов

Иногда вам нужно добавить несколько пробелов, обойдясь без многократного нажатия соответствующей клавиши на клавиатуре. Для выполнения такой задачи Liberty **BASIC** предлагает специальную функцию $PACE$:

```
SPACE$(X)
```

В этом примере X представляет собой количество пробелов, которые необходимо добавить. В следующей программе функция SPACE$ просто добавляет пять пробелов между словом Hello (Привет) и фразой Bo the cat (кот Bo):

```
Astring$ = "Bo the Cat."
•PRINT "Hello" = SPACE$(5) + Astring$
END
```

Удаление символов из строки

Если у вас длинная строка, вам понадобится только ее часть. Например, строка содержит чье-то имя и фамилию, а вам необходима только фамилия. Для удаления одного или нескольких символов из строки воспользуйтесь одной из следующих функций Liberty **BASIC.**

- ✓ Функция LEFT$ (string, X) удаляет X символов, начиная с левого края строки.
- ✓ Функция RIGHTS (string, X) удаляет X символов, начиная с правого края строки.
- ✓ Функция MID$ (string, X, Y) удаляет Y символов, начиная с символа номер X от левого края строки.

Для того чтобы увидеть, как работают эти функции, выполните следующую программу и посмотрите на результат:

```
FakeName$ = "John Barkley Doe"
FirstName$ = LEFT$(FakeName$, 4)
PRINT "This is the first name = "; FirstName$
LastName$ = RIGHT$(FakeName$, 3)
PRINT "This is the last name = "; LastName$
MiddleName$ = MID$( FakeName$, 6, 7)
PRINT "This is the middle name = "; MiddleName$
END.
```

Эта программа просто "вырезает" имя, второе имя и фамилию из строки John Barkley Doe.

Поиск строки в рамках другой строки

Если у вас длинная строка, вам понадобится определить расположение в ней какого-то слова или фразы. Предположим, у вас есть строка, содержащая целый перечень имен:

```
"John, Julia, Matt, Mike, Sam, Chris, Karen"
```

Если вам необходимо определить расположение в этой строке имени Matt, воспользуйтесь командой INSTR, как показано ниже:

```
Names$ = "John, Julia, Matt, Mike, Sam, Chris, Karen"
Position = INSTR(Names$, "Matt", 1)
PRINT "The name Matt is located in position = "; Position
END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. **Первая строка создает** строковую **переменную Name$ и присваивает ей значение John, Julia, Matt, Mike, Sam, Chris, Karen.**
2. **Вторая строка приказывает присвоить переменной Position значение, возвращенной функцией INSTR.**

   Функция INSTR как будто говорит компьютеру: "Начиная с определенного места, я просматриваю строку, которая представляется переменной Name$, и ищу в ней слово Matt". В нашем примере слово Matt начинается в строке с 14-й позиции.
3. **Третья строка выводит сообщение:** The name Matt is located in position (Имя Matt приведено на позиции)= 14.
4. **Четвертая строка сообщает компьютеру о завершении программы.**

Для того чтобы воспользоваться функцией INSTR, вам необходимо указать следующие три характеристики.

- ✓ Позицию, с которой будет проводиться поиск.

  В предыдущем пример поиск проводится прямо с позиции 1, которая представляет самое начало строки. Однако вы можете приказать компьютеру проводить поиск, начиная с любой позиции. Если вы прикажете компьютеру начать поиск с 20-й позиции, он никогда не найдет слово Matt в строке Name$.

- ✓ Строку, в которой необходимо провести поиск.
- ✓ Строку (фразу), которую необходимо найти,

Строки чувствительны к регистру, что означает, что с точки зрения компьютера Matt и MATT — это совершенно разные вещи. В следующей программе функция INSTR не найдет фразу MATT, так как считает, что Matt и MATT - это не одно и то же:

```
Names$ = "John, Julia, Matt, Mike, Sam, Chris, Karen"
Position - INSTR(Names$, "MATT", 1)
PRINT "The name Matt is located in position = "; Position
END
```

Если функция INSTR не нашла строку, которая вам нужна, ее значение будет равно нулю.

Преобразование строк в числа (и наоборот)

Строка может содержать буквы, символы и цифры. Чаще всего для хранения цифр строки используются, когда вам необходимо сохранить такие сведения, как номер телефона или адрес. Если вы хотите, чтобы компьютер напечатал номер телефона, вам необходимо сохранить этот номер как строку:

```
PRINT "555-1212"
```

После выполнения этой команды на экране отображается номер 555-1212. То, что номер телефона является строкой, компьютер определит по наличию кавычек. А что же произойдет, если вы выполните следующую команду:

```
PRINT 555-1212?
```

Liberty BASIC истолкует эту команду как: "Отними 1212 от 555 и отобрази полученный результат (равный −657) на экране".

Если у вы хотите, чтобы число трактовалось как строка, воспользуйтесь такой функцией Liberty BASIC, как STR$, которая выглядит следующим образом:

```
STR$(число)
```

Функция STR$ указывает Liberty BASIC на то, что число должно быть превращено в строку. Например, число 34 с помощью функции STR$ превращается в строку 34.

Для того чтобы вывести на печать строку и число, обычно приходится использовать команду PRINT совместно с разделителем в виде точки с запятой:

```
BossIQ = 12
PRINT "This is the IQ of your boss = "; BossIQ
END
```

Однако вы можете воспользоваться функцией STR$, чтобы преобразовать значение переменной BossIQ в строку:

```
BossIQ - 12
NewString$ = "This is the IQ of your boss = " + STR$(BossIQ)
PRINT NewString$
END
```

Основное различие между первым и вторым примерами состоит в том, что число BossIQ теперь стало частью строки NewString$.

Конечно, Liberty BASIC позволяет вам преобразовать и строки в числа. Вы не сможете выполнять со строкой (например, 4 6) никаких математических операций. Однако можно превратить строку в число, если воспользоваться функцией Liberty BASIC VAL, которая выглядит следующим образом:

```
VAL("Строка")
```

Функция VAL указывает Liberty BASIC на необходимость преобразования строки, являющейся ее аргументом, в число. Например, строку "45" легко преобразовать в число 45.

Если вы примените функцию VAL к строке, не содержащей чисел, например строке "Hello", функция VAL вернет нулевое значение.

Для того чтобы увидеть функцию VAL в действии, выполните следующую программу:

```
YearBorn$ = "1964"
PROMPT "You were born in "; YearBorn$
Year = 1999 - VAL(YearBorn$)
NOTICE "In 1999, you were this old - "; Year
END
```

Если вы запустите эту программу, на экране отобразится диалоговое окно с сообщением You were born in (Вы родились в) 1964. **Затем на экране отобразится** сообщение In 1999, you were this old (В 1999 вам было) 35.

Глава 9

Принятие решений с помощью управляющих операторов

В этой главе...

- Знакомство с булевыми выражениями
- Использование оператора IF THEN
- Использование оператора Select Case

Основное назначение программы — заставить компьютер вести себя определенным образом. Самые примитивные программы действуют точно так же, как и первом запуске, например отображают на экране фразу "Hello, world!" ("Здравствуй, мир!") или имя вашего любимого кота.

Такие примитивные программы очень хорошо подходят для изучения основ программирования, однако они совершенно бесполезны во всех остальных случаях, так как большинство программ должно принимать данные и изменять свое поведение в зависимости от этих данных.

Например, многие банки используют компьютеры для анализа факторов риска, прежде чем предоставлять денежные ссуды. Если ваш доход превышает определенную сумму и вы не оказывались банкротом на протяжении последних двух лет, компьютер, скорее всего, "одобрит" предоставление вам кредита, но не сделает это для человека без определенного рода деятельности.

Каждый раз, когда вы предоставляете программе различные данные, программа может выдавать совершенно различные ответы. Для того чтобы предоставить программе возможность принимать решения, вы должны использовать *управляющие операторы.*

Знакомство с булевыми выражениями

Когда вы принимаете какое-нибудь решение, например, что съесть на завтрак, вы задаете себе определенный вопрос: "А хочу ли я яичницу?". Если ответ положительный, вы отправитесь на кухню или в соответствующий ресторан.

Компьютеры работают приблизительно так же, хотя люди задают вопросы, а компьютеры проверяют *булевы выражения.* Булево выражение — это нечто, принимающее всего два значения — истина и ложь, или нуль и "не нуль".

Булевы выражения являются частью булевой алгебры, названной в честь Джорджа Буля (George Boole). (Если вы будете усердно учиться и создадите собственный раздел математики, его обязательно назовут в вашу честь.) Простейшие булевы выражения сравнивают два значения, как показано в табл. 9.1.

Таблица 9.1. Примеры булевых выражений

Булево выражение	Что означает	Булево значение
4<54	4 меньше 54	Истина
4>54	4 больше 54	Ложь
4=54	4 равно 54	Ложь
4<=54	4 меньше или равно 54	Истина
4>=54	4 больше или равно 54	Ложь
4054	4 не равно 54	Истина

Такие символы, как <, >, =, <=, >= и <>, называются *операторами отношений.*

Попробуйте запустить следующую программу, чтобы угадать, какое предложение будет напечатано:

```
IF (4 < 54) THEN
  PRINT "This prints out on-screen."
ELSE
  PRINT "Why did you pick this sentence?"
END IF
END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру вычислить булево выражение (4<54). Поскольку это истина, компьютер переходит к следующей строке программы.
2. Вторая строка приказывает компьютеру отобразить на экране сообщение This prints out on-screen. (Это напечатано на экране.). После этого компьютер пропускает третью и четвертую строки и переходит к пятой.
3. Третья строка как бы говорит компьютеру: если булево выражение окажется неверным, переходи к выполнению четвертой строки программы.
4. **Четвертая строка приказывает компьютеру отобразить на экране** сообщение Why did you pick this sentence? (Почему вы выбрали это предложение?). **Поскольку булево выражение** (4<54) **никогда не даст ответ** "ложь", третья **и четвертая строки программы никогда не выполняются.**
5. Пятая строка просто указывает на завершение оператора IF THEN ELSE (о котором подробно мы поговорим в подразделе "IF THEN ELSE" дальше в настоящей главе.)
6. Шестая строка сообщает компьютеру о завершении программы.

Переменную назначить булевому выражению можно следующим образом:

```
Guilty = (4 < 54}
```

В этом примере переменной Guilty присваивается значение true. Инструкция говорит компьютеру следующее: "Булево выражение (4<54) правильно. Поэтому значением переменной Guilty будет true".

Liberty BASIC совершенно ничего не известно о различии между правдой (true) и ложью (false). Так как Liberty BASIC не может назначить значение true булеву выражению, поэтому он просто присваивает ему значение -1, которое в данном случае соответствует true. Если Liberty BASIC необходимо присвоить булеву выражению значение false, на самом деле присваивается значение 0.

Попробуйте выполнить следующую программу:

```
Guilty = (4 < 54)
IF Guilty THEN
  PRINT "Slap him on the wrist and let him
        go."
ELSE
  PRINT "This sentence never prints out."
END IF
END
```

Каждый раз, когда вы запускаете эту программу, на печать выводится предложение Slap him on the wrist and let him go. (Пожмите ему руку и отпустите.).

Использование переменных в булевых выражениях

Пример программы из предыдущего раздела использовал булево выражение (4<54), которое достаточно бесполезно, так как всегда правильно. Каждый раз программа будет выводить на печать одно и то же предложение.

Для обеспечения большей гибкости булевы выражения обычно сравнивают две переменные или одну переменную и одно фиксированное значение, как показано в следующих примерах:

```
(MyIQ < AnotherIQ)
(Taxes < 100000)
```

Значение первого из двух выражений (MyIQ < AnotherIQ) зависит от конкретных значений переменных MyIQ и AnotherlQ. Запустите следующую программу, чтобы увидеть, как она выполняет различные наборы инструкций в зависимости от значений переменных MyIQ и AnotherlQ:

```
PROMPT "What is your IQ"; MyIQ
PROMPT "What is the IQ of another person";
       AnotherlQ
IF (MyIQ > AnotherlQ) THEN
  PRINT "I'm smarter than you are."
ELSE
  PRINT "You have a higher IQ to make up your
        lack of common sense."
END IF
END
```

Если вы выполните эту программу и введете различные значения для переменных MyIQ и AnotherIQ, программа поведет себя одним из двух возможных способов, выводя на печать либо предложение I'm smarter than you are. (А ведь я сообразительнее тебя.), либо You have a higher IQ to make up your lack of common sense. (У тебя IQ выше, так как тебе нужно компенсировать недостаток здравого смысла.).

Если вы используете переменные в булевых выражениях, вы предоставляете компьютеру большую гибкость при принятии решений.

Использование булевых операторов

Анализ одного булевого выражения может оказаться достаточно проблематичным, как показано в следующем примере:

```
PROMPT "How much money did you make"; Salary
PROMPT "How much money did you donate to
       political candidates"; Bribes
IF (Salary > 500) THEN
  IF (Bribes > 700) THEN
       PRINT "You don't need to pay any taxes. "
  END IF
END IF
END
```

Программа может только вывести на печать сообщение You don't need to pay any taxes. (Вам не нужно платить никаких налогов.), если оба булевых выражения (Salary>500 и Bribes>700) будут правильными.

Вместо того чтобы заставлять компьютер вычислять булевы выражения по одному, вы можете заставить его вычислять сразу несколько булевых выражений, если воспользуетесь *булевыми операторами*. Булев оператор — это не что иное, как объединение двух или нескольких булевых выражений для представления значения true или false.

В любом языке программирования используются следующие булевы операторы.

- ✓ AND
- ✓ OR
- ✓ XOR
- ✓ NOT

Оператор AND

Оператор AND (И) связывает вместе два булевых выражения. Например, вы можете переписать программу из предыдущего раздела, используя булев оператор AND:

```
PROMPT "How much money did you make"; Salary
PROMPT "How much money did you donate to
       political candidates"; Bribes
IF (Salary > 500) AND (Bribes > 700) THEN
  PRINT "You don't need to pay any taxes."
END IF
END
```

В данном примере программа выводит на печать сообщение You don't need to pay any taxes. (Вам не нужно платить никаких налогов.), если оба булевых выражения (Salary>500 и Bribes>700) будут правильными. Если одно из них окажется неправильным, на печать не выводится ничего.

Запустите программу, текст которой приведен выше, используя значения из табл. 9.2, и посмотрите на полученный результат.

Таблица 9,2. Результат использования оператора AND зависит от значений переменных

Значение переменной Salary	Значение переменной Bribes	Что сделает программа
100	100	Ничего
900	100	Ничего
100	900	Ничего
900	900	Выведет сообщение на печать

Оператор AND может представлять значение true только в том случае, если оба булева выражения, которые он связывает, также равны true.

Для того чтобы проиллюстрировать работу оператора AND, программисты любят использовать *таблицу истинности,* которая позволяет определить результат использования оператора в зависимости от значений обоих булевых выражений. В табл. 9.3 представлена таблица истинности для следующих двух выражений:

```
(Булево выражение 1) AND (Булево выражение 2}
```

Таблица 9.3. Таблица истинности для оператора AND

Значение булевого выражения 1	Значение булевого выражения 2	Значение (Булево выражение 1) AND (Булево выражение 2)
False (Ложь)	False (Ложь)	False (Ложь)
True (Истина)	False (Ложь)	False (Ложь)
False (Ложь)	True (Истина)	False (Ложь)
True (Истина)	True (Истина)	True (Истина)

Оператор OR

Оператор OR (или) связывает два булевых выражения, но выдает значение true (истина) в том случае, если значение true (истина) соответствует хотя бы одному из них. Для того чтобы проиллюстрировать работу оператора OR, выполните следующую программу:

```
PROMPT "How far can you throw a football";
       Football
PROMPT "What is your IQ"; IQ
IF (Football > 50) OR (IQ <= 45) THEN
  PRINT "You have what it takes to become a
        professional athlete!"
END IF
END
```

В этом случае программа напечатает сообщение You have what it takes to become a professional athlete! (У вас есть все необходимое для карьеры профессионального спортсмена!) при **выполнении любого из** булевых выражений

(Football > 50 **или** IQ <= 45). Только в том случае, если оба выражения окажутся неправильными, программа ничего не напечатает.

Запустите программу, текст которой приведен выше, используя значения из табл. 9.4, и посмотрите на полученный результат.

*Таблица **9.4.** Результат использования оператора **OR** зависит от значений переменных*

Значение переменкой Football	Значение переменной IQ	Что сделает программа
5	70	Ничего
70	70	Выведет сообщение на печать
5	5	Выведет сообщение на печать
70	5	Выведет сообщение на печать

Оператор OR может представлять значение false только в том случае, если оба булева выражения, которые он связывают, также равны false.

В табл. 9.5 представлена таблица истинности для применения оператора OR для связывания следующих двух выражений:

```
(Булево выражение 1) OR (Булево выражение 2)
```

*Таблица **9.5.** Таблица истинности для оператора OR*

Значение булева выражения 1	Значение булева выражения 2	Значение (Булево выражение 1) OR (Булево выражение 2)
False (Ложь)	False (Ложь)	False (Ложь)
True (Истина)	False (Ложь)	True (Истина)
False (Ложь)	True (Истина)	True (Истина)
True (Истина)	True (Истина)	True (Истина)

Оператор XOR

Оператор XOR (исключающее или) связывает два булевых выражения, но дает значение true только в том случае, если одно из выражений имеет значение true, а другое — false. В качестве примера использования этого оператора выполните следующую программу:

```
PROMPT "Is your spouse around (Type 1 for Yes
       or 0 for No)"; SpouseHere
PROMPT "Is your best friend around (Type 1
       for Yes or 0 for No)"; BestfriendHere
IF (SpouseHere = 0) XOR (BestfriendHere = 0) THEN
   PRINT "You won't be lonely tonight!"
END IF
END
```

Программа выведет сообщение You won't be lonely tonight! (Вы не будете сегодня одиноки!) **только в том случае,** если первое **выражение** (SpouseHere=1) будет **равно** true, **а второе** (BestfriendHere=0) будет **равно** false, **или наоборот** – SpouseHere=0 и BestfriendHere=1.

Если ваша супруга (SpouseHere=1) и ваш **лучший** друг (BestfriendHere=1) **окажутся рядом, ничего не произойдет. Точно так же, если ваша** супруга (SpouseHere=0) и ваш лучший **друг** (BestfriendHere=0) **отсутствуют, программа ничего не** напечатает.

Запустите **программу,** текст **которой приведен выше,** используя **значения из табл. 9.6, и посмотрите на полученный результат.**

Таблица 9.6. Результат использования оператора XOR зависит от значений переменных

Значение переменной SpounseHere	Значение переменной BestfriendHere	Что сделает программа
0	0	Ничего
1	0	Выведет сообщение на печать
0	1	Выведет сообщение на печать
1	1	Ничего

Оператор XOR представляет **значение** false **только в том** случае, если **оба булева выражения, которые он связывают, равны** false или true.

В табл. 9.7 представлена таблица **истинности для оператора** XOR **при связывании** следующих **двух** выражений:

```
(Булево выражение 1} XOR (Булево выражение 2)
```

Таблица S, 7, Таблица истинности для оператора XOR

Значение булева выражения 1	Значение булева выражения 2	Значение (Булево выражение 1) XOR (Булево выражение 2)
False (Ложь)	False (Ложь)	False (Ложь)
True (Истина)	False (Ложь)	True (Истина)
False (Ложь)	True (Истина)	True (Истина)
True (Истина)	True (Истина)	False (Ложь)

Оператор NOT

Оператор NOT (НЕ) **влияет только на одно булево выражение.** Если **булево выражение равно** true. **оператор** NOT делает **его равным** false. Если булево **выражение равно** false, **оператор** NOT делает **его равным** true, т.е. **оператор** кот **изменяет** значение **булевого выражение на противоположное.**

Например, значение приведенного ниже булевого **выражение равно** true:

```
(4 < 54)
```

А значение этого булевого выражение равно false:

```
NOT(4 < 54)
```

Если вы хотите присвоить значение false **переменной** Guilty, **это можно** сделать следующим **образом:**

```
Guilty = NOT(4 < 54)
```

Эта инструкция говорит компьютеру: "Значение булевого выражения (4<54) **равно** true. **Однако оператор** NOT **изменяет его значение с** true на false, **поэтому значе-**

ние переменной Guilty равно false". Для того чтобы проиллюстрировать работу оператора NOT, выполните следующую программу:

```
Guilty = NOT($ < 54)  ' The value of Guilty is false
IF Guilty THEN
  PRINT "This sentence never prints out."
ELSE
  PRINT "The defendant is not guilty because
        he's rich."
END IF
END
```

Каждый раз, когда вы выполняете эту программу, на печать отправляется сообщение The defendant is not guilty because he's rich. (Подзащитный невиновен, так как он богат.).

Знакомство с операторами IF THEN

Проще всего определить, какие действия выполняет компьютер, — использовать оператор IF THEN. Этот оператор проверяет, выполняется определенное условие или нет. Тогда компьютер должен выполнить одну или несколько инструкций.

Оператор IF THEN выглядит следующим образом:

```
IF (Булево выражение) THEN
  ' Следуй одной из перечисленных здесь инструкций
END IF
```

В качестве примера использования этого оператора давайте рассмотрим следующую программу:

```
I PROMPT "Do you eat hot dogs? (Type Y for Yes
        or N for No)"; Answer$
 IF (Answer$ - "Y") THEN
   PRINT "I have a nice hot dog you might
         like."
 END IF
 END
```

Только если вы введете Y (именно в верхнем регистре), программа напечатает сообщение I have a nice hot dog you might like. (У меня есть замечательный хот-дог, который вам обязательно понравится.).

Если вы хотите, чтобы компьютер следовал только одной определенной инструкции, указанной после оператора IF THEN (например, выполнил команду PRINT из последнего примера), вы можете сократить оператор IF THEN следующим образом:

```
: PROMPT "Do you eat hot dogs? (Type Y for Yes or N for No)";
 Answer$
 IF (AnswerS = "Y") THEN PRINT "I have a nice
    hot dog you might like."
 END
```

Если вы хотите, чтобы компьютер выполнил две или несколько инструкций, указанных после оператора IF THEN, вам необходимо заключить их между командами IF и END IF, как показано ниже.

```
PROMPT "Do you eat hot dogs? (Type Y for
       Yes or N for No)"; Answer$
IF (Answers >= 1) THEN
   PRINT "You have my sympathies."
   PRINT "Have you ever thought of
         getting"
   PRINT "your head examined real
               soon?"
END IF
END
```

IF THEN ELSE

Оператор IF THEN приказывает компьютеру следовать определенным инструкциям только при выполнении какого-то условия. Если условие не выполняется, компьютер просто игнорирует все соответствующие инструкции.

Оператор IF THEN ELSE немного отличается от оператора IF THEN, так как приказывает компьютеру выполнить один набор инструкций в случае выполнения условия, и другой набор — в случае его невыполнения. Оператор IF THEN ELSE выглядит следующим образом:

```
IF (Boolean expression) THEN
   'Следуй одной или нескольким из перечисленных здесь инструкций
ELSE
   ' Если условие не выполняется, тогда выполни эти инструкции
END IF
```

В качестве примера давайте рассмотрим следующую программу:

```
PROMPT "How long were you in medical school";
       Answer
IF (Answer > 4) THEN
   PRINT "Congratulations! You should be able to"
   PRINT "play a good game of golf in no time."
ELSE
   PRINT "You may not have been in medical school for"
   PRINT "very long, but at least should know"
   PRINT "how to put on a white lab coat."
END IF
END
```

В отличие от оператора IF THEN **оператор** IF THEN ELSE **заставляет компьютер выполнить какой-нибудь набор инструкций. В** этой **программе,** если **ответ** будет **превышать четверку, на экране отобразится** сообщение Congratulations! YOU should be able to play a good game of golf in no time. (Поздравляем! Бы должны суметь принять необходимое решение достаточно быстро.).

Если же ответ меньше **четверки, на экране отобразится** сообщение You may not have been in medical school for very long, but at least should know how to put on a white lab coat. (Вы обучаетесь в медицинской школе еще недостаточно долго, но уже должны знать, как надеть белый халат.).

Оператор IF THEN ELSE всегда заставляет компьютер выполнять какой-нибудь набор инструкций. Если же вы используете обычный оператор IF THEN, компьютер не предпринимает никаких действий.

IF THEN ELSE IF

Оператор IF THEN ELSE IF перечисляет два или больше наборов инструкций, хотя, в зависимости от значений каждого булевого выражения, компьютер может все равно не выполнить никаких действий. Оператор IF THEN ELSE IF выглядит следующим образом:

```
IF (Булево выражение 1) THEN
   ' Следуй одной или нескольким перечисленным здесь инструкциям
ELSE IF (Булево выражение 2) THEN
   'Следуй одной или нескольким перечисленным здесь инструкциям
END IF
```

В качестве примера использования оператора IF THEN ELSE IF введите и выполните следующую профамму:

```
PROMPT "How long did you go to law school?";
       Answer
IF (Answer > 4} THEN
   PRINT "Congratulations! You now know enough to"
   PRINT "sue your law school for wasting your time."
ELSE IF (Answer > 2) THEN
   PRINT "You may not have finished law school yet"
   PRINT "but that doesn't mean you can't lie about it."
END IF
END
```

Если пользователь введет значение, превышающее 4, программа отобразит на экране сообщение Congratulations! You now know enough to sue your law school for wasting your time. (Поздравляем! Вы уже должны знать достаточно много, чтобы преследовать руководство школы за ненужную трату времени.)

Обратите внимание: компьютер выполнит набор инструкций, а после этого прекратит дальнейшую проверку, хотя условия, указанные после команды ELSE IF, также правильны.

Если пользователь введет число 3 или 4, программа отобразит на экране сообщение You may not have finished law school yet but that doesn't mean you can't lie about it." (Вы еще не закончили школу, но это не означает, что вы не можете рассказывать о ней небылицы.) **А что же произойдет,** если **пользователь введет значение** 2, **1 или О? В** данном **случае не произойдет совершенно ничего.**

Оператор IF THEN ELSE может предложить целый ряд условий, как показано ниже:

```
IF (Булево выражение 1) THEN
   ' Следуй одной или нескольким перечисленным здесь инструкциям
ELSE IF (Булево выражение 2) THEN
   ' Следуй одной или нескольким перечисленным здесь инструкциям
ELSE IF (Булево выражение 3) THEN
   ' Следуй одной или нескольким перечисленным здесь инструкциям
END IF
```

Для того чтобы исследовать работу программы, запустите следующую профамму, введите числа 10, 4 и 1 и посмотрите на результат:

```
I PROMPT "How many millions did your government
         waste last year"; Answer
IF (Answer > 8) THEN
   PRINT "Congratulations! You must live in a true"
```

```
   PRINT "democracy."
ELSE IF (Answer > 2) THEN
   PRINT "You government isn't spending enough. Tell them"
   PRINT "to start buying weapons they don't need."
ELSE IF (Answer > 0) THEN
   PRINT "Your government obviously concerned with"
   PRINT "staying under budget, which means it's not"
   PRINT- "likely to last much longer."
END IF
END
```

Если вы введете 10, что означает выполнение булевого выражения (Anwser>8), **программа отобразит на экране сообщение** Congratulations ! You must live in a true democracy. (Поздравляем! Вы должны жить в настоящей демократической стране.)

Если вы введете 4, что означает выполнение булевого выражения (Anwser>2), **программа отобразит на экране сообщение** You government isn't spending enough. Tell them to start buying weapons they don't need. (Ваше правительство тратит недостаточно много. Скажите ему, что пора приобретать совершенно не нужное вооружение.).

Если вы введете 1, что означает выполнение условия ELSE IF (Anwser>2), **программа отобразит на экране сообщение** Your government obviously concerned with likely to staying under budget, which means it's not last much longer. (Ваше правительство определенно решило уложиться в выделенный бюджет, значит, оно вряд ли долго пробудет у власти.)

Использование оператора Select Case

Перечисление нескольких условий при использовании IF THEN ELSE IF весьма утомительная задача, как показано ниже:

```
PROMPT "How old are you"; Answer
IF (Answer > 21) THEN .
   PRINT "Congratulations! You must be able to rent a"
   PRINT "car in some states."
ELSE IF (Answer > 20) THEN
   PRINT "You can't rent a car, but you're
         preapproved""
   PRINT "for 20 different credit cards."
ELSE IF (Answer. > 19) THEN
   PRINT "You're still officially a teenager."
ELSE IF (Answer > 18) THEN
   PRINT "You're old enough to join the military and"
   PRINT "fire an automatic rifle, but you still can't"
   PRINT "buy beer legally. Figure that one out."
ELSE IF (Answer > 17) THEN
   PRINT "You can see R-rated movies on"
   PRINT "your own (but you've probably done that for years)."
END IF
END
```

Поэтому многие языки программирования вместо оператора IF THEN ELSE IF **предлагают оператор** SELECT CASE, **который выглядит** следующим **образом:**

```
' SELECT CASE Переменная
CASE Значение1
   ' Следовать этим инструкциям, если переменная = Значение1
CASE Значение2
   'Следовать этим инструкциям, если переменная = Значение2
END SELECT
```

В отличие от других версий BASIC, Liberty BASIC не позволяет использовать оператор SELECT CASE. Дальнейший материал настоящего раздела касается такого диалекта BASIC, как QBASIC компании *Microsoft*.

Как и оператор IF THEN ELSE IF, оператор SELECT CASE предлагает несколько инструкций в зависимости от указанного значения переменной. Если вы перепишете предыдущую программу с использованием оператора SELECT CASE, она примет следующий вид:

```
INPUT "How old are you"; Answer
SELECT CASE Answer
CASE 21
   PRINT "Congratulations! You must be able to rent a"
   PRINT "car in some states."
CASE 20
   PRINT "You can't rent a car, but you're preapproved"
   PRINT "for 20 different credit cards."
CASE 19
   PRINT "You're still officially a teenager."
CASE 18
   PRINT "You're old enough to join the military and"
   PRINT "fire an automatic rifle, but you still can't"
   PRINT "buy beer legally. Figure that one out."
CASE 17
   PRINT "You can see R-rated movies on"
   PRINT "your own (but you've probably done that for years)."
END SELECT
| END
```

Если пользователь введет 21, программа отобразит на экране сообщение Congratulations! You must be able to rent a car in some states. (Поздравляем! Вы должны суметь взять напрокат автомобиль в некоторых штатах США.). Если пользователь введет 20, программа отобразит на экране сообщение You can't rent a car, but you're pre-approved for 20 different credit cards. (Вы не можете взять напрокат автомобиль, но вы имеете право использовать больше 20 кредитных карточек.). Если пользователь введет **19,** программа отобразит на экране сообщение You're still officially a teenager. (По закону вы еще подросток.). Если пользователь введет 18, программа отобразит на экране сообщение You're old enough to join the military and fire an automatic rifle, but you still can't buy beer legally. Figure that one out. (Вы достаточно взрослы, чтобы вступить в армию и стрелять из автомата, но вы все еще не имеете права приобретать пиво. Попробуйте узнать, почему именно.). Если пользователь введет **17,** программа отобразит на экране сообщение You can see R-rated movies on your own but you've probably done that for years. (Вам еще нельзя смотреть фильмы для взрослых, но вы уже наверняка смотрите их не первый год.).

Конечно же, если пользователь введет любое значение, не перечисленное оператором SELECT CASE, например 22 или **16,** оператор SELECT CASE не выполнит ни одной инструкции.

Для того чтобы убедиться в том, что компьютер выполнит как минимум одну инструкцию оператора SELECT CASE, просто добавьте команду CASE ELSE:

```
INPUT "How old are you"; Answer
SELECT CASE Answer
CASE 21
  - PRINT "Congratulations! You must be able to rent a"
    PRINT "car in some states."
CASE 20
   PRINT "You can't rent a car, but you're pre-approved"
   PRINT "for 20 different credit cards."
CASE 19
   PRINT "You're still officially a teenager."
CASE 18
   PRINT "You're old enough to join the military and"
   PRINT "fire an automatic rifle, but you still can't"
   PRINT "buy beer legally. Figure that one out."
CASE 17
   PRINT "You can see R-rated movies on" -
   PRINT "your own (but you've probably done that for  years)."
CASE ELSE
   PRINT "This sentence prints out if the user does NOT"
   PRINT "type numbers 17, 18, 19, 20, or 21."
END SELECT
:END
```

Если **вы** запустите **эту программу** и введете **21, программа отобразит на экране сообщение** Congratulations! You must be able to rent a car in some states. (Поздравляем! Вы должны суметь взять напрокат автомобиль в некоторых штатах США.). Если вы введете **20, программа отобразит на экране сообщение** YOU can't rent a car, but you're pre-approved for 20 different credit cards. (Вы не можете взять напрокат автомобиль, но вы имеете право использовать больше 20 кредитных карточек.). **Если вы** введете **19, программа отобразит на экране** сообщение You're still officially a teenager. (По закону вы еще подросток.). Если **вы** введете **18, программа отобразит на экране сообщение** You're old enough to join the military and fire an automatic rifle, but you still can't buy beer legally. Figure that one out. (Вы достаточно взрослы, чтобы вступить в армию и стрелять из автомата, но вы все еще не имеете права приобретать пиво. Попробуйте узнать, почему именно.). **Если вы введете 17, программа** отобразит на **экране сообщение** You can see R-rated movies on your own but you've probably done that for years. (Вам еще нельзя смотреть фильмы для взрослых, но вы уже наверняка смотрите их не первый год.). Если **вы введете любое другое** число **(например, 54 или 97), программа отобразит на экране сообщение** This sentence prints out if the user does NOT type numbers 17, 18, 19, 20, or 21. (Это предложение будет напечатано только в том случае, если пользователь НЕ введет числа 17, 18, 19, 20, 21.)

Проверка диапазона значений

Вы очень часто будете использовать оператор SELECT CASE для проверки того, равна ли переменная определенному значению, например числу 21 или строке "да". Однако иногда вам понадобится запустить целый набор инструкций в том случае, если значение переменной находится в определенном диапазоне, например между 3 и 18. В данном случае вам пригодится команда то:

```
CASE 1 TO 10
```

Эта команда позволяет проверить, попадает ли значение переменной в диапазон от 1 до 10, как показано в следующей программе.

```
INPUT "How many water balloons do you have";
      Answer
SELECT CASE Answer
CASE 1 TO 10
  PRINT "You need more water balloons."
CASE 11 TO 1000
  PRINT "Now you need a target."
CASE ELSE
  PRINT "What are you? A peace-loving hippie freak?"
END SELECT
END
```

В этом примере, если пользователь ввел число от 1 до 10, программа отобразит на экране сообщение You need more water balloons. (Вам необходимо приобрести дополнительные баллоны.). Если пользователь **ввел число от** 11 **до** 1000, **программа отобразит на** экране **сообщение** Now you need a target. (Теперь вам необходимо найти цель.). **Если пользователь ввел нуль,** отрицательное **число или любое число** больше **1000, программа отобразит на экране сообщение** What are you? A peace-loving hippie freak? (И кто вы такой? Несчастный хиппи — любитель мира?).

Убедитесь в том, что диапазоны не перекрываются разными частями оператора SELECT CASE, как показано ниже:

```
SELECT CASE Answer
CASE 1 TO 10
  PRINT "This always prints."
CASE 5 TO 10
  PRINT "This never prints."
END SELECT
```

В этом примере оператора SELECT CASE **программа отобразит на экране** сообщение This always prints. (Это предложение будет печататься всегда.), если пользователь введет **число от 1 до 10, но никогда не отобразит на экране сообщение** This never prints. (Это предложение никогда не будет напечатано.) — **условие** CASE 5 TO 10. **Это связано с тем, что первое условие оператора** SELECT CASE **выполняется первым, не оставляя** второму **условию ни одного шанса быть выполненным.**

Проверка оператора отношений

Иногда проверка точного значения или диапазона значений оказывается слишком ограниченной. Вы можете, например, сравнить переменную с определенным значением, воспользовавшись так хорошо известными *операторами отношений.* Оператор отношений позволяет оператору *SELECT CASE* определить, переменная больше чем (>), больше чем или не равна определенному значению.

Для того чтобы воспользоваться оператором отношений, вы должны использовать команду IS, как показано в следующем примере:

```
CASE IS <= 5
```

Эта команда проверяет, переменная меньше значения 5 или равна ему, как показано в следующей программе:

```
INPUT "How many cats do you own"; Answer
SELECT CASE Answer
CASE IS <= 5
  PRINT "You need more cats."
CASE IS > 5
  PRINT "Are you out of your mind?"
EMD SELECT
END
```

Если пользователь введет число, меньшее или равное 5, программа напечатает сообщение You need more cats. (Вам стоит завести еще несколько кошек.). Если пользователь введет число, большее 5 или равное 5, программа напечатает сообщение Are you out of your mind? (Вы в своем уме?).

Убедитесь в том, что используемые операторы отношений не конфликтуют с другими частями оператора SELECT CASE, что произошло в следующем примере:

```
SELECT CASE Answer
CASE IS < 10
  PRINT "This always prints."
CASE IS < 12
  PRINT "This prints only if the user types 11."
END SELECT
```

В данном случае программа выведет на печать сообщение This always prints, (Это предложение будет всегда печататься.), если пользователь введет число меньше 10, или сообщение This prints only if the user types 11. (Это предложение будет напечатано только в том случае, если пользователь введет число 11.), если пользователь введет число 11.

Не стоит использовать оператор SELECT CASE в C/C++ или Java

В BASIC и многих других языках программирования, таких как Pascal, оператор SELECT CASE выполняет только один набор инструкций, если обнаруживает соответствующее совпадение, например отправляет на печать сообщение You're still officially a teenager. (По закону вы еще подросток.) в примере на языке QBASIC из раздела "Использование оператора SelectCase", если пользователь введет число 19.

Однако языки программирования C/C++ и Java ведут себя несколько иначе. Если вы используете эти языки профаммирования, вы должны специально приказать компьютеру перестать выполнять инструкции оператора SELECT CASE (технически на языках программирования C/C++ и Java это называется использованием *переключателей*), воспользовавшись командой break.

Рассмотрите, например, следующую программу, написанную на C:

```
#include <stdio.h>
main ()
{
  char akey;
       printf ("Type a lower case letter ");
  scanf(" ");
- scanf ("%c" , &akey);
  switch (akey) {
       case 'a' .: printf ("You pressed the A key.n");
```

```
        case 'b' : printf ("You pressed the B key).\n");
    }
}
```

Если вы запустите эту программу и нажмете клавишу <A>, программа напечатает следующее:

You pressed the A key (Вы нажали клавишу A)

You pressed the A key (Вы нажали клавишу A)

При использовании C/C++ или Java компьютер выполнит все инструкции оператора-переключателя, начиная с первого совпадения и до конца. Чтобы убедиться в том, что программа, написанная на C/C++ или Java, прекратила выполнение всех инструкций оператора-переключателя, вы должны добавить в текст программы команду break:

```
#include <stdio.h>
main {}
{
  char akey;
  printf ("Type a lower case letter ");
  scanf(" ");
; scanf ("%c" , &akey);
  switch (akey) {
  case 'a' : printf ("You pressed the A key.\n");
  break;
  case 'b' : printf ("You pressed the B key.\n");
  }
}
```

Если вы действительно планируете писать профаммы на C/C++ или Java, вы должны никогда не забывать о целом ряде отличий между программами, написанными на C/C++ или Java, и программами, написанными на BASIC.

Глава 10

Использование циклов

В этой главе...

- Создание циклов с помощью команды WHILE WEND
- Создание циклов с помощью команды FOR NEXT

В общем говоря, программисты стремятся к тому, чтобы компьютер выполнял как можно больше работы, а они — как можно меньше. В идеале вы должны писать очень небольшие программы, не только потому, что их проще отлаживать и при необходимости модифицировать, но и потому, что при их создании вам придется вводить меньше текста.

Один из способов сокращения числа символов, которые должны вводить программисты, состоит в использовании *циклов*. Основная идея в том, чтобы заставить компьютер повторять выполнение одной или нескольких инструкций. Например, обратите внимание на следующую программу, которая выводит на экран числа от 1 до 5:

```
PRINT 1
PRINT 2
PRINT 3
PRINT 4
PRINT 5
END
```

Если вы хотите, чтобы программа вывела на экран числа от 1 до 5000000, представляете, что вам придется сделать? Вам придется набрать пять миллионов инструкций. Так как вы наверняка не хотите печатать столько инструкций, вы можете использовать циклы для того, чтобы заставить компьютер многократно выполнять определенные инструкции. Компьютеру предстоит выполнить серьезную работу. Рассмотрите следующий пример, в котором используется цикл FOR NEXT:

```
FOR I = 1 TO 5
   PRINT I
NEXT I
END
```

Эта программа выводит на экран следующее:

```
1
2
3
4
5
```

Как видно, когда вы используете цикл, программа становится намного меньше.

Если вы запустите эту программу, она выполнит то же самое, что и первый вариант программы. Однако программа, в которой используется цикл, может напечатать числа от 1 до 5 или от I до 5000000 (достаточно изменить всего одно число в программе). Циклы заставляют компьютер интенсивно работать и при этом избавляют вас от необходимости вводить дополнительные команды.

Хотя циклы и позволяют создавать более короткие программы, недостаток состоит в том, что в них тяжелее разобраться, чем в простой последовательности инструкций. Если вы создаете циклы, убедитесь в том, что написали доходчивые комментарии относительно работы программы. (Подробно об использовании комментариев мы говорили в главе 7, "Переменные, константы и комментарии".)

Цикл заставляет компьютер выполнять многократно одни и те же инструкции, однако, конечно же, компьютер должен знать, когда ему остановиться. Для того чтобы сообщить компьютеру об этом, вам необходимо указать *условие,* которое должно представлять значение true или false.

В мире математики и программирования все, что представляет значение true или false, называется *булевым выражением;* подробно о булевых выражениях мы говорили в главе 9, "Принятие решений с помощью управляющих операторов". В качестве примера булевого выражения можно привести выражение 4>9,48 (что в булевой алгебре представляет значение false).

Создание циклов с помощью команды WHILE WEND

Циклы очень удобны при повторном выполнении одной или нескольких инструкций. Конечно же, создавая цикл, вы должны указать условие, по которому он будет прерываться.

Одна из самых распространенных проблем, связанных с циклами, связана с так называемыми *бесконечными циклами,* когда компьютер выполняет набор инструкций и никогда не останавливается. Если программа приступила к выполнению бесконечного цикла, создается впечатление, что она простонапросто зависла. Обычно для прекращения бесконечного цикла приходится перегружать компьютер.

Единственный способ заставить цикл прекратить выполняться — проверка выполнения определенного условия. Для этого в Liberty BASIC используется специальный цикл WHILE WEND:

```
WHILE (Булево выражение = true)
   ' Одна или несколько инструкций
WEND
```

Для **того чтобы задать выполнение одной или нескольких инструкций, вам достаточно поместить** их между **командами** WHILE и WEND, как показано **в следующем примере:**

```
I = 1
WHILE I < 5
   PRINT "The square of "; I; " is "; I * I
   I = I + 1
WEND
END
```

Эта программа выполняет следующие действия.

1. Первая строка создает переменную I и присваивает ей значение 1.

2. Вторая строка сообщает компьютеру о начале цикла, а также о том, что цикл, инструкции которого заключены между командами WHILE и WEND, должен выполняться до тех пор, пока будет оставаться правильным условие I<5.

3. Третья строка приказывает компьютеру вывести на экран сообщение The square of 1 is 1 (Квадрат 1 равен 1) после первого выполнения цикла WHILE WEND, сообщение The square cf 2 is 4 (Квадрат 2 равен 4) после второго выполнения цикла WHILE WEND и т.д.

4. Четвертая строка приказывает компьютеру увеличивать значение переменной I на 1, чтобы она была равна 1 + 1, т.е. 2.

5. Пятая строка приказывает компьютеру выполнять проверку условия I<5. Если условие не выполняется, программа переходит к шестой строке. Если же оно выполняется (значение переменной I равно 1, 2, 3 или 4), программа возвращается к началу цикла, т.е. второй строке. Компьютер выполняет цикл четыре раза, выводя на экран следующее:

```
The square of 1 is 1 (Квадрат 1 равен 1)
The square of 2 is 4 (Квадрат 2 равен 4)
The square of 3 is 9 (Квадрат 3 равен 9)
The square of 4 is 16 (Квадрат 4 равен 16)
```

6. Шестая строка сообщает компьютеру о завершении программы.

Бесконечный цикл N1: не нужно забывать изменять булево выражение в цикле

При использовании циклов самая важная задача, которая стоит перед программистом, — избежать возникновения бесконечных циклов. *Бесконечный цикл* возникает в тех ситуациях, когда булево выражение никогда не изменяется, а значит, компьютер не может прекратить выполнение инструкций цикла, что и показано на следующем примере;

```
I = 1
WHILE I < 5
   PRINT "This loop never ends."
WEND
END
```

Этот цикл **никогда не закончится, поскольку значение переменной I не** изменяется. **Таким образом, булево выражение** I<5 **всегда остается правильным, а инструкции** цикла WHILE WEND не прекращают **выполняться. В данном** случае **программа постоянно выводит на экран** сообщение This loop never ends. (Этот цикл никогда не закончится.).

Для того чтобы исправить ситуацию, вам необходимо добавить в цикл WHILE WEND инструкцию, которая изменяла бы значение переменной I, например, инструкцию I=I+1, как показано ниже:

```
I = 1
WHILE I < 5
   PRINT "This loop eventually ends."
   I = I + 1
WEND
END
```

Бесконечные циклы — это достаточно частая ошибка, не позволяющая программам функционировать должным образом. Вы можете определить наличие бесконечного цикла по тому, что программа зависла и для ее перезапуска вам никак не обойтись без перезагрузки компьютера.

Бесконечный цикл N2: не нужно забывать задавать булево выражение перед началом цикла

Еще один вариант бесконечного цикла возникает в том случае, если вы забыли определить значение переменной за пределами цикла WHILE WHEND, как показано в следующем примере:

```
WHILE I < 5
   I = 1
   PRINT "This loop never ends."
   I = I + 1
WEND
END
```

В этом примере значение переменной I всегда определяется равным 1 в самом начале цикла WHILE WHEND, а затем увеличивается до 2 в его конце. Однако при каждом выполнении цикла WHILE WEND значение переменной I становится равным 1, а значит, булево выражение I<5 остается правильным и выполнение цикла никогда не прекращается.

Для того чтобы исправить поведение этого цикла WHILE WEND, строку I=1 необходимо вынести за его пределы:

```
I = 1
WHILE I < 5
   PRINT "This loop eventually ends."
   I = I + 1
WEND
END
```

В этом примере значение переменной I задается равным 1 еще до того, как компьютер приступит к выполнению цикла WHILE WEND. После того как цикл начнет выполняться, значение переменной I будет постоянно возрастать до тех пор, пока не станет равным 5, а значит, булево выражение I<5 перестанет быть правильным; в результате выполнение цикла WHILE WEND прекратится.

Для того чтобы избежать возникновения циклов в вашей программе, вам следует не забывать о двух следующих моментах.

- ✓ Определите значения всех переменных, используемых в булевом выражении, до того как начнется цикл WHILE WEND. Это позволит вам удостовериться в том, что выполнение цикла началось с необходимого значения переменной.
- ✓ Убедите в том, что значения переменных изменяются в пределах цикла WHILE WEND. Это позволит вам избежать возникновения бесконечных циклов.

Создание циклов с помощью команды FOR NEXT

Условный цикл, такой как WHILE WEND, прекращает свое выполнение только при выполнении (или не выполнении) определенного условия. Поэтому количество раз, которые будет выполняться цикл, может варьироваться.

Однако иногда вам понадобится цикл, который будет выполняться строго определенное количество раз. Хотя вы и можете использовать условные циклы, гораздо удобнее использовать цикл FOR NEXT, который на Liberty BASIC выглядит следующим образом:

```
FOR counter = начальное_зыачение TO конечное_значение
   ' Одна или несколько инструкций
NEXT
```

Этот цикл приказывает компьютеру повторять инструкции, указанные между командами FOR и NEXT, определенное количество раз. Количество повторений определяется начальным и конечным значениями переменной-счетчика. Типичный цикл FOR NEXT выглядит следующим образом:

```
FOR I = 1 TO 10
  PRINT "The square of "; I; "is "; I * I
NEXT
END
```

Если вы запустите эту программу, произойдет следующее.

1. Первая строка создаст переменную I и заставит компьютер выполнять инструкции цикла десять раз.
2. Вторая строка приказывает компьютеру вывести на экран сообщение The square of 1 is 1 (Квадрат 1 равен 1). **При каждом выполнении этой строки значение переменной** I будет другим, и в результате **программа выведет на экран** следующие **сообщения:**

```
The square of 1 is 1 (Квадрат числа 1 равен 1)
The square of 2 is 4 (Квадрат числа 2 равен 4)
The square of 3 is 9 (Квадрат числа 3 равен 9)
The square of 4 is 16 (Квадрат числа 4 равен 16)
The square of 5 is 25 (Квадрат числа 5 равен 25)
The square of 6 is 36 (Квадрат числа 6 равен 36)
The square of 7 is 49 (Квадрат числа 7 равен 49)
The square of 8 is 64 (Квадрат числа 8 равен 64)
The square of 9 is 81 (Квадрат числа 9 равен 81)
The square of 10 is 100 (Квадрат числа 10 равен 100)
```

3. Третья строка прикажет компьютеру вернуться к выполнению второй строки.
4. **Четвертая строка сообщит компьютеру о завершении программы.**

Если начальное значение переменной превышает конечное значение, цикл FOR NEXT вообще не выполняется, как показано в следующем примере:

```
FOR counter = 8 TO 2
   ' Этот цикл не сработает вообще
NEXT
```

Использование разных начальных значений

Чаще всего в циклах FOR NEXT значение переменной изменяется от 1 до другого фиксированного значения, например 10. Однако в циклах FOR NEXT в качестве начального значения переменной-счетчика может использоваться любое другое число, как показано ниже:

```
FOR I - в to 14
   PRINT "The value of I =";I
NEXT
END
```

В результате выполнения цикла на экране будут отображены следующие сообщения:

```
IThe value of I = 8
The value of I = 9
The value of I = 10
The value of I = 11
The value of I = 12
The value of I = 13
The value of I = 14
```

В качестве начального или конечного значения переменной-счетчика можно использовать и отрицательные числа, как показано ниже:

```
FOR counter = -5 TO 3
   ' Одна или несколько инструкций
NEXT
```

Если у вас нет веских причин использовать другие значения в качестве начального или конечного значения переменной-счетчика, лучше всего начинать выполнение цикла со значения 1. Тогда любому человеку будет намного проще определить, сколько раз выполняется тот или иной цикл.

Изменение шага

В цикле FOR NEXT значение переменной-счетчика обычно увеличивается. Например, следующий цикл выполняется четыре раза:

```
FOR counter - 1 TO 4
   ' Одна или несколько инструкций
NEXT
```

Для того чтобы значение переменной-счетчика изменялось на другое число, в цикле FOR NEXT можно использовать команду STEP:

```
FOR counter = 1 TO 4 STEP шаг
   ' Одна или несколько инструкций
NEXT
```

Например, если вы хотите, чтобы значение переменной-счетчика увеличивалось на 2, цикл FOR NEXT должен выглядеть следующим образом:

```
FOR counter = 1 TO 8 STEP 2
   PRINT "The value of I = "; I
NEXT
```

Если вы запустите **этот** цикл, он будет **выполняться не** восемь, **а четыре раза, а на экране отобразятся** следующие сообщения:

```
IThe value of I = 1
 The value of I = 3
 The value of I = 5
 The value of I = 7
```

Вы можете изменить "направление" изменений **значения** переменной-счетчика, **используя отрицательное значение** шага. **Только в этом** случае **начальное значение переменной-счетчика может быть** больше **конечного, как** показано ниже:

```
FOR counter = 6 TO -8 STEP -3
   PRINT "The value cf I = "; I
NEXT
```

Как цикл FOR NEXT выглядит в программе, написанной на С

Поскольку многие предпочитают писать программы на С, я решил привести вам пример использования цикла FOR NEXT в программе, написанной на С:

```
main O
{
  int counter;
  for )counter = 1; counter <= 5; counter++) {
  printf ("The square of Id is %d\n", counter,    counter * counter);
  }
}
```

Цикл FOR NEXT как бы говорит компьютеру: "Создай целую переменную counter и присвой ему значение 1 (counter=I). Затем продолжай увеличивать значение этой переменной на 1 (counter++) до тех пор, пока булево выражение (counter <= 5) не перестанет выполняться. Затем прекращай выполнение цикла". (В данном случае цикл FOR NEXT выполняется ровно пять раз.)

Если версия цикла FOR NEXT на языке С кажется вам несколько зашифрованной, так оно и есть. Теперь вы понимаете, почему изучать основы программирования лучше на BASIC, чем на С.

Если вы запустите эту программу, цикл FOR NEXT **выполнится ровно пять раз, а на экране отобразятся следующие** сообщения:

```
jThevalue of I = 6
 The value of I = 3
The value of I = 0
 The value of I = -3
•Thevalue of I = -6
```

Часть III

Дополнительные приемы программирования на Liberty BASIC

В этой части...

В настоящей части вы найдете много полезного и интересного материала, по- i священного созданию графических изображений, воспроизведению звука, сохране- I нию данных на дискете или жестком диске, созданию интерфейса пользователя и многому другому. Познакомившись с подобными средствами Liberty BASIC, вы сможете создавать действительно полезные и интересные программы, практически ! ничем не уступающие коммерческим продуктам.

Для того чтобы максимально разобраться со всеми этими средствами, убедитесь, | что вы набрали тексты всех приведенных мной программ и обязательно их выпол- нили. Кроме того, постарайтесь понять отличия программ, написанных на Liberty BASIC, от программ, написанных на других языка программирования, таких как C/C++ или Java, Чем больше программ вы напишете на Liberty BASIC при 1 изучении основ программирования, тем проще вам будет перейти к использованию другого языка программирования, например C/C++, в будущем.

Глава 11

Использование подпрограмм

В этой главе...

- Что такое структурное программирование
- Разделение программы на модули
- Создание подпрограмм
- Создание функций

Программирование — это скорее искусство, чем строгая наука. Основная цель программирования — создание программы как можно меньшего размера, которая использовала бы ресурсы компьютера (память, жесткий диск и т.д.) как можно интенсивнее.

Хотя некоторые люди обладают настоящим талантом в написании небольшого, компактного и быстро выполняемого кода (говоря языком, принятым в кругах программистов), остальным при написании программ необходима помощь.

Небольшие программы проще писать; в них также намного проще разбираться. Если вы напишете огромную программу, содержащую несколько тысяч строк инструкций, как в ней потом разобраться? Вам лучше продумать структуру программы с самого начала, чем потратить массу времени на написание огромной программы, которая, к тому же, и не заработает.

Оставим неправильный подход к написанию программ в прошлом

Написать программу можно одним из миллиона, а то и больше способов. Нет в мире двух людей, которые бы совершенно одинаково написали программу, решающую одну и ту же задачу (при условии, если один из них не скопирует программу у другого, конечно же).

Раньше люди писали программы без какого-либо предварительного планирования, что равносильно попытке написать роман, запустив текстовый процессор и непрерывно набирая текст до завершения всего произведения. Конечно, таким образом вы можете создать и настоящий шедевр, но, к сожалению, чаще всего в результате такого подхода получается что-то неудобоваримое.

Точно так же программисты писали программы, вводя команды в окне компилятора до тех пор, пока не получат работоспособную программу. После того как простая программа заработала, в нее добавляли новые строки.

К сожалению, программисты часто добавляли инструкции в программы без предварительного планирования. Некоторые программисты добавляли новые инструкции в начале программы; другие — в конце программы; остальные распределяли новые инструкции между существующими, что превращало определение новых инструкций в программе в невыполнимую задачу.

Подобная практика неорганизованного написания программ часто приводила к созданию нечитаемых программ, в которые нельзя вносить изменения или исправле-

ния (однако компании продолжают зарабатывать деньги, продавая программы, несмотря ни на какие обстоятельства).

Одна из причин того, что программисты привыкли писать программы, в которых практически невозможно разобраться, состоит в использовании специальной команды GOTO. Эта команда указывает компьютеру на необходимость перехода к определенной части программы. Например, команда GOTO LebelOne приказывает компьютеру немедленно перейти к выполнению части программы, которая начинается с метки LebelOne.

Команда GOTO позволила программистам создавать программы, которые заставляли компьютер перескакивать с одного места в программе к другому в поисках инструкций. Программы с использованием команды GOTO во многом похожи на роман, на второй странице которого вам предлагают перейти к странице 349, на странице 349 вам предлагают перейти к странице 34, страница 34 предлагает перейти к странице 125 и т.д.

Поскольку постоянные скачки от одной части программы к другой столь же запутанны, как и попытки проследить переплетения макарон, написанные подобным образом программы назвали "макаронными".

Например, рассмотрите следующую программу, написанную на Liberty BASIC:

```
GOTO [LabelOne]
[LabelFour]
  PROMPT "How much money do you have?"; MyCash
  GOTO [LabelThree]
[LabelTwo]
  END
; [LabelOne]
GOTO [LabelFour]
[LabelThree]
  PRINT "You owe me = "; MyCash - .95
  GOTO [LabelTwo]
```

Попытка разобраться в том, что же выполняет данная программа, приведет к определенным проблемам, поскольку ее инструкции достаточно запутаны. Однако ниже я все-таки попытаюсь рассказать вам, как работает программа.

1. Первая строка приказывает компьютеру перейти к части программы, которая начинается с метки LabelOne (она находится в седьмой строке программы).
2. Компьютер переходит к седьмой строке программы.
3. Восьмая строка приказывает компьютеру перейти к части программы, которая начинается с метки LabelFour (она находится во второй строке программы).
4. Компьютер переходит ко второй строке программы.
5. Третья строка отображает на экране диалоговое окно с вопросом: How much money do you have? (Какой денежной сумкой вы располагаете?), после чего ждет, пока пользователь введет какое-нибудь число, которое программа присваивает переменной MyCash.
6. Четвертая строка приказывает компьютеру перейти к части программы, которая начинается с метки LabelThree (она находится в девятой строке программы).
7. Компьютер переходит к девятой строке программы.
8. Десятая строка отображает сообщение: You owe me = (вы должны мне следующую сумму), за которым следует значение переменной MyCash, умноженное на 0,95.

9. Одиннадцатая строка приказывает компьютеру перейти к части программы, которая начинается с метки LabelTwo (она находится в пятой строке программы).
10. Компьютер переходит к пятой строке программы.
11. Шестая строка сообщает компьютеру о завершении программы.

Предыдущая программа, в которой используется команда GOTO, полностью эквивалентна следующей простой программе:

```
PROMPT "How much do you have?"; MyCash
PRINT "You owe me = "; MyCash * .95
END
```

На заре программирования таких операторов, как IF THEN или SELECT CASE (подробно об этих инструкциях мы говорили в главе 9, "Принятие решений с помощью управляющих операторов"), просто не существовало. На протяжении многих лет команда GOTO предлагала программистам единственный способ сообщить компьютеру о необходимости прекратить выполнение определенных инструкций и перейти к выполнению других инструкций. Никто не запрещает использовать команду GOTO в ваших программах, но постарайтесь делать это как можно реже.

Знакомство со структурным программированием

Вы никогда не найдете "единственно верный" вариант написания программы, однако программистами все-таки удалось создать несколько способов, как можно подойти к написанию программ более или менее организованно. Один из таких способов написания программ называется *структурным программированием;* его основная идея состоит в том, чтобы организовать программу таким образом, чтобы в ней использовалось всего три вида инструкций (ни одна из них не должна быть аналогом команды GOTO). Используя только следующие виды инструкций, вы поможете себе и другим разобраться в написанных вами программах намного проще и быстрее.

- ✓ Последовательные инструкции
- ✓ Инструкции ветвления
- ✓ Циклические инструкции

Ниже мы подробно рассмотрим каждую из них.

Последовательные инструкции

Простейший способ как-то организовать используемые в программе инструкции, состоит в их *последовательном* размещении, одна за другой:

```
PROMPT "How much stuff did you steal last year"; Amount
TaxesOwed = Amount * .95
PRINT "This is how much tax you owe ="; TaxesOwed
END
```

К сожалению, вы не сможете написать любую программу как один большой список инструкций. Если компьютеру необходимо принять какое-то решение, ваша

программа должна выбрать один из нескольких наборов инструкций. Программы, в которых присутствует подобная возможность выбора, называются *программами с ветвлением.* В других ситуациях вам понадобится, чтобы компьютер последовательно выполнял один и тот же набор инструкций, поэтому использование цикла (например, цикла FOR NEXT) оказывается намного проще, чем многократное указание одного и того же набора инструкций

Инструкции ветвления

Инструкции ветвления (например, оператор IF THEN) позволяют компьютеру выполнять один из нескольких наборов различных инструкций в зависимости от определенного условия. (Более подробно об операторе IF THEN и циклах вообще мы говорили в главе 9, "Принятие решений с помощью управляющих операторов".) Например, следующая программа вычисляет два различных налога, в зависимости от того, являетесь вы политиком или нет:

```
PROMPT "How much stuff did you steal last year"; Amount .
TaxesOwed = Amount * .95
PROMPT "Are you a professional criminal (Y or N)"; Answer
IF (Answer = "N") THEN
  PRINT "This is how much tax you owe ="; TaxOwed
ELSE
  PRINT "Lawyers and politicians don't need to pay taxes."
END
```

Инструкции ветвления предлагают два или больше вариантов набора инструкций, один из которых компьютер должен выполнить в той или иной ситуации. В результате инструкции ветвления оказываются сложнее для понимания, так как необходимо разобраться в том, какой набор инструкций компьютер выполняет в тот или иной момент.

Циклические инструкции

Иногда компьютер должен последовательно выполнять одни и те же инструкции. Вместо того чтобы многократно вводить эти инструкции, используйте цикл, например FOR NEXT ИЛИ WHILE WELD.

Цикл FOR NEXT выполняется определенное количество раз. Цикл WHILE WELD выполняется до тех пор, пока выполняется или прекратит выполняться определенное условие. Таким образом, количество раз, которое выполняется цикл WHILE WELD, изменяется от нуля до бесконечности (о циклах подробно мы говорили в главе 10, "Использование циклов").

Например, приведенная ниже программа запрашивает пароль, проверяет его правильность (правильный пароль — строка "open"), и повторяет эти действия до тех пор, пока пользователь не введет правильный пароль:

```
PROMPT "What is the password"; Password$
WHILE Password$ <> "open"
  PRINT "Wrong password, moron. Try again."
  PROMPT "What is the password"; Password$
WEND
PRINT "You typed the correct password!"
END
```

В циклах сложнее разбираться, чем, например, в последовательных инструкциях или инструкциях ветвления, так как не всегда удается легко определить, сколько будет выполняться тот или иной цикл. Основное назначение циклов — избавить программиста от необходимости вводить одни и те же инструкции (подробности в главе 10, "Использование циклов").

Практикуемся в структурном программировании

Основная причина организации программ в виде участков последовательных инструкций, инструкций ветвления или циклических инструкций состоит в том, чтобы сделать программу как можно понятнее для других. Если другие люди сумеют разобраться в том, как же работает написанная вами программа, они смогут внести в нее необходимые изменения, исправления или улучшения.

То, что вы автор программы, еще не означает, что вы легко разоберетесь в ней в дальнейшем. Если вы написали программу, состоящую из нескольких тысяч строк, вы наверняка забудете, как работает та или иная часть программы — особенно в том случае, если вы отложили программу в сторону на длительный срок и работали над другим проектом. Поэтому написание простых для понимания программ оказывается чрезвычайно важным преимуществом как для вас самих, так и для других программистов, которым придется вносить в написанные вами программы какие-то коррективы.

Для того чтобы понять, насколько структурное программирование упрощает чтение программ, посмотрите на следующую программу, которая состоит из последовательных инструкций, инструкций ветвления или циклических инструкций:

```
  `Последовательные инструкции
•• PRINT "This program prints a message, of your"
 PRINT "choosing, on the screen."
 PROMPT "What message do you want to appear"); Message$
 PROMPT "Display message in all UPPERCASE (type 0) or lowercase (type
        1)?"; WhatCase$
к -
 `Инструкции ветвления
•- IF WhatCase$ = "U" THEN
   Messages = UPPER$(Message$)
 END IF
 IF WhatCase$ - "1" THEN
   Message$ = LOWER$(Message$)
 END IF
 `Циклические инструкции
 FOR I = 1 TO 15
   PRINT SPACE$(I + 4); Message$
 NEXT
 END
```

Представьте себе, что последовательные инструкции, инструкции ветвления и циклические инструкции представляют собой строительные блоки, из которых состоит любая программа. Если вы написали программу с использованием только последовательных инструкций, инструкций ветвления или циклических инструкций, в ней будет намного проще разобраться (как вам, так и любому другому программисту) и внести необходимые изменения.

Написание модульных программ

Если вы планируете писать сложные программы, которые будут контролировать траекторию полета спутника или проводить поиск месторождений нефти на океанском дне, вероятность того, что такие программы будут громадными, достаточно высока. (Многие коммерческие программы содержат по несколько тысяч строк. Такие действительно большие программы, как Windows 2000, содержат несколько миллионов строк, что превращает поиск ошибок в них в чрезвычайно сложную проблему.)

Если вы достаточно амбициозны, вы можете написать программу, представляющую собой один большой список инструкций. Однако, чем больше будет ваша программа, тем сложнее в ней разобраться, а также ее написать или внести необходимые изменения. Написание большой программы в виде одного набора инструкций сродни попыткам построить дом из одного песка. Очень вероятно, что одна из частей такой структуры (например, стена) окажется слабее, чем остальные, а это приведет к катастрофе.

Вместо того чтобы создавать одну большую программу, программисты создают несколько программ меньшего размера и объединяют их вместе (это напоминает использование кирпичей при постройке дома). Таким образом, если одна из частей программы не работает, вы можете переписать ее или заменить, оставив остальную часть программы без изменений.

В компьютерном мире небольшие программы, образующие одну большую программу, называются *подпрограммами*. Подпрограммы иногда называют *модулями,* что привело к появлению термина *модульное программирование*.

Каждая модульная программа содержит как минимум одну подпрограмму, которая называется *основной программой*. Основная программа обычно не выполняет никакой другой задачи, а только сообщает компьютеру, в какой последовательности должны использоваться различные подпрограммы для выполнения той или иной задачи.

Подпрограмма обычно решает какую-нибудь одну задачу, например перемножение двух чисел или проверка, указал ли пользователь правильный пароль. В случае действительно сложных программы вы работаете с несколькими подпрограммами, которые, в свою очередь, разделяются на несколько более мелких подпрограмм.

Например, предположим, вам необходимо написать программу, которая должна проникнуть в другой компьютер. Основная задача, поставленная перед программой, формулируется достаточно просто:

```
Проникновение а другой компьютер.
```

Конечно же, вы не можете компьютеру непосредственно приказать проникнуть в другой компьютер, так как он просто не знает, как это сделать, Вы должны компьютеру подробно объяснить, какие именно действия он должен предпринять. Это означает разделение обшей, основной задачи на несколько более мелких подзадач, как показано ниже.

1. Найти телефонный номер доступа к другому компьютеру.
2. Узнать пароль доступа к другому компьютеру.
3. Получив доступ, сообщить об этом пользователю.

В идеале для решения каждой из этих задач необходимо использовать отдельную подпрограмму, совершенно не зависящую от других. После того как вы заставили корректно функционировать каждую из этих подпрограмм, можно объединить их в одну большую работающую программу.

Программисты используют подпрограммы по следующим двум причинам.

- **Для упрощения написания больших программ.** Написав несколько небольших программ для решения определенных задач, вы сможете объединить их в качестве подпрограмм в одну большую программу.
- **Для сохранения повторяющихся инструкций в определенном месте.** Иногда вам нужно запускать один и тот же набор инструкций по несколько раз подряд. Вместо того чтобы писать инструкции каждый раз, когда в них возникает необходимость, вы можете написать их всего один раз и сохранить в виде отдельной подпрограммы, которая будет вызываться в необходимое время в рамках основной программы.

Основное преимущество разделения большой программы на несколько подпрограмм состоит в том, что вы можете легко вносить изменения в целую программу, изменив всего одну из небольших подпрограмм. Небольшие подпрограммы намного проще для понимания и внесения изменений, поэтому они значительно упрощают понимание и всей большой программы в целом.

Для того чтобы создать подпрограмму на Liberty BASIC, вам необходимо выполнить следующие действия.

- Написать все необходимые инструкции подпрограмм после команды END основной программы.
- Идентифицировать начало подпрограммы с помощью уникального имени, заключенного в квадратные скобки [например, `так`].
- Идентифицировать конец подпрограммы с помощью команды RETURN, которая прикажет компьютеру перейти обратно к той части программы, которая выполнялась до запуска подпрограммы.

Вы можете использовать практически любые имена подпрограмм, однако старайтесь использовать описательные имена. Если подпрограмма предназначена для отображения какого-то предостережения на экране, вы можете присвоить ей имя [warning].

Команда RETURN идентифицирует окончание подпрограммы и приказывает компьютеру вернуться к основной программе.

Поэтому при использовании Liberty BASIC типичная подпрограмма выглядит следующим образом:

```
Инструкции основной программы
'--Инструкции основной программы
END

[Подпрограмма]
Инструкции подпрограммы
Инструкции подпрограммы
RETURN
```

В данном случае компьютер полностью игнорирует подпрограмму, так как начинает выполнение основной программы сверху вниз, с самой первой инструкции и выполняет все последующие инструкции до тех пор, пока не дойдет до команды END, которая прикажет компьютеру прекратить выполнение программы раньше, чем он достигнет любой из инструкций подпрограммы.

Поэтому, если вы хотите, чтобы компьютер выполнил инструкции подпрограммы, вам необходимо использовать команду GOSUB:

```
GOSUB [Имя подпрограммы]
```

Эта команда не делает ничего иного, а только приказывает компьютеру перейти к выполнению подпрограммы, имя которой указано в квадратных скобках. Если вы хотите, чтобы инструкции подпрограммы были выполнены, вам придется добавить в текст основной программы команду GOSUB, как показано ниже:

```
Инструкции основной программы 1
GOSUB [Подпрограмма]
Инструкции основной программы 2
END

[Подпрограмма]
Инструкции подпрограммы
Инструкции подпрограммы
RETURN
```

В этом примере сначала выполняются инструкции основной программы первой группы, а после применения команды GOSUB начинается выполнение инструкций подпрограммы. Затем команда RETURN возвращает компьютер к строке программы, которая следует сразу после команды GOSUB. После того компьютер выполняет инструкции основной программы второй группы, после чего выполнение программы завершается по достижении команды END.

Для того чтобы увидеть в действии настоящую программу, введите и выполните программу, текст которой приведен ниже:

```
NOMAINWIN
PROMPT "What is your password?"; password$
IF password$ = "open" THEN
  NOTICE "You typed a valid password."
ELSE
  NOTICE "Invalid password."
  GOSUB [hackeralert]
END IF
END

[hackeralert]
PROMPT "Are you a hacker? (Y or N)?"; answer$
IF answer$ = "Y" THEN
  NOTICE "The police are on their way to pick you up now."
ELSE
  NOTICE "Then you roust be incompetent."
END IF
RETURN
```

В этом примере инструкции основной программы начинаются с команды NOMAINWIN и заканчиваются командой END. Подпрограмма [hackeralert] начинается в 11-й строке со своего названия и заканчивается на 18-й строке командой RETURN. Подпрограмма [hackeralert] выполняется только в том случае, если вы указали неправильный пароль в диалоговом окне с вопросом What is your password? (Введите пароль).

В идеале вы должны стараться делать подпрограммы как можно меньшего размера, чтобы они умещались на экране монитора. Чем проще будет подпрограмма, тем легче ее прочитать, отладить или внести в нее необходимые изменения.

Если вы используете подпрограммы в программе, написанной на Liberty BASIC, на самом деле вы разделяете большую программу на меньшие части, хотя вся программа сохраняется в виде одного файла, например PACMAN.BAS или BLACKJAK.BAS.

Имейте в виду, что в различных языках программирования одни и те же термины означают совершенно разное. Например, если вы разделите программу на несколько частей, то в Liberty BASIC такие части называются *подпрограммами*. Однако в языке программирования C эти же подпрограммы уже называются *функциями*. Поэтому, если вы используете различные языки программирования, убедитесь в том, что вы применяете соответствующие термины в каждом из них; в противном случае у вас возникнут недоразумения при общении с другими программистами.

Использование процедур

При использовании Liberty BASIC вы можете создавать подпрограммы, т.е. небольшие программы, выполняющие одну определенную задачу. К сожалению, подпрограммы выполняют одни и те же действия, сколько бы вы их ни запускали.

Для большей гибкости создайте подпрограммы, которые получали бы от пользователя какие-нибудь данные и вычисляли новый результат, базируясь именно на этих данных. Вы можете, например, написать подпрограмму, которая будет перемножать два одинаковых числа. В большинстве случаев вам не нужно умножать само на себя одно и то же число при каждом запуске подпрограммы. Ведь подпрограмма окажется намного полезнее, если вы будете вводить другое число при каждом ее запуске, и заставите ее изменять результат работы в зависимости от введенных данных.

В большинстве языков программирования, в частности C/C++, Java или Pascal, можно создавать подпрограммы двух типов.

- ✓ Процедуры. Подпрограммы этого типа содержат инструкции, которые принимают данные, введенные пользователем, и используют их при выполнении определенных задач, например, при проверке правильности введенного пароля или перемещении указателя мыши по экрану.
- ✓ Функции. Подпрограммы этого типа вычисляют и возвращают всего одно значение, например вычисляют куб введенного числа или подсчитывают количество слов во введенном предложении. Пример — команды cos (X) и SIN (X), используемые в Liberty BASIC.

Определение процедуры

Процедура состоит из следующих двух или трех частей.

- ✓ **Имя процедуры**
- ✓ Одна или несколько инструкций, выполняемых процедурой
- ✓ Любые данные, которые должна использовать процедура (необязательно)

В Liberty BASIC процедура выглядит следующим образом:

```
SUB Имя_процедуры Данные
    ' Одна или несколько инструкций
END SUB
```

Если процедура не получает никаких данных от основной программы или подпрограммы, вы можете не указывать никаких данных, как показано ниже:

```
SUB Имя_процедуры
    ' Одна ими несколько инструкций
END SUB
```

Передача данных процедуре

Процедура функционирует подобно небольшой программе, выполняющей одно или несколько заданий. Хотя они и являются частью большей программы, некоторые процедуры функционируют совершенно независимо. Вы можете, например, написать процедуру, которая не будет выполнять ничего, а только отображать сообщение на экране. Если процедура функционирует независимо от основной программы, вам достаточно указать только ее имя и инструкции, которые она должна выполнять:

```
SUB Имя процедуры
  PRINT "Эта процедура не выполняет никаких действий."
END SUB
```

Этот пример процедуры очень похож на пример подпрограммы, написанной на Liberty BASIC. После запуска процедура только выполнит предписанные ей инструкции, не используя при этом никаких данных, получаемых от остальных частей программы.

Однако во многих случаях процедуре для выполнения определенных действий, например проверки правильности введенного пользователем пароля, какие-то данные все-таки необходимы. Если для работы процедуры ей необходимо предоставить какие-то данные, вы должны создать специальную переменную для их хранения, как показано в следующем примере:

```
SUB Имя процедуры Переменная
  ' - Одна или несколько операций
END SUB
```

Когда другая часть программы вызывает эту процедуру, она передает ей определенные данные. После получения этих данных процедура сохраняет их в качестве значения переменной.

Список переменных, предназначенных для сохранения переданных процедуре данных, называется *списком параметров*.

Каждая переменная из списка параметров представляет определенную часть данных, например номер строки. Если другая часть программы передает процедуре два фрагмента данных, в списке параметров процедуры должно присутствовать ровно две переменные. Например, ниже приведена процедура, использующая три переменные:

```
SUB Имя процедуры Переменная1, Переменная2, Переменная3
   ' Одна или несколько инструкций
END SUB
```

Обязательно убедитесь в том, что объявили все необходимые строковые переменные с использованием знака доллара, а числовые переменные — без него. Соответствующий пример приведен ниже.

```
SUB Имя процедуры Name$, Age
   ' Одна или несколько инструкций
END SUB
```

Первая строка определяет имя процедуры, а также создает две переменные, первая из которых предназначена для хранения строковых данных, а вторая — числовых.

Вызов процедуры

После того как вы создали процедуру, она будет выполнять содержащиеся в ней инструкции. Обычно процедура не выполняет никаких действий (как настоящий политик) до тех пор, пока не получит соответствующие указания от компьютера. Говоря техническим языком, когда вы приказываете процедуре выполнить инструкции, вы осуществляете *вызов процедуры*. Вы как бы говорите: "Эй, глупая процедура! Немедленно выполни все необходимые инструкции!"

Если вы хотите запустить процедуру BurnOutMonitor, вам следует использовать команду CALL. Большинство версий BASIC позволяет вам делать это двумя способами:

```
CALL BurnOutMonitor
```

Если для работы процедуры BurnOutMonitor ей необходимо передать определенные данные, вам следует использовать следующий вариант:

```
CALL BurnOutMonitor 45
```

Если вы используете команду CALL, убедитесь в том, что правильно указали имя процедуры, в том числе и правильно использовали прописные и строчные буквы. Например, с точки зрения языка программирования Liberty BASIC, DisplayMessage и displaymessage — это имена двух разных процедур.

Для того чтобы помочь вам разобраться в работе процедур, я предлагаю выполнить следующую программу. Обратите внимание, что процедура начинается после команды END. Данная программа спрашивает у пользователя его имя, после чего отображает один из двух возможных вариантов сообщения: Имя must be a moron (должен быть настоящим глупцом). ИЛИ Имя sounds like an idiot to me (звучит, по-моему, совершенно глупо)..

```
NOMAINWIN
PROMPT "Give me the name of someone you hate:"; enemy$
CALL DisplayMessage enemy$
END

SUB DisplayMessage stuff$
  X = INT(RND(1) *2) + 1
  IF X = 1 THEN
     NOTICE stuff$ + " must be a moron."
  ELSE
     NOTICE stuff$ + " sounds like an idiot to me."
  END IF
END SUB
```

Liberty BASIC интерпретирует эту программу следующим образом.

1. Первая строка предотвращает отображение на экране основного окна.
2. Вторая строка отображает на экране диалоговое окно с сообщением Give me the name of someone you hate: (Укажите имя ненавистного человека:). Введенные **пользователем данные присваиваются переменной** enemy$.
3. **Третья строка вызывает процедуру** DisplayMessage **и передает ей данные, сохраненные в виде переменной** enemy$. **После запуска** процедуры **выполняются строки** с SUB DisplayMessage stuff$ ДО END SUB.

4. Четвертая строка завершает программу.
5. Пятая строка представляет собой первую строку процедуры DisplayMessage, которая принимает строковое значение и присваивает его переменной stuff$.
6. Шестая строка создает произвольное число (или 1, или 2) и присваивает его значение переменной X.
7. Строки с седьмой по одиннадцатую отображают два различных диалоговых окна, используя имя, указанное переменной stuff$.
8. Двенадцатая строка указывает на завершение процедуры. Когда компьютер достигает этой строки, он возвращается к основной программе в строке, следующей после команды CALL DisplayMessage. В данном случае это команда END, завершающая выполнение программы.

использование функций

Функция — это специальная процедура, которая выполняет только вычисление значения. Например, если вы хотите вычислить куб числа, используйте следующие инструкции:

```
Куб = Число * Число * Число
```

Но если вам необходимо вычислять куб числа неоднократно, вам придется каждый раз вводить формулу Число * Число * число. Как вы уже знаете, подобный подход приводит к возникновению ошибок, особенно при внесении изменений в будущем.

Определение функции

Вместо того чтобы вводить одни и те же инструкции несколько раз (при этом используя разные числа), вы можете сохранить их в виде функции. Затем вы просто передает данные функции, которая выполняет необходимые действия и возвращает результат. Функция состоит из следующих четырех частей.

- ✓ Имя
- ✓ Одна или несколько инструкций, позволяющих получить значение
- ✓ Одна строка, назначающая значение (или выражение, представляющее значение)
- V Любые данные, необходимые для работы функции

При использовании Liberty BASIC типичная функция выглядит следующим образом:

```
FUNCTION Имя_функции (Данные)
    Одна или несколько инструкций
  Имя_функции = значение
• END FUNCTION
```

Не ставьте пробел между именем функции и левой скобкой, так как Liberty BASIC не сможет выполнить программу.

Передача данных функции

Практически все функции должны получать данные от других частей программы. Когда какая-то часть программы запускает (вызывает) функцию, она должна передать

ей определенные данные. Для того чтобы убедиться в том, что функция получила все данные, переданные ей другой частью программы, вы должны указать в скобках одну или несколько переменных, как показано ниже:

```
FUNCTION Имя_функции (Переменная)
    Одна или несколько инструкций
  Имя_функции = значение
END FUNCTION
```

В приведенном примере функция получит определенные данные (число или строку), переданные другой частью программы.

Если вам необходимо, чтобы функция получала несколько данных от других частей программы, разделите имена переменных запятыми, как показано ниже:

```
FUNCTION Имя_функции(Переменная1, Переменная2, Переменная3)
    Одна или несколько инструкций
  Имя_функции = значение
END FUNCTION
```

Не забывайте определить тип данных, передаваемых функции, — например, число или строка. (О строковом типе данных мы подробно говорили в главе 8, "Забавы с числами и строками".) Полное объявление функции выглядит следующим образом:

```
FUNCTION Имя_функции(Note$, Salary)
    Одна или несколько инструкций
  Имя_функции = значение
END FUNCTION
```

Первая строка функции определяет ее имя и создает две переменные — Note$ и Salary. Переменная Note представляет строку, а переменная Salary — число с одинарной точностью, например 3,14.

Вызов функции

Конечно же, вы захотите использовать функцию, созданную в рамках программы. Функция представляет одно значение, например целое число или число двойной точности, поэтому ее можно использовать точно так же, как и переменную.

Предположим, вы создали функцию и назвали ее Cube, как показано ниже:

```
FUNCTION Cube(Число)
  Cube = Число * Число * Число
END Function
```

Вы можете использовать эту функцию как любую переменную, как показано в следующем примере:

```
PRINT Cube(3)
```

Или так:

```
MyValue = Cube(3)
```

Ниже подробно рассмотрен процесс интерпретации компьютером команды PRINT Cube(3).

1. Строка PRINT Cube(3) приказывает компьютеру запустить функцию Cube и передать ей в качестве данных число 3. После этого результат, возвращенный функцией, выводится на экран.

2. **Сразу после этого компьютер приступает к** поиску **функции** Cube. **Первая строка** функции **сообщит компьютеру об ее имени, а также о том, что** функции **в качестве параметра нужно** передать **целое число, представленное переменной** Number. **В нашем** примере переменная Number представляет **число 3.**

3. **Вторая строка функции** Cube **прикажет компьютеру умножить значение переменной** Number **само на себя три раза,** после чего **присвоить полученный результат функции** Cube. **В нашем примере переменная** Number **представляет** число 3, поэтому **значение функции** Cube **равно** 3*3*3, или **же** 27.

4. **Третья строка** функции Cube **сообщает компьютеру о** завершении **выполнения** функции, **что приводит к возврату к основной программе. В** данном **случае выполняется команда** PRINT Cube(3).

5. **Поскольку** значение Cube(3) **равно** 27, **компьютер воспринимает команду** PRINT Cube(3) как PRINT *21.* **В конечном** итоге **на экране отображается число** 27.

Глава 12

Создание картинок и звуков

В этой главе...

- Создание графического объекта
- Забавы с "черепашьей" графикой
- Рисование кругов и прямоугольников
- Добавление звука

Раньше результат работы компьютерной программы можно было увидеть только напечатанным на листе бумаги. Со временем основная часть программ стала отображать полученные результаты на экране монитора. И только после того, как мониторы стали неотъемлемой частью компьютера, программисты смогли реализовать все свои идеи, создав программы, в которых используются красивые рисунки и звук.

В настоящей главе я рассматриваю несколько способов создания красочных программ на языке программирования Liberty BASIC. Как только в вашей программе появятся цветные рисунки и звуки, пользователи тут же скажут, что программа стала красивее и ее легче использовать.

Создание графического объекта

Перед тем как создать рисунки и отобразить их на экране, необходимо создать *графический объект,* который состоит из окна или части окна, в котором на экране отображается рисунок. Создать графический объект можно двумя способами.

- Создать отдельное графическое окно
- ✓ Создать графическое поле в существующем окне

Для создания отдельного графического окна используется команда OPEN, пример которой приведен ниже:

```
OPEN "Область рисования" FOR Graphics AS #graphWin
```

Основное отличие между графическим окном и обычным окном состоит в том, что в графическом окне вместо текста отображается рисунок.

Если вы не хотите создавать отдельное графическое окно, можно создать графическое поле внутри обычного окна. Для создания графического поля в окне необходимо воспользоваться следующими командами:

```
GRAPHICBOX #main.graph, 10, 10, 150, 150
OPEN "Обычное окно" FOR Window AS #main
```

Команда GRAPHICBOX работает следующим образом:

```
GRAPHICBOX #Windowhandle.boxname, xpos, ypos, width, height
```

Ниже рассмотрены элементы этой программы.

1. Команда GRAPHICBOX приказывает компьютеру создать поле, в котором будет отображаться рисунок. Данное поле появится в окне, которому присвоено имя #Windowhandle.
2. Элемент boxname присваивает графическому полю имя.
3. Переменные xpos и ypos определяют расположение графического поля (по оси *X* и оси *Y* соответственно). Отсчет начинается с верхнего левого угла окна, которому присвоено имя #Windowhandle.
4. Переменные width и height определяют ширину и высоту графического поля.

Создав графическое окно или графическое поле, можно приступать к работе.

Использование "черепашьей" графики

"Черепашья" графика получила свое название благодаря идее размещения воображаемой роботизированной черепахи в центре экрана. Для того чтобы что-нибудь нарисовать, программа должна сказать роботизированной черепахе, когда необходимо начать рисовать линию. Для того чтобы завершить рисование, необходимо сказать, чтобы черепаха оторвала карандаш от бумаги.

В языке программирования LOGO используется идея "черепашьей" графики. С ее помощью детей обучает основам программирования.

В языке программирования Liberty BASIC (как и в других языках программирования, использующих "черепашью" графику), есть четыре основные команды для рисования.

1. Оторвать карандаш от бумаги (для прекращения рисования).
2. Опустить карандаш на бумагу (для начала рисования).
3. Перейти вперед на определенное расстояние (если карандаш опушен, рисуется линия).
4. Повернуть карандаш на сколько-то градусов.

Воспользовавшись командами для перемещения, поворота, поднятия и опускания карандаша, можно "заставить" воображаемую черепаху-робота рисовать на экране различные рисунки, состоящие из прямых линий и прямоугольников.

В языке программирования Liberty BASIC есть несколько специальных команд для создания "черепашьей" графики, *а* именно:

- ✓ UP. Оторвать карандаш от бумаги (не рисовать)
- ✓ DOWN. Опустить карандаш на бумагу (рисовать)
- ✓ HOME. Переместить черепаху (карандаш) к центру области рисования
- ✓ GO. Переместить черепаху (карандаш) вперед в текущем направлении
- ✓ GOTO. Переместить черепаху (карандаш) в определенную точку. Если карандаш опущен, рисуется линия
- ✓ PLACE. Переместить черепаху (карандаш) в определенную точку. Даже если карандаш опушен, линия рисоваться не будет
- ✓ TURN. Повернуть черепаху (карандаш) на сколько-то градусов.
- ✓ NORTH. Повернуть черепаху (карандаш) прямо на север
- ✓ POSXY. Выяснить текущие координаты *X* и *Y* черепахи (карандаша)

Для того чтобы увидеть, как на самом деле работает "черепашья" графика, воспользуйтесь программой для рисования флажка (или цифры 4, если вам так больше нравится) в центре экрана, как показано на рис. 12.1.

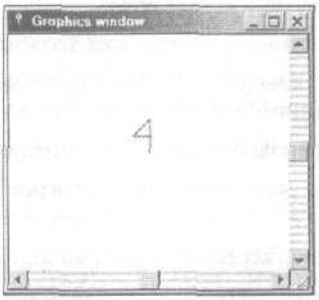

Рис. 12.1. С помощью "черепашьей" графики можно рисовать простейшие фигуры

```
NOMAINWIN
WindowHeight = 300
WindowWidht = 250

OPEN "Graphics window" FOR Graphics AS #main

PRINT #main, "HOME"
PRINT #main, "DOWN"
PRINT #main, "NORTH"
PRINT #main, "GO 35"
PRINT #main, "TURN 225"
PRINT #main, "GO 25"
PRINT #main, "TURN 225"
PRINT #main, "GO 20"
PRINT #main, "FLUSH"

PRINT #main, "trapclose [quit]"
WAIT

[quit]
CON-FIRM "Are you sure want to quit?"; quit$
IF quitS - "no" THEN WAIT
CLOSE #main
END
```

Ниже описаны элементы рассмотренной программы.

1. **Первая строка "приказывает" компьютеру не отображать основное окно.**
2. **Вторая и третья линии задают высоту (300) и ширину (250)** окна, **создаваемого с** помощью **четвертой линии.**
3. **С** помощью **четвертой линии создается графическое окно, в котором будет текст с заголовком Graphics window. Данное окно** идентифицирует **дескриптор ttmain.**

4. Пятая строка программы перемещает черепаху в основное положение (в центр окна).
5. В шестой строке черепахе приказывают опустить карандаш и приготовиться к рисованию.
6. В седьмой строке сказано, что черепаха должна направляться на север.
7. В восьмой строке говорится, что черепаха должна переместиться на 35 пикселей в выбранном направлении, т.е. на север.
8. В девятой строке черепахе приказывают повернуть на 225° направо.
9. В десятой строке черепахе приказывают переместиться вперед на 25 пикселей.
10. В одиннадцатой строке черепахе приказывают повернуть на 225° направо.
П. В двенадцатой строке черепахе приказывают переместиться вперед на 20 пикселей.
12. В тринадцатой строке черепахе отдают команду FLUSH. Это специальная команда, позволяющая созданный черепахой рисунок оставлять в окне даже в том случае, если вы измените размер окна или переместите его.
13. В четырнадцатой строке используется команда trapclose, которая говорит компьютеру, что после того, как пользователь закроет программу, компьютер должен перейти к инструкции для выключения, которая подписана как [quit].
14. В пятнадцатой строке компьютеру отдан приказ ждать до тех пор, пока пользователь что-нибудь сделает.
15. Строки с шестнадцатой (в ней только команда [quit]) по девятнадцатую говорят компьютеру, что он должен делать после того, как пользователь закроет программу. В данном случае появится диалоговое окно Confirm, значит пользователь вышел из программы. Если это действительно так, программа закроет окно с дескриптором #main.
16. Двадцатая строка означает окончание программы.

Если вы не будете использовать команду FLUSH, "черепашья" графика может полностью исчезнуть, как только пользователь изменит размер окна.

Вместо того чтобы вводить отдельные команды в каждой линии, можно написать несколько команд для построения "черепашьей" графики, после чего разместить их в одной строке, разделив точкой с запятой. При этом вы сэкономите достаточно много времени и места, как это видно из следующего примера:

```
PRINT flmain, "HOME"
PRINT ttmain, "DOWN"
PRINT #main,  "NORTH"
PRINT #main,  "GO 35"
PRINT tfmain, "TURN 225"
```

Для того чтобы сберечь место, объедините все команды в одну строку:

```
PRINT #main, "HOME; DOWN; NORTH; GO 35; TURN 225"
```

Для того чтобы разделить одну длинную строку на несколько коротких, воспользуйтесь знаком подчеркивания (_), который как бы говорит программе Liberty BASIC: "Как только ты увидишь знак подчеркивания, перейди на следующую строку. На ней находится продолжение предыдущей строки".

Поэтому следующую строку:

```
PRINT #main, "HOME; DOWN; NORTH; go 35; TURN 225"
```

можно сделать короче, записав ее следующим образом:

```
PRINT ttmain, "HOME; DOWN;
        NORTH; GO 35; TURN 225"
```

Для того чтобы ускорить рисование линии, воспользуйтесь командой GOTO сразу же после команды DOWN:

```
PRINT #MAIN, "DOWN; GOTO x y"
```

Данная строка говорит компьютеру, что необходимо опустить карандаш на бумагу и нарисовать линию из текущего положения черепахи (карандаша) до точки с координатами *X* и *Y*.

Задание толщины линии

Чтобы разнообразить рисунок, можно изменять толщину линий, для этого воспользуйтесь командой SIZE:

```
PRINT tfWindowhandle, "size X"
```

Ниже описаны элементы рассмотренной программы.

1. Элемент #Windowhandle задает графическое окно для настройки толщины следующей нарисованной черепахой линии.
2. Команда size X определяет толщину линии. Здесь X — число, определяющее толщину линии (например, 3 или 8). Если данная команда не используется, толщина линии будет равна единице (1).

Для того чтобы увидеть, каким образом изменяется толщина линии, попытайтесь запустить следующую программу. В ней рисуются две линии разной толщины: пять (5) и десять (10):

```
NOMAINWIN
WindowHeight = 300
WindowWidht = 250

OPEN "Graphics window" FOR Graphics AS #main

PRINT flmain, "HOME; DOWN; NORTH"
PRINT #main, "SIZE 5"
PRINT #main, "GO 35; TURN 90; GO 35; TURN 90"
PRINT #main, "SIZE 10"
PRINT #main, "DOWN; GO 35"
PRINT #main, "FLUSH"

PRINT #main, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure want to quit?"; quitS
IF quit$ = "no" THEN WAIT
CLOSE #main
END
```

Выбор цвета линии

Поскольку черно-белый рисунок не всегда радует глаз, возможно, что вы захотите раскрасить свое произведение. Для того чтобы изменить цвет линии, необходимо воспользоваться командой COLOR:

```
PRINT #Windowhandle, "COLOR color"
```

Ниже описаны элементы рассмотренной программы.

1. Элемент #Windowhandle определяет графическое окно для настройки цвета следующей нарисованной черепахой линии.
2. **Команда** COLOR color **задает** цвет линии. **Здесь** color **— это один из следующих** цветов: black (черный), blue (синий), brown (коричневый), cyan (голубой), darkblue (темно-синий), darkcyan (темно-голубой), darkgray (темно-серый), darkgreen (темно-зеленый), darkpink (темно-розовый), darkred (темно-красный), green (зеленый), lightgray (светло-серый), palegray (бледно-серый), pink (розовый), red (красный), white (белый) и yellow (желтый).

Для того чтобы увидеть, каким образом изменяется цвет линии, попытайтесь запустить следующую программу. В ней рисуются две линии разного цвета:

```
NOMAINWIN
WindowHeight = 300
WindowWidht = 250

OPEN "Graphics window" FOR Graphics AS #main

PRINTftrnain,"HOME; DOWN; NORTH"
PRINT flmain, "COLOR darkgreen"
PRINT flmain, "GO 35; TURN 90; GO 35; TURN 90"
PRINT ftmain,'"COLOR DARKPINK"
PRINT ftmain, "DOWN; GO 35"
PRINT ttmain, "FLUSH"

PRINT #main, "trapclose [quit]"
WAIT

 [quit]
CONFIRM "Are you sure want to quit?"; quit$
IF quit$ = "no" THEN WAIT
CLOSEflrnain
END
```

Рисование окружностей

Поскольку рисунки, состоящие из одних прямых линий, несколько утомительны, попытайтесь разнообразить свои произведения с помощью окружностей. Для этого в программе Liberty BASIC используется команда CIRCLE:

```
PRINT #Windowhandle, "CIRCLE R"
```

Ниже описаны элементы рассмотренной программы.

1. Элемент tWindowhandle задает графическое окно, в котором будет окружность.
2. Команда CIRCLE R приказывает компьютеру нарисовать окружность, центр которой находится в текущем положении карандаша. Радиус окружности равен R, где R — число (например, 35 или 90).

Если вы хотите нарисовать окружность, границы которой окрашены в определенный цвет, необходимо воспользоваться командой COLOR, поставив ее перед командой CIRCLE:

```
• PRINT #Windowhandle, "COLOR darkpink; CIRCLE R"
```

Также можно закрасить окружность, воспользовавшись командами BACKCOLOR и CIRCLEFILLED:

```
PRINT #Windowhandle, "BACKCOLOR yellow; CIRCLEFILLED
R"
```

Для того чтобы увидеть, как нарисовать круг, попытайтесь запустить следующую программу. Она рисует две окружности (рис. 12.2).

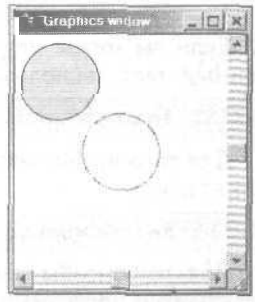

Рис. 12.2. Две окружности, нарисованные с помощью "черепашьей" графики

```
NOMAINWIN
WindowHeight = 300
WindowWidht = 250
OPEN "Graphics window" FOR Graphics AS #main

PRINT #main, "HOME; DOWN"
PRINT tmain, "COLOR red; CIRCLE 40"
PRINT #main, "PLACE 45 50"
PRINT #main, "COLOR darkblue; BACKCOLOR yellow; CIRCLEFILLED 40"
PRINT #main, "FLUSH"

PRINT tmain, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure want to quit?"; quit$
IF quit$ = "no" THEN WAIT
CLOSE tmain
END
PRINT #Windowhandle, "BOX x y"
```

В седьмой строке рассмотренной выше программы используется команда PLACE. Она перемещает черепаху (карандаш) в определенную точку без рисованиялинии.

Рисование прямоугольников

Вы уже умеете рисовать круги. Liberty BASIC также позволяет рисовать прямоугольники. Для того чтобы нарисовать прямоугольник, воспользуйтесь командой BOX:

```
PRINT #Windowhandle, "BOX x y"
```

Ниже описаны элементы рассмотренной программы.

1. Элемент #Windowhandle задает графическое окно, в котором будет нарисованный черепахой прямоугольник.
2. Команда BOX x y приказывает компьютеру нарисовать прямоугольник, один угол которого находится в месте, в котором сейчас черепаха (карандаш), а противоположный — в месте с координатами x и y.

Если вы хотите нарисовать прямоугольник, границы которого окрашены в определенный цвет, воспользуйтесь командой COLOR, поставив ее перед командой BOX:

```
PRINT    flWindowhandle,    "COLOR  red;  BOX  x  y"
```

Для того чтобы закрасить прямоугольник, воспользуйтесь командами BACKCOLOR и BOXFILLED:

```
PRINT #Windowhandle, "BACKCOLOR pink;  BOXFILLED  x  y"
```

Для того чтобы увидеть, как нарисовать прямоугольник, попытайтесь запустить следующую программу. В ней рисуются два прямоугольника:

```
NIMAINWIN
WindowHeight = 300
WindowWidht = 250

OPEN "Graphics window" FOR Graphics AS #main

PRINT #main,  "HOME;  DOW
PRINT frmain, "COLOR red;  BOX  190"
PRINT fmain,  "PLACE  45  50"
PRINT #main,  "COLOR darkblue; BACKCOLOR pink;  BOXFILLED 80  80"
PRINT #main,  "FLUSH"

PRINT ftmain, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure want to quit?"; guit$
IF quit$ = "no" THEN WAIT
CLOSE #main
END
```

Отображение текста

В графическом окне можно размешать не только линии, окружности и прямоугольники, но и текст. Для того чтобы в графическом объекте отобразить текст, необходимо переместить черепаху (карандаш) в место, в котором должен появиться текст. После этого необходимо ввести текст, разместив перед ним обратную косую черту.

```
NIMAINWIN
WindowHeight = 300
WindowWidht = 250

OPEN "Graphics window" FOR Graphics AS #main

PRINT #main,  "HOME"
PRINT #main,  "\This is an"
```

```
PRINTtfmain, "\example of text"
PRINT #main, "FLUSH"

PRINT ftmain, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure want to quit?"; quit$
IF quit$ - "no" THEN WAIT
CLOSE #main
END
```

Обратная косая черта (\) используется для отображения текста и начала новой строки, как показано на рис. 12.3. Чтобы отобразить на экране несколько строк, не надо постоянно перемешать черепаху (карандаш) на новое место.

Для того чтобы внести некоторое разнообразие, можно раскрасить текст или фон. Чтобы изменить цвет текст, воспользуйтесь командой COLOR:

```
PRINT #main, "HOME"
PRINT #main, "COLOR red"
PRINT #main, "\This is an"
PRINT #main, "\example of text"
PRINT #main, "FLUSH"
```

Во второй строке программы компьютер получает приказ раскрасить текст в красный двет.

Если вы хотите изменить фон, на котором находится текст, воспользуйтесь командой BACKCOLOR:

```
PRINT #main, ''HOME"
PRINT teain, ''BACKCOLOR red"
PRINT #main, ''\This is an"
PRINT #main, ''\EXAMPLE OF TEXT"
PRINT #main, ''FLUSH"
```

Рис. 12.3. Созданный текст, отображенный в графическом окне

Во второй строке программы компьютер получает приказ поместить текст на красном фоне.

Добавление звука

В программах часто используется звук. Это делается для того, чтобы привлечь внимание пользователя к некоторым этапам выполнения программы или просто развлечь его. Программы, в которых используется звуковое сопровождение, кажутся намного интереснее.

Создание звукового сигнала

В Liberty BASIC можно создавать простые (и часто надоедливые) звуковые сигналы. Для этого необходимо воспользоваться командой BEEP:

```
PROMPT "How many beeps do you want to hear"; Answer
FOR I = 1 TO Answer
  BEEP
NEXT
END
```

В программах команду BEEP можно использовать в том случае, если пользователь должен в определенный момент нажать какую-либо клавишу или выбрать определенную команду.

Проигрывание файлов в формате .WAV

Команда BEEP используется только для извлечения простых звуков, поэтому Liberty BASIC предлагает команду PLAYWAVE. С ее помощью можно проигрывать файлы в формате .WAV, загруженные из Internet.

Некоторые файлы в формате .WAV можно найти в каталоге C:\Windows\Media на вашем компьютере.

Для того чтобы воспользоваться командой PLAYWAVE, необходимо определить, какой файл в формате .WAV вы хотите прослушать и как именно вы хотите это сделать. Есть три режима прослушивания файлов в формате .WAV.

- ✓ SYNC. Временно останавливается выполнение программы до тех пор, пока не закончится воспроизведение файла в формате .WAV
- ✓ ASYNC. Позволяет продолжить выполнение программы во время воспроизведения файла в формате .WAV
- ✓ LOOP. Файл в формате .WAV будет воспроизводиться до тех пор, пока компьютер не получит команду PLAYWAVS "", что означает прекращение воспроизведения

Команда PLAYWAVE выглядит следующим образом:

```
PLAYWAVE filename, mode
```

Если вы хотите воспроизвести файл tada.wav, который находится в каталоге C:\Windows\Media, необходимо ввести следующую команду:

```
PLAYWAVE "C:Windows\Media\tada.wav", SYNC
```

В данном примере будет воспроизводиться файл tada.wav. При этом программа временно прекратит работу до тех пор, пока не закончится воспроизведение.

Убедитесь в том, что при использовании команды PLAYWAVE вы указали правильное размещение каталога, в котором находится файл в формате .WAV.

Для того чтобы увидеть, как в действительности работает программа Liberty BASIC, попытайтесь запустить следующую программу:

```
NIMAINWIN
FILEDIALOG "Pick a .WAV file to play.", "*.wav", filename$
PLAYWAVE filename$, SYNC
END
```

Ниже описано, что делает каждая строка рассмотренной программы.

1. Первая строка говорит Liberty BASIC не отображать основное окно.
2. Вторая строка отображает диалоговое окно, в котором можно выбрать файл в формате .WAV.
3. Третья строка запускает файл, выбранный в предыдущем диалоговом окне.
4. Четвертая строка завершает выполнение программы.

Глава 13

Сохранение и получение информации из файлов

В этой главе...

- Сохранение текста
- Использование файлов с произвольной выборкой
- Использование каталогов

Каждой программе для работы необходимо получать данные от какого-то внешнего источника (например, человека, вводящего текст с клавиатуры), потом она использует и преобразует их к необходимому формату (например, создает квартальный отчет). Для временного хранения данных программы используют переменные, которые сохраняются в памяти компьютера. По завершении работы программы компьютер удаляет из памяти все данные, освобождая пространство, необходимое для работы других программ.

А что делать, если вы хотите хранить данные постоянно? Например, многие компьютерные игры сохраняют наилучшие результат, чтобы играющим в будущем было к чему стремиться. Если программа сохраняет данные на дискете или на жестком диске компьютера, на самом деле она сохраняет их в отдельном файле.

Сохранение данных в текстовых файлах

Простейшие данные, которые сохраняются программой, представляют собой обычный текст, который состоит из букв, цифр и символов (#, ~, < или &), вводимых с клавиатуры. Любой файл, который содержит только текст, называется *текстовым*. Если вы решили сохранять, записывать или получать данные из текстового файла, вам необходимо помнить о том, что считывание информации из таких файлов всегда начинается с самого начала. По этой причине текстовые файлы часто называют *последовательными*.

Поскольку текстовые файлы содержат только буквы, цифры или символы, вы можете свободно обмениваться ими с пользователями других компьютеров, например компьютеров, работающих под управлением Linux, Macintosh или Windows. Если вы используете команду Сохранить как своего любимого текстового процессора, вы обязательно найдете возможность сохранить документ в виде текстового файла. Однако при этом не стоит забывать, что сохранение документа текстового процессора приводит к потере любого форматирования текста, такого как подчеркивание или использование специальных шрифтов.

Создание нового текстового файла

Прежде чем вы сохраните любые данные в виде файла, вы (очевидно) должны сначала этот файл создать. Для создания текстового файла при использовании Liberty BASIC служит следующая команда:

```
OPEN "Имя_файла" FOR OUTPUT AS #Handle
```

Вот что происходит в ходе выполнения этой строки.

1. Команда OPEN приказывает компьютеру создать новый текстовый файл с указанным именем (например, STUFF.TXT или C:\WINDOWS\STUFF.TXT).
2. Такая часть команды, как FOR OUTPUT, приказывает компьютеру приготовиться к выводу результатов выполнения программы в текстовый файл.
3. #Handle — это "маркер", по которому вы определяете текст в файле.

Например, если вы хотите создать файл STUFF.TXT и сохранить его на дискете, воспользуйтесь такой командой:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #Secrets
```

Это строка приказывает Liberty BASIC создать текстовый файл STUFF.TXT на дискете и назначить ему имя #Secrets.

Каждый раз, когда вы используете команду OPEN для создания нового или открытия существующего текстового файла, обязательно командой CLOSE закройте ее. Без этого ваша программа может вызвать зависание компьютера.

Помещение информации в текстовый файл

После того как вы создали текстовый файл, используйте команду PRINT для занесения в него данных. Работая с командой PRINT, Liberty BASIC обычно отображает данные на экране, поэтому прикажите Liberty BASIC сохранять их в текстовом файле, задав идентификатор последнего, как показано ниже:

```
PRINT #Secrets, "Эта строка сохраняется в текстовом файле."
```

Эта команда прикажет Liberty BASIC найти текстовый файл, на который указывает идентификатор #Secrets, **и помещает в него строку:** Эта строка сохраняется в текстовом файле.

Вся программа выглядит следующим образом:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #Secrets
PRINT #Secrets, "Эта строка сохраняется в текстовом файле."
CLOSE ttSecrets
END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру создать на дискете текстовый файл STUFF.TXT. Затем этому файлу присваивается идентификатор #Secrets.
2. **Вторая строка приказывает компьютеру найти файл с идентификатором** #Secrets **и поместить в него строку:** Эта строка сохраняется в текстовом файле.
3. Третья строка приказывает компьютеру закрыть файл с идентификатором #Secrets.
4. Четвертая строка сообщает компьютеру о завершении программы.

Чтобы оценить, работает ли программа должным образом, сохраняя строку: Эта строка сохраняется в текстовом файле в текстовом файле STUFF.TXT, запустите программу Проводник из состава Windows и щелкните на значке файла, чтобы открыть его.

Добавление новых данных в существующий текстовый файл

Если вы используете команду OPEN, как показано ниже, Liberty BASIC точно знает, что вы намерены сохранять данные в текстовом файле:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #Secrets
```

Эта строка говорит Liberty BASIC следующее: "Создай на дискете новый файл и подготовься к записи данных".

А если файл STUFF.TXT уже существует? Тогда команда OPEN прикажет Liberty BASIC удалить все данные в файле STUFF.txt и подготовить его для сохранения данных.

Если вы хотите сохранить новые данные в текстовом файле, который уже содержит какую-то информацию, вы должны использовать команду APPEND:

```
OPEN "A:\STUFF.TXT" FOR APPEND AS #Secrets
```

Если вы уже запустили предыдущую программу на Liberty BASIC, которая создает текстовый файл STUFF.TXT, попробуйте запустить следующую программу, чтобы увидеть, как добавлять новые данные без уничтожения существующих:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #Secrets
PRINT  ftSecrets,  "Эта строка сохраняется в текстовом файле,"
CLOSE    flSecrets
OPEN "A:\STUFF.TXT" FOR APPEND AS #Secrets
PRINT #NewStuff, "Эти данные добавляются в существующий файл."
CLOSE #NewStuff
END
```

Получение данных из текстового файла

Конечно же, сохранять данные в текстовом файле — это замечательно, но до тех пор, пока эти данные не потребуются вам в следующий раз. К счастью, Liberty BASIC содержит команду, которая позволяет получать данные из текстового файла. В результате появляется возможность отображать эти данные на экране еще раз.

Для получения данных из текстового файла используется сочетание команд INPUT и OUTPUT, как показано ниже:

```
OPEN "A:\STUFF.TXT" FOR INPUT AS #Retrieve
```

Затем вы используете команду INPUT **для считывания данных из текстового файла:**

```
INRUT #FileHandle, Variable$
```

В этом примере #FileHandle представляет собой идентификатор файла, используемый вами раньше с помощью команды OPEN, а Variables — имя строковой переменной, которой временно присваивается значение, равное строке, извлеченной из текстового файла.

Если все это кажется вам слишком сложным, просто запустите эту программу, чтобы увидеть, как это все работает:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #myfile
INPUT "What is your name  "; Name$
PRINT ftmyfile, Name$
```

```
CLOSE #myfile
OPEN "A:\STUFF.TXT" FOR INPUT AS #file2
LINE INPUT #file2, YourName$
PRINT "This is the name you stored in the text file = " ("Это имя,
сохраненное вами в текстовом файле - "); YourName$
CLOSE #file2
END
```

Команда INPUT извлекает данные только по одной строке, начиная с первой строки в файле. Так как текстовые файлы обычно содержат много строк, используйте цикл или специальную команду EOF (аббревиатура от End-Of-File — конец файла).

Для того чтобы увидеть, как это все работает, попробуйте запустить следующую программу, которая сохраняет в текстовом файле три строки, а затем считывает их оттуда:

```
OPEN "A:\STUFF.TXT" FOR OUTPUT AS #ufo
• PRINT tufo, "Isn`t this exciting?"
PRINT #ufo, "Another line bites the dust."
PRINT tufo, "The last line in the text file."
CLOSE #ufo
OPEN "A:\STUFF.TXT" FOR INPUT AS #bigfoot
I = 1
WHILE EOF(#bigfoot) 0 0
  LINE INPUT tbigfoot, OneLine$
  PRINT OneLine$
  I = I + 1
WEND
CLOSE tbigfoot
END
```

Цикл WHILE WEND покажется вам несколько странным; более подробные сведения о циклах изложены в главе 10, "Использование циклов". Сейчас давайте рассмотрим следующий пример:

```
OPEN "A:\STUFF.TXT" FOR INPUT AS tbigfoot
I = 1
WHILE EOF(#bigfoot) = 0
   INPUT tbigfoot, OneLine$
   PRINT OneLine$
   I = I + 1
WEND
CLOSE tbigfoot
```

Вот как это все работает.

1. Первая строка кода приказывает компьютеру открыть текстовый файл STUFF.TXT, присвоить ему идентификатор tbigfoot и приготовиться считывать из него данные.
2. Вторая строка приказывает компьютеру создать переменную I и присвоить ей значение 1.
3. Третья строка сообщает компьютеру о начале цикла WHILE WEND, который закончится только по достижении конца файла с идентификатором tbigfoot. По достижении параметром EOF ненулевого значения цикл завершается.

4. Четвертая строка приказывает компьютеру считать одну строку из файла с идентификатором #bigfoot и присвоить ее в качестве переменной OneLine$. Сразу после команды INPUT компьютер переходит к следующей строке.
5. Пятая строка приказывает компьютеру вывести на печать значение переменной OneLine$.
6. Шестая строка приказывает компьютеру увеличить значение переменной I на единицу.
7. Седьмая строка сообщает компьютеру о завершении цикла WHILE WEND.
8. Восьмая строка приказывает компьютеру закрыть файл с идентификатором #bigfoot.

Сохранение данных в файлах с произвольной выборкой

Текстовые файлы очень подходят для сохранения и извлечения данных построчно, но как быть, если вам необходимо извлечь только последнюю строку из текстового файла; ведь компьютеру придется перебрать все строки, пока он доберется до нужной вам последней строки. Эта процедура во многом напоминает поиск любимой песни на аудиокассете. В обоих случаях вам придется проделать путь от начала файла (кассеты) до конца.

Для решения этой проблемы программисты придумали так называемые *файлы с произвольной выборкой*. В отличие от текстовых файлов, когда все данные помещаются в один большой файл, файлы с произвольной выборкой разделяют файл на небольшие части (называемые *записями*), каждая из которых содержит единицу данных. Каждая единица данных называется *полем*. Таким образом, если вам необходимо получить данные, содержащиеся в конце файла с произвольной выборкой, вы можете перейти сразу к нужному полю, а не просматривать весь файл целиком. Разница между хранением данных в текстовом файле и файле с произвольной выборкой показана на рис. 13.1.

Основное преимущество файлов с произвольной выборкой перед текстовыми заключается в скорости извлечения информации.

Создание нового файла с произвольной выборкой

Вы очень часто используете файлы с произвольной выборкой для сохранения связанных между собой данных, таких как имя человека, его адрес, идентификационный код или телефонный номер. Для сохранения данных в файле с произвольной выборкой вам необходимо определить следующие элементы.

- ✓ Сколько различных категорий информации вы намерены сохранять, таких как имя, возраст и адрес проживания.
- ✓ Сколько символов будет использоваться каждым элементом данных. Например, для имени человека можно выделить 15 символов, а для возраста — только 2.

После того как вы определите, сколько символов выделить каждой категории информации, вам необходимо указать общее количество символов, которые будут использоваться в одной записи, воспользовавшись командой FIELD, как показано ниже:

```
FIELD #Filehandle, X AS name$, Y AS age
```

Эта строка сообщает Liberty BASIC следующее: каждая запись содержит такие сведения, как имя человека и его возраст; имя может содержать *X* символов, а возраст — *Y*. Так как команда FIELD, помимо всего прочего, определяет и идентификатор файла, в котором будут сохраняться данные, ее следует использовать сразу после команды OPEN, как показано ниже:

```
OPEN "Имя_файла" FOR RANDOM AS #FileHandle LEN = Size
```

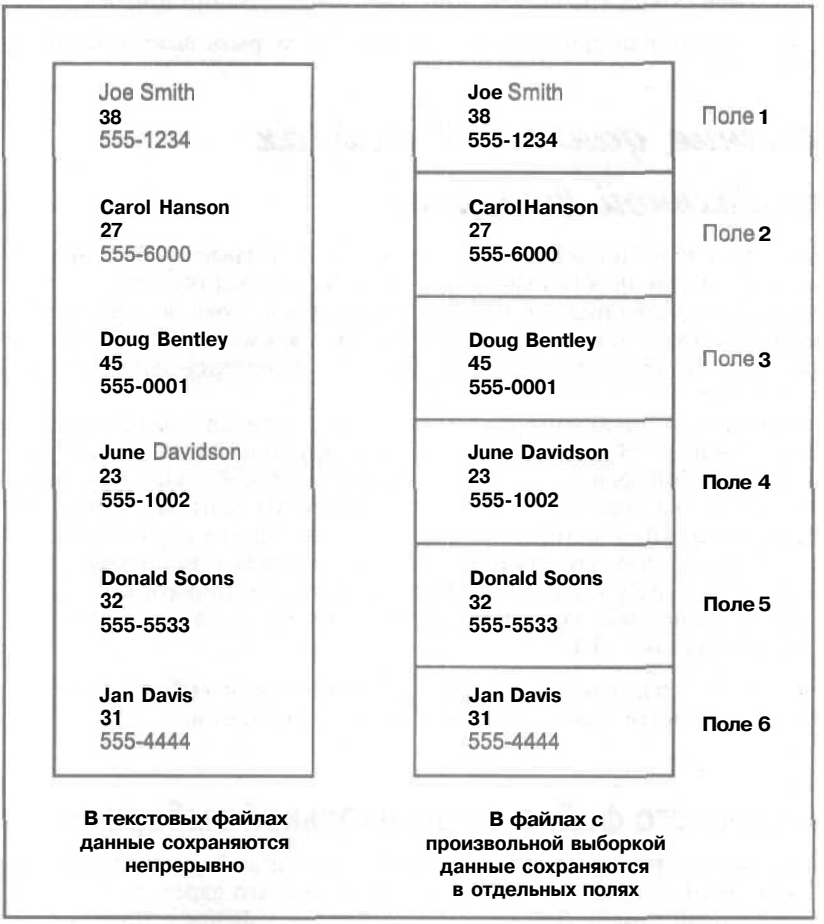

Рис. 13.1. В текстовых файлах данные сохраняются непрерывно от начала до конца; в файлах с произвольной выборкой данные сохраняются в отдельных полях

Вот что происходит при выполнении этой команды.

1. Команда OPEN приказывает компьютеру создать новый файл с произвольной выборкой с указанным именем (например, TRASH.DAT или C:\WINDOWS\TRASH.DAT).
2. Команда FOR RANDOM указывает компьютеру, что этот файл будет с произвольной выборкой.

3. #FileHandle — это идентификатор, указывающий на реальный файл, в котором и будут сохраняться данные. Хотя вы уже определили имя файла с произвольной выборкой, Liberty BASIC следует знать и идентификатор файла.
4. Команда LEN = Size сообщает компьютеру такие сведения, как общая длина записи. (Именно благодаря этим сведениям компьютер узнает, сколько записей он должен пропустить, прежде чем доберется до интересующей вас записи в середине файла.)

Например, для того чтобы создать файл TRASH.DAT, в котором будут сохраняться такие сведения, как имя (15 символов) и возраст (2 символа), вы можете использовать следующую команду:

```
OPEN "A:\TRASH.DAT" FOR RANDOM AS #garbage LEN = 17
```

Эта команда приказывает компьютеру создать файл с произвольной выборкой TRASH.DAT на дискете, назначить ему идентификатор #garbage и определить длину записи равной 17 символов.

Каждый раз, когда вы будете использовать команду OPEN для создания нового или открытия существующего файла с произвольной выборкой, вы должны обязательно командой CLOSE ее закрыть. Если вы не воспользуетесь этой командой, ваша программа может вызвать зависание компьютера.

Сохранение данных в файле с произвольной выборкой

После того как вы преодолели все сложности и создали файл с произвольной выборкой, вам не мешало бы узнать, как же сохранять данные в подобных файлах.

Для помещения данных в файл с произвольной выборкой воспользуйтесь следующей командой:

```
, PUT #Filehandle, Номер_записи
```

В этом примере ttFilehandle представляет собой идентификатор, присвоенный файлу с помощью команды OPEN, а Номер_записи определяет порядок, в котором данные будут сохраняться в файле с произвольной выборкой. (Запись номер 1 представляет собой первый набор данных в файле, запись номер 2 — второй набор данных в файле и т.д.)

Объединив все эти строки, мы получаем следующую программу:

```
OPEN "a:\stuff.dat" FOR random AS #losers LEN - 25
FIELD  fllosers,  15 AS name$, 2 AS age$, 8 AS phone$
FOR I =1 TO 3
  PROMPT "Type a name:"; name$
  PROMPT "What is this person's age?";  age$
PROMPT "What is this person`s phone number:"; phone$
       PUT #losers, I
NEXT I
CLOSE     ttlosers
END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру создать файл с произвольной выборкой STUFF.DAT на дискете, назначить ему идентификатор ttlosers и определить длину записи равной 25 символов.

2. Вторая строка приказывает компьютеру создать запись для файла с произвольной выборкой с идентификатором #losers. Каждая запись будет содержать такие сведения, как имя (15 символов), возраст (2 символа) и телефонный номер (до 8 символов). Общая длина записи составляет 25 символов.
3. Третья строка запускает цикл FOR NEXT и просит пользователя ввести все необходимые элементы записи.
4. Строки с четвертой по шестую отображают диалоговые окна, в которых пользователю предлагается указать имя, адрес и телефонный номер соответственно.
5. Седьмая строка использует команду PUT для сохранения таких сведений, как имя, адрес и телефонный номер, в файле с произвольной выборкой с идентификатором #losers. При первом запуске цикла FOR NEXT значение переменной I равно единице, поэтому первые имя, адрес и телефонный номер сохраняются в записи номер 1. При втором запуске цикла FOR NEXT значение переменной I равно двум, поэтому вторые имя, адрес и телефонный номер сохраняются в записи номер 2, и т.д.
6. Восьмая строка сообщает о завершении цикла FOR NEXT.
7. Девятая строка приказывает компьютеру закрыть файл с идентификатором ttlosers.
8. Десятая строка сообщает компьютеру о завершении программы.

Если вы запустите эту программу, на экране ничего не отобразится. Для того чтобы увидеть, какие же данные сохранены в файле с произвольной выборкой STUFF.DAT, вам необходимо извлечь сведения из этого файла; о том, как это сделать, я расскажу в следующем разделе.

Получение данных из файла с произвольной выборкой

Сохранив данные в файле с произвольной выборкой, вы можете извлечь их оттуда с помощью следующей команды:

```
GET #Filehandle, Номер_записи
```

В этом примере ttFilehandle представляет собой идентификатор, присвоенный файлу с помощью команды OPEN, а Номер_записи определяет порядок, в котором данные будут извлекаться из файла с произвольной выборкой. (Запись номер 1 представляет собой первый набор данных в файле, запись номер 2 — второй набор данных в файле и т.д.)

Все вышесказанное станет совершенно понятным, когда вы выполните следующую программу:

```
OPEN "a:\stuff.dat" FOR random AS #losers LEN - 25"
FIELD #losers, 15 AS name$, 2 AS age$, 8 AS phone$
FOR I =1 TO 3
  GET ftlosers, I
  PROMPT "Name = "; name$
  PROMPT "Age = "; age$
PROMPT "Phone = "; phone$
        PRINT
NEXT I
CLOSEtflosers
END
```

Если вы запускали программу из предыдущего подраздела "Сохранение данных в файле с произвольной выборкой" и трижды сохранили такие сведения, как имя, возраст и телефон в файле STUFF.DAT, вам легко извлечь эти данные из файла при первой необходимости.

1. Первая строка приказывает компьютеру создать файл с произвольной выборкой STUFF.DAT на дискете, назначить ему идентификатор #losers и определить длину записи равной 25 символов.
2. Вторая строка приказывает компьютеру создать запись для файла с произвольной выборкой с идентификатором ttlosers. Каждая запись будет содержать такие сведения, как имя (15 символов), возраст (2 символа) и телефонный номер (до 8 символов). Общая длина записи составляет 25 символов.
3. Третья строка запускает цикл FOR NEXT и просит пользователя ввести все необходимые элементы записи.
4. Четвертая строка использует команду GET для получения таких сведений, как имя, адрес и телефонный номер в файле с произвольной выборкой с идентификатором #losers. При первом запуске цикла FOR NEXT значение переменной I равно единице, поэтому первые имя, адрес и телефонный номер получаются из записи номер 1. При втором запуске цикла FOR NEXT значение переменной I равно двум, поэтому вторые имя, адрес и телефонный номер получаются из записи номер 2, и т.д.
5. Строки с пятой по восьмую используют команду GET для получения всех записей.
6. Девятая строка сообщает о завершении цикла FOR NEXT.
7. Десятая строка приказывает компьютеру закрыть файл с идентификатором ttlosera.
8. Одиннадцатая строка сообщает компьютеру о завершении программы.

Основное преимущество файлов с произвольной выборкой состоит в произвольном извлечении записей. Попробуйте запустить следующую программу, написанную на Liberty BASIC, которая предлагает вам ввести номер записи, которую необходимо извлечь, а затем извлекает указанную запись и выводит ее содержание на экран:

```
OPEN "a:\stuff.dat" FOR random AS ftlosers LEN - 25
FIELD fllosers, 15 AS name$, 2 AS age$, 8 AS phone$
PROMPT "What record do you want to retrieve data from?"; recordnum
GET #losers,recordnum
PROMPT "Name = "; name$
PROMPT "Age = "; age$
PROMPT "Phone - "; phone$
PRINT
CLOSE ftlosers
END
```

Создание и удаление каталогов

Каждый раз, когда программа сохраняет данные на диске, она записывает их в файл в определенном каталоге. Все диски содержат как минимум один каталог, который называется *корневым*. Если вы работаете с жестким диском C, корневым каталогом на нем будет каталог C:\. Если вы работаете с Windows, на жестком диске навер-

няка есть каталог C:\Windows. Пример типичного жесткого диска, содержащего несколько каталогов, приведен на рис. 13,2.

Термин *каталог* означает то же самое, что и термин *папка,* используемый при работе с Windows. По какой-то неизвестной мне причине в те времена, когда люди работали с операционной системой MS DOS, пространство жестких дисков разделялось на каталоги. В наши дни царствования Windows люди также используют каталоги, однако Windows называет их папками, чтобы упростить работу с компьютером.

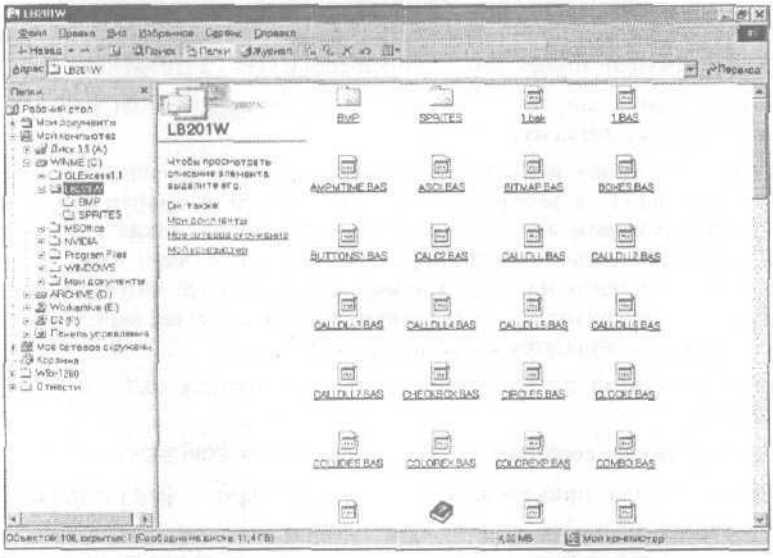

Рис. 13.2. Пространство большинства жестких дисков разделено на каталоги (папки) и подкаталоги (вложенные папки)

Создание каталога

Многие программы сохраняют данные в файле, который, в свою очередь, сохраняют в существующем каталоге. Однако вам понадобится создать новый каталог, чтобы программа имела возможность безопасно сохранять все свои данные в каталоге, в котором они не перепутаются с другими файлами. Для создания каталогов воспользуйтесь следующей командой:

```
Имя_переменной = MKDIR Имя_каталога
```

Здесь Имя_каталога — это имя, содержащее букву диска и название каталога, который вы решили создать. Если команда MKDIR успешно создала каталог, переменной Имя_переменной присваивается нулевое значение. Если команде MKDIR не удалось создать каталог, переменной Имя_переменной присваивается отличное от нуля значение. Например, если вы хотите создать каталог TEMP на дискете (диске A), воспользуйтесь следующей командой;

```
X = MKDIR "A:\TEMP"
```

Удаление файлов и каталогов

Если какой-то каталог вам больше не нужен, удалите его. Но, прежде чем вы удалите каталог, вы должны удалить все файлы и подкаталоги, которые в нем создали.

При использовании Windows можно удалить **папку (также называемую** каталогом); **при этом автоматически** удаляются **все файлы и вложенные папки. Однако при** обращении **к Windows с** помощью команды **языка** программирования операционная система не позволит вам удалить **файлы из каталога (папки). Это** предотвращает удаление файлов **программой по ошибке.**

Удаление нужных файлов действительно может **привести к проблемам, поэтому используйте** следующую команду **осторожно:**

```
KILL Имя_файла
```

В данном случае Имя_файла **определяет диск, каталог и файл, который вы решили удалить. Например, если вы хотите** удалить **файле** RAMFILE.SYS **в папке** TEMP на дискете, **воспользуйтесь** следующей **командой:**

```
KILL "A:\TEMP\RAMFILE.BAS"
```

Вместе с командой KILL **можно использовать подстановочные символы "*" и "?". Например, для удаления всех файлов на диске A воспользуйтесь** следующей **командой:**

```
KILL "A:\*.*"
```

Для удаления **на диске A всех** файлов **с расширением .EXE воспользуйтесь следующей командой:**

```
1 KILL "A:\*.EXE"
```

После удаления всех файлов вы сможете удалить **и каталог, воспользовавшись** следующей **командой:**

```
Имя_переменной = RMDIR Имя_каталога
```

Здесь Имя_каталога **— это имя, содержащее букву** диска и название каталога, **который вы решили** удалить. **Если команда** RMDIR успешно удалила каталог, переменной Имя_переменной **присваивается** нулевое **значение. Если команде** RMDIR **не удалось удалить** каталог, **переменной** Имя_переменной **присваивается отличное от нуля значение. Например,** если **вы хотите удалить файл** KITTY **на диске A, воспользуйтесь следующей командой:**

```
X = RMDIR "A:\KITTY"
```

Глава 14

Создание интерфейса пользователя

В этой главе...

- Создание окна
- Добавление новых меню в окно
- Размещение в окне элементов управления
- Использование диалоговых окон

Как и следует из его названия, *интерфейс пользователя* действует как посредник между пользователем и программой. Пользователь вводит команды с помощью интерфейса пользователя, который передает эти команды части программы, действительно выполняющей какую-то работу. После этого программа возвращает данные интерфейсу пользователя для отображения на экране, чтобы пользователь смог их увидеть.

Проектирование окна

В любой программе для Windows самый распространенный элемент — это окно, т.е. прямоугольник, который отображается на экране. Окна создаются для выполнения двух следующих функций.

- ✓ Отображение команд, выполняемых программой, в виде раскрывающихся меню и кнопок, которые пользователю нужны для выбора каких-то вариантов или ввода информации.
- ✓ Отображение информации на экране, благодаря чему пользователь может прочитать ее, например график изменения котировки акций или текст, вводимый самим пользователем.

Создание нового окна

Для того чтобы создать окно, вам необходимо использовать команду OPEN, а также определить текст, который будет отображаться в строке заголовка окна, как показано ниже.

```
 NOMAINWIN
 OPEN "Titlebatext" FOR Window AS #1
 PRINT #1, "trapclose [quit]"
;WAIT

I [quit]
 CONFIRM "Are you sure that you to quit?"; quit$
 IF quit$ = "no" THEN WAIT
 CLOSE #1
I END
```

Эта программа, написанная на Liberty BASIC, приказывает компьютеру выполнить следующее.

1. Первая строка приказывает Liberty BASIC не отображать основное окно.
2. Вторая строка использует команды OPEN и FOR WINDOW *AS* для создания окна, отображения подписи в строке заголовка, а также присвоения окну идентификатора #1.
3. Третья строка приказывает компьютеру выполнить инструкции, указанные сразу после метки [label], если пользователь предпримет попытку закрыть окно. Чтобы определить попытку пользователя закрыть окно с идентификатором #1, используется команда PRINT, однако на экран при этом не выводится никакой информации.
4. Четвертая строка использует команду WAIT для того, чтобы приказать компьютеру ждать каких-то действий от пользователя.
5. Пятая строка содержит метку [label].
6. Шестая строка отображает диалоговое окно с вопросом Are you sure that you to quit? (Вы уверены, что хотите завершить работу программы?). При этом в окно автоматически добавляются кнопки Yes (Да) и No (Нет). Если пользователь щелкает на кнопке Yes (Да), программа присваивает значение "yes" переменной quits. Если же пользователь щелкает на кнопке No (Нет), программа присваивает значение "по" переменной quits.
7. Седьмая строка проверяет, равно ли значение переменной quit$ "no". Если это так, команда WAIT используется для того, чтобы заставить пользователя предпринять какие-то действия.
8. Восьмая строка закрывает окно с идентификатором #1. Эта строка выполняется только в том случае, если пользователь щелкает на кнопке Yes (Да) в диалоговом окне, отображенном на экране при выполнении шестой строки.
9. Девятая строка сообщает компьютеру о завершении работы.

Если вы запустите эту программу, на экране отобразится диалоговое окно, для закрытия которого вам придется предпринять какие-то действия.

Каждый раз, когда вы будете использовать команду OPEN для создания нового окна, вы должны обязательно командой CLOSE ее закрыть. Если вы не воспользуйтесь этой командой, ваша программа может привести к зависанию компьютера.

Определение размеров и расположения окна

Когда вы создаете новое окно, Liberty BASIC отображает его в той или иной части экрана. Если вы хотите указать точное расположение окна, вам потребуются команды UpperLeftX и UpperLeftY для определения координат верхнего левого края окна по горизонтали и вертикали соответственно. Если вы хотите, чтобы верхний левый край окна находился ровно на 225 пикселей сверху и на 139 пикселей слева по отношению к верхнему левому краю экрана (рис. 14.1), выполните следующую программу:

```
NOMAINWIN
UpperLeftX - 139
```

```
UpperLeftY = 225
OPEN "Titlebar" FOR Window AS #1
PRINT #1, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ - "no" THEN WAIT
CLOSE #1
 END
```

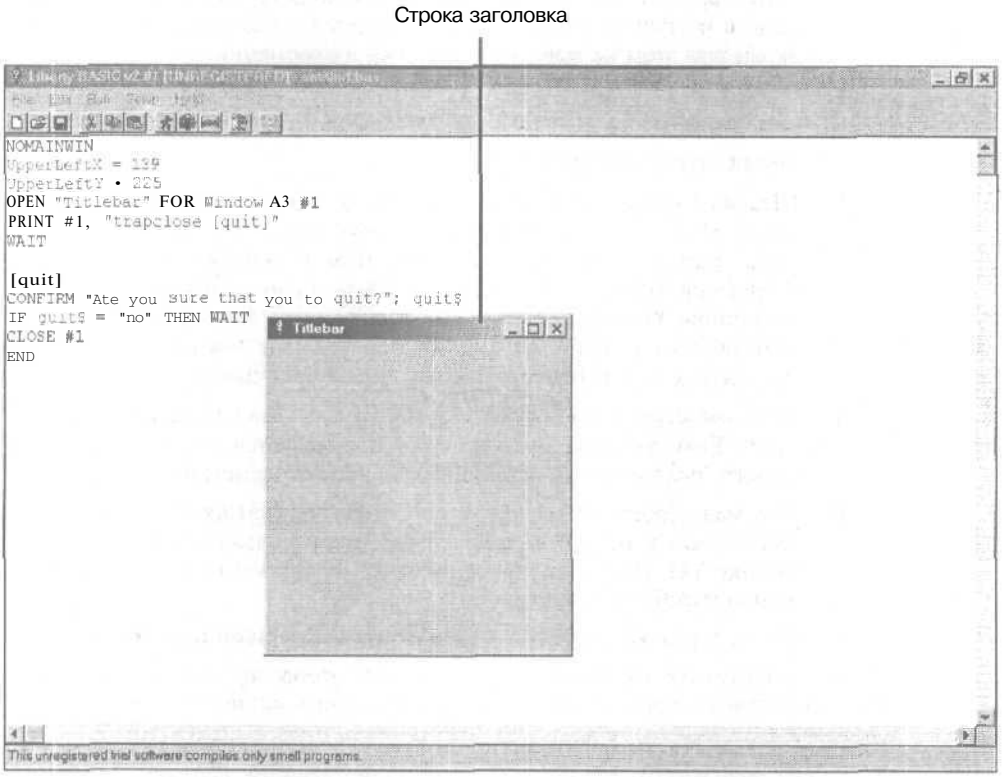

Рис. 14.1. Создав окно, задайте текст, который будет отображаться в строке заголовка, расположение на экране, а также размер окна

Команды UpperLeftX и UpperLeftY определяют координаты верхнего левого края окна по горизонтали и вертикали соответственно. Если вы хотите определить точные размеры окна, воспользуйтесь командами WindowHeight и WindowWidth для определения размера окна в пикселях, как показано ниже:

```
NOMAINWIN
UpperLeftX = 139
UpperLeftY = 225
WindowWidht - 550
WindowHeight = 275
! OPEN "Titlebar" FOR Window AS #1
PRINT #1, "trapclose [quit]"
```

```
WAIT
 [quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ - "no" THEN WAIT
CLOSE #1
END
```

Вы должны определить размеры и расположение окна до использования команды OPEN, предназначенной для создания окна на экране.

Добавление цветов

Liberty BASIC также позволяет вам указывать фоновый цвет окна с помощью команды BackgroundColor$, присваивая ей значение одного из следующих цветов.

White (Белый)
Black (Черный)
Lightgray (Светло-серый) или Palegray (Бледно-серый)
Darkgray (Темно-серый)
Yellow (Желтый)
Brown (Коричневый)
Blue (Синий)
Cyan (Голубой)
Darkblue (Темно-синий)
Red (Красный)
Darkred (Темно-красный)
Pink (Розовый)
Darkpink (Темно-розовый)
Green (Зеленый)
Darkgreen (Темно-зеленый)
Darkcyan (Темно-голубой)

Например, если вы хотите создать окно с розовым фоном, воспользуйтесь следующими командами:

```
NOMAINWIN
UpperLeftX = 139
UpperLeftY = 225
WindowWidht = 550
WindowHeight = 275
BackgroundColor$ = "Pink"
OPEN "Строка заголовка" FOR Window AS #1
PRINT ttl, "trapclose [quit]"
WAIT

 [quit]
; CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ = "no" THEN WAIT
CLOSEttl
 END
```

Добавление раскрывающихся меню в окно

Большинство окон содержат *раскрывающиеся меню*, которые позволяют пользователям выбирать нужные команды. Для создания раскрывающегося меню вам необходима команда MENU, которая позволяет определить следующие параметры.

- ✓ Название меню, например File (Файл), Edit (Правка) или Help (Справка)
- ✓ Команды меню, которые отображаются под названием меню, например Edit (Изменить), Print (Печать) или Cut (Вырезать)

I ✓ Метка, которая позволяет программе определить, какие действия необходимы после выбора пользователем определенной команды меню

Типичная команда для создания меню выглядит следующим образом:

```
MENU #WindowHandle, "Название_меню", "Команда1", [command1]
```

В данном примере tfwindowHandle — это идентификатор окна, в которое должно быть добавлено меню, Название_меню представляет собственно название меню, а команда Команда1 появляется после того, как пользователь щелкнет на названии раскрывающегося меню. Для того чтобы продемонстрировать вам реальный пример, давайте рассмотрим следующую программу:

```
NOMAINWIN
MENU #1, "&File", "&Open", [asOpen], "&Print", [asPrint], "&Exit", fquit]
OPEN "Menu Example" FOR Window AS #1
PRINT #1, "trapclose [quit]"
,WAIT

i[quit]
 CONFIRM "Are you sure that you to quit?";
        quitS
 IF quit$ = "no" THEN WAIT
 CLOSE #1
  END

 [asOpen]
 NOTICE "Open command chosen"
-WAIT

 [asPrint]
 NOTICE "Print command chosen"
 WAIT
```

Эта программа работает следующим образом.

1. Первая строка приказывает Liberty BASIC не отображать основное окно.
2. Вторая строка определяет раскрывающееся меню, которое появляется в окне с идентификатором #1. Меню называется File (Файл) и содержит такие команды, как Open (Открыть), Print (Печать) и Exit (Выход).

Если вы хотите, чтобы какая-нибудь буква в названии меню или команды отображалась подчеркнутой, поместите знак амперсанда (&) перед этой буквой, например &Print. Любая подчеркнутая буква — это горячая клавиша, например <Alt+Ф> для выбора меню Файл. После выбора меню пользователю достаточно нажать подчеркнутую букву в названии команды, чтобы ее выбрать, например клавишу <ы> для команды Выход.

3. Третья строка приказывает компьютеру создать окно и присвоить ему идентификатор #1.
4. Четвертая строка приказывает выполнить инструкции, указанные сразу после метки [label], если пользователь предпримет попытку закрыть окно.
5. Пятая строка использует команду WAIT для того, чтобы приказать компьютеру ждать каких-то действий от пользователя.

6. Шестая строка содержит метку [label].
7. Седьмая строка отображает диалоговое окно с вопросом Are you sure that you to quit? (Вы уверены, что хотите завершить работу программы?). При этом в окно автоматически добавляются кнопки Yes (Да) и No (Нет). Если пользователь щелкает на кнопке Yes (Да), программа присваивает значение "yes" переменной quit$. Если же пользователь щелкает на кнопке No (Нет), программа присваивает значение "no" переменной quits.
8. Восьмая строка проверяет, равно ли значение переменной quit$ "no". Если это так, команда WAIT используется для того, чтобы заставить пользователя предпринять какие-то действия.
9. Девятая строка закрывает окно с идентификатором #1. Эта строка выполняется только в том случае, если пользователь щелкает на кнопке Yes (Да) в диалоговом окне, отображенном на экране при выполнении шестой строки.
10. Десятая строка сообщает компьютеру о завершении работы.
11. Одиннадцатая строка определяет метку [asOpen]. После того как пользователь выберет команду Open (Открыть) из меню File (Файл), компьютер немедленно перейдет к этой метке для получения дальнейших инструкций.
12. Двенадцатая и тринадцатая строки приказывают компьютеру отобразить диалоговое окно с сообщением Open command chosen (Выбрана команда Открыть). Затем ожидаются какие-то действия со стороны пользователя.
13. Четырнадцатая строка определяет метку [asPrint]. После того как пользователь выберет команду Print (Печать) из меню File (Файл), компьютер немедленно перейдет к этой метке для дальнейших инструкций.
14. Пятнадцатая и шестнадцатая строки приказывают компьютеру отобразить диалоговое окно с сообщением Print command chosen (Выбрана команда Печать). Затем ожидаются какие-то действия со стороны пользователя.

Для добавления дополнительных меню в окно (рис. 14.2) используйте команду MENU несколько раз, как показано ниже:

```
NOMAINWIN
MENU #1, "&File", "&Open", [asOpen], "&Print", [asPrint], "E&xit", (quit]
MENU #1, "&Help", "&Contents", [asContents], "&About", [asAbout}
OPEN "Menu Example" FOR Window AS #1
PRINT #1, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ = "no" THEN WAIT
CLOSE #1
END

[asOpen]
NOTICE "Open command chosen"
WAIT

[asPrint]
NOTICE "Print command chosen"
WAIT

; [asContents]
```

```
NOTICE "Contents command chosen"
WAIT
[asAbout]
NOTICE "About command chosen"
WAIT
```

Каждый раз, когда вы будете добавлять новые команды в меню, вам обязательно следует использовать новую метку для указания инструкций, которые будут выполняться компьютером после выбора соответствующей команды.

Рис. 14,2. Liberty BASIC способен создавать раскрывающиеся меню

Создание контекстных меню

В дополнение к раскрывающимся меню многие программы предлагают *контекстные меню,* которые появляются рядом с указателем мыши после щелчка правой кнопкой мыши. Для создания контекстного меню в Liberty BASIC используется команда POPUPMENU:

```
POPUPMENU "команда1", [label1], " команда2", [label2]
```

Для того чтобы сделать команду POPUPMENU проще для понимания, поместите каждую команду и соответствующую метку на отдельную строку, используя в конце каждой строки знак подчеркивания (_). Знак подчеркивания укажет Liberty BASIC на то, что в следующей строке содержатся дополнительные инструкции. Вы можете использовать знак подчеркивания для разделения длинной строки на несколько коротких, как показано ниже:

```
POPUPMENU_
"команда1", [label1],_
"команда2", [label2],_
"команда3", [label3]
```

Вот что делают перечисленные выше команды.

1. Команда POPUPMENU приказывает компьютеру создать контекстное меню в текущем расположении указателя мыши.
2. Команда1, команда2 и команда3 — это действительные названия команд, которые появляются в меню. Вы можете добавить новые команды, поэтому контекстное меню может содержать 5, 7 или даже 15 различных команд.

3. Метки [label1] и [label2] определяют инструкции, которые выполняются компьютером после выбора пользователем той или иной команды меню. Так, если пользователь выберет команда1, компьютер немедленно приступит к исполнению инструкций, указанных после метки [label1].

Если вы хотите добавить горизонтальную линию для разделения команд в контекстном меню, просто добавьте символ "|" вместо команды и опустите метку, которая должна следовать после него, как показано ниже:

```
POPUPMENU_
"команда1, [label1],_
|,_
"команда3", [label3]
```

Для того чтобы увидеть контекстные меню в действии, попробуйте выполнить следующую программу.

```
NOMAINWIN
POPUPMENU_
   "Destroy the planet",  [one],_
   "Poison the environment",  [two],_
   |,_
   "Get elected to political office",  [three]

NOTICE "Nothing selected."
END

[one]
NOTICE "Launching missiles now."
END

[two]
NOTICE "Spilling oil into the oceans."
END

[three]
NOTICE "Fooled the public once more."
END
```

Добавление в окно управляющих элементов

Хотя для выбора пользователем команд можно использовать меню, вы наверняка захотите добавить в окно кнопки, флажки, переключатели и текстовые окна. Для того чтобы это сделать, вам необходимо определить следующее.

- ✓ Имя управляющего элемента
- ✓ Текст, который будет отображаться рядом
- ✓ Метка для идентификации инструкций, которые должен выполнить компьютер в случае выбора пользователем определенного управляющего элемента
- ✓ Ширина и высота управляющего элемента
- ✓ Расположение управляющего элемента

Создание кнопок

Кнопка — это серый прямоугольник с подписью, например OK или Cancel (Отмена). Для создания кнопки вам следует воспользоваться командой BUTTON:

```
I BUTTON #Windowhandle.имя_кнопки, "Подпись к кнопке",
   [branchLabel], UL, xpos, ypos, ширина, высота
```

Рис. 14.3. Контекстные меню предоставляют пользователям дополнительные возможности выбора

Если вы не укажете ширину и высоту, размеры кнопки автоматически подгоняются под размеры подписи, определенной для этой кнопки.

Вот как работает эта команда.

1. Команда BUTTON приказывает компьютеру создать кнопку в окне с идентификатором #Windowhandle.
2. Имя_кнопки определяет уникальное имя кнопки.
3. Подпись к кнопке действительно есть на кнопке.
4. Метка [branchLabel] идентифицирует инструкции, которые должен выполнить компьютер после щелчка пользователем на кнопке.
5. Параметр UL указывает, что кнопка должна быть создана в верхнем левом углу окна. Помимо параметра UL, вы также можете использовать параметры UR (верхний правый), LL (нижний левый), LR (нижний правый).
6. Параметры xpos и ypos определяют расположение кнопки по осям *X* и *Y* по отношению к выбранному углу окна.
7. Параметры ширина и высота определяют ширину и высоту кнопки соответственно.

Для того чтобы увидеть создание кнопки, запустите следующую программу. Результат выполнения программы приведен на рис. 14.4.

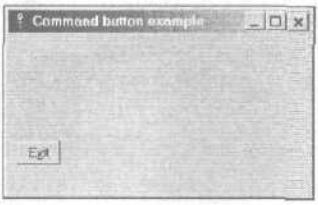

Рис. 14.4. Если вы создаете кнопку, определите ее размер и расположение

```
NOMAINWIN
WindowHeight = 200
WindowWidht = 250
BUTTON #main.mybutton. "E&xit", [quit], LL, 10, 10, 45, 25
j OPEN "Command button example" FOR WINDOW AS #main
PRINT #main, "trapclose [quit]"
WAIT

[quit]
CONFIRM "Are you sure that you to quit?"; quit$
IF quit$ = "no" THEN WAIT
CLOSE #main
 END
```

Помимо создания настоящих кнопок, Liberty BASIC также позволяет вам определять в качестве кнопки растровое изображение. Для превращения растрового изображения в кнопку используется команда BMPBUTTON:

```
BMPBUTTON #Windowhandle.имя_кнопки, "bitmap filename",
  [branchLabel], UL, xpos, ypos
```

Вот как работает эта команда.

1. Команда BMPBUTTON приказывает компьютеру создать кнопку в окне с идентификатором ttWindowhandle.
2. *Имя_кнопки* определяет уникальное имя кнопки.
3. Bitmap filename определяет каталог и имя файла, содержащего необходимое растровое изображение, например C:\Liberty Basic\face.bmp.
4. Метка [branchLabel] **идентифицирует инструкции, которые должен выполнить компьютер** после **щелчка пользователем на кнопке.**
5. Параметр UL указывает, что кнопка должна быть создана в верхнем левом углу окна. Помимо параметра UL, вы также можете использовать параметры UR (верхний правый), LL (нижний левый), LR (нижний правый).
6. Параметры xpos и ypos определяют расположение кнопки по осям *X* и *Y* по отношению к выбранному углу окна.

Для того чтобы увидеть создание кнопки, запустите следующую программу:

```
NOMAINWIN
WindowWidth = 400
BMPBUTTON #main.picturebtn, "vwsignon.bmp", [Exit], UL, 10, 40
OPEN "BITmap Button Example" -FOR WINDOW AS #main
WAIT
 [quit]
CONFIRM "Are you sure that you to quit?";
        quits
IF quit$ = "no" THEN WAIT
CLOSE #main
END
```

Отображение текста

Иногда вам нужно отобразить в окне текст, представляющий собой сообщение или инструкции, адресованные пользователю. В Liberty BASIC текст, который не может изменить пользователь, называется *статическим*; для его создания используется STATICTEXT:

```
STATICTEXT #Windowhandle.staticname, "Статический текст", xpos, ypos, j
ширина, высота
```

Вот как работает эта команда.

1. Команда STATICTEXT приказывает компьютеру отобразить текст в окне с идентификатором #Windowhandle. При этом тексту присваивается уникальное имя staticname.
2. Статический текст определяет текст, который будет отображаться в окне.
3. Параметры xpos и ypos определяют расположение текста по осям *X* и *Y* по отношению к верхнему левому углу окна.
4. Параметры ширина и высота определяют ширину и высоту текста соответственно.

Для изменения статического текста можно использовать команду PRINT:

```
PRINT #Windowhandle.staticname, "Новый текст"
```

Для того чтобы увидеть все это, попробуйте выполнить следующую программу:

```
NOMAINWIN
 STATICTEXT #main.static, "Old text", 10, 10, 75, 25
 BUTTON #main.mybutton, "Change", [change], LL, 10, 10, 45, 25
 OPEN "Real-life example" FOR Window AS #main
 PRINT #main, "trapclose [quit]"
 WAIT

• [change]
 PRINT ttmain. static, "New text"
i WAIT

 [quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ = "no" THEN WAIT
CLOSE #main
END
```

В результате выполнения этой программы на экране отображается окно, в котором содержится текст Old text (Старый текст). После того как вы щелкнете на кнопке Change (Изменить), в окне отобразится текст New text (Новый текст).

Создание флажков

Иногда вам нужно предоставить пользователю несколько вариантов выбора. В этом случае используйте флажки, рядом с каждым из которых должна быть поясняющая надпись (рис. 14.5).

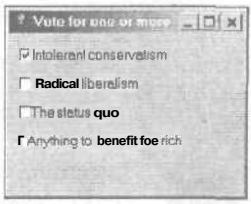

Рис. 14.5. Флажки позволяют предоставить пользователю несколько вариантов выбора

Для создания флажков предназначена команда CHECKBOX:

```
CHECKBOX #Windowhandle.boxname, "Подпись к флаж-
;ку", [set],
   [reset], xpos, ypos, width, height
```

Вот как работает эта команда.

1. Команда CHECKBOX приказывает компьютеру создать флажок в окне с идентификатором ttWindowhandle.
2. Boxname определяет уникальное имя флажка.
3. Подпись к флажку определяет текст, отображаемый рядом с флажком.
4. Метка [set] идентифицирует инструкции, которые должен выполнить компьютер, если пользователь установит флажок. Метка [reset] идентифицирует инструкции, которые должен выполнить компьютер, если пользователь сбросит флажок.
5. Параметры xpos и ypos определяют расположение флажка по осям *X* и *Y* по отношению к верхнему левому углу окна.
6. Параметры ширина и высота определяют ширину и высоту флажка соответственно.

Для того чтобы увидеть создание флажка, запустите следующую программу. Результат выполнения программы приведен на рис. 14.5.

```
NOMAINWIN
WindowWidth = 250
WindowHeight = 200
CHECKBOX     ftl.checkl, "Intolerant conservatism", [set], [reset], 10,
       10, 250, 25
CHECKBOX #1.check2, "Radical liberalism", [set], [reset], 10, 40,
       250, 25
CHECKBOX #1.check3, "The status quo", [set],[reset], 10, 70, 250, 25
CHECKBOX #1.check4, "Anything to benefit the rich", [set], [reset],
       10, 100, 250, 25
OPEN "Vote for one or more" FOR Window AS 41
PRINT #1, "trapclose [quit]"
WAIT

[set]
NOTICE "Are you sure live in a democracy?"
•WAIT
```

```
 [reset]
I NOTICE "Good move!"
 WAIT

'. [quit]
 CONFIRM "Are you sure that you to quit?";
         quit$
 IF quit$ - "no" THEN WAIT
 CLOSE #1
  END
```

Значение флажка может быть равно set (если флажок установлен) или reset (если флажок сброшен). Для определения состояния флажка используйте две следующие команды:

```
PRINT #1.cboxname, "Value?"
INPUT #1.cboxname, result$
```

Команда PRINT **получает значение от** флажка **с идентификатором** cboxname, а команда INPUT **присваивает это значение переменной** result$.

Создание переключателей

В отличие от флажков, которые предлагают выбрать несколько вариантов из предложенного списка, переключатели позволяют выбрать всего один. Для создания переключателя используется команда RADIOBUTTON, действие которой во многом напоминает действие команды CHECKBOX, как показано ниже:

```
RADIOBUTTON #Windowhandle.radioname, "Подпись к переключателю", -
[set], [reset], xpos, ypos, width, height
```

Чтобы увидеть переключатели в работе, просто запустите следующую программу (результат ее выполнения показан на рис. 14.6). После щелчка на кнопке Check radio button 1 (Проверить переключатель 1) на экране отображается соответствующее сообщение.

```
NOMAINWIN
WindowWidth = 250
WindowHeight = 200
RADIOBUTTON #1.radio1, "Intolerant conservatism", [set], [reset], 10,
      10, 250, 25
RADIOBUTTON #1.radio2, "Radical liberalism", [set], [reset], 10, 40,
      250, 25
RADIOBUTTON #1.radio3, "The status quo", [set], [reset], 10, 70, 250,
      25
RADIOBUTTON #1.radio4, "Anything to benefit the rich", [set],
 [reset], 10, 100, 250, 25
 BUTTON #1.bitton1, "Check radio button 1", [test], LL, 50, 3
 OPEN "Vote for one or more" FOR Window AS #1
 PRINT #1, "trapclose [quit]"
;WAIT

 [test]
 PRINT #1.radio1, "Value?"
; INPUT#1.radio1, testS
 NOTICE "The value of radio button 1 is "; testS
```

```
WAIT

[set]
WAIT

  [quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ - "no" THEN WAIT -
CLOSE #1
END
```

Значение флажка может быть set (если флажок установлен) или reset (если флажок сброшен). Для определения состояния флажка используйте две следующие команды:

```
PRINT #1.radiobuttonname, "Value?"
INPUT #1.radiobuttonname, result$
```

Команда PRINT **получает значение от переключателя с идентификатором** radiobuttonname, **а команда** INPUT **присваивает это значение переменной** result$.

Создание текстовых полей

Текстовые поля представляют собой небольшие окна, которые просто содержат текст или позволяют пользователям вводить текст. Для создания текстовых полей используется команда TEXTBOX:

```
TEXTBOX #Windowhandle.textboxname, xpos, ypos, width, height
```

Вот как работает эта команда.

1. Команда TEXTBOX приказывает компьютеру создать текстовое поле в окне с идентификатором #Windowhandle.
2. Textboxname определяет уникальное имя текстового поля.
3. Параметры xpos и ypos определяют расположение текстового поля по осям *X* и *Y* по отношению к верхнему левому углу окна.
4. Параметры ширина и высота определяют ширину и высоту текстового поля соответственно.

Если вы хотите добавить текст в текстовое поле, вам необходимо использовать команду PRINT, указав идентификатор окна, имя текстового поля и собственно текст, который должен в нем содержаться, как показано ниже:

```
PRINT #1.text1, "Этот текст отображается в текстовом поле text1."
```

Для получения текста из текстового поля вам необходимо использовать следующие две команды:

```
PRINT #1.text1, "!contents?"
 INPUT #1.text1, stuff$
```

Для того чтобы увидеть, как работают текстовые поля, выполните следующую программу, отображающую текстовое поле и меню, которое позволяет вам удалить текстовое поле или отобразить его содержимое в диалоговом окне Notice.

```
NOMAINWIN
WindowWidth = 250
WindowHeight = 200
TEXTBOX tfl.textl, 25,25,200,100
MENU #1, "&Options", "Clear text box", [clear], _
  "&Display text from text box", [display], _
  "E&xit", [quit]
OPEN "Text box example" FOR Window AS #1
PRINT ttl.textl, "Initial text."
PRINT #1, "trapclose [quit]"
WAIT
[clear]
PRINT #1.textl, ""
WAIT
[display]
PRINTtfl.textl,-"!contents?"
INPUT #1.textl, stuff$
NOTICE "Text in text box = " + stuffS
WAIT
[quit]
CONFIRM "Are you sure that you to quit?"; quit$
IF quitS - "no" THEN WAIT
CLOSE #1
END
```

Создание списков

Для того чтобы предоставить пользователям возможность выбрать один из нескольких вариантов, используйте флажки или переключатели. Списки особенно полезны, когда нужно отобразить несколько элементов, которые не очень удобно представлять в виде группы флажков или переключателей. Кроме того, списки позволяют избавиться от ошибок ввода, так как пользователь просто щелкает на необходимом варианте, а не вводит какой-то текст (а значит, рискует допустить ошибку).

Для создания списков используется команда LISTBOX, как показано ниже:

```
LISTBOX #Windowhandle.listboxname, array$(), [action], xpos, ypos,
ширина, высота
```

Вот как работает эта команда.

1. Команда LISTBOX приказывает компьютеру создать кнопку в окне с идентификатором #Windowhandle.
2. Listboxname определяет уникальное имя кнопки.
3. Array$ представляет *массив*, содержащий все элементы, которые должны быть в списке.
4. Метка [action] идентифицирует инструкции, которые должен выполнить компьютер после выбора пользователем одного из элементов списка.
5. Параметры xpos и ypos определяют расположение списка по осям *X* и *Y* по отношению к верхнему левому углу окна.
6. Параметры ширина и высота определяют ширину и высоту списка соответственно.

Для того чтобы выбрать элемент из списка, на нем необходимо дважды щелкнуть. Если вы хотите, чтобы выбор элемента из списка осуществлялся с помощью одного щелчка, вам необходимо использовать следующую команду:

```
PRINT #handle.listboxname, "singleclickselect"
```

Эта команда приказывает компьютеру разрешить пользователю выбирать элементы из списка listboxname с помощью одного щелчка.

После того как пользователь выберет элемент из списка, для идентификации выбора пользователя используются следующие две команды:

```
PRINT ttl.listboxl, "selection?"
INPUT fll.listbcxl. selected$
```

Для того чтобы увидеть использование списков, выполните следующую программу, которая отображает два списка, как показано на рис. 14.6. Поскольку размер верхнего списка меньше, чем общее число элементов в нем, к нему автоматически добавляется полоса прокрутки. Кроме того, верхний список позволяет пользователю выбирать элементы с помощью следующей команды:

```
PRINT   ttl.listl,   "singleclickselect"
```

С другой стороны, второй список заставляет пользователей для выбора элементов шелкать дважды.

```
NOMAINWIN
array$(0) = "Mystery meat"
array$(1) = "Cat food"
array$(2) = "  Something pink and artificially preserved"
array$(3) = "  Liquid mush"
array$(4) = "Sugar and artificial coloring"

WindowWidth = 380
".WindowHeidht = 240

LI3TBOX #1.list1, array&(), [Action1], 40, 10, 300, 70
LISTBOX #1.list2, array&(), [Action1], 40, 100, 300, 90
OPEN "Here are your choices for dinner tonight" FOR Window AS #1
PRINT #1.list1, "singleclickselect"
PRINT #1, "trapclose [quit]"
WAIT

[Action1]
PRINT #1.list1, "selection?"
INPUT #1.list1, selected$
NOTICE "You chose = + selected$
iWAIT

[Action2]
PRINT #1.list2, "selection?"
INPUT #1.list2, selected$
NOTICE "You chose = " + selected$
WAIT

[quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ = "no" THEN WAIT
CLOSE #1
END
```

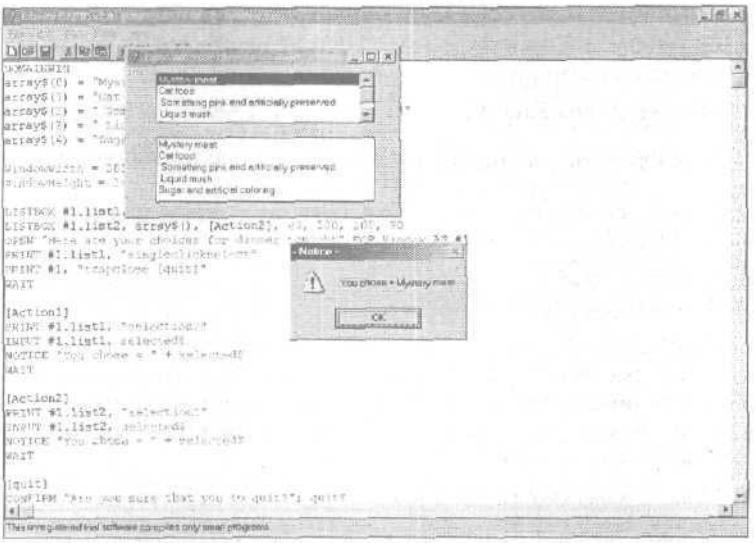

Рис. 14.6. Списки предоставляют пользователю несколько возможностей выбора на совсем небольшом пространстве

Создание раскрывающихся списков

Раскрывающиеся списки во многом напоминают обычные списки, но при этом занимают на экране еще меньше места. После того как пользователь щелкнет в раскрывающемся списке, на экране появляется несколько вариантов выбора, как показано на рис. 14.7.

Для создания раскрывающегося списка используется команда COMBOBOX, которая во многом похожа на команду LISTBOX:

```
COMBOBOX ttWindowhandle.comboboxname, array$(), [action], xpos, ypos,
width, height
```

Рис. 14.7. Раскрывающиеся списки занимают совсем немного места

После того как пользователь выберет элемент из раскрывающегося списка, используйте следующие две команды, чтобы определить, что же выбрал пользователь:

```
PRINT ttl.conboboxl, "selection?"
INPUT ttl.comboboxl, selected$
```

Для того чтобы увидеть раскрывающиеся списки в действии, выполните следующую программу:

```
NOMAINWIN
array$(0) = "Mystery meat"
array$(1) = "Cat food"
array$(2) = " Something pink and artificially preserved"
array$(3) = " Liquid mush"
array$(4) = "Sugar and artificial coloring"

WindowWidth = 300
WindowHeidht = 240

COMBOBOX #1.combol, array$(, [Action], 40, 50, 216, 100
OPEN "Here are your choices for dinner tonight" FOR WINDOW AS #1
PRINT #1, "trapclose [quit]"
WAIT

[Action]
PRINT ftl.combol, "selection?"
INPUT tfl.combol, selected$
NOTICE "You chose = " + selected$
WAIT

[quit]
CONFIRM "Are you sure that you to quit?"; quit$
IF quit$ = "no" THEN WAIT
CLOSE #1
END
```

Создание групп

Группы — это не что иное, как сгруппированные другие управляющие элементы, как показано на рис. 14.8. Группы особенно полезны, если вам необходимо разделить переключатели и флажки, например. Для создания групп используется следующая команда:

```
GROUPBOX #Windowhandle, "Groupbox text", xpos, ypos, ширина, высота
```

Для того чтобы увидеть использование групп в действии, выполните следующую программу:

```
NOMAINWIN
WindowWidth = 250
WindowHeidht = 200
GROUPBOX #1, "Your choices are", 5, 5, 225, 120
RADIOBUTTON #1.radio2, "Radical liberalism", [set], [reset], 10, 30,
        150, 15
RADIOBUTTON #1.radio3, "The status quo", [set], [reset], 10, 60, 150, .
        15
RADIOBUTTON ftl.radio4, "Anything to benefit the rich", [set],
        [reset], 10, 90, 150, 15
BUTTON ftl.buttonl, "Exit program", [quit], LL, 50, 1
```

```
OPEN "Vote for one or more" FOR WINDOW AS #1
PRINT #1, "trapclose [quit]"
WAIT

[set]
WAIT

[quit]
CONFIRM "Are you sure that you to quit?";
        quit$
IF quit$ - "no" THEN WAIT
CLOSE #1
END
```

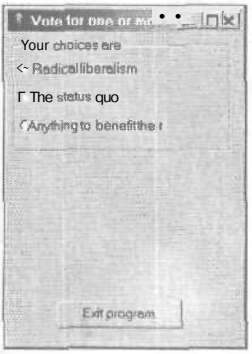

Рис. 14.8. Группы позволяют объединить несколько управляющих элементов, например переключателей

Глава 15

Отладка программ

В этой главе...

- Что такое ошибки программы
- Обнаружение синтаксических ошибок
- Ошибки выполнения программы
- Что такое логические ошибки

Ни один человек не может написать программу, которая будет работать всегда правильно. Это связано с тем, что программа отдает компьютеру четкие подробные инструкции. И если в процессе выполнения программы компьютер получит хотя бы одну неправильную инструкцию, он не будет знать, что делать дальше. Это приведет к ошибке выполнения программы или даже к поломке программы.

При неправильной работе программы программисты говорят: "В программе есть *ошибки"*. Для того чтобы исправить эти ошибки, необходимо провести *отладку* программы.

Независимо от размера программы необходимо найти все ошибки, которые мешают правильной работе. Несмотря на то, что все ошибки в программе убрать нельзя, язык Liberty BASIC (как и другие компиляторы языков) включает в себя средства, позволяющие отследить явные ошибки и ликвидировать их. После этого программа будет работать правильно.

Анатомия ошибки

Ошибки можно разделить на три категории.

- **Синтаксические ошибки.** Ошибка данного типа появляется, когда команду вводят неправильно (например, вместо PRINT написать PRRINT) или ставят слишком много запятых
- **Ошибки выполнения программы.** Данная ошибка появляется во время выполнения программы (например, если необходимо ввести возраст, а пользователь вводит отрицательное число)
- **Логические ошибки.** Данные ошибки появляются в том случае, если написанные инструкции не работают должным образом, а компьютер старается выполнить программу любым образом. При этом результат выполнения программы будет ошибочным

Синтаксические ошибки

Если в правильной команде сделана ошибка, будет добавлен или пропущен какой-либо символ (запятая или пробел), то говорят, что допущена синтаксическая ошибка. Если команда PRINT написана с ошибками, Liberty BASIC выделит строку, в которой есть эта ошибка. Ошибку можно исправить позже. Если программа Liberty BASIC обнаружит синтаксическую ошибку, она выделит строку с ней, а также отобразит сообщение об ошибке в нижнем левом углу окна программы (рис. 15.1).

Несмотря на то, что синтаксические ошибки обычно являются основной причиной неправильной работы профаммы, следите и за своими ошибками при вводе. Это связано с тем, что компьютер будет пытаться выполнить ошибочные инструкции. Если вы ошибетесь при вводе имени переменной, это приведет к некоторым проблемам. Рассмотрим следующую программу:

```
PROMPT "How many times do I need to tell you no"; Answeer$
PRINT "Your reply - "; Answer$
END
```

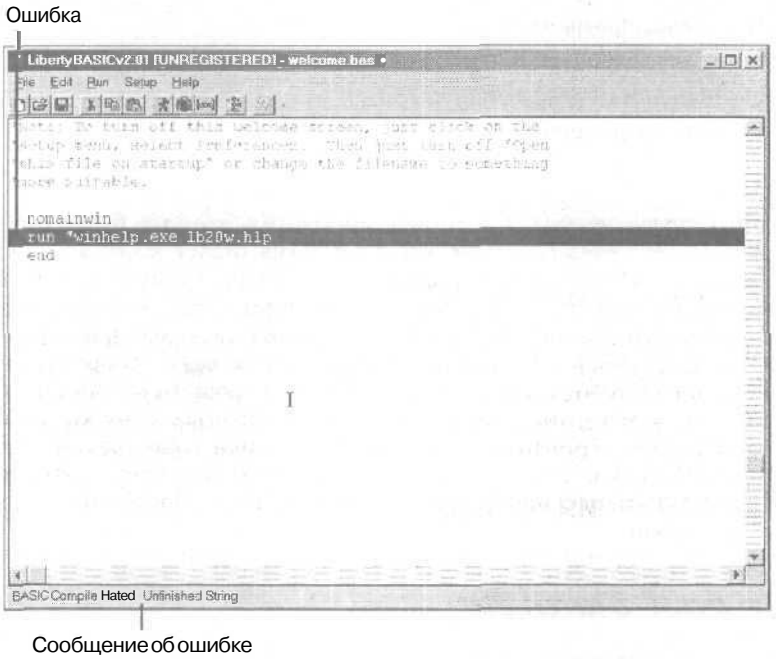

Рис. 15.1. Программа Liberty BASIC выделила строку, в которой есть ошибка

В рассмотренной выше программе задается вопрос: "How many times do I need to tell you no? ("Сколько раз я тебе должен говорить Нет?")". **После этого профамма** сохранит **введенные пользователем сведения в** виде **переменной** Answeer$ **(переменная** Answer$, написанная **с ошибкой). Для программы Liberty BASIC** Answeer$ и Answer$ **— это две разные переменные, поэтому в переменной** Answer$ не будет данных. **После запуска программы сведения, введенные пользователем, не будут напечатаны, так как в профамме есть ошибка.**

Кроме ошибок в названии переменных, необходимо следить за верхним и нижним реестром букв в именах переменных. Для профаммы Liberty BASIC переменная Answer$ отличается от переменной answer$, так как первая переменная начинается со строчной буквы "A", а вторая — с прописной.

Поскольку одна ошибка в имени переменной приводит к неправильной работе программы, многие программисты предпочитают использовать сокращенные имена переменных. Не поступайте так! Те несколько секунд, которые вы сбережете при использовании коротких имен переменных, не стоят времени, потраченного на их расшифровку.

Уничтожение космического зонда Mariner 1

Синтаксические ошибки необходимо обязательно обнаруживать, так как они отдают компьютеру неправильные команды. Многие ошибки обнаруживаются сразу во время выполнения программы, Однако самые серьезные синтаксические ошибки — это те, которые обнаруживаются через определенное время. Программы, содержащие такие ошибки, могут некоторое время работать правильно.

В 1982 году NASA (National Aeronautics and Space Administration — Национальный комитет по аэронавтике и исследованию космического пространства) отправил космический зонд для изучения планеты Венера. Однако ракетоноситель, несущий данный космический зонд, во время полета сбился с курса. и NASA пришлось его взорвать. И все это произошло только из-за того, что в цикле FOR NEXT вместо команды

```
FOR I = 1, 3
```

была команда

```
FOR I = 1. 3.
```

Как видите, в ней вместо запятой поставлена точка.

Вместо того чтобы приказать компьютеру выполнить цикл три раза, команда присвоила переменной / значение 1.3. В результате этой ошибки ракетоноситель получил неправильные инструкции. NASA потерял ракету, на которую были затрачены миллионы долларов, и ее груз.

Такие ошибки очень трудно обнаружить. Если в программе есть ошибки, она будет работать, хотя и неправильно. Если, запустив программу, вы поймете, что она работает неправильно, приступайте к поиску ошибок или лишних знаков, таких как кавычки, запятые, буквы верхнего или нижнего регистров в именах переменных, круглые скобки, а также всего, что приводит к неправильной работе программы.

Ошибки выполнения программы

Ошибки выполнения программы — небольшие ошибки, скрытые в самой программе. Такая программа может работать правильно долгое время до тех пор, пока она не столкнется с непонятным для нее значением (например, если вместо того, чтобы ввести свой возраст, пользователь введет отрицательное число). К сожалению, ошибки выполнения программы можно обнаружить не сразу, а только после того, как они уже приведут к поломке программы.

Именно поэтому все компании, производящие программное обеспечение, проводят тестирование своих программ, выпуская их бета-версии. *Бета-версия* — это версия программы, которая предоставляется специальным людям (бета-тестерам) для обнаружения тех ошибок, которые не найдены специалистами компании.

Ошибки выполнения программы проявляются только после того, как программа получит непонятные для нее данные. Поэтому для обнаружения таких ошибок лучше всего несколько раз запускать программу до тех пор, пока какое-либо значение не приведет к ошибке.

Случаи смерти из-за прибора Therac-25

Therac-25 — это прибор, созданный для нормировки доз радиации, получаемых пациентами во время лечения. Для того чтобы не допустить чрезмерного облучения, в состав данного прибора входило программное обеспечение, контролирующее получаемые пациентами дозы. Но в нем оказалась фатальная ошибка выполнения программы.

Therac-25 работал в двух режимах — рентгеновские лучи и пучок электронов. Если специалист, работающий на данном приборе, сначала выбирал режим работы с рентгеновскими лучами, аппарат Therac-25 переходил в режим работы с высоким уровнем анергии. Если после этого специалист переключался на режим работы с пучком электронов, не отключив при

этом предыдущий режим, прибор начинал выдавать пучок электронов с высоким уровнем энергии. Это приводило к чрезмерному облучению пациентов.

Только после того, как несколько пациентов получили завышенную дозу облучения, в правильной работе прибора возникли серьезные сомнения. Была обнаружена ошибка выполнения программы. Но для пациентов, получивших высокие дозы облучения, было уже слишком поздно.

Например, если программа предлагает ввести возраст, введите большое число (например, 60 000). После этого введите нуль. И под конец введите отрицательное число (например, −9 489). Воспользовавшись таким нехитрым тестом, вы сможете обнаружить ошибки выполнения программы.

Логические ошибки

Изо всех ошибок, которые могут вкрасться в профамму, самые коварные *логические ошибки*. Синтаксические ошибки легко найти. Для этого необходимо внимательно просмотреть профамму и исправить все ошибки. Ошибки выполнения программы также просто исправить с помощью несложных тестов, одни из которых описан раньше.

Однако логические ошибки можно обнаружить только после того, как инструкция написана полностью. Если вы считаете, что инструкция написана правильно, для того чтобы найти и исправить все логические ошибки, необходимо просмотреть каждую строку программы. Таким образом, вы сможете обнаружить, где допущена ошибка, или просто решить задачу по-другому.

Из-за того, что обнаружить логическую ошибку достаточно трудно, программа Liberty BASIC предлагает несколько функций для отладки программы. С их помощью вы легко справитесь с поставленной задачей. Есть два основных способа обнаружения логических ошибок — *выполнение программы в пошаговом режиме* и *трассировка*.

Почему потонул военный корабль Sheffield

При проверке программы на наличие логических ошибок необходимо проверить всю профамму. Несмотря на то, что программа работает прекрасно во время выполнения всех тестов, может возникнуть ситуация, когда она откажется работать.

Подобная ошибка была во время войны за Фолклендские острова между Великобританией и Аргентиной. На британском военном корабле *Sheffield* использовалась компьютерная система защиты корабля от воздушных и ракетных атак. Для того чтобы данная система не атаковала ракеты, принадлежавшие британским вооруженным силам, компьютеры были запрограммированы таким образом, чтобы они не обращали внимание на "дружеские" ракеты. Под определение "дружеских" ракет попали все ракеты, стоявшие на вооружении британского военно-морского флота, включая ракеты, произведенные во Франции.

К несчастью для Великобритании, на вооружении аргентинского военного флота были точно такие же ракеты. И компьютер, отвечающий за безопасность корабля *Sheffield,* воспринимал эти ракеты как "дружеские". После нескольких прямых попаданий корабль затонул.

Выполнение программы в пошаговом режиме

При выполнении программы в пошаговом режиме просматривают каждую строку программы. Благодаря этому можно понять, что в данный момент делает программа. Как только программа сделает что-либо неправильно, вы будете точно знать, на каком этапе выполнения это произошло.

Для того чтобы выполнить программу Liberty BASIC в пошаговом режиме, сделайте следующее.

1. Загрузите программу, которую необходимо выполнить в пошаговом режиме.
2. Выберите комаду Run⇨Dedug (Выполнение⇨Отладка) в строке меню программы Liberty BASIC (или нажмите <Alt+F5>).

Появится окно Debugging (Отладка), показанное на рис. 15.2. Если появится основное окно, можете закрыть его.

3. Щелкните на кнопке Step (Шаг),

После каждого щелчка на кнопке Step (Шаг) программа Liberty BASIC будет переходить к следующей строке программы.

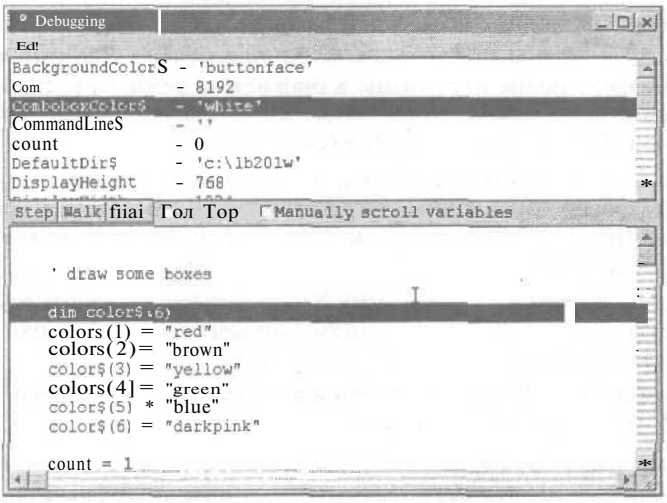

Рис. 15.2. Окно Debugging позволяет обнаружить ошибки в программе

Если вы щелкнете на кнопке Walk (Последовательное выполнение), Liberty BASIC запустит программу. Она будет выполняться очень быстро. Если вы щелкнете на кнопке Run (Выполнить), Liberty BASIC будет выполнять программу, не выделяя выполняемые строки. Если такое выполнение программы слишком быстро для вас, щелкните на кнопке Step (Шаг). Программа будет выполняться пошагово, строка за строкой. Благодаря этому можно пропускать большие куски программы, в правильности которых вы не сомневаетесь.

Для того чтобы понять, каким образом изменится значение переменной после выполнения программы в пошаговом режиме, вставьте несколько команд PRINT в различных местах программы. Например, рассмотрим следующий фрагмент программы:

```
PRINT totalamount
totalamount = (totalamount *salestax) + totalamount
PRINT totalamount
```

В данном примере первая команда PRINT показывает значение переменной totalamount перед ее изменением. Вторая команда PRINT показывает значение переменной totalamount после ее изменения. Если значение переменной totalamount изменится существенным образом при переходе от первой команды PRINT ко второй, будет понятно, что ошибка находится между двумя командами PRINT.

Трассировка программы

Выполнение программы в пошаговом режиме гарантирует скучное времяпрепровождение, особенно если вы догадываетесь, где именно в программе находится ошибка. Вместо того чтобы выполнять каждую строку программы, можно провести трассировку.

Для этого выполните следующее.

1. Откройте программу, отладку которой необходимо провести.
2. Введите команду TRACE в том месте программы, с которого необходимо начать трассировку.

   При использовании команды TRASE можно воспользоваться следующими тремя параметрами.

   - TRASE 0 останавливает выполнение программы. Для того чтобы возобновить выполнение программы в пошаговом режиме, воспользуйтесь кнопками Step (Шаг), Walk (Последовательное выполнение) или Run (Выполнить) в окне Debugging (Отладка).
   - TRASE 1 запускает программу в пошаговом режиме. При этом вмешательство пользователя не обязательно. Использование данной команды аналогично действию кнопки Walk (Последовательное выполнение) окна Debugging (Отладка)..
   - TRASE 2 запускает программу без отображения исполняемых строк. Использование данной команды аналогично действию кнопки Run (Выполнение) окна Debugging (Отладка).
3. Выберите Run⇨Debug (Выполнение⇨Отладка) из строки меню Liberty BASIC (или нажмите <Alt+F5>).

   Появится окно Debugging (Отладка).
4. Щелкните на кнопке Walk (Последовательное выполнение).

   Программа Liberty BASIC запустит программу с начала и будет выполнять ее до тех пор, пока не дойдет ло первой команды TRASE 0. Затем программа выделит команду TRASE 0. С этого момента можно просматривать программу в пошаговом режиме. Для этого необходимо щелкнуть на кнопке Step (Шаг). Для того чтобы перейти к оставшейся части программы, используйте кнопки Walk (Последовательное выполнение) или Run (Выполнить).

Команду TRASE можно размешать в любом месте программы. Данная команда не влияет на работу самой профаммы. Запущенная программа просто игнорирует команду TRASE. Используйте команду TRASE только в том случае, если действительно необходимо отладить программу.

Часть IV

Знакомство со структурами данных

"Это новый текстовый процессор. Он позволяет не только проверять грамматику, но и политическую корректность, религиозную направленность текста и многое другое..."

В этой части...

Какую программу вы ни написали бы, ей обязательно понадобится сохранять данные в памяти компьютера. Конечно, программа не может просто забрасывать туда данные, ведь подобный подход быстро приведет к неразберихе.

Для упрощения организации памяти компьютера программы используют *структуры данных*. Структура данных — это просто способ упорядочения данных, позволяющий при необходимости легко получать необходимые сведения. Простейшей структурой данных является обычная переменная. К более сложным относятся массивы, записи, связанные списки и объекты. В настоящей части мы рассмотрим их.

Глава 16

Сохранение информации в массивах

В этой главе...

- Создание массивов
- Использование массивов для хранения данных
- Создание многомерных массивов
- Создание динамических массивов

Когда вы собираетесь сохранить данные, можно воспользоваться переменными с описательными именами, например `Номер_телефона`, `Название_фильма` или Имя. После создания переменной необходимо определить, какой тип данных будет в ней храниться — строка, целое число или число с одинарной точностью.

Однако иногда необходимо сохранить список схожих между собой данных. Например, если вы написали программу для сохранения имен всех учеников в классе, вам понадобится создать несколько похожих переменных для хранения этих имен, как показано в следующем примере:

```
Name1$ = "Patrick DeGuire"
Name2$ - "Jordan Preston Wang"
Name3$ = "Michael Elizondo "
Name4$ = "Bo the Cat"
```

В действительности такой способ несколько неудобен, так как программа получается загроможденной. Для того чтобы решить эту проблему, программисты предложили использовать *массивы*. Массив можно представить как список (пример одного из массивов приведен на рис. 16.1).

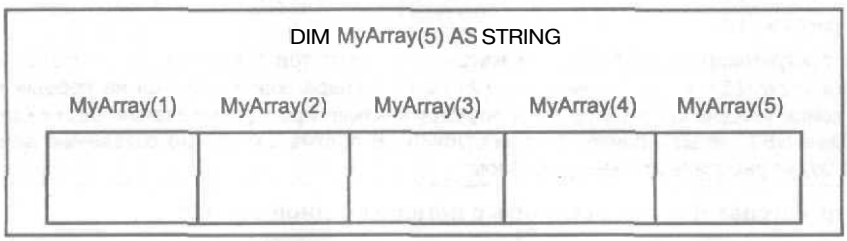

Рис. 16.1. В массиве каждому элементу присвоено свое имя

Создание массива

Обычная переменная может хранить только одну порцию данных, например число или имя. При попытке сохранения в данной переменной новой порции данных, старые данные заменяются новыми.

В отличие от обычной переменной, в массиве могут храниться несколько порций данных. Они будут храниться там до тех пор, пока каждая порция данных не заменит-

ся на данные того же типа, например строку, целое число или число с одинарной точностью. Для того чтобы создать массив, его необходимо определить. Для этого используется команда DIM:

```
DIM ArrayName (Number)
```

В приведенном выше примере ArrayName — это любое правильное имя переменной, а Number — соответствует общему числу элементов, которые необходимо сохранить в массиве.

Как и в других переменных, в массиве могут храниться либо цифры, либо строки. Для того чтобы указать, что в массиве будут храниться строки, необходимо добавить знак доллара ($) в конце имени массива:

```
DIM ArrayName$ (Number)
```

Массив состоит из следующих трех частей.

- ✓ Имя
- ✓ Тип данных. В массиве могут храниться только данные определенного типа, например строки или цифры
- ✓ Количество элементов, которые хранятся в массиве. (В единичном элементе может храниться только одна порция данных.) Иногда число, определяющее количество элементов, называют *индексом массива*

Как определить массивы в языках программирования C/C++, Java и Visual Basic.NET

При использовании **таких** языков программирования, **как Liberty BASIC и Pascal, для определения размера массива берут цифры,** начиная **с** единицы. **Например, в языке** программирования Liberty **BASIC можно определить массив следующим образом:**

```
DIM AgeArray(3)
```

В языке программирования Liberty BASIC элемент `AgeArray` **содержит три элемента:** `AgeArray(1)`, `AgeArray(2)` и `AgeArray(3)`.

Однако при **использовании языков программирования** C/C++, **Java или** Visual Basic.NET **для определения массивов берут** цифры, начиная **с нуля. Например, в языке программирования С определить массив можно следующим образом:**

```
int agearray(3)
```

В языке программирования С данный массив содержит три элемента: `int agearray(0)`, `int agearray(1)` и `int agearray(2)`, **а** не **четыре, как покажется на** первый взгляд. **Если в конце концов вы решитесь использовать языки** программирования C/C++, Java или Visual **Basic.NET, не забывайте об этом отличии. В противном случае созданные** вами **массивы не будут работать** должным образом.

Размер массива можно определить с помощью одной строки:

```
DIM CatArray$ (45)
```

В данной строке создается массив, который содержит 45 элементов, начиная с первого — `CatArray$(1)` и заканчивая последним `CatArray$(45)`.

Сохранение (и удаление) данных в массиве

После создания массива он пустой. Поскольку массив может содержать несколько порций данных, необходимо определить место в массиве, в котором они будут храниться.

Предположим, вы создали массив, в котором содержатся пять различных целых чисел:

```
DIM IQArray$(5)
```

Обычно для того чтобы поместить число 93 в переменную, используется следующая команда:

```
MyVariable = 93
```

Однако из-за того, что в массиве содержатся несколько порций данных, необходимо задать точно место в массиве, в котором они будут размещены. Если вы хотите сохранить данные в третьем элементе массива, воспользуйтесь следующей командой:

```
;IQArray(3) = 93
```

Данная команда приказывает компьютеру сохранить число 93 в третьем элементе массива IQArray.

Если вы попытаетесь сохранить данные в месте, в котором уже находятся данные, программа Liberty BASIC просто сотрет старые данные и заменит их новыми.

Для того чтобы убрать данные из определенного места массива, необходимо указать их точное расположение:

```
YourIQ = IQArray (3)
```

Если в третьем элементе массива IQArray сохранено число 93, значение YourIQ также будет равно 93.

В массиве можно сохранять несколько порций данных. Поэтому программисты используют циклы для того, чтобы легче сохранять и удалять данные в массивах. Без циклов придется задавать точное место в массиве, в котором сохраняются (или из которого удаляются) данные, как показано в следующем примере:

```
NameArray(l) = "Mike Ross"
NameArray(2) = "Bill McPherson"
NameArray(3) = "John Smith"
```

Однако можно задать расположение в массиве, воспользовавшись командой WHILE-WEND или циклом FOR NEXT, как показано в следующем примере:

```
FOR I = 1 TO 3
  NumberArray(I) = 125
NEXT I
```

Цикл FOR NEXT сохраняет число 125 в первом, втором и третьем элементе массива NumberArray.

Для того чтобы понять, каким образом в массиве сохраняются данные (или удаляются), попытайтесь запустить следующую программу:

```
DIN Name Array$(3)
 FOR I - 1 TO 3
```

```
  PROMPT "Type the name of someone you hate:"; Enemy$
  NameArray$(I) = Enemy$
NEXT I
FOR I = 1 TO 3
PRINT NameArray$(I) + " sound like the name of a moron."
NEXT I
END
```

Ниже описано, что делает каждая строка программы.

1. Первая строка создает массив NameArray, который содержит три различные строки.
2. Вторая строка запускает цикл FOR NEXT, который выполняется три раза.
3. Третья строка отображает на экране диалоговое окно Prompt, в котором пользователю предлагается ввести имя своего врага. Введенное пользователем имя будет помещено в переменную Enemy$.
4. Четвертая строка говорит компьютеру, что необходимо сохранить значение переменной Enemy$ в массиве NameArray. Сначала, после запуска цикла FOR NEXT, значение переменной EnemyS сохранится в качестве элемента NameArray(1). Потом значение переменной EnemyS сохранится в качестве элемента NameArray(2). И наконец, значение переменной EnemyS будет сохранено в качестве элемента NameArray(3).
5. Пятая строка завершает выполнение цикла FOR NEXT,
6. Шестая строка запускает цикл FOR NEXT, который выполняется три раза.
7. Седьмая строка печатает значение NameArray, а также строку: sound like the name of a moron. (звучит достаточно глупо.). Сначала печатается значение, сохраненное в качестве элемента NameArray$(1), затем значение, сохраненное в качестве элемента NameArray(2), а в конце — значение, сохраненное в NameArray(3).
8. Восьмая строка завершает выполнение цикла FOR NEXT.
9. Девятая строка говорит компьютеру, что программа завершила работу.

Создание многомерных массивов

Простейший массив содержит не более чем один список элементов, как показано в следующем примере:

```
DIM PetArray$(5)
```

Данная команда создает массив, в котором содержится пять строк (см. рис. 16.1). Массив, содержащий один список данных, называется *одномерным массивом*. Для задания размера одномерного массива используется одно число, как показано в следующем примере:

```
DIM ArrayName(X)
```

Liberty BASIC также позволяет создавать двумерные массивы, положение элементов в которых задается двумя числами. Программа для создания двумерного массива выглядит следующим образом:

```
DIM ArrayName(X, Y)
```

С помошью данной команды создается двумерный массив, показанный на рис. 16.2.

Ниже приведен пример двумерного массива:

```
DIM VictimArray(10, 9) AS STRING
```

С помощью данной команды создается двумерный массив, состоящий из 90 ячеек (10x9), предназначенных для хранения строковых элементов.

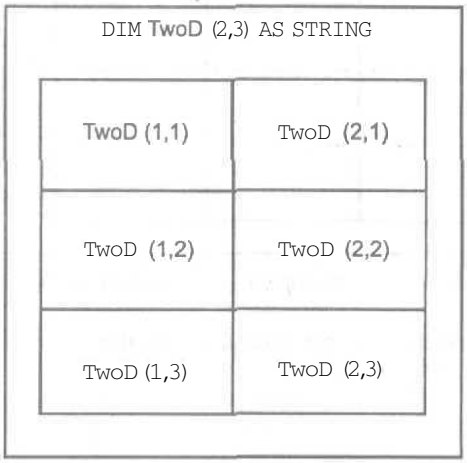

Рис. 16.2. Для задания размера двумерного массива используется два числа

Язык программирования Liberty BASIC поддерживает использование только одномерных и двумерных массивов. Другие языки программирования позволяют создавать многомерные массивы. Для того чтобы в другом языке программирования (например, Visual Basic) создать трехмерный массив, необходимо задать три размера. Пример трехмерного массива приведен в следующем примере (также обратите внимание на рис. 16.3):

```
DIM ArrayName(X, Y, Z)
```

Для того чтобы сохранить или убрать данные в многомерном массиве, необходимо задать три числа, которые определяют точное расположение в массиве.

Для того чтобы создать массив, состоящий из шести строк, воспользуйтесь следующей программой:

```
DIM VictimArray$(2, 3) AS STRING
FOR I = 1 TO 2
  FOR J - 1 TO 3
  PROMPT "Who do you want to hurt";   Enemy$
    VictimArray$(I, J) = Enemy$
  NEXT J
NEXT I
PROMPT "Type X location of the array item that you want to print,
       such as 1"; X
PROMPT "Type Y location of the array item that you want to print,
       such as 1"; Y
```

```
PRINT VictimArray$(X, Y) + "deserves to be hurt the most,"
END
```

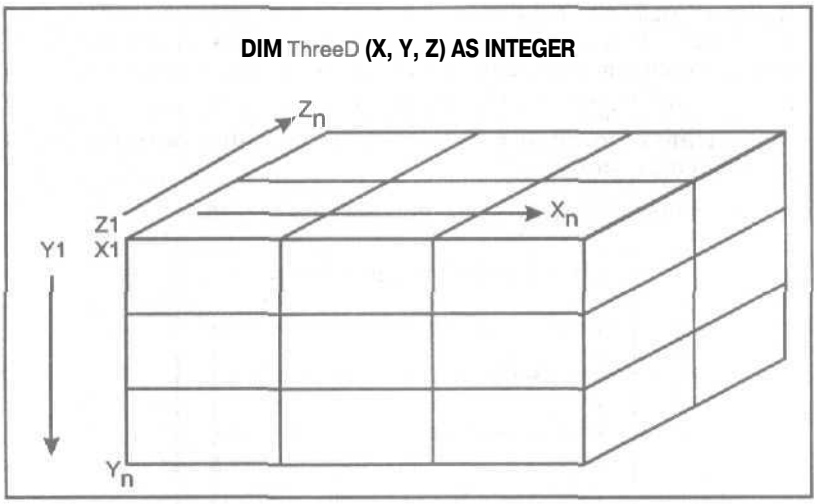

Рис. 16.3. Для задания размера трехмерного массива необходимо три числа

Данная программа предлагает вам ввести шесть имен. Предположим, что вы введете шесть имен в следующем порядке:

```
Mike Ross
Bill McPherson
Jon Markey
Bobby Lee
Tom Clark
Roger Smith
```

Программа сохранит имена в том порядке, в котором вы их ввели:

```
VictimArray$(1, 1) = Mike Ross
VictimArray$(1, 2) = Bill McPherson
VictimArray$(1, 3) = Jon Markey
VictimArray$(2, 1) = Bobby Lee
VictimArray$(2, 2) = Tom Clark
VictimArray$(2, 3) = Roger Smith
```

Если программа выдаст сообщение "Type X and Y location of the array item that you want to print, such as 1, 3" и вы введете **значение 2,1, программа напечатает следующее** сообщение:

```
"Bobby Lee deserves to be hurt the most."
```

Из-за того что сохранение и удаление данных в многомерных массивах не всегда просто, используйте их только в случае крайней необходимости.

Создание динамических массивов

В большинстве языков программирования можно создавать только *статические массивы*. Статический массив характеризуется постоянным размером. Использование статических массивов приводит к следующим проблемам.

- ✓ После создания массива определенного размера в дальнейшем нельзя изменить этот размер (если вы создали слишком маленький или слишком большой массив)
- ✓ Статический массив всегда занимает определенный объем памяти независимо от того, есть в нем данные или нет

Несмотря на эти проблемы, статические массивы подходят для большинства программ. Однако если необходимо изменять размер массива или убрать из него данные (для освобождения памяти), лучше создать *динамический массив*.

Основное преимущество динамического массива состоит в том, что можно изменять его размер и удалять из него данные в процессе работы программы. К недостаткам можно отнести то, что для выполнения любой из перечисленных выше операций необходимо ввести специальную команду.

В отличие от некоторых языков программирования (таких как старая версия языка программирования BASIC — GW-BASIC), язык программирования Liberty BASIC позволяет создавать динамические массивы. Создав массив, вы всегда сможете изменить его размер. Для того чтобы изменить размер массива, воспользуйтесь командой REDIM, как показано в следующем примере:

```
REDIM VictimArray$(44)
```

Эта команда приказывает компьютеру изменить размер массива VictimArray$ таким образом, чтобы в нем содержалось 44 элемента.

Изменение размера массива в Liberty BASIC приведет к уничтожению всех сохраненных в массиве данных.

Изменение размера динамического массива позволяет изменять размер массива, но не тип данных. Если в первоначальном массиве сохранены строки, в массиве с измененным размером также будут храниться строки.

После изменения размера массива в нем можно разместить новые данные. Для того чтобы понять, каким образом работает динамический массив, воспользуйтесь следующей программой:

```
DIM LoserArray$(3)
FOR I = 1 TO 3
  PROMPT "Who is incompetent?"; MyBoss$
  LoserArray&(I) = MyBoss$
NEXT I
FOR J - 1 TO 3
  PRINT LoserArray$(J)
NEXT J
REDIM LoserArray$(7)
LoserArray$(7) = "Bobby Lee"
PRINT LoserArray$(7) + " is a pathetic character."
END
```

Ниже описано, что делает каждая строка программы.

1. Первая строка создает массив LoserArray, который содержит три различных строковых элемента.
2. Вторая строка запускает цикл FOR NEXT, который выполняется три раза.
3. Третья строка отображает на экране диалоговое окно Prompt, в котором пользователю задается вопрос Who is incompetent? (Кто больше всего не компетентен?). Любое введенное пользователем имя будет сохранено в качестве значения переменной MyBoss$.
4. Четвертая строка приказывает компьютеру сохранить значение переменной MyBoss$ в массиве LoserArray.
5. Пятая строка останавливает выполнение цикла FOR NEXT.
6. Шестая строка запускает цикл FOR NEXT, который выполняется три раза.
7. Седьмая строка печатает содержимое массива LoserArray. После первого выполнения цикла FOR NEXT будет напечатано первое введенное вами имя, после второго выполнения цикла — второе имя, а поле третьего выполнения — третье введенное вами имя. После этого в массиве LoserArray будет три элемента.
8. Восьмая строка останавливает выполнение цикла FOR NEXT.
9. Девятая строка удаляет все содержимое массива LoserArray и изменяет размер этого массива таким образом, что в нем содержатся семь (7) элементов.
10. Десятая строка сохраняет строку Bobby Lee в качестве седьмого элемента массива LoserArray.
11. Одиннадцатая строка печатает содержимое массива LoserArray(7), в котором находится строка Bobby Lee, и соединяет ее со строкой is a pathetic character. (достаточно жалкий персонаж.). В результате отображается сообщение Bobby Lee is a pathetic character. (Бобби Ли достаточно жалкий персонаж.).
12. Двенадцатая строка приказывает компьютеру завершить выполнение программы.

Можно изменить размер многомерного размера массива, но нельзя изменить его размерность. Создав двумерный массив, нельзя сделать из него одномерный.

Глава 17

Сохранение связанных данных в виде записи

В этой главе...

- Создание записей
- Добавление и извлечение данных из записи
- Комбинация записей и массивов

С помощью массивов удобно сохранять данные одного типа (целое число или строка) в виде списка. Однако некоторые данные необходимо сохранять в массивах, в которых каждому элементу соответствует определенная строка и номер. Из-за того что в одном и том же массиве нельзя сохранять данные различных типов (более подробная информация о массивах приведена в главе 16), необходимо использовать другую структуру данных, которая называется *записью.*

Структура данных ~ это выстроенные в определенном порядке элементы, содержащие определенную информацию (слова или цифры). К простейшей структуре данных можно отнести переменную, которая содержит небольшой набор данных. Массив — это более сложная структура данных. В нем содержится список данных одного типа.

Запись сохраняет связанные данные под одним именем переменной. Например, если вы захотите сохранить имена, адреса и телефонные номера всех своих друзей (или врагов), можно создать несколько различных переменных, как показано в следующем примере:

```
Name1$ = "Bo Katz"
Address1$ = "123 Main Street"
Phone1$ = "555-1234"
Salary1 = 55000
Name2$ = "Roger Wilco"
Address2$ = "948 Manchester Road"
Phone2$ = "555-4587"
Salary2 = 29000
```

Чем больше имен, адресов и телефонных номеров необходимо сохранить, тем больше различных переменных нужно создать. Так как создание переменных — достаточно сложная операция, можно создать записи, которые упрощают сохранение связанных данных под одним именем переменной. Запись позволяет группировать связанные данные и оперировать ими. При этом вам не придется работать с отдельными данными, такими как имена, адреса, телефонные номера или размер зарплаты.

Программа Liberty BASIC не поддерживает использование записей. Поэтому в настоящей главе в качестве примеров я использую программы, написанные на языке программирования QBASIC. Не пытайтесь запускать их. Просто разберитесь в них и попытайтесь понять основные принципы их работы.

Создание записи

Запись состоит из имени и одной или нескольких переменных.

Например, если необходимо создать запись для сохранения имен, адресов и телефонных номеров, воспользуйтесь следующей программой:

```
TYPE RecordName
  FullName AS STRING * 15
  Address AS STRING * 25
  Phone AS STRING * 14
  Salary AS SINGLE
END TYPE
```

Данная запись приказывает компьютеру сделать следующее.

1. Первая строка говорит компьютеру: "Начинается запись RecordName".
2. Во второй строке создается переменная FullName, которая может содержать до **15** символов.
3. В третьей строке создается переменная Address, которая может содержать до 25 символов.
4. В четвертой строке создается переменная Phone, которая может содержать до 14 символов.
5. В пятой строке создается переменная Salary, которая может содержать число с одинарной точностью.
6. Шестая строка говорит компьютеру: "Заканчивается запись RecordName".

Создав первую запись, вы не сможете тут же воспользоваться ею. Говоря техническим языком, запись — это *определенный пользователем тип данных.* Перед использованием данных необходимо создать переменную, содержащую информацию, которую нужно сохранить в записи. Это же делается при создании переменной, в которой хранится целое число или строка.

В следующем примере показано, что для того, чтобы воспользоваться записью, сначала необходимо определить ее, а затем создать переменную:

```
TYPE EmployeeRecord
  FullName AS STRING * 15
  Address AS STRING * 25
  Phone AS STRING * 14
  Salary AS SINGLE
END TYPE
DIM Workers AS EmployeeRecord
```

В рассмотренном примере команда DIM говорит компьютеру; "Создай переменную Workers, в которой содержатся данные, определенные записью EmployeeRecord".

Только после того, как задана переменная, определяющая запись, можно помещать данные в запись.

Управление данными в записях

Для того чтобы добавить и извлечь данные из записи, необходимо задать два связующих элемента.

- ✓ Имя переменной, которая определяет запись
- ✓ Имя переменной записи, в которой хранятся данные или из которой данные извлекаются

Сохранение данных в записи

Для сохранения данных в записи необходимо задать имя переменной (она определяет запись) и специальную переменную записи, как показано в следующем примере:

```
RecordVariable.Variable = Data
```

Предположим, что запись определена так:

```
TYPE BomberInfo
  NickName AS STRING * 16
  MissijnsFlown AS INTEGER
  Crew AS INTEGER
END TYPE
```

Переменная, содержащаяся в данной записи, определяется следующим образом:

```
DIM B17 AS Bomberlnfo
```

Если необходимо сохранить сроку в переменной NickName, воспользуйтесь следующей командой:

```
B17.NickName = "Brennan`s Circus"
```

Данная команда говорит компьютеру: "Посмотри на переменную B17 и сохрани строку Brennan в переменной NickName".

Извлечение данных из записи

Для извлечения данных из записи необходимо указать имена записи и соответствующей переменной:

```
Variable = RecordVariable.Variable
```

Предположим, есть определенная запись и соответствующая ей переменная:

```
TYPE Bomberlnfo
  NickName AS STRING * 15
  MissijnsFlown AS INTEGER
  Crew AS INTEGER
END TYPE
DIM B24 AS BomberInfo
```

*** Как создаются записи на языке С**

Различные языки программирования по-разному создают аналогичные структуры данных. На языке программирования С записи (они в языке С называются *структурой*) создаются следующим образом:

```
struct bomberinfo {
char nickname[15]
int  missionsflown
int  crew
} B24;
```

С помощью этой небольшой программы создается структура bomberinfo. Структура bomberinfo может сохранять псевдоним, содержащий до 15 символов, целое число в переменной missionsflown и другое целое число в переменной crew. Кроме того, с помощью данной программы создается переменная B24, которая представляет данную структуру.

Если в записи B24 уже сохранены данные, их можно извлечь с помощью следующей команды:

```
GetName = B24.NickName
'. GetHistory = B24. MissijnsFlown
GetCrew = B24.Crew
```

Использование записей в массивах

Несмотря на то, что в записи хранится несколько связанных фрагментов данных под одним и тем же именем, записи сами по себе не очень подходят для хранения списков связанных данных, таких как имена, адреса и телефонные номера. Для того чтобы решить эту проблему, создайте массив, содержащий запись, используя следующую команду:

```
DIM ArrayName(Number) AS RecordType
```

Более подробная информация о массивах приведена в главе 16.

Поскольку массивы — это просто список определенных типов данных, можно определить массив следующим образом:

```
TYPE GulliblePeople
   FullName AS STRING * 15
   CashAvailable AS SINGLE
END TYPE
DIM PotemtialVictims(3)AS GulliblePeople
```

Данный фрагмент программы говорит, что компьютеру необходимо сделать следующее.

1. **Первая строка** создает **запись** GullliblePeople.
2. Вторая строка создает строковую переменную FullName, которая может содержать до15 символов.
3. **Третья строка создает переменную** CashAvailable, **которая может** содержать **число с одинарной точностью.**
4. Четвертая строка заканчивает запись GullliblePeople.
5. Пятая строка создает массив, который может содержать три записи, размещенные в записи GullliblePeople. (Чтобы познакомиться с этим массивом, обратитесь к главе 16, рис. 16.1.)

В следующем примере приведена программа, написанная на языке программирования QBASIC. Это пример реально работающей программы для создания записей.

```
TYPE GulliblePeople
  FullName AS STRING * 15
  CashAvailable AS SINGLE
END TYPE
DIM PotemtialVictims(2) AS GulliblePeople

FOR I * 1 TO 2
  PRINT "Type the name of someone you want to con:"
  INPUT PotemtialVictims(I).FullName
  PRINT "How much money do think you can get?"
  INPUT PotemtialVictims(I).CashAvailable
NEXT I
PRINT
PRINT "Here is your list of future con victims:"

FOR I = 1 TO 2
  PRINT PotemtialVictims(I).FullName
  PRINT PotemtialVictims(I).CashAvailable
NEXT I
END
```

Эта программа говорит, что введено два имени и два числа. Первое имя сохранено **в переменной** PotemtialVictims(1).FullName, **а первое** число — **в качестве значения переменной** PotemtialVictims(1).CashAvailable. **Второе имя сохранено в качестве значения** переменной PotemtialVictims(2).FullName, **а второе** число — **в качестве значения переменной** PotemtialVictims(2).CashAvailable.

Глава 18

Связанные списки и указатели

В этой главе...

- Указание на данные
- Как работают связанные списки
- Создание связанных списков
- Создание структур данных с помощью связанных списков

При создании массива вы должны указать его точный размер. Если массив будет слишком маленьким, программе просто не хватит места для сохранения всех данных. Однако, если массив слишком велик, вы просто зря потратите такую ценную память компьютера.

В качестве компромиссного варианта программисты придумали *связанные списки*. Точно так же как и массив, связанный список может содержать целый список элементов. Однако, в отличие от массива, связанный список при необходимости можно увеличивать или уменьшать, поэтому вам не приходится думать о том, сколько памяти выделить связанному списку при проектировании программы.

Некоторые языки программирования, в частности BASIC, не позволяют создавать связанные списки. Хотя вы не сможете создавать или использовать связанные списки при работе с Liberty BASIC, вам следует знать об их существовании. Ведь связанные списки представляют собой достаточно распространенную структуру данных для некоторых языков программирования, например Java и C/C++.

Начнем с указателя

Массив предоставляет для хранения данных строго определенное число ячеек. Если вы создали массив на пять ячеек, но который будет содержать максимум два элемента, вы просто зря потратите память компьютера. При использовании массивов небольшого размера эта проблема не столь серьезна. Но если вы используете большие многомерные массивы, занимаемый ими огромный объем памяти не позволит программе работать должным образом на компьютерах, оснащенных недостаточным объемом памяти.

Еще одна проблема состоит в том, что вам непросто переупорядочивать данные в массиве. Например, предположим, у вас массив, содержащий список имен, как показано на рис. 18.1. Если вы захотите упорядочить их в алфавитном порядке, вы должны извлечь имена из массива и снова поместить их туда, но уже согласно алфавиту.

Для решения обеих проблем программисты создали связанные списки. Связанный список может увеличиваться или уменьшаться в зависимости от того, какой объем данных вы в нем сохраняете, поэтому использует память компьютера намного эффективнее, чем массив.

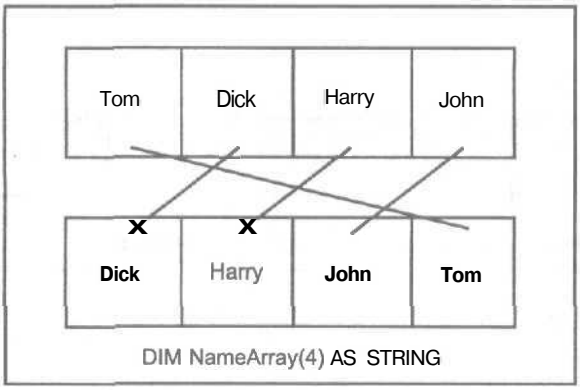

Рис. 18.1. Для того *чтобы переупорядочить данные в массиве, массив сначала необходимо опустошить, а затем снова наполнить данными*

Кроме того, массив можно представить в виде большой коробки, разделенной на секции для хранения отдельных элементов данных, а связанный список можно представить в виде отдельных коробок (называемых *узлами),* которые вы легко объединяете, как показано на рис. 18.2.

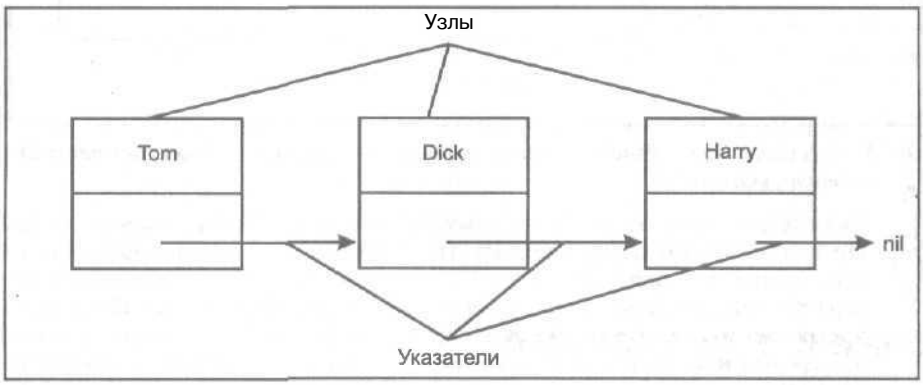

Рис. 18.2. Связанный список состоит из узлов, каждый из которых содержит данные и указатель на следующий узел в списке

Указатель в последнем узле связанного списка обозначает специальное значение, которые называется *нулем* (nil) и представляет собой конец списка.

В отличие от массивов для переупорядочения данных, в связанном списке его не нужно сначала опустошать, а затем снова наполнять. Для этого достаточно переупорядочить указатели, как показано на рис. 18.3.

Хотя переменные обычно содержат строки или цифры, указатель содержит *адрес памяти,* который во многом напоминает обычный почтовый адрес. Считывая сведения об адресе памяти, компьютер получает возможность определить, в каком узле связанного списка содержатся необходимые данные.

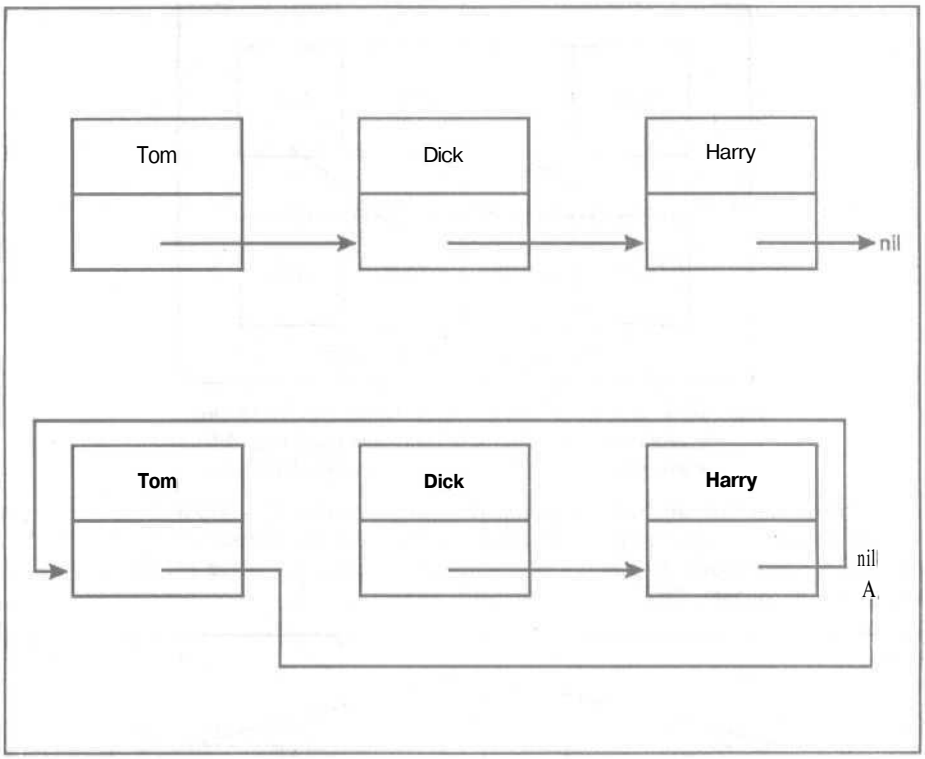

Рис. 18.3. Для переупорядочения данных в связанном списке достаточно переупорядочить указатели

Указатели играют одну из основных ролей при программировании на C/C++. Поскольку указатель представляет адрес памяти, вы можете совершенно запутать программу, написанную на C/C++, если всего один указатель будет содержать неправильный адрес памяти. При возникновении подобной ситуации программа получает возможность обратиться напрямую к памяти компьютера, что сродни исследованию вашего мозга с помощью длинной острой иглы. Неправильное использование указателей (как и неправильное вмешательство в головной мозг) внесет неразбериху в память компьютера, а значит, приведет к сбоям в работе компьютера или его зависанию.

Составные части связанного списка

Связанный список состоит из нескольких узлов, каждый из которых содержит следующее.

- ✓ Указатель на следующий узел связанного списка
- ✓ Запись для сохранения данных в связанном списке

Liberty BASIC не позволяет вам создавать указатели, поэтому все дальнейшие примеры программ приведены для языка программирования Pascal, который достаточно прост для понимания, поскольку создан в учебных целях. Если тема связанных списков кажется вам слишком сложной, изучите фрагменты программ, написанных на Pascal, а также рисунки, приведенные в настоящей главе, чтобы понять, как же работают связанные списки, хотя бы в общих чертах.

Для того чтобы создать связанный список с помощью языка программирования Pascal, вы должны сначала создать указатель, как показано ниже:

```
TYPE
   PointerName = ^RecordType;
```

Эта команда приказывает компьютеру создать тип данных PointerName. Этот тип данных может содержать адреса памяти, определяющие расположение тип записи RecordType.

После создания указателя на запись вы должны создать собственно запись. Например, следующая запись содержит две переменные типа string — FullName и Address, а также переменную Next типа pointer:

```
TYPE
   PointerName = ^RecordType;
   RecordType = RECORD
   FullName : string[15];
   Address : string[25];
   Next : PointerName;
 - END
```

Определив указатель и запись, задайте переменную, которая эту запись представляла бы. Эта переменная создает каждый узел связанного списка, как показано ниже:

```
VAR
   Node : PointerName;
```

Объединив все эти фрагменты кода, вы получите следующую программу:

```
PROGRAM LinkedLists;
TYPE
   PointerName = ^RecordType;
   RecordType = RECORD
   FullName : string[15];
   Address : string[25];
   Next : PointerName;
   END
VAR
   Node : PointerName;
```

Эта программа, написанная на Pascal, приказывает компьютеру выполнить следующее.

1. Первая строка сообщает компьютеру о том, что программа называется LinkedLists.
2. Вторая строка, которая начинается со слова TYPE, сообщает компьютеру о том, что будут определены указатель и запись.
3. Третья строка определяет тип данных PointerName, указывающий на любую запись типа RecordType.
4. Четвертая строка определяет имя записи типа RecordType.
5. Пятая строка приказывает компьютеру создать переменную FullName, которая может содержать до 15 символов.
6. Шестая строка приказывает компьютеру создать переменную Address, которая может содержать до 25 символов.

7. Седьмая строка приказывает компьютеру создать переменную Next, которая будет указывать на следующую запись типа RecordType.
8. Восьмая строка сообщает компьютеру о завершении определения записи типа RecordType.
9. Девятая строка, которая начинается со слова VAR, сообщает компьютеру о том, что определены одна или несколько переменных.
10. Десятая строка приказывает компьютеру создать переменную Node, которая будет представлять указатель на запись типа RecordType. (Эта переменная-указатель определяет один узел в связанном списке.)

Не переживайте, если связанные списки кажутся вам чем-то загадочным. О том, как работают связанные списки, вы получите достаточное представление, если изучите последующие рисунки настоящей главы.

Создание связанного списка

После того как вы определили запись для хранения данных, указатель для этой записи, а также переменную, представляющую каждый узел записи, вам необходимо создать связанный список. Для создания списка вам необходимо выполнить следующие шаги, которые проиллюстрированы на рис. 18.4.

1. Создание узла.
2. Сохранение данных в узле.
3. Упорядочение указателей при вставке нового узла в начале, середине или конце связанного списка.

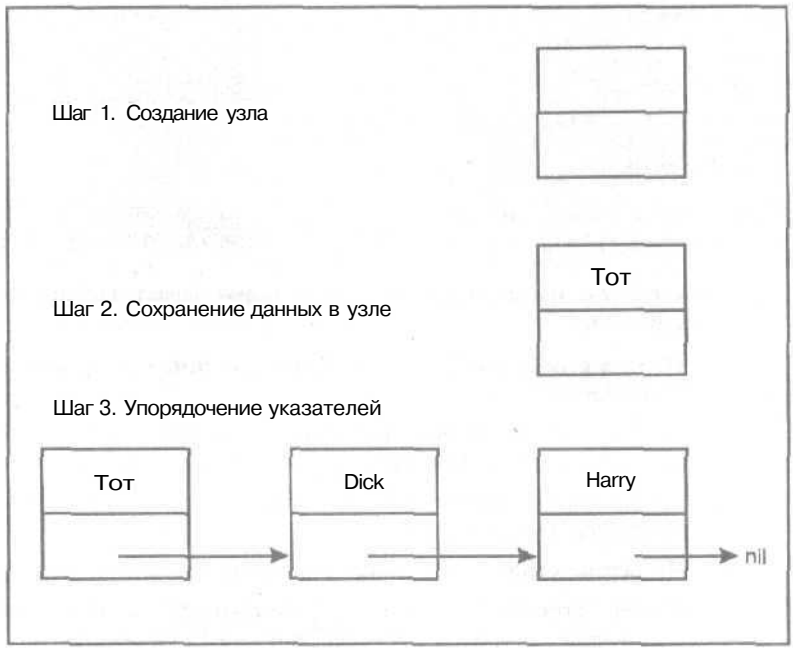

Рис. 18.4. Для создания связанного списка необходимо выполнить эти три шага

Чтобы понять, как написанная на Pascal программа создает узел и сохраняет в нем данные, изучите следующий фрагмент кода программы:

```
PROGRAM LinkedLists;
TYPE
  PointerName = ^RecordType;
  RecordType = RECORD
  FullName : string[15];
  Address : string[25];
  Next : PointerName;
  END
VAR
  Node : PointerKarne;
BEGIN
  New(Node);
  Node^.FullName : = 'Jonathan Blake';
  Node^.Address := '837 Dead-End Avenue';
  Node^.Next := nil;
END.
```

Начиная с команды BEGIN эта программа приказывает компьютеру выполнить следующее.

Если вы хотите разобраться с кодом программы, находящимся до слова BEGIN, обратитесь к подразделу "Составные части связанного списка".

1. Первая строка, которая начинается со слова BEGIN, сообщает компьютеру о начале ряда инструкций, которые ему необходимо выполнить.
2. Вторая строка создает новый узел. При этом узел не содержит никакой информации.
3. Третья строка сохраняет имя Jonathan Blake в качестве значения переменной Fullname записи узла.
4. Четвертая строка сохраняет адрес 837 Dead-Snd Avenue в качестве значения переменной Address записи узла.
5. Пятая строка присваивает значение nil указателю Next. Это значение говорит о том, что указатель, собственно, ни на что не указывает. Использование значения nil предотвращает такую ситуацию, когда указатель указывает на какую-то другую ячейку памяти. Если вы создадите второй узел, указатель Node^. Next будет указывать именно на него.

Управление связанным списком

Если вы сохранили данные в массиве, после чего удалили какие-то данные в его середине, на месте этих данных образуется пустое пространство. Хотя в этой части массива не будет никакой информации, компьютеру все равно необходимо выделять ей определенный объем памяти. Хуже того, при поиске данных вам обязательно придется помнить о подобной пустой "ловушке", как показано на рис. 18.5.

Удаление данных в связанном списке проходит намного проще — для этого необходимо выполнить всего два действия.

1. Переупорядочение указателей таким образом, чтобы ни один из них не указывал на данные, которые вы решили убрать.
2. Удаление узла, содержащего данные, которые вам больше не нужны.

Процесс удаления данных в связанном списке проиллюстрирован на рис. 18.6.

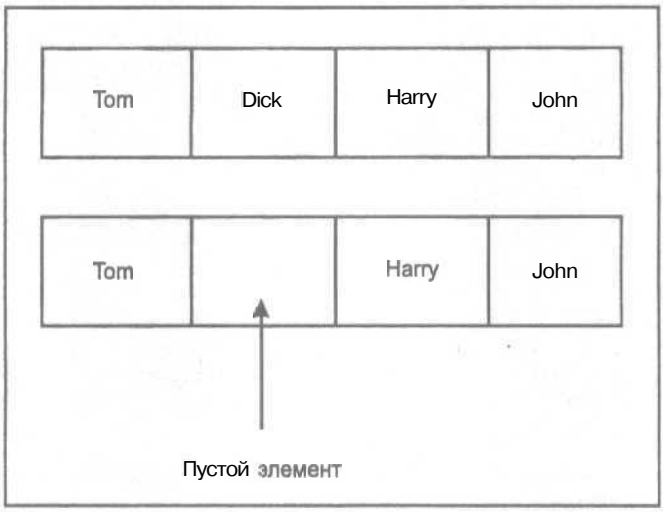

Рис. 18.5. Удаление данных в центре массива приводит к появлению пустой "ловушки"

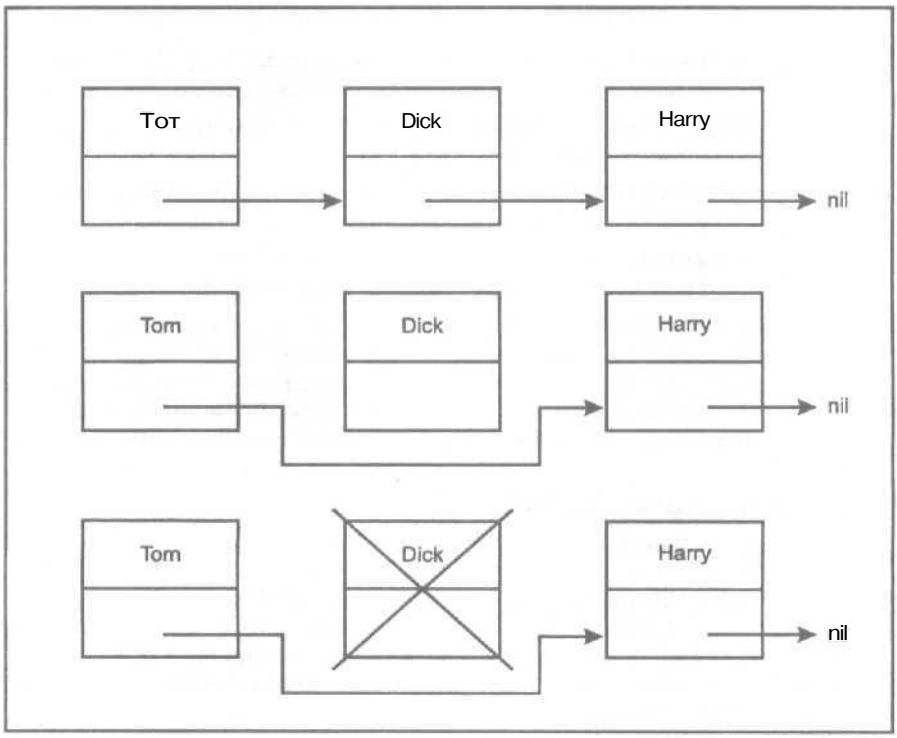

Рис. 18.6. Удаление данных в связанном списке не приводит к появлению пустой "ловушки", что позволяет сэкономить память компьютера

Написание действительных инструкций, которые заставили бы компьютер переупорядочить указатели и убрать узел, оказывает совсем непростой задачей. Поэтому, несмотря на то, что массивы приводят к неэффективному использованию памяти, с ними намного проще работать. С другой стороны, связанные списки обеспечивают эффективное использование памяти, но ими очень сложно управлять, так как вам приходится писать очень много инструкций для выполнения такой простой, казалось бы, задачи, как переупорядочение указателей.

Создание структуданных с помощью связанных списков

Простейшие связанные списки называются *однонаправленными;* каждый узел содержит данные и один указатель на следующий узел в связанном списке (см. рис. 18.2).

Проблема с однонаправленными связанными списками состоит в том, что каждый узел может указывать только на следующий узел в списке. Если вы захотите найти предыдущий узел, вы не сможете этого сделать. Например, возвращаясь к рис. 18.2, средний узел (содержащий имя *Dick),* указывает только на узел, содержащий имя *Harry.* Но вы не можете определить, какое имя стоит перед именем *Dick.* В подобной ситуации массивы оказываются намного проще в использовании, так как они всегда позволяют определить, какие данные содержатся в последующей или предыдущей ячейке по отношению к выбранной.

Вы можете упорядочивать связанные списки многими способами для создания различных типов структур данных. Так как структуры данных не выполняют никакой другой задачи, кроме хранения данных, выбор правильной структуры данных значительно упрощает написание программы (и наоборот).

Программа — личный ассистент, предназначенная для хранения имен и адресов, например, использует связанный список записей. Программа в состоянии расширить или уменьшить связанный список, в зависимости от того, сколько имен пользователь сохранил. Какой тип структуры данных вы будете использовать, во многом зависит от того, программу какого типа вы решили написать. Именно по этой причине связанные списки играют такую большую роль — они позволяют создавать огромное количество типов данных.

Не стоит переживать по поводу подробных сведений о создании различных структур данных с помощью связанных списков. Просто запомните, что различных типов данных существует великое множество; со многими из них вам предстоит разобраться, если вы намерены всерьез заниматься программированием.

Двунаправленные связанные списки

Проблема с однонаправленными связанными списками состоит в том, что каждый узел может указывать только на следующий узел в списке, как уже говорилось. Для решения этой проблемы можно использовать *двунаправленные связанные списки,* каждый узел которых содержит данные и два указателя. Один из них указывает на следующий узел, а второй — на предыдущий, как показано на рис. 18.7.

Программа — личный ассистент для хранения имен и адресов явно бы "предпочла" двунаправленные связанные списки, что позволило бы пользователю прокручивать список при просмотре имен и адресов.

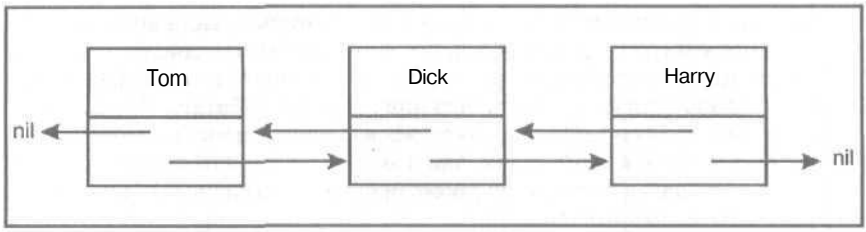

Рис. 18.7. В двусвязанных списках каждый узел содержит данные и два указателя, один из которых указывает на следующий узел, а второй — на предыдущий

Создание двунаправленных связанных списков подразумевает написание инструкций для управления в два раза большим числом указателей, что еще увеличивает вероятность того, что какой-нибудь указатель будет на несуществующий или ненужный узел. В любом случае указатели играют важную роль, поскольку неправильное их использование приведет к серьезным ошибкам в работе программ.

Кольцевые связанные списки

Типичный двунаправленный связанный список можно представить в виде прямоугольника, у которого есть начало и конец (см. рис. 18.7). Однако вы можете объединить оба конца однонаправленного связанного или двусвязанного списков, чтобы создать кольцевой связанный список, как показано на рис. 18.8.

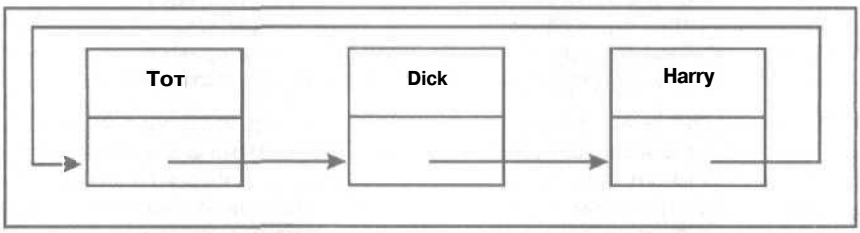

Рис. 18.8. В кольцевых связанных списках вы не найдете ни начала, ни конца

Кольцевые связанные списки особенно полезны в программах для создания презентаций, когда необходимо отображать данные непрерывно. Такие программы отображают определенную последовательность данных на экране с начала до конца, а затем снова начинают с начала.

Стеки

Стек — это специальная разновидность однонаправленного связанного списка, который позволяет вам добавлять и удалять узлы только в конце или начале. В качестве неплохого реального примера стеков можно привести создание торта: вы добавляете коржи, только укладывая их один на другой.

Как пример использования стеков в математических вычислениях можно привести обратную польскую запись (Reverse Polish Notation — RPN), которая реализована во многих калькуляторах. Например, если вы хотите посчитать выражение (1+2) *3, используя обратную польскую запись, вам необходимо ввести следующее выражение:

```
1 2 + 3 *
```

Рис. 18.9. В стек данные добавляются только с одного конца

На рис. 18.9 показано, каким образом используется стек при вычислении формулы (1+2) *3 при использовании обратной польской записи. Вот как это все работает.

1. Число 1 помещается в первую ячейку стека сверху.
2. Число 2 помещается в первую ячейку стека сверху, а число 1 опускается вниз на одну ячейку.
3. Число 2 извлекается из стека, затем извлекается число I. После этого оба числа суммируются и полученный результат, число 3, помещается в первую ячейку стека сверху.
4. Число 3 помещается в первую ячейку стека сверху, а результат предыдущего вычисления, также равный 3, опускается на одну ячейку вниз.
5. Число 3 извлекается из стека, затем извлекается второе число 3. После этого оба числа перемножаются и полученный результат, число 9, помещается в первую ячейку стека сверху.

Стеки часто называют *связанными списками LIFO* (Last In First Out, т.е. последним пришел — первым обслужен). Это означает, что данные, помещенные в стек последними, извлекаются из него и обрабатываются первыми.

Не стоит переживать, если вы не понимаете логики обратной польской записи. Главное, чтобы вы поняли, как работают стеки, а с обратной польской записью пусть возятся инженеры — разработчики калькуляторов.

Очереди

Очередь — это специальный связанный список, который следует двум правилам. Первое состоит в том, что узлы можно добавлять только с одного конца списка. Второе правило состоит в том, что узлы удаляются только с другого конца.

Очереди часто называют *связанными списками FIFO* (First In First Out, т.е. первым пришел — первым обслужен). Это означает, что данные, помещенные в стек первыми, извлекаются из него и обрабатываются также первыми (рис, 18.10).

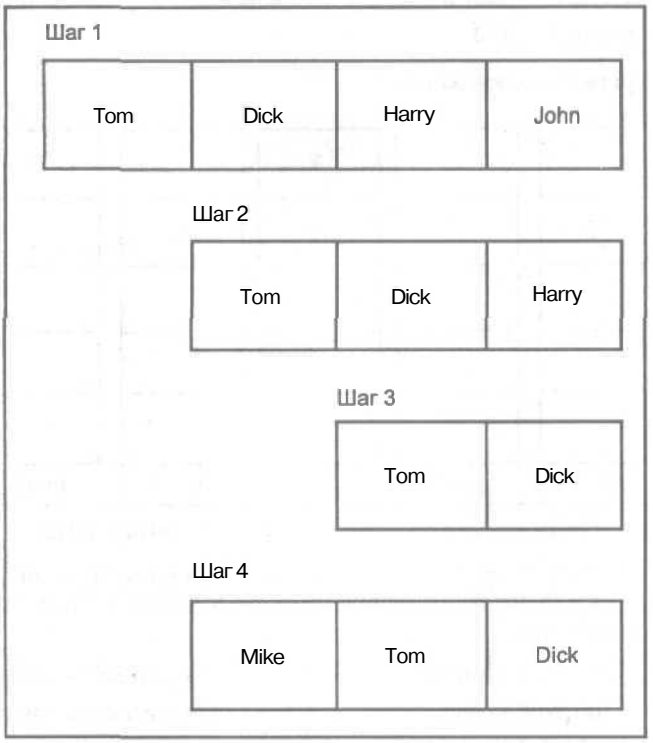

Рис. 18.10. В очереди данные добавляются с одного конца, а удаляются с другого

Ниже приводится пояснение того, что же представлено на рис. 18.10.

1. Первый шаг иллюстрирует очередь, в которой имя *John* стоит первым, имя *Harry* — второе, имя *Dick* — третье, а имя *Tom* — четвертое и последнее.
2. Второй шаг иллюстрирует удаление имени *John*. Все остальные имена перемешаются на одну позицию ближе к началу очереди.
3. Третий шаг иллюстрирует удаление имени *Harry*. Все остальные имена перемешаются на одну позицию ближе к началу очереди.
4. Четвертый шаг иллюстрирует добавление нового имени *Mike*. Так как это новый элемент очереди, он помешается в ее конец.

Деревья

Связанные списки используются для создания линейных или круговых структур. Однако существуют и нелинейные структуры, например *деревья,* в которых один узел *(корень)* представляет верх дерева, а остальные узлы *(листья)* находятся под ним, как показано на рис. 18.11.

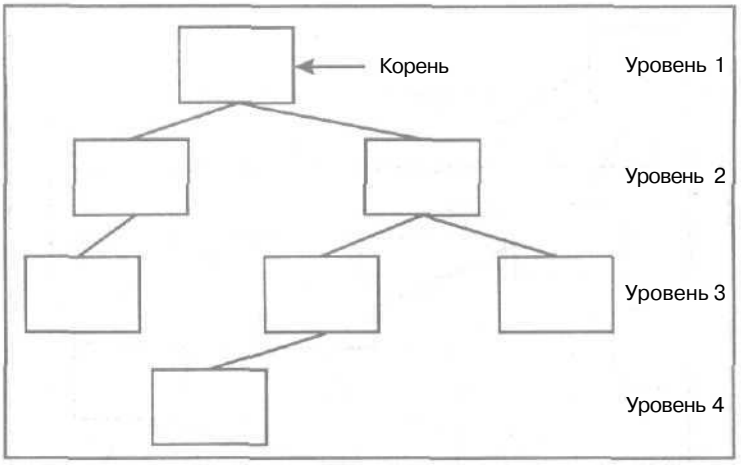

Рис. 18.11. Дерево как нелинейный связанный список

В дереве узел может не указывать ни на один другой узел, а может и указывать на несколько узлов. Для упрощения многие программисты используют *бинарные деревья,* в которых каждый узел ссылается на нуль, один или два других узла.

Деревья очень часто используются в программах, имитирующих искусственный интеллект. Например, в игре в шахматы используется дерево, корень которого представляет возможное действие первого игрока, находящиеся под ним узлы-листья (уровня J) представляют потенциальные действия второго игрока. Затем узлы-листья (уже уровня 2) представляют потенциальные действия первого игрока, а узлы-листья (уровня 3) — потенциальные действия второго игрока и т.д.

Графы

Граф — это связанный список, в котором каждый узел может ссылаться на одни иди несколько узлов, не образуя при этом никакой четкой структуры, например списка. На рис. 18.12 показан типичный граф.

Программисты очень часто используют графы в *нейронных сетях,* специальных программах, имитирующих работу человеческого мозга. В данном случае каждый узел представляет собой нейрон, а связи между узлами — синапсы, связывающие нейроны.

Графы, деревья, стеки, круговые связанные и двусвязанные списки — все это различные варианты упорядочения и использования связанных списков. Чем сложнее программы, тем вероятнее, что вы будете использовать в них подобные структуры данных.

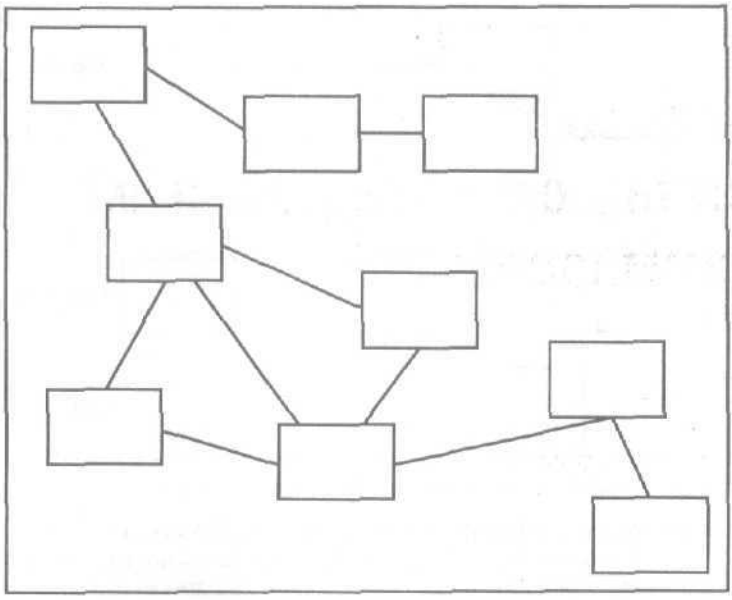

Рис. 18.12. Граф допускает любые способы связывания узлов

Глава 19

Знакомство с объектно-ориентированным программированием

В этой главе...

- Проблемы программного обеспечения
- Как упростить написание программ
- Разбиение программ на объекты
- Выбор объектно-ориентированного языка программирования

Объектно-ориентированное программирование — это последняя причуда (или, говоря правильным компьютерным языком, последняя *методология)* для достижения следующих характеристик, которые программисты хотят видеть в своих творениях.

- ✓ Простота и скорость создания
- ✓ Простота понимания и внесения изменений
- ✓ Надежность и отсутствие ошибок

Специалисты часто используют аббревиатуру ООП вместо слишком длинной фразы "Объектно-ориентированное программирование".

Liberty BASIC не объектно-ориентированный язык программирования, а значит, вы не сможете поэкспериментировать с объектами, используя этот язык программирования. Хотя большинство версий BASIC не поддерживает объекты, компания *Microsoft* добавила средства объектно-ориентированного программирования в последнюю версию Visual Basic. Поэтому, если вы хотите использовать объекты, но при этом еще и работать с BASIC, подумайте об использовании Visual Basic и приобретении книги *Visual Basic For Dummies* (написанной вашим покорным слугой и выпущенной издательством *IDG Books Worldwide, Inc.).* Для демонстрации возможностей объектно-ориентированного программирования во всей настоящей главе я использую примеры программ, написанные на C++.

Проблемы программного обеспечения

Независимо от того, насколько велики вычислительные ресурсы компьютера, вся эта мощь ограничена программным обеспечением, которое управляет работой компьютера. Наибольшей проблемой, связанной с программным обеспечением, является *надежность.* Под надежностью понимается работа программы без сбоев, зависаний, а также неправильных действий. Особенно надежность оказывается критичной для систем реального времени, таких как системы управления полетами самолетов, компью-

теров, обслуживающих корпоративные Web-узлы, а также различных бухгалтерских программ, в работе которых потеря одного бита при сбое может обернуться многомиллионными потерями. Надежность программного обеспечения настолько важна, потому что человеческие жизни и (что очень важно с точки зрения корпораций) миллионы долларов зависят от работоспособности компьютеров.

Конечно же, надежное программное обеспечение окажется совершенно бесполезным, если вы не сможете написать его вовремя. Из-за конфликтов между требующими компаниями и работающими программистами профаммы очень часто начинают продавать еще до того, как они полностью протестированы на надежность работы (а это значит, что профамма вообще не будет работать должным образом), или же профаммы не успевают попасть на рынок вовремя, а пользователи отдают предпочтение конкурирующим программньЕМ продуктам. (А те программисты, которые долго ждут, все равно не могут стопроцентно гарантировать беспроблемную работы профаммы, даже если потратили на ее тестирование не один год.)

Какими бы привлекательными ни были новая технология, язык программирования или стиль написания программ, разработчики профаммного обеспечения всегда сталкиваются с одной и той же проблемой — программы должны работать надежно, но при этом появляться на рынке как можно быстрее. Это приводит к тому, что любое программное обеспечение содержит ошибки, не позволяющие ему работать максимально эффективно.

Как упростить написание програлми

Чтобы вы прочувствовали всю "головную боль", сопровождающую разработку программного обеспечения, следует хотя в общих чертах знать, как профаммное обеспечение развивалось во времени. Когда компьютеры только появились, профаммы были относительно малы. Поэтому программисты писали профаммы, обходясь без какого-либо предварительного планирования. Завершив написание программы, они запускали ее и смотрели, работает она или нет. Если она не работала, профаммисты переписывали ее наново, запускали и снова смотрели, работает программа или нет.

Подобный метод проб и ошибок хорошо подходил для написания небольших программ, однако по мере того, как они становились больше и сложнее, переписывание программ становилось все более сложным мероприятием, которое, к тому же, приводило к возникновению других ошибок. Чем больше профамма, тем больше мест, в которых скрываются ошибки, что означает потенциальную невозможность профаммы работать вообще.

Современные компьютерные профаммы часто содержат по несколько миллионов строк, поэтому применять к ним метод проб и ошибок вообще нельзя. Написание большой программы без предварительного планирования сродни попытке построить небоскреб без чертежей.

Профаммисты очень быстро разработали новую стратегию. Вместо того чтобы пытаться написать одну большую программу, они решили написать несколько небольших профамм (которые назвали *подпрограммами).*

Основная идея состояла в том, что небольшую программу проще написать и отладить, поэтому вам остается только написать несколько небольших программ, объединить их и получить большую профамму, которая будет гарантированно работать, как показано на рис. 19.1.

Какой бы привлекательной ни казалась идея написания небольших подпрофамм и объединения их наподобие строительных блоков, проблемы все еще оставались. Вы могли разделить профамму на небольшие подпрограммы, однако инструкции, сохраненные в одной подпрофамме, все равно могли манипулировать данными, используемыми другой подпрограммой, как показано на рис. 19.2.

Рис. 19.1. Разделение большой программы на несколько небольших подпрограмм позволяет значительно упростить написание программ

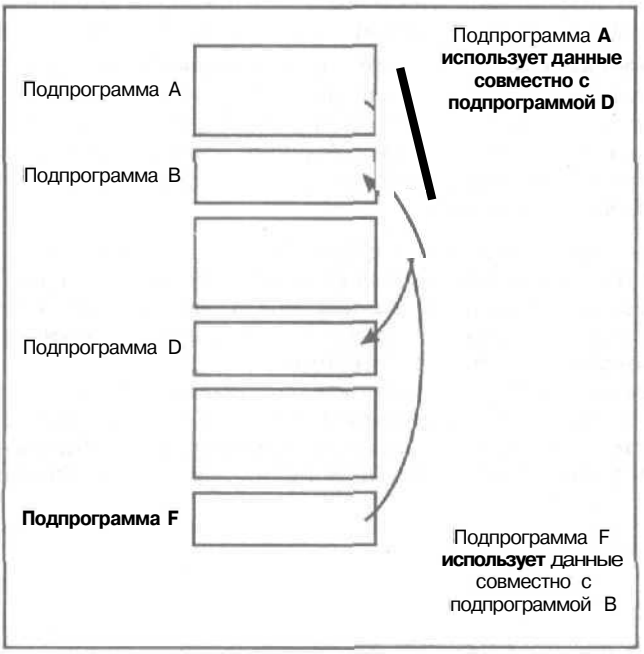

Рис. 19.2. Инструкции из одной подпрограммы могут случайно изменить данные, используемые другой подпрограммой

Для решения этой, а также ряда других проблем, программисты решили изолировать подпрограммы, превращая их в независимо скомпилированные файлы. Таким образом, можно избежать одновременного использования одних и тех же данных несколькими подпрограммами, хотя они точно так же работали вместе, образуя одну большую программу.

Развивая идею изолирования подпрограмм в виде независимых частей (их специалисты называются *модулями*), программисты пришли к идее объектов, а значит, и к такому понятию, как *объектно-ориентированное* программирование.

Разбиение программ на объекты

После того как программисты преуспели в разбиении больших программ на небольшие подпрограммы, следующим логическим шагом стало выделение данных и инструкций, манипулирующих данными, в отдельную единицу, — другими словами, *объект*.

Программа состоит из одного или нескольких объектов, каждый из которых является единицей, содержащей следующие два элемента.

1. ✓ Данные (известные и как *свойства*)
2. ✓ Инструкции (известные и как *методы)* для манипулирования этими данными

Поскольку объект не зависит от никакой другой части программы, объектно-ориентированный подход к программированию обладает следующими преимуществами.

- ✓ Надежность. Если программа не работает, вам достаточно изолировать ошибку в виде отдельного объекта и отладить работу только этого объекта вместо того, чтобы пытаться отладить работу целиком всей программы, которая содержит миллионы строк кода.
- ✓ Возможность повторного использования. Поскольку объекты являются автономными единицами, вы можете (теоретически) скопировать объект из одной программы в другую. Возможность повторного использования объектов не только упрощает создание новых программ, но ускоряет этот процесс, поскольку старые проверенные объекты уже гарантированно работают.

В программе, спроектированной традиционным способом, вы часто сохраняете данные в одном месте, а манипулирующие ими инструкции — в другом. На рис, 19.3 показана традиционная программа, разделенная на подпрограммы (показаны прямоугольниками). Каждая программа получает доступ к необходимым данным, а также к данным, используемым другой подпрограммой.

В противоположность этому, объектно-ориентированная программа (объекты показаны на рис. 19.3 эллипсами) объединяет данные и манипулирующие ими инструкции в одном месте (этот процесс называется *инкапсуляцией).* Инкапсуляция не позволяет одной части программы пересекаться с данными, используемыми другой частью программы.

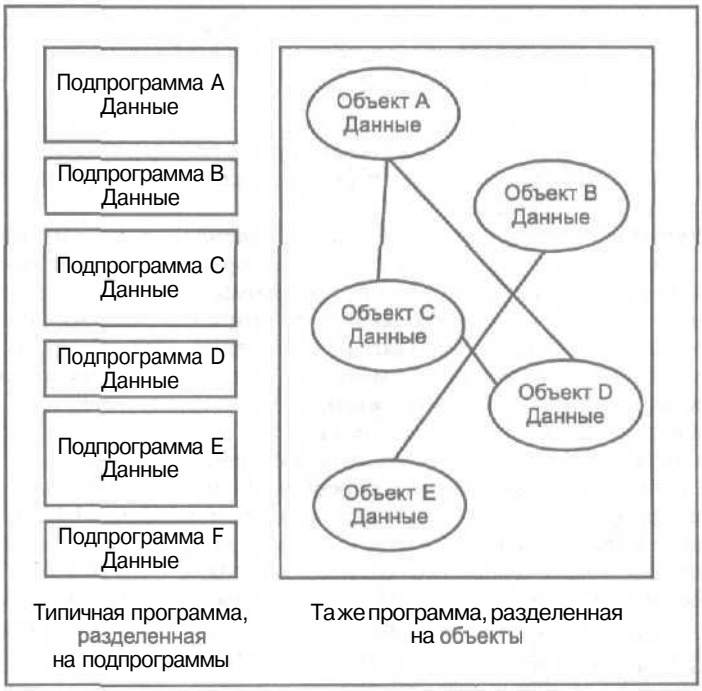

Рис. 19.3. В объектно-ориентированных программах объекты изолируют данные от других объектов

Как использовать объекты

Одна из самых больших проблем программирования состоит в том, что программы необходимо постоянно модифицировать. Многие программы используются годами, если не десятилетиями. Вместо того чтобы писать совершенно новую программу вместо уже существующей, многие компании предпочитают модифицировать проверенные временем программы. Теоретически модификация существующей программы (которая уже работает) занимает намного меньше времени и позволяет получить намного более стабильный результат, чем создание новой программы с нуля.

5 Наследование объектов

Объекты характеризуются надежностью и возможностью повторного использования благодаря такой концепции, как *наследование*. Основная идея наследования заключается в поощрении повторного использования программистами уже готового кода.

На заре программирования вы могли разделить программу на несколько небольших подпрограмм. Изучив одну из них, вы приходили к выводу, что после внесения определенных изменений эту же подпрограмму можно использовать в другой программе. К сожалению, внесение изменений в подпрограмму приводило к тому, что она переставала вообще работать.

При модифицировании существующей подпрограммы для предотвращения внесения в ее код ошибок объекты используют наследование. Наследование позволяет вам копировать существующий объект, после чего добавлять к этой копии новый код, не внося никаких изменений в готовый код объекта. Таким образом, новая копия "наследует" все данные и программный код, но при этом позволяет вам модифицировать объект для решения несколько иных задач.

Наследование не только избавляет от случайных ошибок в программном коде, но и позволяет намного быстрее создавать программы. Копирование и модифицирование объекта — это намного проще, чем создание объекта с нуля.

К сожалению, постоянное модифицирование существующей программы приводит к такому же результату, как написание романа по одной странице 300 разными людьми. В результате получается сумбурная структура, в которой нельзя понять, где заканчивается одна часть и начинается другая.

Объектно-ориентированное программирование кажется в такой ситуации очень привлекательным. Для внесения изменений в объектно-ориентированную программу достаточно извлечь объект, содержащий средства программы, которые вы решили изменить, переписать или заменить его, после чего возвратить этот объект опять в программу.

Лучше всего то, что объекты позволяют намного проще определить, в какую именно часть программы необходимо вносить изменения. Например, предположим, компьютерная ифа отображает на экране каких-то ужасных пришельцев, уничтожить которых можно, только попав им в голову. Вам необходимо изменить только ту часть программы, которая отвечает за отображение пришельцев и их перемещение по экрану.

Если ифа написана традиционным способом (с разделением программы на небольшие подпрограммы), вы должны обнаружить подпрограмму, которая контролирует внешний вид пришельца, а также подпрограмму, которая контролирует его перемещения. Даже если вам и удастся это сделать, все равно придется еще и исследовать взаимодействие этих подпрограмм с другими частями программы, что практически невозможно сделать с миллионами строк. Звучит устрашающе? Так оно и есть, особенно если на внесение изменений в игру, содержащую очень много строк кода, вам отведено меньше двух недель.

Но если ифа написана в объектно-ориентированном стиле, поставленная перед вами задача значительно упрощается. Все, что вам необходимо сделать, — это найти объект, представляющий пришельца. Этот объект содержит все подпрограммы, в которые вам необходимо внести изменения. Измените объект, вставьте его снова в программу, и все — ваша работа выполнена.

Как создать объект

Первая часть создания объекта — определение *класса*, который во многом напоминает запись. (О записях подробно мы говорили в главе 17, "Сохранение связанных данных в виде записи".) Класс определяет данные и манипулирующие ими инструкции. Определив класс, вы можете создать один или несколько базирующихся на нем объектов, используемых в вашей профамме.

Liberty BASIC не предлагает команд для создания объектов, поэтому все приведенные примеры написаны на языке программирования C++. Не пытайтесь разобраться в синтаксисе команд C++; просто изучите код, чтобы понять, как все это работает в целом.

Ниже приведен пример определения **класса на** C++:

```
Class monster
{
public:
   int x_coordinate:
   int y_coordinate;
   void moveme(int, int);
   void initialize tint, int);
};
```

Сокрытие и предоставление **данных объекту**

Поскольку объектам необходимо взаимодействовать, они могут классифицировать свои данные и инструкции по одной из трех категорий: *частные, открытые* и *защищенные.*

Если объект определяет данные и инструкции как частные, их может использовать только он сам. Никакой другой объект сделать этого не сможет. (Именно по этой причине подобные объекты называются частными.)

С другой стороны, другие объекты могут использовать открытые данные и инструкции. Объекты используют подобные данные и инструкции для взаимодействия и обмена данными,

Защищенные данные и инструкции во многом напоминают частные, однако существует одно очень важное отличие: если вы используете наследование для копирования существующего и создания нового объекта, новый объект наследует только открытые и защищенные данные и инструкции. Любые частные данные и инструкции остаются у старого объекта.

Этот простой написанный на C++ код приказывает компьютеру выполнить следующее.

1. Первая строка сообщает компьютеру о том, что создается новый класс `monster`.
2. Вторая строка начинает определение класса.
3. Третья строка сообщает компьютеру о том, что все указанные ниже сведения являются открытыми, а значит, могут использоваться любой частью программы.
4. Четвертая строка создает переменную `x_coordinate` типа `integer`.
5. Пятая строка создает переменную `y_coordinate` типа `integer`.
6. Шестая строка сообщает компьютеру о том, что объект содержит подпрограмму (метод) `moveme`, использующую две переменные типа `integer`.
7. Шестая строка сообщает компьютеру о том, что объект содержит подпрограмму (метод) `initialize`, использующую две переменные типа `integer`.
8. Восьмая строка завершает определение класса.

Класс — это не объект. Для создания объекта вам необходимо определить переменную, которая будет представлять определенный класс.

Написание методов объекта

После того как вы объявили методы, которые сохраняются в классе, необходимо написать действительные инструкции, которые подпрограмма будет выполнять. Я использовал определение класса из предыдущего раздела в следующем примере:

```
Class monster
{
public:
   int x_coordinate:
   int y_coordinate;
```

```
  void moveme(int, int);
  void initialize(int, int);
};
```

Этот класс определяет два метода — moveme и initialize. Для того чтобы эти методы действительно выполнили какие-то действия, вы должны **их** полностью описать после определения класса, как показано ниже:

```
void monster::moveme(int new_x, int new_y)
{
   x_coordinate = x coordinate + new_x;
   y_coordinate = y_coordinate + new_y;
}
void monster::initialixe(int init_x, int init_y)
{
   x_coordinate - init_x;
   y_coordinate = init_y;
}
```

Метод initialize определяет координаты *X* и *Y* любого объекта, полученного из класса monster. Вы можете использовать метод moveme для перемещения объекта на определенное расстояния по оси *X* (вправо или влево) и оси *Y* (вверх или вниз).

Создание объекта

После того как вы определили класс и написали все методы, объявленные в его рамках, вам необходимо определить переменные, представляющие этот класс. В объектно-ориентированном программировании каждая подобная переменная представляет собой настоящий объект.

Для создания объекта вы объявляете переменную, которая будет его представлять. В приведенной ниже программе я определяю объект zombie, относящийся к классу monster;

```
#include <iostrem.h>
class monster
{
public:
   int x_coordinate:
   int y_coordinate;
   void moveme(int, int);
   void initialize(int, int);
};
void monster: :moveme(int new_x, int new_y)
{
   x_coordinate = x_coordinate 4 new_x;
   y_coordinate = y_coordinate 4 new_y;
}
void monster::initialixe(int init_x, int init_y)
{
        x_coordinate = init_x;
  -y_coordinate = init_y;
}
void main()
{
monster zombie;
   zombie.initialize(12, 15);
   cout << "The X-location of the zombie is " <<
```

```
  zombie.x_coordinate << "\n";
  cout << "The Y-location of the zombie is " <<
  zombie.y_coordinate << "\n";
  zombie.moveme (34, 9) ;
  cout << "The new X-location of the zombie is " <<
  zombie.x_coordinate << "\n";
  cout << "The new Y-location of the zombie is " <<
  zombie.y_coordinate << "\n";
}
```

Основная программа, написанная на C++, начинается со строки void main(). Каждая строка приведенной программы выполняет следующее.

1. Первая строка сообщает компьютеру, что начинается программа, написанная на C++.
2. Вторая строка сообщает компьютеру, что начинается список всех инструкций основной программы.
3. Третья строка приказывает компьютеру создать объект zombie класса monster.
4. **Четвертая строка запускает метод** initialize, **который определяет координаты** *X* и **Y объекта** zombie **(12 и 15 соответственно).**
5. **Пятая строка отображает на экране сообщение** The X-location of the zombie is (Координата X зомби равна) 12.
6. Шестая строка отображает на экране сообщение The Y-location of the zombie is (Координата Y зомби равна) 15.
7. Седьмая строка запускает метод moveme, который перемещает объект zombie на 34 единицы вдоль оси *X* и на 9 единиц вдоль оси *Y*.
8. Восьмая строка отображает на экране сообщение The new X-location of the zombie is (Новая координата X зомби равна) 46.
9. **Девятая строка отображает на экране сообщение The** new Y-location of the zombie is (новая координата Y зомби равна) 24.
10. Десятая строка завершает основную программу на C++.

Несмотря на то, что текст программы, написанной на C++, кажется достаточно запутанным, вы должны запомнить всего две детали.

- ✓ При создании объекта сначала необходимо определить класс
- ✓ Класс содержит данные и методы, манипулирующие данными объекта

Выбор объектно-ориентированного языка программирования

После того как вы познакомились с общими принципами, лежащими в основе объектно-ориентированного программирования, вам наверняка не терпится попробовать собственные силы в написании объектно-ориентированных программ. Liberty BASIC не поддерживает объектно-ориентированное программирование, поэтому вам придется обратиться к другому языку программирования. При этом вы можете выбрать один из двух видов языков программирования.

- ✓ Гибридный объектно-ориентированный язык программирования
- ✓ Истинный объектно-ориентированный язык программирования

Гибридный объектно-ориентированный язык программирования — это обычный язык программирования, в который добавили ряд объектно-ориентированных средств, К наиболее популярным гибридным языкам программирования относятся Pascal (реализованный в виде Delphi), BASIC (реализованный в виде Visual Basic) и C++.

Основные термины объектно-ориентированного программирования

Хотя вы еще не стали экспертом в объектно-ориентированном программировании, изучив материал настоящей, совсем небольшой по объему главы, вы, по крайней мере, поняли основные преимущества объектов, значительно упрощающих модификацию и повторное использование программ.

Если вы планируете продолжить изучение программирования, вам стоит обратить внимание именно на объектно-ориентированное программирование (по крайней мере до тех пор, пока не появится новая методология, еще упрощающая написание программ). Для того чтобы получить более или менее полное представление об объектно-ориентированном программировании, вам не мешало бы знать ряд основных терминов.

- ✓ *Инкапсуляция* — объединение всех связанных данных и манипулирующих ими инструкций в одном месте
- ✓ *Наследование* — передача данных и инструкций от одного объекта к другому, созданному на его основе
- ✓ *Метод* — это подпрограмма, манипулирующая данным в рамках одного объекта
- ✓ *Объект* — это набор данных и манипулирующих ими инструкций, представляющий собой автономную единицу

Основное преимущество использования гибридного языка программирования состоит в том, что, если вы уже знакомы с каким-то языком программирования (Pascal, BASIC или C++), вы сможете быстро разобраться с добавленными к нему объектно-ориентированными средствами. Если вы еще не уверены в преимуществе объектно-ориентированного программирования, напишите небольшую часть программы с помощью объектов, а также программу целиком, используя "устаревшие" методы программирования.

Конечно же, основной недостаток гибридных языков состоит в том, что они позволяют программистам смешивать традиционные и объектно-ориентированные приемы программирования, а это приводит к тому, что структура программ становится слишком запутанной. Гибридные языки программирования позволяют программистам не использовать вообще, использовать только некоторые или же все объектно-ориентированные подходы, поэтому написанная с помощью подобного языка программа не использует всех преимуществ объектов; кроме того, в коде подобной программы намного сложнее разобраться.

Именно по этой причине многие люди предпочитают истинные объектно-ориентированные языки программирования, которые заставляют программиста работать только с объектами. К наиболее популярным истинным объектно-ориентированным языкам программирования относятся

- ✓ SmallTalk,
- ✓ Eiffel,

- ✓ C#,
- ✓ Java.

Если вы еще не решили, отдать предпочтение традиционному языку программирования (и использовать его объектно-ориентированный гибрид) или перейти к использованию истинного объектно-ориентированного языка программирования, подумайте о преимуществах разделения программы на объекты. Сами по себе объектно-ориентированные подходы не упрощают написание программного обеспечения и не делают его более надежным, однако они дают более полные сведения о проблемах, возникающих при написании программного обеспечения; кроме того, вы знаете, каким образом объектно-ориентированное программирование решает эти проблемы.

По большому счету, абсолютно никого не интересует, какой язык программирования вы используете, или же вы создаете программы с помощью объектно-ориентированного языка программирования, или сидя на кухне в три часа ночи, слушая радио. Самое важное — написать работоспособное программное обеспечение, причем вовремя. Если вы достигнете этой цели, вы можете сконцентрироваться на результатах работы и опустить мелочи, о которых слишком часто думают ваши коллеги.

Часть V

Алгоритмы: объясните компьютеру, что от него требуется

"Мы пришли почистить код программы"

В этой части...

Программа — это просто список инструкций, и вы можете создать инструкции, которые объяснят компьютеру, что именно он должен сделать, используя миллион способов. Например, если вы хотите рассказать знакомому, как ему добраться от аэропорта до вашего дома, вы наверняка дадите ему два-три различных варианта, как это сделать. Каждый вариант обязательно позволит знакомому добраться до вашего дома, однако один вариант будет проще, второй — быстрее, а третий обеспечит массу положительных эмоций.

В мире программирования определенный способ выполнения определенной задачи называется *алгоритмом*. Выбрав самый быстрый набор инструкций (алгоритм), вы сможете сделать программу быстрее и намного эффективнее. В настоящей части книги я рассмотрю несколько основных алгоритмов, позволяющих решать различные задачи, что позволит вам профессиональнее подойти к написанию собственных программ.

Глава 20

Сортировка

В этой главе...

- Несколько вариантов сортировки данных
- Выбор алгоритма сортировки

Программы чаще всего получают данные из внешнего мира (например, от человека, сидящего за клавиатурой), выполняют с ними определенные действия, после чего возвращают их в определенном, удобном для пользователя, формате.

В промежутке между стадиями получения данных и возвращения результатов их обработки программа должна сохранить данные в памяти или на жестком диске, используя при этом определенную структуру данных. (О структурах данных подробно мы говорили в главах 16–19.) Прежде чем программа вернет полезный результат, ей понадобится сначала отсортировать их определенным образом.

Например, база данных оказывается бесполезной, если позволяет сохранять информацию, но не позволяет ее переупорядочивать. Например, вам нужно упорядочить данные по алфавиту, по фамилии, по коду города или по любому другому критерию. Ваша программа в данном случае просто обязана уметь сортировать данные.

Хотя сортировка покажется достаточно простой темой, на самом деле все оказывается не так легко. Это связано, в первую очередь, с тем, что программе необходимо провести сортировку данных как можно быстрее. В конце концов, разве назовешь полезной программу, которая тратит пять минут на сортировку 15 имен.

По этой причине определенная часть компьютерных центров сосредоточилась именно на разработке максимально эффективных методов сортировки *(алгоритмов сортировки)*. Сортировку данных проводят многие программы, поэтому знать о различных алгоритмах сортировки и уметь их использовать должен любой программист. В настоящей главе вы введете текст нескольких программ на Liberty BASIC и увидите, как выполняет свою работу тот или иной алгоритм сортировки.

Ученые разработали массу разнообразных алгоритмов сортировки, однако при этом все равно не существует одного универсального алгоритма сортировки, пригодного для всех программ для решения любых задач. Наиболее предпочтительный алгоритм сортировки частично зависит от данных, которые ему придется сортировать, и от структур данных, используемых программой для хранения данных.

Измерение эффективности с помощью представления "большого О".

Для измерения эффективности определенных алгоритмов ученые создали так называемое представление "большого О". В частности представление "большого О" определяет время выполнения какого-то алгоритма (например, алгоритма сортировки), зависящее от числа элементов, которое этот алгоритм должен обработать.

Например, если у вас есть алгоритм сортировки имен в алфавитном порядке, время выполнения алгоритма зависит от количества имен. В представлении "большого О" подобная за-

висимость выражается в виде O(N), где о обозначает "порядок величины" (order of magnitude), а N — число обрабатываемых элементов.

Способ, используемый программистами для определения представления "большого О" определенного алгоритма, **зависит** от времени **выполнения** алгоритма **и количества элементов,** которые он должен **обработать. Например,** если **время выполнения алгоритма и количество обрабатываемых** элементов (N) **выражается как** N2+N+1, **представление "большого** О" будет иметь вид O(N2).

При вычислении представления "большого **О" для определенного алгоритма вы выбираете элемент, скорость изменения которого максимальна (в** данном **случае это N2), и игнорируете всю оставшуюся часть выражения. (Конечно, если вы выберете неправильное выражение** для представления **алгоритма,** полученное **представление** "большого О" **также окажется неправильным.)**

Программисты часто используют представление "большого О" **для** определения **среднего и** худшего сценария при изучении поведения **алгоритма при управлении типичным количеством элементов и очень большим количеством элементов.**

Неудивительно, **что некоторые алгоритмы лучше подходят для обработки относительно** небольшого числа элементов и совершенно **теряют свои достоинства при увеличении элементов. Другие алгоритмы оказываются** беспомощными **при сортировке очень запутанного списка элементов.**

Программисты изучают **недостатки и преимущества** того **или иного алгоритма, используя представление "большого О", которое** значительно **упрощает поиск алгоритма, подходящего для выполнения определенной** задачи.

Сортировка методом вставок

Предположим, вы играете в карточную игру и сдающий сдает карты именно вам. Получив две карты, увы тут же стремитесь каким-то образом упорядочить их. После получения третьей карты вы захотите упорядочить ее по отношению к предыдущим двум. Вы сортируете каждую дополнительную карту по отношению к уже отсортированным. Именно так и работает метод сортировки, называемый методом вставок (рис. 20.1). С точки зрения компьютера алгоритм сортировки методом вставок работает следующим образом.

1. Сравниваются два элемента в списке, после чего они взаимно упорядочиваются.
2. Анализируется следующий элемент в списке, после чего он сортируется по отношению к двум раньше отсортированным элементам.
3. Для каждого дополнительного элемента в списке повторяется п. 2 до тех пор, пока не будет достигнут конец списка.

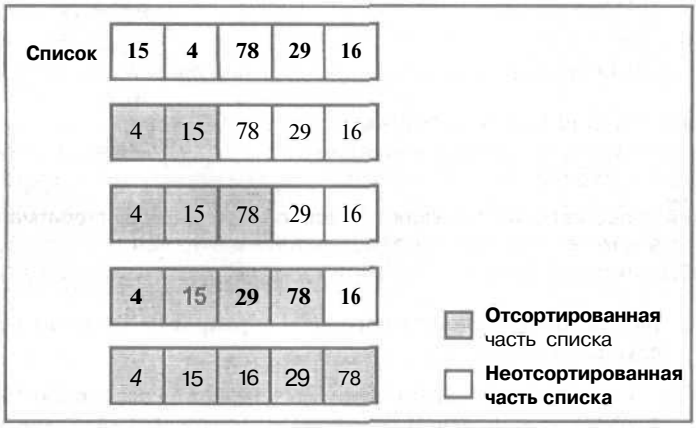

Рис. 20.1. Сортировка методом вставок изменяет расположение каждого элемента по отношению к уже отсортированным элементам в списке

Для того чтобы увидеть сортировку методом вставок в действии, попробуйте выполнить следующую программу:

```
MaxSize = 5
REDIM MyArray(MaxSize)

FOR I = 1 TO MaxSize
  MyArrya(I) = INT(RND(1) * 100) + 1
  PRINT MyArray(I); SPACES(1);
NEXT I
PRINT "(Initial array)"

FOR ArrayPos - 2 TO MaxSize
  TempValue = MyArray(ArrayPos)
  StopNow = 0
  Count = 1
  Time2Stop = 0
  WHILE (Time2Stop - 0)
     IF TempValue ,MyArray(Count) THEN
        FOR J = ArrayPos TO Count STEP -1
          MyArray(J) = MyArray(J - 1)
        NEXT J
        MyArray(Count) = TempValue
        StopNow — 1
        FOR I = 1 TO MaxSize
           PRINT MyArray(I); SPACE$(1);
        NEXT I
        PRINT
     END IF
     Count = Count + 1
     IF _StopNow - 1) OR (Count = ArrayPos) THEN
        Time2Stop = 1
     END IF
  WEND
NEXT ArrayPos
FOR I = 1 TO MaxSize
  PRINT MyArray(I); SPACE$(1);
```

```
NEXT I
PRINT "Sorted array)"
END
```

Результат выполнения программы имеет следующий вид:

```
57 89 77 3 21 (Initial array)
58 77 89 3 21
59 57 77 89 21
60 21 57 77 89
3 21 57 77 89 (Sorted array)
```

Программа, использующая алгоритм сортировки методом вставки, работает следующим образом.

1. Строки с первой по седьмую создают переменную MaxSize, значение которой равно 5; создают массив MyArray для хранения пяти целых чисел; генерируют произвольное число; создают произвольное число от 1 до 100 и сохраняют его в массиве MyArray; после чего выводят массив на экран вместе со строкой Initial array (Первоначальный вид массива).
2. Восьмая строка является началом цикла FOR NEXT, который перебирает элементы массива со второго, используя переменную ArrayPos.
3. Девятая строка создает переменную TempValue и присваивает ей значение элемента массива MyArray, на который указывает переменная ArrayPos. В начале выполнения цикла FOR NEXT значение переменной TempValue равно второму элементу массива MyArray.
4. Десятая строка создает переменную StopNow и присваивает ей нулевое значение. Эта переменная используется в дальнейшем для того, чтобы сообщить компьютеру о том, что он уже переместил число на нужное место в массиве.
5. Одиннадцатая строка создает переменную Count и присваивает ей значение, равное единице. Эта переменная используется при определении того, куда именно в массиве необходимо переместить значение переменной TempValue.
6. Двенадцатая строка создает переменную Time2Stop и присваивает ей нулевое значение. Эта переменная используется в дальнейшем для того, чтобы сообщить программе о том, что массив уже полностью отсортирован.
7. Тринадцатая строка является началом цикла WHILE WEND, она проверяет равенство нулю значения переменной Time2Stop. Если это условие справедливо, все инструкции цикла выполняются.
8. Четырнадцатая строка является началом цикла IF THEN, проверяющего значение переменной TempValue (она представляет число, которое вы решили отсортировать), которое должно быть меньше значения переменной Count. При первом запуске цикла проверяется, меньше ли второй элемент массива первого элемента.

9. Строки с пятнадцатой по семнадцатую образуют цикл FOR NEXT, который перемешает каждый элемент массива на одну позицию вправо для освобождения места в массиве для значения переменной TempValue.

10. Восемнадцатая строка перемешает значение переменной TempValue на новое место в массиве, указанное переменной Count.

И. Девятнадцатая строка присваивает переменной StopNow значение, равное 1. Это сообщает компьютеру о том, что он правильно изменил место расположения значения переменной TempValue.

12. Строки с двадцатой по двадцать третью выводят на экран частично отсортированные варианты массива.

13. Двадцать четвертая строка является последней строкой цикла IF THEN, который начался с четырнадцатой строки.

14. Двадцать пятая строка увеличивает значение переменной Count.

15. Строки с двадцать шестой по двадцать восьмую проверяют, равно ли значение переменной StopNow единице, и равно ли значение переменной Count значению переменной ArrayPos. Если это так, переменной Time2Stop присваивается значение, равное 1.

16. Двадцать девятая строка является последней строкой цикла WHILE WEND, который начался с тринадцатой строки.

17. Тридцатая строка является последней строкой цикла FOR NEXT, который начался с восьмой строки.

18. Строки с тридцать первой по тридцать четвертую выводят на экран окончательный вариант массива вместе с сообщением Sorted array (Отсортированный вариант массива).

19. Тридцать пятая строка сообщает компьютеру о завершении программы.

Сортировка пузырьковым методом

Алгоритм пузырьковой сортировки получил свое название от того, что при его использовании отдельные элементы как будто "переносятся" на свои места. Алгоритм пузырьковой сортировки постоянно исследует массив, упорядочивая соседние элементы, до тех пор пока не отсортирует весь массив целиком, как показано на рис. 20.2. Компьютер выполняет алгоритм пузырьковой сортировки следующим образом.

1. Сначала сравниваются и упорядочиваются два первых элемента в списке.
2. Затем компьютер переходит к следующему элементу в списке и сортирует его по отношению к последнему элементу из раньше отсортированной пары.
3. Для каждого последующего элемента в списке повторяется п. 2 до тех пор, пока не будет исследован весь список.
4. Повторяются п. 1—3 до тех пор, пока не будет отсортирован весь список.

Один из недостатков алгоритма пузырьковой сортировки заключается в необходимости многократной проверки всего списка целиком до полной сортировки всего списка (см. рис. 20.2).

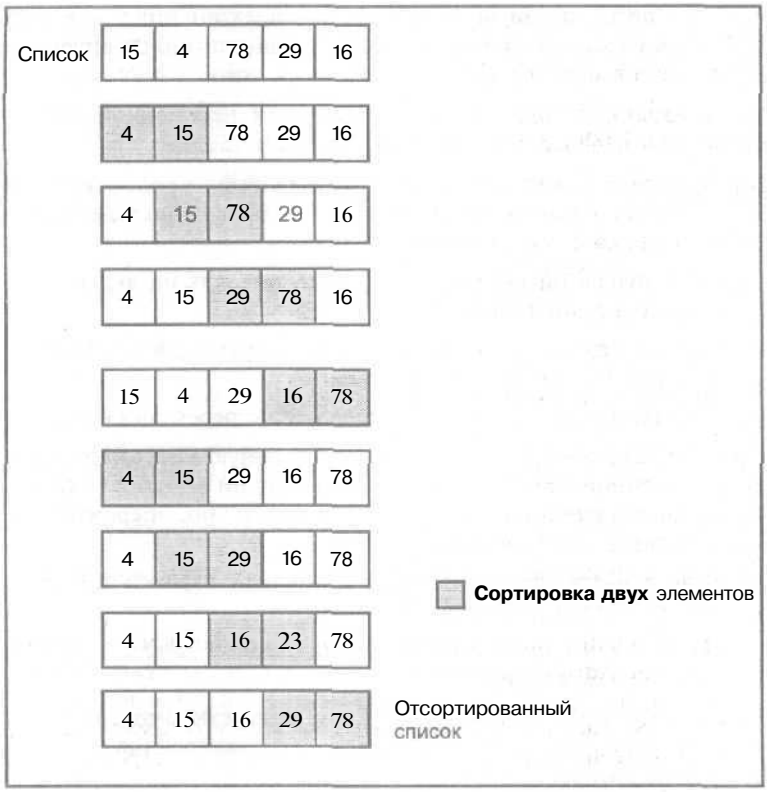

Рис. 20.2. Алгоритм пузырьковой сортировки исследует каждый элемент в списке и сортирует его по отношению к соседним элементам

Для того чтобы увидеть пузырьковую сортировку в действии, попробуйте выполнить следующую программу:

```
MaxSize = 5
REDIM MyArrau(MaxSize)
FOR I = 1 TO MaxSize
  MyArrya(I) = INT(RND(1) * 100) + 1
   PRINT MyArray(I) ; SPACE$(1);
NEXT I
PRINT "(Initial array)"

Pass = 1
Time2Stop - 0
WHILE (Time2Stop) = 0
  NoSwap = 1
  FOR I = 1 TO (MaxSize - Pass)
     IF MyArrayd) > MyArray(I + 1) THEN
        TempValue = MyArray(I)
        MyArray(I) = MyArrayd + 1)
        MyArray(I + 1) = TempValue
        NoSwaps = 0
        FOR J = 1 TO MaxSize
           PRINT MyArray(J); SPACE$(1);
```

```
        NEXT J
        PRINT
      END IF
  NEXT I
  IF NoSwaps = 1 THEN
     Time2Stop - 1
       END IF
  WEND
FOR I = 1 TO MaxSize
  PRINT MyArray(I); SPACE$(1);
NEXT I
PRINT "(Sorted array)"
END
```

Результат выполнения программы имеет следующий вид:

```
5 19 61 26 27 (Initial array)
5 19 26 61 27
5 19 26 27 61
5 19 26 27 61 (Sorted array)
```

Работа программы, использующей алгоритм пузырьковой сортировки, подробно описана ниже.

1. Строки с первой по седьмую создают переменную MaxSize, значение которой равно 5; создают массив MyArray для хранения пяти целых чисел; генерируют произвольное число; создают произвольное число от 1 до 100 и сохраняют его в массиве MyArray; после чего выводят массив на экран вместе со строкой Initial array (Первоначальный вид массива).
2. Восьмая строка создает переменную Pass и присваивает ей значение 1.
3. Девятая строка создает переменную Time2Stop и присваивает ей нулевое значение.
4. Десятая строка является началом цикла WHILE WEND, который проверяет равенство нулю значения переменной Time2Stop. Если это условие справедливо, все инструкции цикла выполняются.
5. Одиннадцатая строка создает переменную NoSwap и присваивает ей нулевое значение.
6. Двенадцатая строка является началом цикла FOR NEXT, который выполняется (5-Pass) раз. При первом запуске цикл выполняется четыре раза, при втором — три и т.д.
7. Тринадцатая строка приказывает компьютеру сравнить значение элемента массива со значением следующего элемента. При первом запуске цикла IF THEN сравниваются первый и второй элементы массива MyArray.
8. Строки с четырнадцатой по шестнадцатую меняют местами два элемента из массива MyArray.
9. Семнадцатая строка присваивает переменной NoSwaps нулевое значение. Эта строка сообщает алгоритму пузырьковой сортировки о том, что произошла перестановка элементов списка, поэтому цикл WHILE WEND нужно выполнить еще раз.

10. Строки с восемнадцатую по двадцать первую выводят на экран частично отсортированные варианты массива.

11. Двадцать вторая строка станет последней строкой цикла IF THEN, который начался с четырнадцатой строки.

12. Двадцать третья строка станет последней строкой цикла FOR NEXT.

13. Строки с двадцать четвертую по двадцать шестую, используя цикл IF THEN, проверяют, равно ли значение переменной NoSwap единице. Если это так, переменной Time2Stop присваивается значение, равное 1.

14. Двадцать седьмая строка является последней строкой цикла WHILE WEND. Выполнение цикла прекращается после того, как значение переменной Time2Stop становится равным 1. Это возможно только после того, как список полностью отсортирован.

15. Строки с двадцать восьмой по тридцать первую выводят на экран окончательный вариант массива вместе с сообщением Sorted array (Отсортированный вариант массива).

16. Тридцать вторая строка сообщает компьютеру о завершении программы.

Алгоритм пузырьковой сортировки замечательно подходит для сортировки небольших списков, однако оказывается слишком медленным при необходимости отсортировать большое количество элементов. Хуже всего, что этот алгоритм оказывается особенно медленным в том случае, если несколько малых чисел находятся в конце списка, а это приводит к многократному выполнению алгоритма.

Сортировка методом Шелла

Проблема, связанная с алгоритмами сортировки методом вставки и пузырьковой сортировки, состоит в том, что они часто должны перемещать элемент с самого конца списка в его начала, что особенно заметно в случае пузырьковой сортировки. Для сокращения времени сортировки можно использовать алгоритм сортировки методом Шелла.

Этот алгоритм работает по принципу "разделяй и властвуй". Вместо того чтобы пытаться сразу отсортировать весь список целиком, алгоритм сортировки методом Шелла разделяет большой список на несколько небольших. После сортировки небольших списков он объединяет их в один большой отсортированный список.

На самом деле алгоритм сортировки методом **Шелла** не выполняет никакой сортировки; он работает совместно с существующим алгоритмом сортировки (вставки или пузырьковой сортировки) для ускорения общего процесса сортировки.

В общих чертах алгоритм сортировки методом Шелла работает следующим образом.

1. Он разделяет длинный список на несколько списков меньшего размера. (На рис. 20.3 показан список, разделенный на три небольших списка. Для создания таких списков алгоритм сортировки методом Шелла использует каждый третий элемент большого списка.)

2. Каждый из полученных списков сортируется с помощью одного из алгоритмов (вставки или пузырьковой сортировки). В примере, показанном на рис. 20.3, первый мини-список содержит числа 15 и 29, которые сортировать не нужно. Второй мини-список содержит числа 16 и 4, поэтому нуждается в сортировке. Третий мини-список содержит только число 78.

3. Все три мини-списка объединяются снова в один большой список. Обратите внимание (см. рис. 20.3), что числа 4 и 16 уже отсортированы.

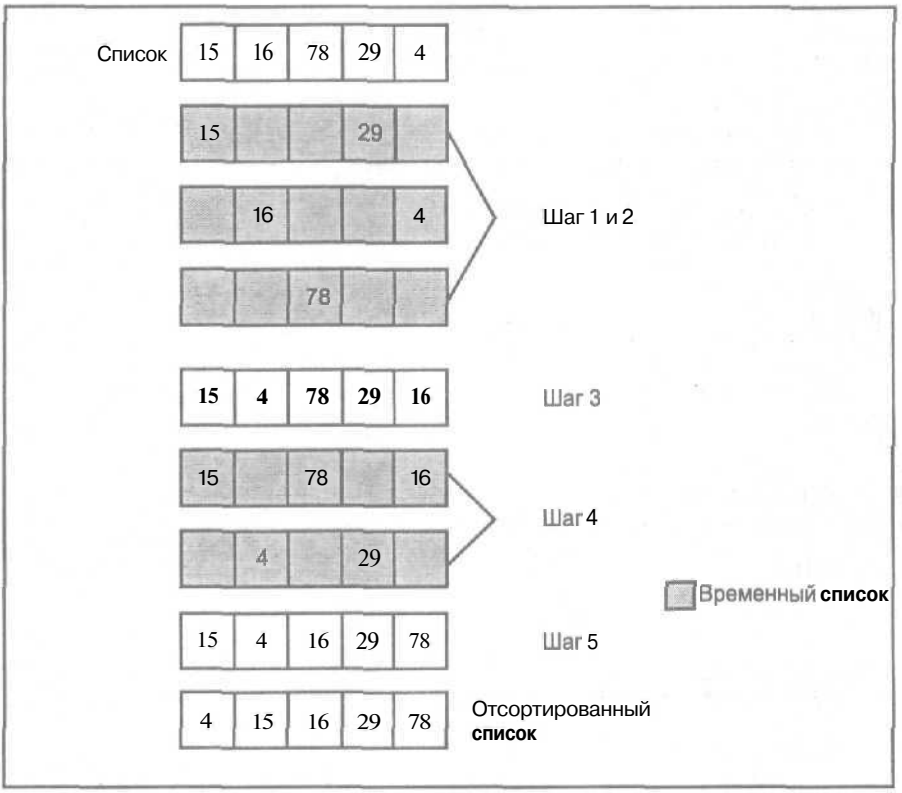

Рис. 20.3. Алгоритм сортировки методом Шелла разделяет большой список на несколько небольших, после чего сортирует каждый из них

4. Он разделяет длинный список на несколько списков меньшего размера, но количество списков уменьшается. На рис. 20.3 показан список, разделенный на два небольших списка; при их создании использовался каждый второй элемент списка.
5. Повторяются п. 2–4 (если это необходимо) до тех пор, пока не останется всего один полностью отсортированный список. Обратите внимание на то, что после сортировки чисел 16 и 78 становится отсортированным и весь список.

Для того чтобы увидеть алгоритм сортировки методом Шелла в действии, выполните следующую программу, которая использует этот алгоритм для разделения списка на несколько небольших списков, а затем применяет алгоритм пузырьковой сортировки к каждому из них.

```
MaxSize = 5
REDIM MyArray(MaxSize)
FOR I = 1 TO MaxSize
  MyArray(I) = INT(RND(1) * 100) + 1
  PRINT MyArray(I); SPACE$(1);
NEXT I
```

```
PRINT "(Initial array)"

X = INT(MaxSize / 2)
WHILE X > 0
  Time2Stop = 0
  Limit = MaxSize - X
  WHILE (Time2Stop = 0)
     Switch = 0
     FOR K = 1 TO Limit
        IF MyArray(K) > MyArray(K + X) THEN
           TEMPX - MyArray(K)
           MyArray(K) = MyArray(K + X)
           MyArray!K + X) = TempX
           Switch = K
        END IF
     NEXT K
    - Limit = Switch - X
     IF Switch = 0 THEN
        Time2Stop = 1
     END IF
  WEND

  FOR I = 1 TO MaxSize
     PRINT MyArray(I); SPACE$(1);
  NEXT I
        PRINT
        X = INT(X / 2)
WEND

FOR I = 1 TO MaxSize
  PRINT MyArray(I); SPACE$(1);
NEXT I
PRINT "(Sorted array)"
END
```

Результат выполнения этой программы имеет приблизительно следующий вид:

```
94 17 70 90 62 (Initial array)
62 17 70 90 94
17 62 70 90 94
17 62 70 90 94 (Sorted array)
```

При первом запуске программы алгоритм сортировки методом Шелла сравнивает числа, содержащиеся в ячейках 1, 3 и 5 массива (94, 70 и 62 соответственно). После сортировки списка выполняется сортировка чисел, содержащихся в ячейках 2 и 4 массива (17 и 90 соответственно). После этого проводится сортировка всего списка.

Чтобы лучше разобраться в работе программы, давайте определим, что же делает каждая ее строка.

1. Строки с первой по седьмую создают переменную MaxSize, значение которой равно 5; создают массив MyArray для хранения пяти целых чисел; генерируют произвольное число; создают произвольное число от I до 100 и сохраняют его в массиве MyArray; после чего выводят массив на экран вместе со строкой Initial array (Первоначальный вид массива).

2. Восьмая строка создает переменную X и разделяет общее количество элементов списка на 2, после чего присваивает полученное целое значение переменной x. В нашем примере значение переменной X равно 2 (MaxSize/2 = 2), что означает, что большой список нужно разделить на два списка.
3. Девятая строка начинает цикл WHILE WEND, который выполняется до тех пор, пока значение переменной X будет больше нуля.
4. Десятая строка создает переменную Time2Stop и присваивает ей нулевое значение.
5. Одиннадцатая строка создает переменную Limit и присваивает ей значение MaxSize-X. При первом выполнении строки значение переменной Limit равно MaxSize-2 или 3.
6. Двенадцатая строка является началом цикла WHILE WEND, она проверяет равенство нулю значения переменной Time2Stop. Если это условие справедливо. все инструкции цикла выполняются,
7. Тринадцатая строка создает переменную Switch и присваивает ей нулевое значение.
8. Четырнадцатая строка является началом цикла FOR NEXT.
9. Пятнадцатая строка приказывает компьютеру сравнить значение элемента массива со значением следующего элемента. При первом запуске цикла IF THEN сравниваются первый и второй элементы массива MyArray.
10. Строки с шестнадцатой по восемнадцатую меняют местами два элемента из массива MyArray.
11. Девятнадцатая строка присваивает значение переменной Switch переменной к.
12. Двадцать первая строка является последней строкой цикла IF THEN, который начался с пятнадцатой строки.
13. Двадцать вторая строка станет последней строкой цикла FOR NEXT.
14. Двадцать третья строка присваивает переменной Limit значение Switch-X.
15. Строки с двадцать четвертой по двадцать шестую проверяют, равно ли значение переменной Switch нулю. Если это так, переменной Time2Stop присваивается значение, равное 1.
16. Двадцать седьмая строка является последней строкой цикла WHILE WEND.
17. Строки с двадцать восьмой по тридцать первую выводят на экран частично отсортированные варианты массива.
18. Тридцать вторая строка делит значение переменной X на 2 и присваивает полученный результат переменной x. Это сообщает алгоритму сортировки методом Шелла о том, на сколько небольших списков нужно разделить большой список.
19. Тридцать третья строка является последней строкой цикла WHILE WEND.
20. Строки с тридцать четвертой по тридцать седьмую выводят на экран окончательный вариант массива вместе с сообщением Sorted array (Отсортированный вариант массива).
21. Тридцать восьмая строка сообщает компьютеру о завершении программы.

Быстрая сортировка

Одним из самых популярных алгоритмов сортировки стал алгоритм *быстрой сортировки*. Метод быстрой сортировки работает следующим образом: из центра списка извлекается одно число, после чего сортируются числа, находящиеся слева или справа от него, как показано на рис. 20.4.

Рис. 20.4. Метод быстрой сортировки делит большой список на небольшие списки по отношению к числу, выбранному в середине списка

После разделения пополам первоначального списка алгоритм быстрой сортировки делит пополам каждую половину списка, произвольным образом выбирая число в середине каждой из них. После разделения списка на много частей и сортировки каждой из них все части объединяются снова в один большой список.

Метод быстрой сортировки работает следующим образом.

1. Выбирается число из середины списка, которое используется для разделения списка на две части. Все числа, которые меньше выбранного числа, оказываются слева, а числа, которые больше его, — справа.
2. Для каждой части, получаемой в результате выбора произвольного числа, выполняется п. 1 до тех пор, пока все элементы списков меньшего размера не окажутся упорядоченными.
3. Все небольшие части объединяются в один список.

Поскольку алгоритм быстрой сортировки повторяет одни и те же шаги для списков всего меньшего и меньшего размера, он использует прием, известный как *рекурсия*. Рекурсия означает, что подпрограмма последовательно запускает сама себя.

Алгоритм быстрой сортировки должен использовать рекурсию, поэтому его следует вынести в отдельную подпрограмму. Таким образом, полная программа быстрой сортировки состоит из основной программы и подпрограммы, как показано ниже:

```
MaxSize = 5
REDIM NumArray (MaxSize)
; FOR I = 1 TO MaxSize
   NumArray (I) = INT(RND(1)*10) + 1
   PRINT NumArray(I) ; " ";
NEXT I
PRINT "(Initial array"
```

```
CALL QSort 1, MaxSize

FOR I = 1TO MaxSize
  PRINT NumArray(I); "";
NEXT I
PRINT "(Sorted array)"
END
```

Основная программа быстрой сортировки выполняет следующее.

1. Строки с первой по седьмую создают массив, содержащий пять произвольных целых чисел, и выводят его на экран.
2. Восьмая строка вызывает подпрограмму QSort, передавая ей значение первого элемента (1) и максимального размера (MaxSize) списка.
3. Строки с девятой по двенадцатую выводят на экран окончательный отсортированный вариант массива.
4. Тринадцатая строка сообщает компьютеру о завершении программы.

Подпрограмма QSort выглядит следующим образом:

```
SUB QSort Start, Finish
  I = Start
  J - Finish
  X = NumArray(INT((I+J/2))
  WHILE, I <= J
         WHILE NumArray(I) < X
       I = I + 1
    WEND
    WHILE NumArray(J) > X
      J = J - 1
    WEND
    IF I <= J THEN
      A = NumArray(I)
      NumArray(I) = NumArray(J)
           NumArray(J) = A
      I = I + 1
      J = J - 1
        END IF
      WEND
  FOR K = 1 TO Finish
    PRINT NumArray(K); "";
  NEXT K
  PRINT
  IF J > Start THEN CALL QSort Start, J
  IF I < Finish THEN CAKK QSort I, Finish
END SUB
```

Подпрограмма QSort работает следующим образом.

1. Первая строка определяет имя подпрограммы (QSort) и данные, необходимые для ее работы (Start и Finish).
2. Вторая и третья строки создают две переменные (I и J), которым присваиваются значения переменных start и Finish соответственно. Переменные I и J необходимы подпрограмме, потому что их значения при выполнении подпрограммы изменяются, а значения переменных Start и Finish — не изменяются.

3. Четвертая строка создает переменную X, которая разделяет массив NumArray пополам, а ее значение становится равным целой части полученного результата.
4. Пятая строка начинает цикл WHILE WEND, который выполняется до тех пор, пока значение переменной *I* будет меньше или равно значения переменной J.
5. Строки с шестой по восьмой увеличивает значение переменной I на единицу до тех пор, пока число, которое содержится в массиве NumArray, будет больше значения переменной X.
6. Строки с девятой по одиннадцатую увеличивают значение переменной J на единицу до тех пор, пока число, которое содержится в массиве NumArray, будет больше значения переменной X. В этом месте программа пытается определить, какие числа меньше, а какие больше числа, выбранного из середины списка при выполнении четвертой строки.
7. Строки с двенадцатой по восемнадцатую сравнивают числа в списке и перемещают их влево или вправо по отношению к числу, выбранному из середины списка при выполнении четвертой строки.
8. Девятнадцатая строка является последней строкой цикла WHILE WEND, который начался с пятнадцатой строки.
9. Строки с двадцатой по двадцать третью выводят на экран частично отсортированные варианты массива.
10. Строки с двадцать четвертой по двадцать пятую рекурсивно запускают подпрограмму QSort, каждый раз предоставляя ей массивы все меньшего и меньшего размера.
11. Двадцать шестая строка сообщает компьютеру о завершении подпрограммы.

Типичный результат выполнения программы имеет такой вид:

```
27 62 5 79 14 (Initial array)
5 62 27 79 14
5 14 27 79 62
5 14 27 62 79
5 14 27 62 79 (Sorted array)
```

При первом запуске программы алгоритм быстрой сортировки выбирает из массива третье число (5). Затем он сортирует оставшиеся числа, в зависимости от того, меньше они или больше, чем 5. Так как все они больше 5, эти числа сохраняются в правой части массива. Затем алгоритм выбирает из массива число (27). Затем он сортирует оставшиеся числа, в зависимости от того, меньше они или больше, чем 27.

Теперь третий оставшийся список меньшего размера состоит из чисел 79 и 62. После их упорядочения алгоритм объединяет все полученные списки вместе.

Выбор алгоритма сортировки

Теперь вы научились при сортировке данных использовать в своих программах алгоритмы сортировки методом вставки, быстрой сортировки, сортировки методом Шелла или быстрой сортировки. Конечно же, ученые-кибернетики постоянно изобретают все новые и новые алгоритмы сортировки с теми или иными преимуществами и недостатками, поэтому относиться к выбору алгоритма сортировки следует достаточно осторожно. Выберите нужный алгоритм, и ваша программа будет работать быстро. Выберите неудачный алгоритм, и пользователь посчитает вашу программу слишком медленной.

Запомните следующие общие рекомендации. Метод вставки лучше всего подходит для сортировки небольших списков. Алгоритм пузырьковой сортировки лучше всего подходит для списков, которые уже каким-то образом упорядочены, а алгоритм быстрой сортировки больше всего подходит для повседневного использования. Для ускорения работы алгоритма методом вставки и пузырьковой сортировки подумайте об их объединении с алгоритмом сортировки методом Шелла.

Глава 21

Поиск

В этой главе...

- Последовательный поиск
- Двоичный поиск
- Хэширование
- Выбор метода поиска

Поиск данных — это еще одна операция (после сортировки данных), используемая программами различного типа. Например, программу для сортировки имен и адресов можно использовать для поиска людей, фамилии которых начинаются на букву М и которые проживают в Киеве.

Для того чтобы упростить поиск, сначала проводится сортировка данных. Более подробная информация о сортировке приведена в главе 20.

Наилучший способ решить задачу — создать *алгоритм*, с помощью которого можно дать компьютеру инструкции для выполнения задачи. Выбор правильного способа сортировки и алгоритма поиска делает работу программы более быстрой и эффективной. Если же вы используете неправильный алгоритм поиска и сортировки данных, программа будет работать очень медленно, независимо от объема данных.

Последовательный поиск

При выполнении *последовательного поиска* проводится просмотр всех элементов в структуре данных (таких как массив или связанный список). Просмотр проводится до тех пор, пока не найден необходимый элемент. Такой тип поиска чем-то похож на поиск ключей от машины в огромном доме, когда вы мечетесь из комнаты в комнату. переворачивая все на своем пути. Несмотря на то, что последовательный поиск в конечном итоге приведет к положительному результату (особенно, если дом состоит всего из нескольких комнат), выполнять его придется достаточно долго.

Если вы проводите последовательный поиск в небольшом списке, необходимый элемент найдется достаточно быстро. Однако как только вы попытаетесь провести последовательный поиск в большом объеме данных, вы столкнетесь с тем, что поиск проходит достаточно медленно. С таким же успехом можно искать ключи от машины по всему Киеву.

Выполнять последовательный поиск можно как с начала, так и с конца списка. При этом поиск проводится до тех пор, пока не найден необходимый элемент. После этого поиск останавливается. Для того чтобы понять, каким образом выполняется последовательный поиск, попытайтесь запустить следующую программу, написанную на языке программирования Liberty BASIC:

```
MaxSize = 5
REDIM MyArray(MaxSize)
MyArray(1) = INT(RND(1) * 10) + 1
```

```
PRINT MyArray(1); SPACE$(1);

FOR I = 2 TO MaxSize
  MyArray(I) = MyArray(I - 1) + INT(RND(1) * 10) + 1
  PRINT MyArray(I); SPACE$(1);
NEXT I
PRINT
INPUT "Which number do you want to find: "; FindMe

FoundIt = 0
FOR J = 1 TO MaxSize
  IF FounsIt = 0 THEN
     PRINT "Checking array location "; J
     IF MyArray(J) = FindMe THEN
        FounDIt = 1
     END IF
  END IF
NEXT J

IF Foundlt = 1 THEN
  PRINT "Found it!"
ELSE
  PRINT "The number you want is not the list."
END IF
END
```

Ниже описано, что делает каждая строка программы.

1. Первая и вторая строки создают переменную MaxSize, устанавливают, что значение переменной MaxSize равно 5 и задают массив MyArray, в котором содержатся пять (значение переменной MaxSize равно пяти) целых чисел.
2. С третьей по девятую строку создаются случайные числа, которые сохраняются в массиве MyArray. При этом каждое следующее произвольно выбранное случайное число будет больше, чем предыдущее. Затем группа линий отобразит весь массив на экране для проверки.
3. Десятая строка предлагает пользователю ввести число, которое необходимо найти. Это число будет сохранено в переменной FindMe.
4. Одиннадцатая строка создает переменную Foundlt и присваивает ей значение 0.
5. С двенадцатой по девятнадцатую строку в массиве MyArray проводится поиск числа, сохраненного в переменной FindIt. При этом печатается каждый проверяемый элемент.
6. С двадцатой по двадцать пятую строку печатается сообщение Found it! (Найдено!), **если искомое** число **найдено.** Если **это число не найдено, выдается** сообщение: The number you want is not the list. (Число, которое вы ищете, в списке отсутствует.).
7. Двадцать шестая строка заканчивает выполнение программы.

Одним из достоинств последовательного поиска является то, что можно работать как с сортированными, так и несортированными списками.

Двоичный поиск

Последовательный поиск начинается с начала списка и продолжается до тех пор, пока нужный элемент списка не найден. Если список уже отсортирован, можно прибегнуть к *двоичному поиску*. (Более подробная информация о сортировке данных приводится в главе 20.)

При выполнении двоичного поиска длинный список (предварительно отсортированный) делится на две части. Предположим, есть отсортированный список, содержащий десять элементов, которые размешены от меньшего элемента (слева) к большему (справа). Для того чтобы провести двоичный поиск в данном списке, его необходимо разделить на две части (пять элементов справа и пять элементов слева).

На рис. 21.1 показан пример двоичного поиска числа 37 в отсортированном списке, состоящем из десяти элементов. Сначала алгоритм выполнения двоичного поиска делит длинный список на две части и проводит поиск искомого числа в центре списка. Так как список состоит из десяти элементов, двоичный поиск происходит в левых пяти элементах списка. Если центральное (пятое) число равно **30,** значит искомое число (37) должно находиться в правой половине списка.

После этого правая половина списка (состоящая из пяти элементов) делится на две части. Центральное число равно **59.** Так как число **59** больше, чем искомое число 37, поиск продолжается в левой части данной половины списка.

В левой части списка два числа — 37 и 45. При выполнении двоичного поиска необходимо проверить только первое число данного списка. К нашему счастью, это как раз и есть искомое число. Поиск завершен.

Рис. 21.1. При двоичном поиске список делится на две части до тех пор, пока не будет найдено необходимое число

Для того чтобы понять, как выполняется двоичный поиск, запустите следующую программу:

```
MaxSize = 5
REDIM MyArray(MaxSize)
MyArray(1) = INT(RND(1) * 10) + 1
PRINT MyArray(1); SPACE$(1);
FOR I = 2 TO MaxSize
  MyArray(I) = MyArray(I - 1) + INT(RND(1) * 10} + 1
```

```
   PRINT MyArray(I); SPACE$(1);
NEXT I
PRINT

INPUT "Which number do you want to find: "; FindMe
Left = 1
Right = MaxSize
Time2Stop = 0
WHILE Time2Stop = 0
   Half = INT((Left + Right) / 2)
   IF FindMt < MyArray(Half) THEN
      Right = Half - 1
   ELSE
      Left = Half + 1
   END IF
   IF (FindMe = MyArray(Half) OR Left > Right) THEN
      Time2Stop = 1
   END IF
.WEND
 IF FindMe = MyArray(Half) THEN
   PRINT "Found it in location "; Half
!ELSE
   PRINT "The number you want is not the list."
 END IF
 END
```

Ниже описано, что делает каждая строка программы.

1. Первая и вторая строки создают переменную MaxSize, устанавливают, что значение переменной MaxSize равно 5 и задают массив MyArray, в котором содержатся пять (значение переменной MaxSize равно пяти) целых чисел.
2. С третьей по девятую строку создаются случайные числа, которые сохраняются в массиве MyArray. При этом каждое следующее произвольно выбранное случайное число будет больше, чем предыдущее. Затем группа линий отобразит весь массив на экране для проверки.
3. Десятая строка предлагает пользователю ввести число, которое необходимо найти. Это число будет сохранено в переменой FindMe.
4. Одиннадцатая строка создает переменную Left и присваивает ей значение 1.
5. Двенадцатая строка создает переменную Right и присваивает ей значение переменной MaxSize, которое равно максимальному размеру массива.
6. Тринадцатая строка создает переменную Time2Stop и присваивает ей значение 0.
7. Четырнадцатая строка запускает цикл WHILE WEND, который выполняется до тех пор, пока в массиве будет найдено число, сохраненное в переменой FindMe, или до тех пор, пока это число не будет найдено.
8. Пятнадцатая строка создает переменную Half, которая делит список пополам и сохраняет результат в виде целого числа.

9. С шестнадцатой по двадцатую строки в левой половине массива проводится поиск числа, сохраненного в переменной FindMe. Если сохраненное в переменной FindMe число меньше, чем центральное число массива, значение переменной Right равно Half-1. В противном случае значение переменной Left равно Half+1. Уменьшая значение переменной Right и увеличивая значение переменной Left, программа просматривает все числа в массиве. Как только значение переменной Left станет больше, чем значение переменной Right, программа "поймет", что в данном массиве искомого числа нет.
10. С двадцать первой по двадцать третью строки выясняется, есть ли в массиве число, содержащееся в переменной FindMe. В этом случае значение переменной Time2Stop станет равно 1.
11. Двадцать четвертая строка заканчивает исполнение цикла WHILE WEND.
12. С двадцать пятой до двадцать девятой строки печатается сообщение Found it in location, если искомый элемент найден. В случае если искомый элемент не найден, выдается сообщение The number you want is not the list.
13. Тридцатая строка заканчивает выполнение программы.

Двоичный поиск можно выполнять только в отсортированном списке.

Хэширование

Найти что-то намного проще, если вы знаете, где вы видели это нечто последний раз. Намного проще найти ключи от машины, если вы всегда вешаете их на крючочек возле двери, чем искать их по всему дому.

Хэширование работает на простом принципе. Если необходимо сохранить элемент, сначала подсчитывается числовое значение (оно называется *хэш-функцией*), которое идентифицирует данный элемент. Затем программа использует это цифровое значение для сохранения элемента в определенном месте в структуре данных (массиве или связанном списке). Теперь вместо того, чтобы проводить поиск элемента по всему связанному списку или массиву, программа просто просматривает определенное место, используя хэш-функцию.

Предположим, необходимо сохранить целое число в массиве. Если сохранить целое число в любом месте массива, для его поиска в дальнейшем придется просмотреть весь массив. С помощью хэширования задача намного упрощается. Сначала с помощью следующей формулы подсчитывается хэш-функция:

```
' HashValue = Число MOD 5
```

Данная формула говорит компьютеру, что сохраняемое число необходимо разделить на пять, а остаток деления использовать в качестве хэш-функции. Таким образом, если вы хотите сохранить число 26, хэш-функция будет 1 (26 / 5 = 5 и остаток 1).

Команда MOD говорит компьютеру, что необходимо разделить данное число и вернуть остаток. Например, команда 26 MOD 5 возвращает значение 1. Так как язык программирования Liberty BASIC не поддерживает использование команды MOD, вместо нее необходимо использовать следующую формулу:

```
HashValue =Число - (INT(Number to Store / 5) * 5
```

Независимо от того, какое число вы сохранили, хэш-функция подсчитает одно из следующих пяти значении: О, 1, 2, 3 или 4. Можно создать массив и использовать хэш-

функцию для определения точного места, в котором сохранен элемент. Хэш-функция для число 26 равна 1, поэтому можно сохранить данное целое число в качестве первого элемента массива, как показано на рис. 21.2.

Значение хэш-функции определяет, где сохраняется элемент в структуре данных.

Если понадобится снова найти элемент 26 в массиве, хэширование рассчитает ту же самую хэш-функцию (равную 1) и скажет программе, что число 26 — первый элемент массива. Для хэширования не требуется просмотр всех элементов списка, а это обеспечивает более быстрый поиск информации, чем при последовательном и двоичном поиске (особенно, если список достаточно длинный).

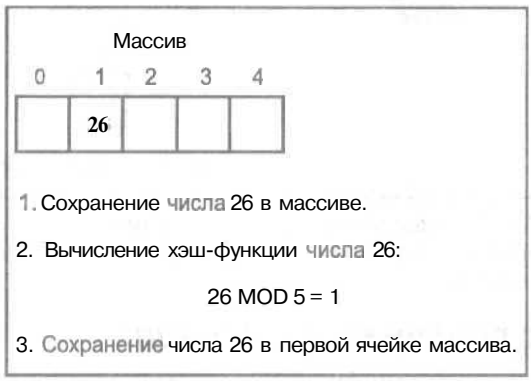

Рис. 21.2. Хэширование рассчитывает значение, определяющее расположение сохраненного числа, что помогает в дальнейших поисках этого числа

Возможные проблемы

В идеале хэш-функция рассчитывает уникальное число. Однако есть вероятность того, что хэш-функция рассчитывает одно и то же число для различных элементов. Например, рассчитываем хэш-функцию с помощью команды MOD 5 (число делится на 5, а в качестве хэш-функции используется остаток деления) для чисел 7 и 32, а в обоих случаях остаток деления равен 2.

Если вы сохраняете много элементов, высока вероятность того, что несколько элементов будут иметь одну и ту же хэш-функцию. Если у двух различных элементов хэш-функция одинаковая, возникает *конфликт*. Для того чтобы разрешить конфликтную ситуацию, необходимо создать структуру данных, в которой можно сохранять несколько элементов с одной и той же хэш-функцией. На рис. 21.3 показаны две структуры данных, позволяющие разрешить данный конфликт, — двумерный массив и одномерный массив связанных списков. (В главе 16 приведена подробная информация о создании и использовании двумерных массивов. В главе 18 приведена информация о связанных списках. Запомните, что программа Liberty BASIC не может создавать связанные списки. Однако другие языки программирования — *C/C++*, C#, Pascal и Java — позволяют создавать связанные списки.)

На рис. 21.3 числа 7 и 32 имеют одинаковую хэш-функцию (она равна 2). Поскольку число 7 располагается в массиве под номером 2 (что соответствует хэш-функции 2), программа должна сохранить число 32 ниже числа 7. Как двумерный, так

и одномерный массивы позволяют профамме сохранять несколько чисел с одинаковой хэш-функцией одно под другим.

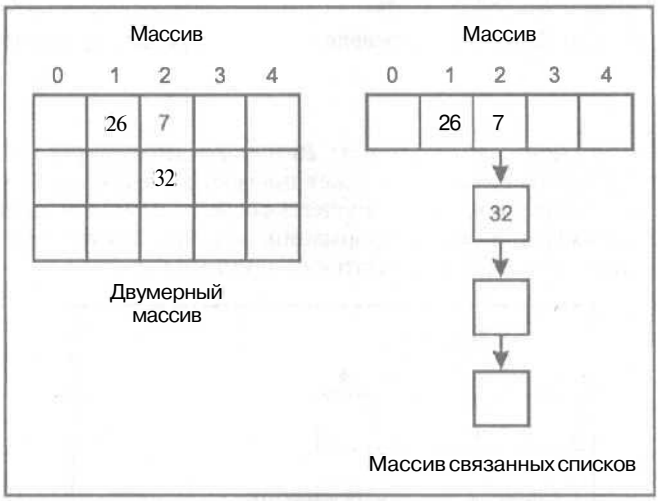

Рис. 21.3. С помощью двумерного и одномерного массивов связанных списков можно разрешить возникший конфликт

Поиск данных с помощью хэш-функции

После того как с помощью хэш-функции проведена сортировка элементов в структуре данных, можно найти любой элемент, рассчитав его хэш-функцию. После расчета хэш-функции необходимо найти элемент в структуре данных, обладающий такой же хэш-функцией.

Если каждому элементу соответствует уникальная хэш-функция, положение этого элемента легко определить. Если же одна хэш-функция соответствует нескольким элементам (см. рис. 20.2), поиск ограничится элементами с одинаковой хэш-функцией. В результате при хэшировании поиск происходит намного быстрее, чем при последовательном или двоичном поиске.

С помощью следующей программы, написанной на языке Liberty BASIC, создается пять случайных целых чисел — 41, **50,** 57, 75, **67,** которые сохраняются в двумерном массиве, как показано на рис. 21.4.

Рис. 21.4. Двумерный массив позволяет сохранить значения, разместив их в соответствии с их хэш-функциями

При создании массива в Liberty BASIC первая координата определяется первым числом, вторая — вторым и т.д. Таким образом, для того чтобы сымитировать массив, в котором содержатся хэш-функции для чисел от 0 до 4, можно воспользоваться следующей программой и добавить 1 к значению функции:

```
MaxSize = 5
REDIM MyArray (MaxSize, MaxSize)
 FOR I = 1 TO MaxSize ' Vertical
   FOR J =1 TO MaxSize   '  Horizontal
   MyArray (I, J) = 0
   NEXT J
NEXT I -

Count = 1
1 FOR J - 1 TO MaxSize
   StopNow = 0
   StoreMe = INT(RND(1) * 100) + 1
   HashValue = StoreMe - (INT(StoreMe / 5 * 5) + 1
   WHILE StopNow <> 1
      IF MyArray (Count, HashValue) - 0 THEN
        MyArray (Count, HashValue) = StoreMe
        StopNow = 1
      ELSE
        Count = Count + 1
      END IF
   WEND
   PRINT StoreMe; SPACE$(1);
NEXT J
 PRINT
 PRINT

FOR I = 1 TO MaxSize ' Vertical
   FOR J = 1 TO MaxSize ' Horizontal
      PRINT MyArray (I, J) ; SPACE$(5);
   NEXT J
   PRINT
NEXT I
END
```

Вот что делает каждая строка программы.

1. Первая и вторая строки создают переменную MaxSize, устанавливают, что значение переменной MaxSize равно 5. Затем они создают двумерный массив MyArray, в котором может храниться двадцать пять (5 x 5) целых чисел.
2. С третьей по седьмую строки массив MyArray заполняется нулями. Обратите внимание на то, что переменная I определяет номер линии, а переменная J — номер столбца, как показано на рис. 21.4. *(Запомните:* в массивах, созданных с помощью языка программирования Liberty BASIC, счет всегда начинается с единицы. Поэтому элемент с номером 0, показанный на рис. 21.4, в действительности соответствует элементу с номером 1 в программе Liberty BASIC; элемент с номером 1 в действительности соответствует элементу с номером 2 в программе Liberty BASIC и т.д.)
3. Восьмая строка создает переменную Count и присваивает ей значение 1.
4. Девятая строка начинает выполнение цикла FORNEXT, с помощью которого запускается алгоритм сохранения целого числа в двумерном массиве.

5. Десятая строка создает переменную stopNow и присваивает ей значение 0.
6. Одиннадцатая строка создает случайное число от 1 до 100 и сохраняет его в переменной StoreMe.
7. Двенадцатая строка рассчитывает хэш-функцию и сохраняет ее в переменной HashValue. Переменная HashValue может иметь следующие значения: 1, 2, 3, 4 или 5.
8. Тринадцатая строка начинает выполнение цикла WHILE WEND, определяя, где в массиве MyArray сохранено число, содержащееся в переменной StoreMe.
9. С четырнадцатой по девятнадцатую строки программа пытается сохранить число из переменной StoreMe в первой строке (она определяется переменной Count) массива MyArray в столбце, который определяется значением переменной HashValue. Если число уже сохранено в этом месте, программа пытается сохранить число в следующей строке. Это продолжается до тех пор, пока не найдено свободное место в массиве.
10. Двадцатая строка заканчивает выполнение цикла WHILE WEND, запущенного в тринадцатой строке.
11. Двадцать первая строка печатает пять случайно созданных программой чисел.
12. Двадцать вторая строка завершает выполнение цикла FOR NEXT, запущенного в девятой строке.
13. Двадцать третья и двадцать четвертая строки печатают две пустые строки.
14. С двадцать пятой по тридцатую строки печатается содержимое массива таким образом, что можно увидеть, как проведена сортировка пяти случайных чисел, созданных программой.
15. Тридцать первая строка заканчивает выполнение программы.

Выбор метода поиска

Понять, как выполняется последовательный поиск, проще всего. Однако при последовательном поиске надеяться на быстрый результат можно только в том случае, если список невелик. Как только список становится больше, время поиска значительно увеличивается. Двоичный поиск приводит к желаемому результату несколько быстрее, чем последовательный. Основным недостатком двоичного поиска является то, что его можно выполнять только в предварительно отсортированных данных.

Хэширование лучше всего подходит для больших объемов данных. Однако при хэшировании необходимо рассчитывать хэш-функцию, которая определяет расположение элемента в массиве. Также вы сталкиваетесь с дополнительной проблемой — два различных элемента могут иметь одинаковую хэш-функцию.

Хочу вам дать совет: используйте последовательный поиск, если вы имеете дело с маленьким списком или неотсортированным списком. В случае отсортированного списка лучше использовать двоичный поиск. Используйте хэширование только в том случае, когда вам не придется постоянно рассчитывать хэш-функцию для каждого элемента.

Глава 22

Оптимизация кода программ

В этой главе...

- Выбор нужной структуры данных
- Выбор нужного алгоритма
- Настройка исходного кода
- Использование быстрого языка программирования
- Оптимизация компилятора

Сама по себе задача заставить программу работать должным образом кажется уже чудом. Однако после того, как вы заставите программу работать таким образом, как она и должна, следующий вопрос состоит в том, использовать ли (или выпустить) программу немедленно или тратить время на ее оптимизацию.

Оптимизация — это попытка достичь следующие три цели (но при этом умудриться не внести в программу никаких ошибок).

- ✓ Заставить программу работать быстрее
- ✓ Сделать объем программы как можно меньше
- ✓ Уменьшить объем памяти, необходимый для работы программы

Обычно компании — разработчики программного обеспечения выпускают новые версии своих продуктов (например, версию 2.0 или 4.0), чтобы захватить большую часть рынка. Через несколько месяцев компании выпускают слегка модифицированные версии программ (например, версию 2.1 или 4.3), чтобы исправить ошибки (что не исключает появления новых ошибок) и оптимизировать программы тем или иным образом. Однако порой складывается впечатление, что разработчикам коммерческих программ об оптимизации кода ничего не известно, поскольку каждой последующей версии для нормальной работы необходимо все больше и больше оперативной памяти и пространства на жестком диске.

Выбор нужной структуры данных

Каждой программе необходимо сохранять данные, поэтому вам просто нужно найти самую подходящую структуру данных для сохранения информации. Массивы очень просто создать, но для их использования вам необходимо знать точное количество элементов, которое в них будет храниться. Если вы сделаете массив слишком маленьким, программа не сможет сохранить все необходимые для ее работы данные. Если вы сделаете его слишком большим, программа займет чересчур много памяти.

Например, если вы создадите большой двумерный массив и сохраните в нем совсем немного элементов, вы потратите зря слишком много памяти. Однако, если вы замените массив графом, вы не потратите ничего зря (хотя при этом вам придется попотеть, чтобы написать инструкции для создания графа и управления им, что намного сложнее создания двумерного массива и управления им).

Еще важнее то, что выбранная вами структура данных оказывает большое влияние на эффективность применения алгоритмов поиска и сортировки в вашей программе. Например, алгоритм поиска в массиве работает намного быстрее, чем переупорядочение указателей в связанном списке.

Подробные сведения о структурах данных вы найдете в главах 16–19. Об указателях и связных списках мы подробно говорили в главе 18.

Выбор нужного алгоритма

Алгоритм сообщает компьютеру, как выполнить ту или иную задачу. Например, представьте себе все способы объяснения друзьям, как добраться из центра к вашему дому. Вы можете порекомендовать им идти по основной дороге, что намного проще выполнить, но требует много времени, или пройти через дворы и по боковым улицам, что займет ощутимо меньше времени, но повысит вероятность запутаться.

Выбор набора инструкций, которые вы даете знакомому, во многом напоминает выбор алгоритма для использования в программе. Например, если вам необходимо отсортировать список, содержащий 30 тысяч имен, алгоритм пузырьковой сортировки окажется медленнее, чем алгоритм быстрой сортировки. (Об алгоритмах пузырьковой сортировки и быстрой сортировки подробно мы говорили в главе 20 "Сортировка".) после того, как вы отсортировали 30 тысяч имен, алгоритм последовательного поиска окажется медленнее, чем алгоритм двоичного поиска. (Об алгоритмах последовательного поиска и двоичного поиска подробно мы говорили в главе 21 "Поиск".)

В качестве еще одного примера выбора подходящего алгоритма можно привести компьютерную игру, отображающую на экране десять лучших результатов. Пока в игру никто не играл, все десять лучших результатов равны нулю. После того как в игру сыграет первый человек, его результат будет отображен в списке первым. Каждый раз, когда в игру сыграет новый человек, игре придется каким-то образом проводить сортировку результатов.

В данном случае наиболее эффективным оказывается алгоритм сортировки методом вставки. После того как игра отобразит два первых результата, этот алгоритм отсортирует их от большего к меньшему. Когда в игру сыграет третий человек, игра сравнит третий результат с двумя предыдущими и поместит его в нужное место. Каждый раз, когда появляется новый результат, алгоритм сортировки методом вставки сравнивает его с полученными результатами для определения необходимого места в списке.

Если вы захотите использовать для сортировки списка десяти лучших результатов алгоритм пузырьковой сортировки, этому алгоритму придется несколько раз исследовать список, сравнивая каждый элемент с его соседями, — а ведь на выполнение подобной операции потребуется намного больше времени, чем на работу алгоритма сортировки методом вставки. В данном случае алгоритм сортировки методом вставки оказывается намного эффективнее.

По мере того как вы будете писать собственные программы, никогда не забывайте, что для решения одной и той же задачи можно использовать несколько разных алгоритмов, но в каждом конкретном случае один из них оказывается быстрее, чем остальные.

Настройка исходного кода

Даже если вы корректно подошли к выбору структур данных и алгоритмов, все равно можно еще оптимизировать программу, если "поколдовать" над ее исходным кодом. Это означает, что вы перепишете части программы таким образом, чтобы они выполнялись быстрее и требовали меньше памяти.

Помещаем наименее вероятное условие в начале

При использовании оператора AND и цикла IF THEN вы сможете объединить одно или два условия, как показано ниже:

```
IF (Булево выражение 1) AND (Булево выражение 2) THEN
   ' Следуй одной или нескольким инструкциям
END IF
```

О булевых выражениях подробно мы говорили в главе 9 "Принятие решений с помощью управляющих операторов".

В данном случае цикл IF THEN будет выполняться, если оба булевых выражения окажутся правильными. Если одно из булевых выражений окажется неправильным, инструкции цикла IF THEN выполняться не будут. Поэтому, если вы планируете использовать оператор AND, поместите то условие, которое не выполнится первым (на ваш взгляд), в начале. Например, если Булево выражение 1 окажется неправильным, компьютер не будет тратить время на проверку Булево выражение 2, так как одно невыполнение выражения Булево выражение 1 делает неправильной всю операцию AND.

Как только компьютер определит, что Булево выражение 1 не выполняется, он не будет проверять Булево выражение 2, а значит, позволит программе работать немного быстрее.

Помещаем наиболее вероятное условие в начале

ИНСТРУКЦИЯМ IF THEN ELSE IF И SELECT CASE ДЛЯ ПРИНЯТИЯ **решения** часто приходится проверять несколько условий, как проиллюстрировано ниже.

```
IF (Булево выражение 1) THEN
   ' Одна или несколько инструкций.-
ELSEIF (Булево выражение 2) THEN
        ' Одна или несколько инструкций
END IF
```

В данном случае при выполнении инструкции IF THEN ELSE IF компьютер начинает с проверки Булево выражение 1. Если оно неправильно, проверяется Булево выражение 2.

Но что, если Булево выражение 1 чаще всего неправильно, а Булево выражение 2 — наоборот? Тогда программа тратит массу времени на проверку Булево выражение 1 (которое чаще всего не выполняется), прежде чем перейти к Булево выражение 2 (которое чаще всего выполняется).

Для того чтобы избавить программу от необходимости постоянно проверять Булево выражение 1, которое чаше всего не выполняется, поместите то из них, которое чаше всего выполняется, в начале, а булево выражение, которое чаще всего не выполняется, в конце инструкции IF THEN SLSE IF, как показано ниже:

```
IF (Булево выражение 2} THEN
   ' Одна,или несколько инструкций
ELSEIF (Булево выражение 1) THEN
        ' Одна или несколько инструкций
END IF
```

Поместив в начале булево выражение, которое чаще всего выполняется, вы избавите компьютер от необходимости тратить время на проверку дополнительного булевого выражения, которое чаще всего не выполняется.

Язык программирования Liberty BASIC не поддерживает инструкции IF THEN ELSE IF И SELECT CASE.

Этот прием также срабатывает и при использовании инструкции SELECT CASE, как показано ниже:

```
SELECT CASE Переменная
CASE Значение1
   ' Одна или несколько инструкций, если значение переменной равно =
Значение1
CASE Значение2
   ' Одна или несколько инструкций, если значение переменной равно =
Значение2
END SELECT
```

Инструкция SELECT CASE проверяет равенство переменной определенному значению (например, Значение1). Поместив наиболее вероятное значение переменной в самом начале инструкции SELECT CASE, вы избежите траты времени на проверку равенства переменной всем другим значениям.

Несмотря на то, что размещение всех наиболее вероятных условий в начале может показаться достаточно тривиальной задачей, каждый раз, когда вам удастся этого добиться, позволит вашей программе работать чуточку быстрее.

Не используйте без надобности цикл **FOR NEXT**

Циклы отнимают массу времени, поэтому убедитесь в том, что выбрали самый оптимальный цикл для решения поставленной задачи. Если вы используете последовательный поиск для нахождения элемента массива, например, работайте с циклом FOR NEXT. Этот цикл заставляет компьютер проверять каждую ячейку массива.

Цикл FOR NEXT выполняется определенное количество раз. Но, как вы думаете, как поведет себя цикл FOR NEXT, если искомым окажется уже первый элемент массива? Цикл FOR NEXT это не интересует: он выполняется строго определенное количество раз,

Если вы используете цикл FOR NEXT, убедитесь в том, что программе действительно необходимо выполнять цикл строго фиксированное число раз; в противном случае используйте команду EXIT FOR для того, чтобы быстрее завершить цикл FOR NEXT.

В следующем примере программы, написанной на QBASIC, команда EXIT FOR используется для того, чтобы побыстрее завершить цикл FOR NEXT при нахождении необходимых данных. Без использования команды EXIT FOR цикл FOR NEXT будет выполняться 30 раз, независимо от того, действительно ли это необходимо, как показано ниже:

```
FoundIt = 0
FOR J = 1 TO 30
". PRINT "Checking array location"; J
  IF MyArray(J) = FindMe THEN
  FoundIt = 1
  EXIT FOR
  END IF
NEXT J
```

Язык программирования Liberty BASIC команду EXIT FOR не поддерживает.

Правильно организуйте циклы

Вы должны поместить все *необходимые* инструкции в цикл... да-да, именно необходимые. Если вы поместите в цикл инструкцию, которая не ифает в нем никакой роли, вы заставите компьютер постоянно выполнять ее при каждом выполнении цикла, что неизбежно приведет к замедлению работы программы.

Рассмотрим следующий пример цикла:

```
FOR J - 1 TO 5000
   I = 0
   IF MyArray(J) = 55 THEN
   PRINT MyArray(J)
'  END IF
NEXT J
```

В данном случае цикл FOR NEXT выполняется 5000 раз, однако инструкция I=0 ни разу не используется в рамках цикла FOR NEXT. Поэтому компьютер исполняет эту инструкцию 5000 раз безо всякой надобности. Для решения проблемы просто удалите инструкцию I=0 из цикла FOR NEXT, как показано ниже:

```
I =O
FOR J = 1 TO 5000 -
   IF MyArray(J) = 55 THEN
   PRINT MyArray(J)
   END IF
NEXT J
```

В программировании о *вложении* говорят в том случае, когда один управляющий элемент или цикл помещается в другой. В предыдущем примере программы, написанной на Liberty BASIC, инструкция IF THEN вложена в цикл FOR NEXT.

Ко вложенным циклам следует относиться особенно осторожно. Если вложить один цикл в другой, вложенный цикл будет выполняться намного чаше, чем внешний. Помещая во вложенный цикл лишние инструкции, вы заставите компьютер выполнять массу ненужных действий.

Почему в программах, написанных на C/C++, так тяжело разобраться?

Программы, которые вы пишете на C/C++, обычно выполняются быстрее и эффективнее, чем программы, написанные на других языках программирования, таких как Pascal или BASIC. Однако языки программирования C/C++ снискали славу языков, на которых создаются самые запутанные программы. Одна из причин такого положения заключается в том, что C/C++ предлагают использовать сокращенные варианты записи операций, однако это приводит к снижению читабельности исходного кода программ.

Вместо того чтобы писать x=x+5 или y=y+23, C/C++ предлагают сделать это следующим образом:

```
X += 5; /*Эквивалент x = x +5*/
y += 23; /*Эквивалент y = y +23*/
```

C/C++ также поддерживают префиксные и постфиксные операторы, позволяющие увеличивать или уменьшать значение параметра на единицу. Пример использования постфиксных операторов приведен ниже:

```
x++;  /*Эквивалент x = x +1*/
y--;  /*Эквивалент y - y -1*/
```

Пример использования префиксных операторов приведен ниже:

```
++x;  /*Эквивалент x = x +1*/
--y;  /*Эквивалент y = y -1*/
```

Если вы решите, что оба варианта выглядят похоже и приводят к одному и тому же результату, вы будете практически правы. Однако основное отличие проявится, если вы объедините префиксные и постфиксные операции в одну формулу, как показано ниже:

```
x = y +z++
```

Эта формула эквивалентна следующим двум операциям:

```
x = y +z
z" = z +1
```

Однако вы можете использовать и префиксный оператор:

```
a = b + --c
```

Эта формула эквивалентна следующим двум операциям:

```
c = c -1
a = b + c
```

C/C++ предлагают достаточно странный сокращенный эквивалент инструкции IF THEN ELSE IF, использующий комбинацию вопросительных знаков и двоеточий:

```
printf("Это число больше = %d\n", (x>y) ? x : y);
```

Эта строка полностью эквивалентна следующей комбинации с использованием инструкции IF THEN ELSE IF:

```
if (x>y)
printf("Это число больше = %d\n", x) ;
else
printf("Это число больше = %d\n",y);
```

Тауой "зашифрованный" короткий эквивалент инструкции IF THEN ELSE IF позволяет профамме быстрее выполняться, однако в нем сложнее разобраться с наскоку. Это справедливо и по отношению любым другим сокращенным вариантам команд.

Используйте правильные типы данных

Для экономии памяти компьютера старайтесь всегда использовать правильные (и наиболее подходящие) типы данных. Многие версии BASIC, например Liberty BASIC, позволяют объявлять все переменные типа integer (например, DIM Num AS INTEGER) или типа long integer (например, DIM Num AS LONG). Переменные типа long integer изменяются в диапазоне от −2147483648 до 2147483647, а переменные типа integer изменяются в диапазоне от −32768 до 32767.

Однако переменные типа long integer занимают намного больше памяти. Поэтому если ваши переменные не содержат огромные значения, объявляйте их как переменные типа long, которые занимают намного меньше памяти.

Старайтесь по возможности использовать встроенные команды

Практически любой язык программирования содержит специальные встроенные команды, которые выполняются намного быстрее, чем их любые эквиваленты, составленные вручную. Например, если необходимо увеличить значение переменной MyNumber на I, вы можете использовать следующую команду:

```
MyNumber = MyNumber + 1
```

В этой команде все правильно, однако многие языки программирования предлагают краткие команды для увеличения значения переменной на 1. Например, при использовании C/C++ эквивалентная команда выглядит следующим образом:

```
MyNumber++
```

Если же вы захотите увеличить значение переменной на 1 при использовании Delphi (который базируется на Pascal), используйте следующую команду:

```
Inc (MyNumber)
```

Если вы используете встроенные команды, вы рискуете сделать исходный код программы более сложным для понимания. Это связано с тем, что далеко не все программисты знакомы со встроенными командами в том или ином языке программирования. Если программист столкнется со встроенной командой, он просто не разберется в том, какие задачи она решает.

Использование быстрого языка программирования

Самый быстрый язык программирования, доступный вам для написания своих программ, — это машинный язык, потом следуют язык ассемблера и C/C++, а затем, с некоторым отставанием, другие языки программирования (Pascal или BASIC). Если вы хотите, чтобы ваши программы выполнялись быстрее, подумайте о переходе на другой язык программирования.

Многие программисты используют более простой язык программирования, Visual Basic, для создания прототипов программ, которые они смогут показать заказчикам, а также для проектирования интерфейса пользователя. После создания прототипа программы у вас два варианта выбора. Во-первых, вы переписываете программу полностью, используя другой язык программирования, например C/C++, но при этом прототип программы, созданный раньше, теряется. Конечно же, подобный процесс требует больше времени и не гарантирует, что программа будет работать должным образом (но, по крайней мере, ее исходный код выглядит неплохо). Во-вторых, вы можете использовать готовый прототип в качестве базиса настоящей работающей программы- Но вместо того, чтобы использовать один язык программирования при написании всей программы, вы пишете на "быстром" языке программирования те части программы, которые будут использоваться активнее других.

Если при создании программы вы используете два или больше языков программирования, вы сможете воспользоваться преимуществами каждого из них. Однако недостатком такого подхода будет необходимость наладить взаимодействие нескольких языков программирования.

Многие сначала создают программу, используя простой язык программирования, например Visual Basic, а затем постепенно переписывают отдельные части программы, используя быстрый язык программирования — C/C++ или язык ассемблера. Программу можно целиком написать на одном языке программирования, но никто не мешает вам переписать ее, используя другой язык программирования.

Оптимизация компилятора

В качестве альтернативы использованию быстрого языка программирования предлагаем использовать более быстрый компилятор. Если вы передадите одну и ту же программу разным компиляторам, каждый из них создаст собственный выполняемый вариант программы, один из них обязательно будет работать быстрее, чем остальные. К сожалению, если вы написали программу на C++ для одного компилятора (например, Microsoft Visual C++), программу нельзя "прогнать" через другой компилятор (например, Borland C++) без внесения в нее серьезных изменений.

Для того чтобы предоставить вам больший контроль над программой, многие компиляторы позволяют настраивать параметры своей работы. Вы можете изменить эти параметры для оптимизации компилятора под конкретную программу, как показано на рис. 22.1.

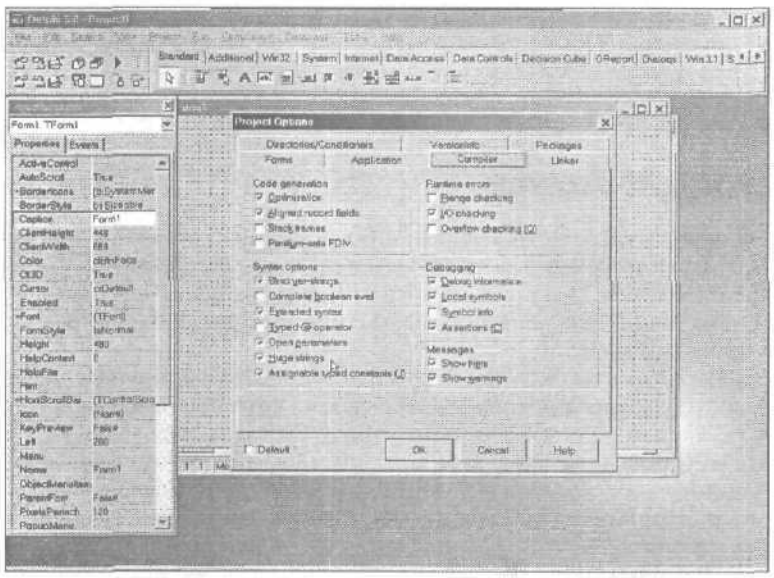

Рис. 22.1. Оптимизация параметров работы компилятора Delphi

Вы должны точно знать, что делаете, прежде чем изменять любые параметры оптимизации компилятора. Многие из этих параметров позволяют ускорить выполнение программы, однако за это придется расплачиваться отключением средств коррекции ошибок компилятора, которые позволяют намного упростить выявление ошибок в работе программы. Если вы отключите средства коррекции ошибок компилятора, программа будет работать быстрее, но в ее работе могут возникать сбои и даже зависания.

Часть VI

Программирование для Internet

"Скажи мне, Тарзан, король джунглей, каково мое будущее?"

В этой части...

Создание компьютерной программы подразумевает написание и выполнение набора инструкций на одном компьютере. Если вы захотите запустить программу на другом компьютере, вам придется предварительно скопировать программу и только после этого запустить ее.

Однако в связи со стремительным распространением Internet появились и новые направления в программировании. Вместо того чтобы писать программу для одного, компьютера, напишите программу, которая будет выполняться на различных компьютерах, подключенных к Internet. Теоретически, любое творение, созданное вами на одном компьютере, может увидеть или запустить другой пользователь на компьютере, находящемся в любой точке мира.

I В настоящей части вы познакомитесь с восхитительным миром языков программирования для Internet. Программы, которые вы создаете с помощью одного из языков программирования для Internet, хранятся в виде исходных кодов на одНом компьютере. Если какой-нибудь другой компьютер получит доступ к исходному коду программы, он сможет выполнить ее и интерпретировать результат.

Наиболее известным языком программирования для Internet является *HTML* (HyperText Markup Language — Язык разметки гипертекста), специальный язык программирования для создания Web-страниц. Для создания более интерактивных Web-узлов предназначены другие языки программирования — *Java* и *JavaScript.* Если вы когда-нибудь мечтали о создании собственного Web-узла, который взаимодействовал бы с пользователем, эта часть — именно для вас.

Глава 23

Забавы с HTML

В этой главе...

- Изучаем основы HTML
- Использование дескрипторов для форматирования текста
- Использование атрибутов дескрипторов
- Создание списков
- Создание гиперссылок
- Использование графики
- Создание интерфейса пользователя

В старые добрые дни использование Internet подразумевало ввод непонятных команд и работу только в текстовом режиме. Затем, в 1992 году физики из швейцарского института физики частиц CERN изобрели World Wide Web (именно от этого выражения и появилась аббревиатура WWW, которая присутствует в адресах большинства Web-страниц, например www.dialektika.com). На самом деле, все, что делает World Wide Web, —добавляет графический интерфейс пользователя к Internet. Вы можете использовать Internet без World Wide Web, но нельзя использовать World Wide Web без Internet.

Основной целью World Wide Web было обеспечение обмена информацией между исследователями. Изначально World Wide Web работала только с текстом, но позволяла создавать *гиперссылки* (текстовые ссылки, указывающие на другие документы, хранящиеся в World Wide Web). Так как обычный текст выглядит достаточно скучно, люди придумали добавлять на Web-страницы картинки, что позволило сделать их гораздо привлекательнее.

Поскольку Internet состоит из огромного числа компьютеров различных типов, программистам понадобилось создать такой интерфейс пользователя World Wide Web, который смогли бы использовать пользователи любых компьютеров. Все компьютеры понимают ASCII-коды, поэтому создали базирующийся на этих кодах язык программирования *HTML* (HyperText Markup Language — Язык разметки гипертекста).

Для создания Web-узла вы вводите HTML-коды и сохраняете их в виде ASCII-файла с расширением .htm или .html. Если пользователь другого компьютера захочет просмотреть ваш Web-узел, он должен воспользоваться специальной программой, которая называется *броузером* и преобразует HTML-коды в замечательный интерфейс пользователя.

Изучаем основы HTML

HTML-коды определяют то, как текст и графические изображения отображаются в окне броузера. Web-страница на самом деле содержит только HTML-код, сохраненный в виде ASCII-файла с расширением .htm или .html. HTML-код состоит из специальных элементов, называемых *дескрипторами,* которые заключаются в угловые скобки. Многие (если не все) дескрипторы используются парами, в которых первый дескриптор начинает определение какого-это элемента страницы, а второй — завершает его, например как при определении заголовка или курсивного начертания текста:

```
<I>Этот текст отображается курсивом.</I>
```

ASCII-файл содержит просто обычные символы безо всякого форматирования, без использования различных шрифтов или подчеркивания. Так как все компьютеры понимают обычные символы (буквы, цифры и символы, которые вы вводите с клавиатуры), вы можете легко переносить текст с одного компьютера на другой, используя ASCII-файлы.

Завершающие дескрипторы всегда содержат косую, например: </I> или </BODY>.

Вы можете помешать дескрипторы между другими дескрипторами, как показано ниже:

```
<B>Этот <I>текст</I> отображается полужирным.</B>
```

В результате вся строка будет отображаться полужирным, а слово *текст* — курсивом., как показано ниже:

Этот *текст* отображается полужирным.

Дескрипторы выступают в роли контейнеров, содержащих текст. Старайтесь не перепутать дескрипторы местами, в противном случае вы рискуете получить непредвиденный результат, как показано ниже:

```
<B>Этот <I>текст отображается</B> полужирным. </I>
```

Этот текст отображается *полужирным.*

В идеале все пары дескрипторов должны быть заключены в другие пары, как показано на рис. 23.1.

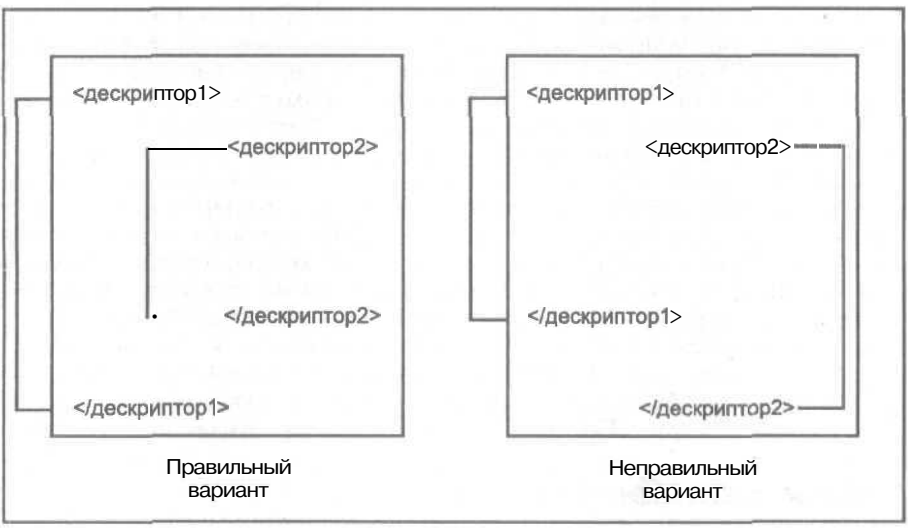

Рис. 23.1. Правильный (слева) и неправильный (справа) способы расположения дескрипторов HTML

HTML-коды превратятся в настоящую шифрограмму, если вы разместите их слишком близко друг к другу, поэтому старайтесь не скупиться на пустые строки и промежутки между ними, чтобы сделать HTML-код страницы удобнее для чтения. Помните, что при интерпретации HTML-кодов в Web-страницу броузер просто игнорирует пустые строки.

Вы можете писать HTML-код Web-страниц в любом текстовом редакторе, таком как Блокнот из состава Windows или даже редакторе Liberty BASIC. Просто не забывайте о том, что полученные файлы необходимо сохранять с расширением .htm или .html. Создав файл с помощью HTML-кодов, вы можете открыть его в окне броузера, воспользовавшись в нем командой Файл⇨Открыть.

Знакомство с самыми важными дескрипторами HTML

Ниже приведены первые дескрипторы HTML, которые содержатся на любой Web-странице:

```
<HTML>
</HTML>
```

Этих два дескриптора просто определяют пустую Web-страницу. Все, что содержится между этими дескрипторами, отображается на Web-странице. Перед дескриптором <HTML> и после дескриптора </HTML> не должно быть никакой информации. Даже если подобная информация и будет в файле, в окне Web-броузера она не отображается.

Создание заголовка

Затем вам следует определить, что будет выступать в качестве заголовка (располагается в верхней части) вашей Web-страницы, воспользовавшись дескриптором <HEAD>, как показано ниже:

```
<HTML>
<HEAD>
</HEAD>
</HTML>
```

Между **дескрипторами** <HEAD> и </HEAD> **помещается заголовок страницы. Если какой-нибудь пользователь** решит **занести вашу** Web-страницу **в список "Избранное", в** списке будет **указан именно этот заголовок. Если вы не укажете заголовок страницы, в списке** будет **имя файла, которое зачастую выглядит достаточно непонятно. В** конце **концов,** что **вам** понятнее — Web-узел, содержащий секретные сведения об ядерном оружии США ИЛИ NK1999.HTM?

Для определения заголовка Web-страницы вы просто помещаете его между соответствующими дескрипторами, как показано ниже:

```
<HTML>
<HEAD>
  <TITLE>Заголовок документа</TITLE>
</HEAD>
</HTML>
```

Для Web-страницы необходимо указывать только один заголовок.

Определение содержимого Web-страницы

После того как вы определили заголовок Web-страницы, необходимо определить ее содержимое с помощью дескрипторов <BODY> и </BODY>, как показано ниже:

```
<HTML>
<HEAD>
  <TITLE>Заголовок документа</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Добавление комментариев

В Liberty BASIC и других языках программирования вы можете добавлять комментарии, поясняющие работу программы, время внесения последних изменений, а также имя автора последних изменений. При написании HTML-кода вы также можете добавлять комментарии к Web-страницам.

Комментарии в окне броузера не отображаются; вы увидите их только при просмотре HTML-кода. Комментарии заключаются в угловые скобки, как показано ниже:

```
<!-- Это комментарий: Твоя мамаша уродлива. -->
```

Фрагмент <!-- свидетельствует о начале дескриптора комментария, а --> — о его завершении.

Использование дескрипторов для форматирования текста

Основные дескрипторы HTML определяют параметры Web-страницы в целом, но, помимо этого, вам необходимо добавить на Web-страницу текст, чтобы пользователям было интересно ее читать. HTML предоставляет следующие специальные дескрипторы для форматирования текста.

- ✓ *Заголовки* разделяют разделы текста на отдельные блоки в рамках определенной темы (точно так же, как и в настоящей книге).
- ✓ *Абзацы* — это блоки текста, содержащие одно или несколько предложений.
- ✓ *Цитаты* очень похожи на абзацы, но соответствующие дескрипторы HTML добавляют до и после них больше свободного пространства, чем для обычных абзацев.
- ✓ *Выделения текста* позволяют отображать текст с особым форматированием или стилем для его выделения.

Все эти дескрипторы форматирования текста подробно рассмотрены в следующих разделах.

Создание заголовка

HTML позволяет вам выбирать любой из шести стилей заголовков. Заголовок первого уровня считается наиболее важным, а заголовок шестого уровня — наименее важным. Пример использования заголовков всех шести уровней приведен на рис. 23.2.

Для создания одного из шести видов заголовков используется одна из следующих пар дескрипторов HTML:

```
<H1>Заголовок 1</H1>
<H2>Заголовок 2</H2>
```

```
<H3>Заголовок 3</H3>
<H4>Заголовок 4</H4>
<H5>Заголовок 5</H5>
<H6>Заголовок 6</H6>
```

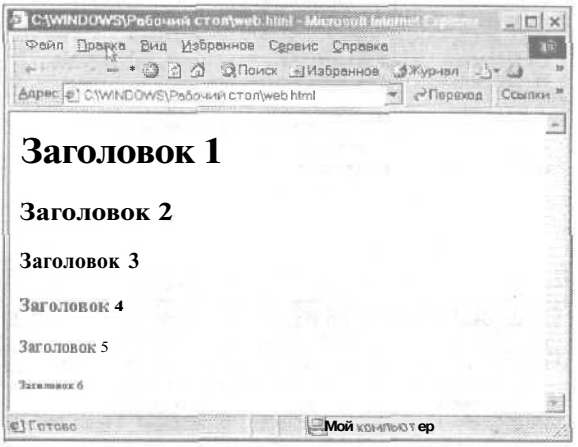

Рис. 23.2. Шесть типов заголовков, которые вы можете создавать с помощью дескрипторов HTML

Обычно в рамках одного заголовка располагается один-два подзаголовка. Например, вы можете использовать два заголовка второго уровня в рамках одного заголовка первого уровня. Или вы можете использовать два заголовка шестого уровня в рамках одного заголовка пятого уровня.

Определение абзаца

Абзац — это фрагмент текста, который отделяется от окружающего текста пустыми строками, или, правильнее, пустым пространством (точно так же, как и на любой странице настоящей книги). Для обозначения начала абзаца используется дескриптор <p>, а для обозначения конца абзаца — дескриптор </P>, как показано ниже:

```
<P>
Этот текст можно считать абзацем.
</P>
```

Если вы ввели текст между двух дескрипторов абзаца, весь абзац может быть одной строкой, от левого до правого поля экрана. Дескрипторы абзаца автоматически форматируют текст так, чтобы он полностью умещался на экране.

Если вы хотите вставить разрыв в середине абзаца, используйте специальный дескриптор
. В отличие от многих других дескрипторов, дескриптор
 "гуляет сам по себе". Пример использования дескрипторов <p>, </P> и
 для добавления разрыва между абзацами приведен на рис. 23.3.

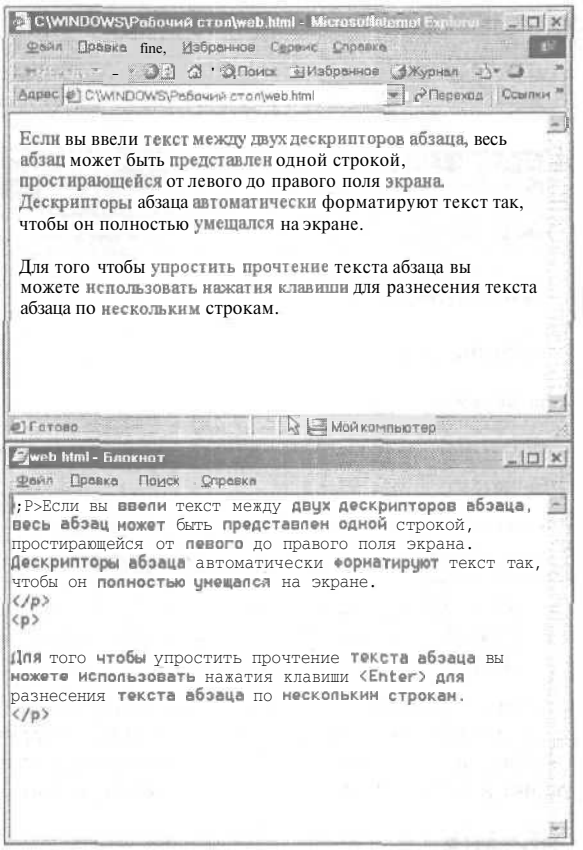

Рис. 23.3. Так абзацы текста отображаются в окне броузера

Выделение цитаты

Если вы хотите, чтобы определенный абзац текста выделялся на фоне всех остальных, определите его в виде цитаты, как показано ниже:

```
<BLOCKQUOTE>
Этот текст выделен.
</BLOCKQUOTE>.
```

Привлекаем внимание к тексту

Дескрипторы абзаца разделяют текст на абзацы (автоматически добавляя пустое пространство), а цитаты позволяют выделять отдельные абзацы на фоне других. Но иногда вам нужно выделить только отдельные фразы и даже слова. Для этого используются следующие пары дескрипторов:

- ✓ <в> и </в> — отображение текста полужирным;
- ✓ <I> и </I> — отображение текста курсивом;
- ✓ <U> и </U> — отображение текста подчеркнутым;
- ✓ <тт> и </TT> — отображение текста машинописным шрифтом;

- ✓ <HR> — отображение горизонтальной линии. (Обратите внимание на то, что дескриптор <HR> используется сам по себе, т.е. без пары.)

Примеры использования этих дескрипторов на Web-странице приведены на рис. 23.4.

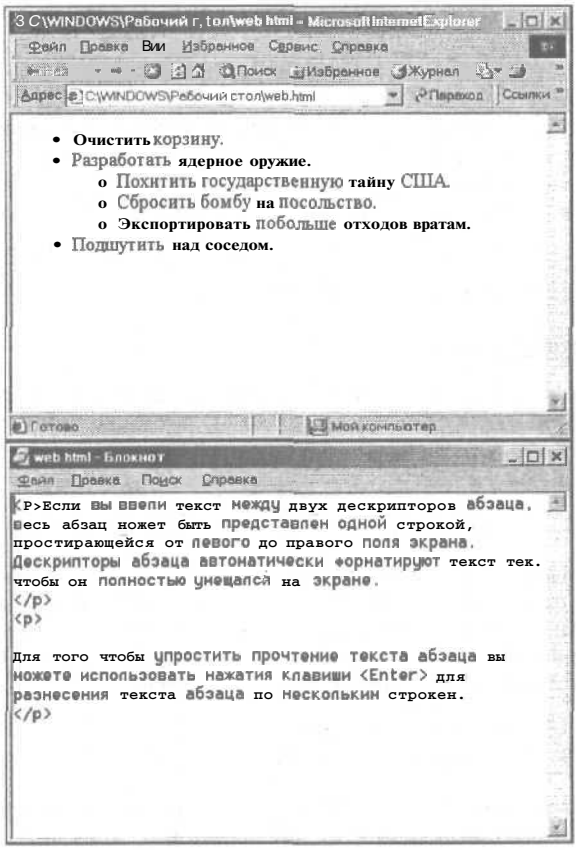

Рис. 23.4. Использование специальных дескрипторов для выделения отдельных фраз

Использование атрибутов дескрипторов

Для улучшения внешнего вида текста вы можете использовать атрибуты. *Атрибут* — это специальная команда, которая добавляется к дескриптору HTML. Атрибут изменяет внешний вид любого текста, определяемого дескриптором. Ниже перечислены часто используемые атрибуты.

- ✓ ALIGN — выравнивание абзаца или заголовка по правому краю, левому краю или по центру;
- ✓ BGCOLOR — изменение цвета фона Web-страницы;
- ✓ TEXT — изменение цвета текста;
- ✓ **LINK — изменение цвета ссылок;**
- ✓ VLINK — изменение цвета ссылок, к которым уже обращался пользователь.

Выравнивание текста

Обычно заголовки и абзацы на Web-странице выравниваются по левому краю, но можно выравнивать их по правому краю или по центру, используя атрибут ALIGN вместе с дескриптором заголовка первого уровня или абзаца, как показано ниже:

```
<P ALIGN="center">
Этот текст выровнен по центру.
</P>
```

Для выравнивания текста по центру вам необходимо использовать служебное слово "center" вместе с атрибутом ALIGN.

Для выравнивания текста по правому или левому краю, вам необходимо использовать служебное слово "right" или "left" вместе с атрибутом ALIGN соответственно:

```
<H1 ALIGN="right">
Этот текст выровнен по правому краю.
```

Использование цвета

Для определения цвета фона или текста вы должны использовать атрибуты BGCOLOR и TEXT. Последние версии броузеров позволяют четко указывать названия цветов, используя такие служебные слова, как red, blue или yellow:

```
<BODY BGCOLOR="white"> (белый цвет фона)
<BODY TEXT="black"> (черный цвет текста)
```

Для большей гибкости вы можете использовать шестизначное число, представляющее значение цвета в модели RGB. Это значение определяет степень проявления каждого из основных цветов — красного, зеленого и синего. Изменяя количество этих цветов, получите разнообразные цвета и оттенки: пурпурный, белый, оранжевый, желтый и т.д. Ниже приведен соответствующий пример:

```
BODY BGCOLOR=" FFFFFF"> (белый цвет фона)
<BODY TEXT="000000"> (черный цвет текста)
```

При определении цветов RGB используются шестнадцатеричные числа, которые изменяются от 0 до F (O, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Нулевое значение означает полное отсутствие цвета, а значение F — его максимальное количество. Вы можете изменять количество красного, синего и зеленого цветов для создания различных оттенков.

Первые две цифры определяют количество красного цвета (R), вторые две цифры — зеленого цвета (G), а последние две цифры — синего цвета (B). Если вы хотите, чтобы цвет фона страницы был насыщенно-красным, используйте следующую команду:

```
<BODY BGCOLOR="FF0000">
```

Если вы хотите, чтобы цвет фона страницы был насыщенно зеленым, используйте следующую команду:

```
<BODY BGCOLOR="00FF00">
```

Если **вы** хотите, чтобы цвет **фона страницы был насыщенно синим,** используйте следующую **команду:**

```
<BODY BGCOLOR="0000FF">
```

Раскрашиваем гиперссылки

Можно изменять и цвет гиперссылок. На многих Web-страницах гиперссылки отображаются более ярким цветом, чтобы привлечь к ним внимание. После того как пользователь воспользуется гиперссылкой, ее цвет изменяется, благодаря чему пользователь знает, что соответствующую Web-страницу он уже посещал. Для изменения цвета гиперссылок используются следующие дескрипторы:

```
<BODY LINK="шестнадцатеричное_значение">
<BODY VLINK="#шестнадцатеричное_значение">
```

Атрибут LINK использует для определения цвета гиперссылки такие же шестнадцатеричные числа, как и для определения цвета фона или текста. Атрибут VLINK делает то же самое, но для уже использованной гиперссылки.

Создание списков

Создание Web-страницы для представления определенных сведений во многом напоминает создание телевизионной рекламы. В обоих случаях вам необходимо предоставить зрителю как можно больше информации, причем в самом привлекательном виде. Многие считают, что большие объемы текста очень сложны для чтения и быстро приводят к утомлению, поэтому подумайте об организации текста Web-страницы в виде списков.

HTML поддерживает следующие виды списков (об использовании каждого из них я обязательно расскажу).

- ✓ *Неупорядоченные списки.* В данном случае перед каждой строкой текста отображается маркер.
- ✓ *Упорядоченные списки.* В данном случае перед каждой строкой текста отображается ее номер.
- ✓ *Списки определений.* В данном случае каждая строка текста начинается с отступа.

Неупорядоченные списки

Для создания неупорядоченного списка вам необходимо использовать два вида дескрипторов HTML. Первая пара дескрипторов <UL> и </UL> определяет неупорядоченный список. Вторым будет дескриптор <LI>, который помечает маркером каждый элемент списка. Изучите следующий пример:

```
<UL>
 <LI>Очистить корзину.
 <LI>Разработать ядерное оружие.
 <LI>Подшутить над соседом.
</UL>
```

Дескриптор <LI> используется без пары, т.е. вы можете использовать его или сам по себе, или в паре с дескриптором </LI>. Все зависит от ваших предпочтений.

Вы также можете создавать и вложенные неупорядоченные списки, как показано ниже:

```
 <UL>
 <LI> Очистить корзину.
 <LI>Разработать ядерное оружие.
   <UL>
```

```
   <LI>Похитить государственную тайну США.
   <LI>Сбросить бомбу на посольство.
   <LI>Экспортировать побольше отходов врагам.
   </UL>
j <LI>Подшутить над соседом.
 </UL>
```

Использование этого кода проиллюстрировано на рис. 23.5. Обратите внимание, что во вложенном неупорядоченном списке используются другие маркеры.

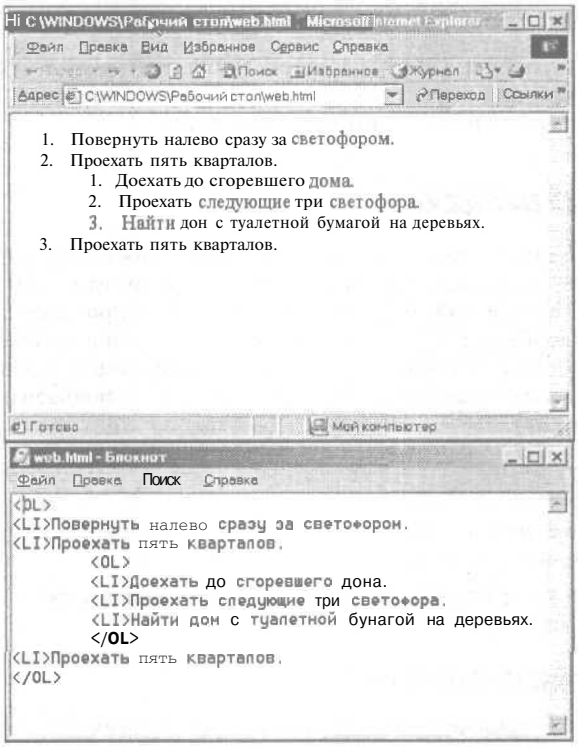

Рис. 23.5. Создание вложенного неупорядоченного списка

Упорядоченные списки

Если в неупорядоченных списках элементы отображаются вместе с маркерами, то в упорядоченных списках они отображаются вместе с цифрами. Первый элемент списка обозначается цифрой 1, второй элемент — цифрой 2 и т. д.

Для создания упорядоченного списка используются такие дескрипторы HTML, как <OL> и </OL>. Для нумерации каждого элемента списка используется дескриптор <LI>. Изучите следующий пример:

```
<OL>
<LI>Повернуть налево сразу за светофором.
<LI>Проехать пять кварталов.
<LI>Бросить тухлое яйцо во входную дверь.
</OL>
```

Вы также можете создавать *вложенные* упорядоченные списки, как показано ниже.

```
<OL>
<LI>Повернуть налево сразу за светофором.
<LI>Проехать пять кварталов.
  <OL>
  <LI>Доехать до сгоревшего дома.
  <LI>Проехать следующие три светофора.
  <LI>Найти дом с туалетной бумагой на деревьях.
  </OL>
<LI>Проехать пять кварталов.
</OL>
```

Использование этого кода проиллюстрировано на рис. 23.6. Обратите внимание, что во вложенном упорядоченном списке используются другие цифры.

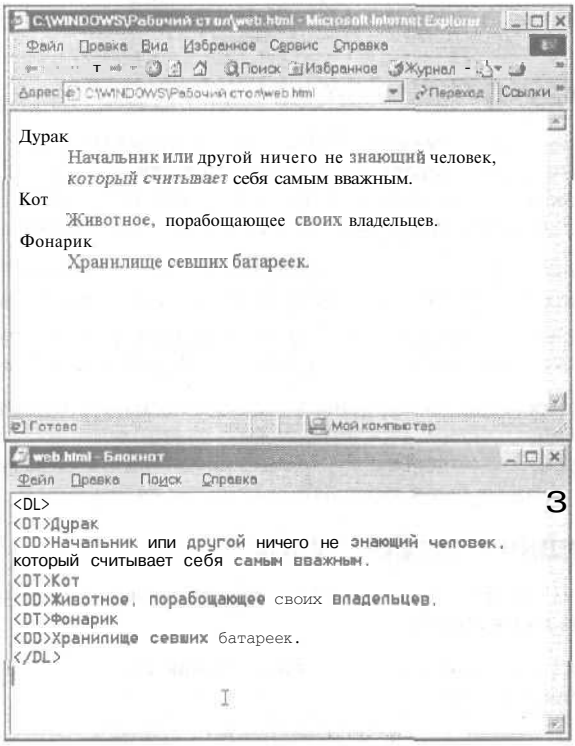

Рис. 23.6. Создание вложенного упорядоченного списка

Списки определений

Списки определения получили такое название из-за того, что их часто используют в словарях, когда в одной строке приводится термин, а в другой — его пояснение. Для создания списка определения используются следующие три типа дескрипторов HTML.

- ✓ Дескрипторы <DL> и </DL> определяют начало и конец списка определений.
- ✓ Дескриптор <DT> отображает строку текста, такую как слово или термин.
- ✓ Дескриптор <DD> отображает определение слова или термина, обозначенного предыдущим дескриптором <DT>.

Для того чтобы разобраться с созданием списка определений, изучите следующий пример кода:

```
<DL>
<DT>Дурак
<DD>Начальник или другой ничего не знающий человек, который считает себя самым важным.
<DT>Кошка
<DD>Животное, порабощающее своих владельцев.
<DT>Фонарик
<DD>Хранилище севших батареек.
</DL>
```

Создание гиперссылок

На каждой действительно хорошей Web-странице должны быть следующие элементы: *информация* (как правило, текстовая), которая предоставляет какие-то важные для пользователя сведения, а также *гиперссылки*, которые позволяют пользователю переходить к Web-страницам, содержащим дополнительные сведения. Гиперссылки бывают двух типов.

- ✓ *Внешние гиперссылки,* которые указывают на другие Web-страницы, которые обычно находятся на других компьютерах (в любой точке мира).
- ✓ *Внутренние гиперссылки*, которые указывают на другие Web-страницы того же Web-узла или на другие части одной и той же Web-страницы.

Для создания гиперссылки вы должны использовать пару специальных дескрипторов-анкеров, таких как <A> и </A>. После первого дескриптора-анкера вы должны указать внешнюю или внутреннюю гиперссылку. Между двумя дескрипторами-анкерами укажите текст или картинку, которые будут выступать в качестве гиперссылки.

Создание внешних гиперссылок

Для внешней гиперссылки атрибут HREF (сокращение от Hypertext REFerence) определяет следующих два элемента.

I ✓ **Адрес, на который** ссылается **внешняя гиперссылка, который выглядит приблизительно так:**

```
http://www.адрес.com
```

I V Текст или картинка, действующие как гиперссылка, на которой пользователь должен щелкнуть для перехода по гиперссылке.

Для использования атрибута HREF вы должны поместить его после первого дескриптора-анкера, как показано ниже:

```
' <A HREF="http://www.dialektika.com">Адрес Web-страницы</A>
```

В этом примере в качестве гиперссылки выступает фраза Адрес Web-страницы. После щелчка на этой фразе пользователь загрузит Web-страницу www.dialektika.com.

Внешние гиперссылки находятся не полностью в вашей власти, поскольку если Web-страница, на которую будет ссылаться гиперссылка, перестанет существовать, пользователь увидит Web-страницу с сообщением об ошибке.

Создание внутренних гиперссылок

Для создания гиперссылки на другую Web-страницу вашего же Web-узла также используется атрибут HREF, но вместо адреса Web-страницы указывается имя файла Web-страницы:

```
; <A HREF="index.html">Указатель</A>
```

В результате в качестве гиперссылки выступает слово Указатель. После щелчка на слове Указатель в окне броузера отображается Web-страница, сохраненная в виде файла index.html.

Ссылка на определенную часть Web-страницы

Одна из проблем, связанных с гиперссылками на другие Web-страницы, состоит в том, что пользователю иногда надо прокрутить Web-страницу, чтобы найти необходимую информацию. Во избежание подобных ситуаций создайте гиперссылку на определенную часть Web-страницы, например на ее середину или конец. Таким образом, гиперссылка быстро приведет пользователя к необходимой информации,

Создание гиперссылки, указывающей на определенную часть Web-страницы, представляет собой двухэтапный процесс.

1. Укажите анкер в той части Web-страницы, которая должна отображаться после щелчка на гиперссылке.
2. Если вы хотите, чтобы гиперссылка направила пользователя к нижней части Web-страницы, разместите дескриптор-анкер в нижней части Web-страницы.
3. Создайте гиперссылку, указывающую на созданный вами анкер.

Для создания анкера необходимо использовать атрибут NAME:

```
<A NAME="TOC">Содержание</A>
```

В данном случае на Web-странице отображается слово Содержание, которому присваивается имя TOC. После создания анкера следующим шагом будет создание указывающей на него гиперссылки.

Имена анкеров *чувствительны к регистру,* т.е. что анкер TOC — это не одно и то же, что анкер toc. Если вы забудете об этом, созданные вами анкеры будут вообще не работать.

Для создания гиперссылки, указывающей на определенный раньше анкер, используйте атрибут HREF, указав имя файла Web-страницы, а также имя анкера, отделенное от имени файла Web-страницы символом #, как показано ниже:

```
<A HREF="index.html#TOC">Переход к первой странице</A>
```

В данном случае на экране отображается гиперссылка Переход к первой странице. После того как пользователь щелкнет на этой ссылке, броузер перейдет к Web-странице index.html и отобразит анкер, определенный именем тос. При этом слово Содержание отображается вверху, независимо от того, расположено оно вверху, в середине или внизу исходной Web-страницы.

Использование графики

Простое отображение текста на Web-странице может показаться слишком скучным, поэтому HTML позволяет отображать на Web-страницах и графические изображения, которые значительно улучшают внешний вид. Изображения могут быть частью Web-страницы или ее фона.

На Web-страницах допускается использование двух типов изображений — GIF (Graphics Interchange Format — Формат графического обмена) и JPEG (Joint Photography Experts Group — Объединенная группа экспертов в области фотографии), так как только эти два формата изображений поддерживаются любыми типами компьютеров.

Размещение картинки на Web-странице

Для отображения изображения на Web-странице вам необходимо использовать дескриптор <1MG> и атрибут SRC, которые позволяют указать компьютеру имя графического файла, который вы хотите отобразить. Изучите следующий пример:

```
<IMG SRC="имя_файла.gif">
```

Для того чтобы предоставить вам больший контроль над размещением картинки по отношению к любому тексту на странице, используйте атрибут ALIGN. Этот атрибут определяет, будет текст отображаться сверху, снизу, справа или слева от изображения:

```
<IMG SRC="имя_файла.gif" ALIGN=middle>
```

Пример различного размещения текста по отношению к изображению приведен на рис.23.7.

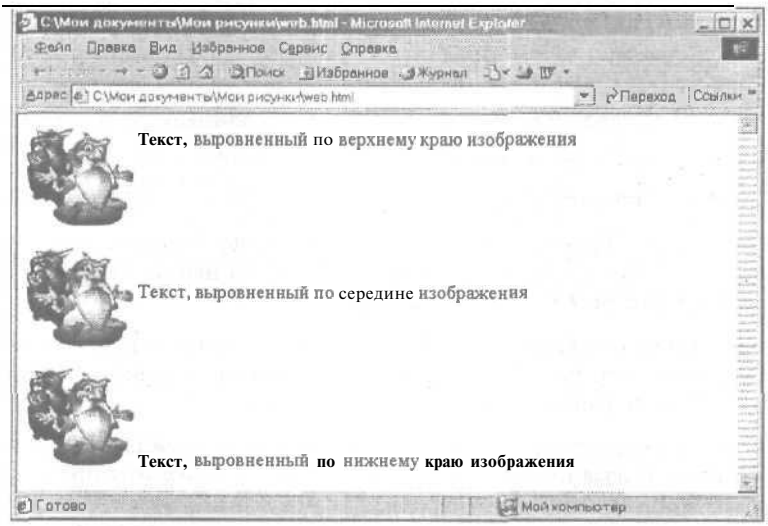

Рис. 23.7. Три различных варианта размещения текста по отношению к изображению

Добавление фонового изображения

Помимо добавления цвета к Web-странице вы можете отобразить изображение на ее фоне. Для этого используется атрибут BACKGROUND дескриптора BODY, как показано ниже:

```
. <BODY "BACKGROUND ="имя_файла.GIF">
```

Создание интерфейса пользователя

Хотя вы обычно используете HTML для отображения текста на экране, можно создать и кое-что поинтереснее — *форму*. Форма позволяет вам отображать *текстовые поля, кнопки* и *флажки* на экране. Для определения формы используются дескрипторы <FORM> и </FORM>, которые помещаются между дескрипторами <BODY> и </BODY>:

```
<HTML>
<BODY>
<FORM>
</FORM>
</BODY>
</HTML>
```

Убедитесь в том, что дескрипторы <FORM> и </FORM> помешены между дескрипторами <BODY> и </BODY>, в противном случае форма просто не отобразится на экране.

Конечно же дескрипторы <FORM> и </FORM> просто определяют форму, к которой вы добавляете элементы интерфейса пользователя, как показано на рис. 23.8. Основные элементы интерфейса пользователя перечислены ниже.

- ✓ *Текстовые поля.* Это поля, в которых пользователь вводит данные.
- ✓ *Кнопки.* Это элементы интерфейса, на которых пользователь щелкает для выполнения определенных действий.
- ✓ *Поля флажков.* Это специальные поля, в которых пользователь может устанавливать или сбрасывать метки.
- ✓ *Переключатели.* Это элементы, среди которых пользователь может установить только один для выбора определенного параметра.

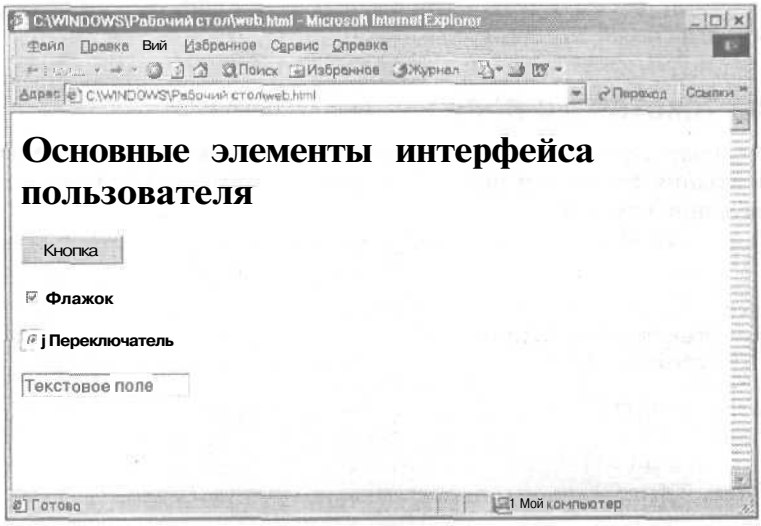

Рис. 23.8. Наиболее распространенные элементы интерфейса пользователя, которые встречаются в формах

Обработка событий

Каждый элемент интерфейса пользователя должен как-то реагировать на действия пользователя. Любое действие, которое пользователь проводит с элементом интерфейса пользователя, например щелчок на кнопке, называется *событием*.

После того как с определенным элементом интерфейса пользователя произойдет какое-то событие, форма должна отреагировать на это событие, отобразив текст, открыв окно и т. д. Ниже перечислены HTML-коды для наиболее распространенных событий.

- ✓ `onAbort`. Это событие происходит после того, как пользователь прекратит загрузку изображения, щелкнув на ссылке или на кнопке Остановить.
- ✓ `onBlur`. Это событие происходит после того, как элемент (текстовое поле или кнопка) перестает быть активным. Это обычно происходит после выбора пользователем другого элемента.
- ✓ `onChange`. Это событие происходит после того, как изменяется содержимое элемента, например текстового поля.
- ✓ `onclick`. Это событие происходит после того, как пользователь щелкнет на определенном элементе, на переключателе или кнопке.
- ✓ `onFocus`. Это событие происходит после того, как пользователь щелкнет на объекте или выделит его с помощью клавиши <Tab>.
- ✓ `onMouseOut`. Это событие происходит после того, как указатель мыши больше не указывает на объект.
- ✓ `onMouseOver`. Это событие происходит после того, как указатель мыши укажет на объект.
- ✓ `onSelect`. Это событие происходит после того, как пользователь выделит текст в текстовом поле.

События связываются с определенными элементами интерфейса пользователя, с кнопками или флажками. Один элемент интерфейса пользователя может реагировать на несколько типов событий.

Создание текстового поля

Текстовое поле содержит готовый текст, но позволяет пользователю и вводить текст. Для создания текстового поля между дескрипторами `<FORM>` и `</FORM>` помещаются следующие команды:

```
<FORM>
<INPUT
  TYPE=text
  NAME="Имя_текстового_поля"
  VALUE="Текст в поле"
  SIZE=integer
  [onBlur="command"]
  [onChange="command"]
  [onFocus="command"]
  [onSelect="command"]
  </FORM>
```

Команда TYPE=text приказывает компьютеру отобразить на экране текстовое поле. Команда NAME назначает этому текстовому полю любое заданное вами имя. Команда VALUE отображает текст в рамках текстового поля. Команда SIZE определяет, сколько символов будет отображаться в текстовом поле без появления полосы прокрутки.

Текстовое поле реагирует на такие события, как onBlur, onChange, onFocus и onSelect. Ниже приведен пример создания текстового поля, в котором отображается сообщение Ой! Вы щелкнули слишком резко! после того, как вы щелкнете в текстовом поле.

```
<HTML>
<BODY>
<FORM>
<INPUT>
  TYPE=text
  NAME="textboxName"
  VALUE="Этот текст отображается в текстовом поле"
  SIZE=30
  onFocus="textboxName.value= 'Ой! Вы щелкнули слишком резко!'">
</FORM>
</BODY>
</HTML>
```

Обратите внимание на то, что после события onFocus используются как одинарные, так и двойные кавычки. В двойные кавычки помещается целиком вся команда, которую должен выполнять компьютер после возникновения события onFocus. В любой команде, заключенной в двойные кавычки, должны стоять и одинарные кавычки, в противном случае команда просто не выполняется.

Создание кнопки

Кнопка — это элемент интерфейса пользователя, щелчок на котором приводит к определенному действию. Для создания кнопки между дескрипторами <FORM> и </FORM> помещается следующий программный код:

```
<FORM>
<INPUT
  TYPE=button
  NAME="имя кнопки"
  VALUE=" Это отображается на кнопке"
  [onBlur="handlerText"]
[onFocus="handlerText"]
</FORM>
```

Команда TYPE=button создает кнопку. Команда NAME назначает этой кнопке любое заданное вами имя. Команда VALUE отображает текст, отображаемый на кнопке.

Кнопка может реагировать на такие события, как onBlur, onclick и onFocus. Ниже приведен пример создания двух кнопок — одной для открытия окна для отображения Web-страницы, сохраненной в файле index.html, и второй для его закрытия.

```
<HTML>
<BODY>
<INPUT
  TYPE=button
  NAME="open"
  VALUE="Открыть окно"
  onClick="mywindow=window.open('index.html')">
<INPUT
  TYPE=button
  NAME="close"
  VALUE="Закрыть окно"
```

```
  onClick="mywindow.close()">
</FORM>
</BODY>
</HTML>
```

Обратите внимание, что команда, **определяемая событием** onclick, **целиком заключается в двойные кавычки. Однако в** команде, **заключенной в** двойные **кавычки, должны быть одинарные кавычки, в противном** случае **команда просто не выполняется.**

Создание флажка

Поля флажков отображают различные **варианты,** среди **которых пользователь выбирает необходимые, установив** соответствующие **флажки. Для создания флажка** между дескрипторами <FORM> и </FORM> **помешается следующий программный код:**

```
<FORM>
<INPUT -.-
  TYPE=checkbox
  NAME="имя_флажка"
  VALUE="checkboxValue"
  [CHECKED]
  [onBlue="handlerText"]
  [onClick="handlerText"]
  [onFocus="handlerText"]>
  textToDisplay
        </FORM>
```

Команда TYPE=checkbox **создает** поле **флажка. Команда** NAME **назначает этому полю любое заданное вами имя. Команда** VALUE определяет текст, **назначенный** флажку. **Команда** CHECKED **отображает флажок** в поле. **Переменная** textToDisplay представляет **любой текст, который вы хотите отобразить рядом с флажком.**

Флажок реагирует на такие события, как onBlur, onClick и onFocus. **Пример создания флажков приведен ниже:**

```
<HTML>
 <BODY>
<H2.Куда ваш компьютер отправится сегодня?</H2>
<FORM> .
<INPUT
  TYPE = checkbox
  NAME="check1" -
  VALUE=99
  onClick="litterbox.value = 'Он будет выброшен
       в мусорную корзину.' ">
  В мусорную корзину
<BR>
<INPUT
  TYPE = checkbox
  NAME="check2"
  VALUE=99
  onClick="litterbox.value = 'Он будет выброшен в окно.' ">
  В окно
<BR>
<INPUT
  TYPE = checkbox
  NAME=" check3" -
```

```
  VALUE=99
  onClick="litterbox.value = 'Он будет сломан и выброшен.' ">
  Разломать его на куски
<BR>
<INPUT
  TYPE = text
  NAME="litterbox"
  VALUE=""
  SIXE = 40>
</FORM>
</BODY>
</HTML>
```

После того как вы установите флажок, чуть ниже появится сообщение, как показано на рис. 23.9.

Если вы вводите текст, который должен отображаться рядом с флажком, не нужно заключать его в кавычки. Если вы это сделаете, кавычки также отобразятся на экране.

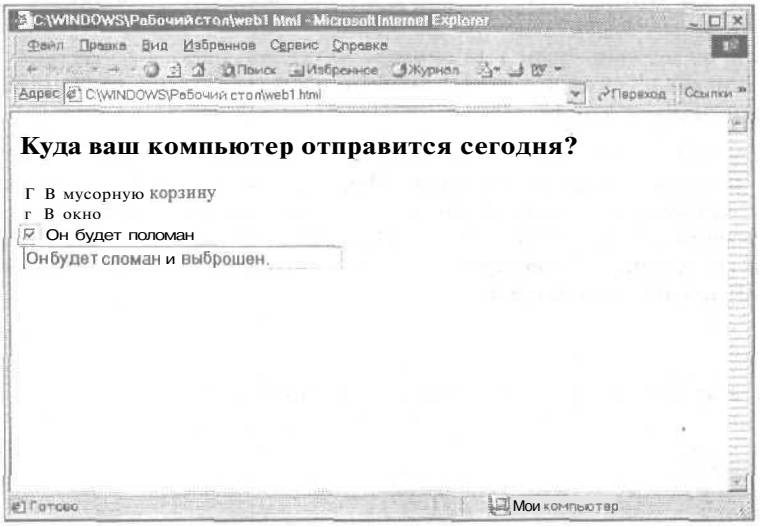

Рис. 23.9. Создание трех флажков и одного текстового поля

Создание переключателя

Переключатель во многом похож на флажок, за исключением того, что в данный момент времени можно выбрать только один переключатель. Переключатели позволяют пользователям отвечать на вопросы, на которые возможен только один ответ, например:

- Каково ваше семейное положение?

Для ответа на вопрос вам необходимы три флажка для следующих ответов.

- ✓ Холост
- ✓ Женат
- ✓ Разведен

Пользователь может выбрать только один из вариантов ответа. Если он установит переключатель Холост, а затем передумает и установит переключатель Разведен, переключатель Разведен будет установлен, что свидетельствует о соответствующем выборе пользователя, а переключатель Холост станет "пустым".

Для того чтобы понять, как создаются переключатели, изучите следующий пример:

```
<FORM>
<INPUT
  TYPE=radio
  NAME="имя переключателя"
  VALUE="buttonValue"
   [CHECKED] .
   [onBlur="handlerTaxt"]
   [onClick="handlerText" j
   [onFocus="handlerText"]>
  textToDisplay
  </FORM>
```

Команда TYPE=radio создает переключатель. Команда NAME назначает этому переключателю любое заданное вами имя.

Если вы хотите, чтобы в группе переключателей отображался только один выбранный переключатель, вы должны присвоить всем переключателям одно и то же имя. Если у всех переключателей одно и то же имя, компьютер обеспечит, чтобы в каждый момент времени можно было выбрать только один переключатель.

Команда VALUE определяет текст, назначенный переключателю. Команда CHECKED отображает переключатель выбранным. Переменная textToDisplay представляет любой текст, который вы хотите отобразить рядом с переключателем.

Переключатель реагирует на такие события, как onBlur, onclick и onFocus. После того как вы установите переключатель, чуть ниже появится сообщение. Вот пример создания трех переключателей:

```
<HTML>
<BODY>
" <H2 .  Куда ваш компьютер отправится сегодня?</H2>
<FORM>
-< INPUT
 • -TYPE = radio
  NAME="groupl"
  VALUE=99
  onClick="litterbox.value = 'Он будет выброшен
       в мусорную корзину. ' ">
  В мусорную корзину
<BR>
<INPUT
  TYPE = radio
  NAME="groupl"
  VALUE=99
  onClick="litterbox.value = 'Он будет выброшен в окно. '">
  В окно
<BR>
<INPUT
  TYPE = radio
  NAME="groupl"
  VALUE=99
  onClick="litterbox.value = 'Он будет сломан и выброшен. '">
```

```
  Разломать его на куски
<BR>
<INPUT
  TYPE - text
  NAME="litterbox"
  VALUE=""
  SIZE = 40>
</FORM>
</BODY>
</HTML>
```

Обратите внимание на то, что все переключатели в предыдущем примере обозначены одним именем `group1`. Таким образом, в каждый момент времени пользователь может выбрать только один из переключателей.

Использование дополнительных средств HTML

Основные HTML-коды, о которых я рассказал в предыдущих разделах настоящей главы, обеспечивают фундаментальные элементы, без которых вам не обойтись при создании и редактировании простых Web-страниц. Однако новые версии HTML предлагают дополнительные средства для создания таблиц, отображения шрифтов, а также разделения Web-страниц на фреймы.

Фреймы позволяют разделить экран Web-броузера на несколько частей, в каждой из которых (во фрейме) отображает отдельная Web-страница или различные части одной и той же Web-страницы.

Хотя эти средства полезны и привлекательны, не забывайте о том, что они работают не со всеми версиями броузеров. Если вы хотите, чтобы ваши Web-страницы увидели все пользователи Internet, ограничьтесь использованием при их создании только основных дескрипторов HTML, о которых я рассказал в настоящей главе.

Глава 24

Создание интерактивных Web-страниц с помощью JavaScript

В этой главе...

- Изучаем основы JavaScript
- Разбираемся с функциями
- Открываем и закрываем окна

HTML-код позволяет создавать замечательные — но при этом абсолютно статические — Web-страницы, напоминающие доски объявлений или рекламные материалы в журналах. Несмотря на то, что такие Web-страницы достаточно функциональны, многие хотят воспользоваться преимуществами компьютера в еше большей степени, для чего создают на своих Web-страницах мини-программы, доступные для всех посетителей этих Web-страниц как небольшие игры или программы анализа котировок акций.

Для создания более интерактивных Web-страниц программисты разработали специализированные языки программирования для Web — *JavaScript* и *VBScript*. Если вы напишете миниатюрную программу, используя язык программирования JavaScript или VBScript, вы сможете разместить ее прямо на своей Web-странице.

Несмотря на столь похожие имена, JavaScript лишь отдаленно напоминает язык программирования Java. JavaScript использует очень простые команды, и программы, написанные на этом языке программирования, могут выполняться только в окне броузера. Java, в свою очередь, использует более сложные команды и может создавать самостоятельные приложения.

Для того чтобы попрактиковаться в написании программ на JavaScript, вы можете использовать простой текстовый редактор (например, текстовый редактор Блокнот из состава Windows или редактор Liberty BASIC), а затем сохранить текст программы в виде файла с расширением `.htm` или `.html`. Потом запустите программу-броузер, выберите команду Файл⇨Открыть, после чего выберите необходимый файл, чтобы увидеть, как броузер интерпретирует команды JavaScript.

Язык программирования JavaScript создан компанией *Netscape*. Для конкуренции с этим языком программирования компания *Microsoft* разработала свой язык программирования, названный VBScript. Этот язык не так популярен, как JavaScript, но он очень прост в использовании, поскольку базируется на языке программирования Visual Basic. Имейте в виду, что если вы напишете программу на JavaScript или VBScript, она не запустится в окне броузера старой версии.

Изучаем основы JavaScript

Для определения начала и конца программы, написанной на JavaScript, вам нужны всего два дескриптора, очень напоминающих дескрипторы HTML, как показано ниже:

```
<script language = "JavaScript">
</script>
```

Вы можете **вставлять программу, написанную на JavaScript, между дескрипторами HTML** <BODY> И </BODY>:

```
<HTML>
<HEAD>
<TITLE>Заголовок документа</TITLE>
</HEAD>
<BODY>
<script language = "JavaScript">
</script>
</BODY>
</HTML>
```

Поскольку старые версии броузеров не всегда понимают команды JavaScript, добавьте две строки после дескриптора <script> и перед дескриптором </script>:

```
<script language = "JavaScript">
<!—
//-->
</script>
```

JavaScript и объекты

Язык программирования JavaScript базируется на *объектах*. (Об объектах подробно мы говорили в главе 19, "Знакомство с объектно-ориентированным программированием".) Объекты обладают тремя характеристиками: *свойствами*, *методами* и *событиями*.

Свойства определяют проявления объекта, например его цвет. Методы определяют действия, которые этот объект может совершать. Наиболее распространен объект `document`, а чаще всего в нем используется команда *write*. События — это то, на что объект может реагировать, например на щелчки мышью.

Вы сможете использовать язык программирования JavaScript, практически ничего не зная об объектах, однако, если вы хорошо разберетесь в объектах, вы намного эффективнее будете использовать JavaScript. А сейчас просто запомните, что JavaScript базируется на объектах, поэтому если увидите странную команду JavaScript вроде `document.write("Привет всем!")`, знайте, что эта команда приказывает объекту document отобразить на экране сообщение Привет всем!.

Эти две строки приказывают старым версиям броузеров трактовать команды JavaScript как комментарии, а значит, просто игнорировать их. Более новые версии броузеров просто-напросто выполняют команды JavaScript, как и любые другие команды, указанные в HTML-документе.

Помимо сохранения команд JavaScript в HTML-файле, вы также можете сохранить их и в отдельном файле. Затем для загрузки и выполнения кода JavaScript используйте атрибут SRC, как показано ниже:

```
<script language = "JavaScript" SRC="program.js">
</script>
```

Этот пример приказывает компьютеру загрузить и выполнить программу, написанную на JavaScript, сохраненную в файле program.js.

Отображение текста

Язык программирования JavaScript содержит специальную команду document.write для вывода сообщений на экран. Для того чтобы понять, как же работает эта программа, рассмотрите следующий пример:

```
document.write("Этот текст отображается на экране.")
```

Если вы хотите внести какое-то разнообразие, используйте обычные дескрипторы HTML для форматирования текста. Например, для отображения текста полужирным просто добавьте соответствующие дескрипторы HTML, как показано ниже:

```
document.write("<B>", Этот текст отображается на экране полужирным.",
"</B>")
```

Команда document.write также позволяет объединять строки с помощью знака "+", как показано ниже:

```
document.write("<B>, Этот текст отображается на экране полужирным.",
"</B>" + "А этот - как обычный текст.")
```

В результате выполнения предыдущей команды на экран выводится следующее сообщение:

```
Этот текст отображается на экране полужирным. А этот •• как обычный
.текст.
```

Создание переменных

В примере использования команды document.write из предыдущего раздела я использовал знак "+" для объединения строк, а также переменные для представления этих строк. В JavaScript для объявления переменной используется волшебная команда var:

```
var имя_переменной
```

В JavaScript вам не нужно объявлять тип переменной; достаточно указать ее имя. После этого вы можете определить переменную для представления строки и использовать знак "+" для объединения строки с этой переменной, как показано ниже:

```
<script language = "JavaScript">
<!—
var mymessage
mymessage = "Золотая рыбка."
Document.write("Какое животное постоянно находится на грани
жизни и смерти? Ответ: " + mymessage)
//-->
</script>
```

Эта программа, написанная на JavaScript, приказывает компьютеру выполнить следующее.

1. Первая строка сообщает компьютеру о том, что все, что заключено между дескрипторами <script> и </script>, является программой, написанной на JavaScript.
2. Вторая строка создает переменную mymessage.
3. Третья строка присваивает переменной mymessage значение Золотая рыбка.
4. **Четвертая строка выводит на экран сообщение** Какое животное постоянно находится на грани жизни и смерти? Ответ: Золотая рыбка.
5. Пятая строка сообщает компьютеру о завершении программы, написанной на JavaScript.

Создание диалоговых окон

Команда document.write очень полезна при отображении текста на экране. Однако JavaScript идет намного дальше, позволяя отображать сообщения в диалоговых окнах. JavaScript позволяет создавать следующие виды диалоговых окон.

- ✓ Диалоговое окно с предупреждением
- ✓ Диалоговое окно с подтверждением
- ✓ Диалоговое окно с запросом

Создание диалогового окна с предупреждением

Чаще всего программы используют *диалоговые окна с предупреждением*. Они обычно отображаются на экране, когда надо уведомить пользователя о том, что какое-то важное событие уже произошло или вот-вот произойдет, как показано на рис. 24.1.

```
alert("Произошел ядерный взрыв. Пришло время эвакуации!")
```

Рис. 24.1. Диалоговое окно с предупреждением, созданное с помощью команды alert языка программирования JavaScript

Команда alert отображает диалоговое окно, которое остается на экране до тех пор, пока пользователь не щелкнет на кнопке OK.

Создание диалогового окна с подтверждением

Диалоговое окно с подтверждением содержит сообщение и предоставляет пользователю возможность щелкнуть на одной из двух кнопок — OK или Отмена. Если пользователь щелкнет на кнопке OK, значение, возвращенное командой confirm, будет равно true. Если пользователь щелкнет на кнопке Отмена, зна-

чение, возвращенное командой confirm, будет равно false. Приведенная ниже программа приводит к созданию диалогового окна с подтверждением, показанного на рис. 24.2.

```
if (confirm("Вы хотите уничтожить всю информацию на жестком диске?"))
  document.write("Идет удаление данных.")
else
document.write("Тогда подождите, пока в работе Windows произойдет
  сбой, и данные будут удалены автоматически.")
```

Рис. 24.2. Диалоговое окно с подтверждением, созданное с помощью команды confirm языка программирования JavaScript

Если пользователь щелкнет на кнопке OK, программа отобразит на экране сообщение Идет удаление данных. Если пользователь щелкнет на кнопке Отмена, программа отобразит на экране сообщение: Тогда подождите, пока в работе Windows произойдет сбой, и данные будут удалены автоматически.

Создание диалогового окна с запросом

Для того чтобы предложить пользователю ввести какие-то данные, многие программы используют *диалоговые окна с запросом,* подобные тому, что показано на рис. 24.3. Диалоговое окно с запросом предлагает пользователю ввести данные, используя команду prompt, как показано ниже:

```
prompt("Сколько раз ваш компьютер зависал сегодня?")
```

Рис. 24.3. Диалоговое окно с запросом, созданное с помощью команды prompt языка программирования JavaScript. По умолчанию задано значение 98

Если вы хотите указать значение по умолчанию, которое пользователь может просто выбрать, не вводя никакой информации самостоятельно (как показано на рис. 24.3), укажите его сразу после сообщения:

```
prompt(("Сколько раз ваш компьютер зависал сегодня?", 98)
```

Если вы не укажете никакого значения по умолчанию, в диалоговом окне с запросом будет указано значение undefined.

Поскольку диалоговое окне с запросом предлагает пользователю ввести определенные данные, вам необходимо создать переменную, которой эти данные будут при-

своены. После создания переменной необходимо сделать так, чтобы она была равна введенным пользователем данным, как показано ниже:

```
<script language = "JavaScript">
<!—
var userdata
userdata = prompt("Сколько раз ваш компьютер зависал сегодня?", 98)
document.write("Пользователь ввел следующие данные = ", userdata)
//-->
</script>
```

Эта программа, написанная на JavaScript, приказывает компьютеру выполнить следующее.

1. Первая строка сообщает компьютеру о том, что все, что заключено между дескрипторами <script> и </script>, является программой, написанной на JavaScript.
2. Вторая строка создает переменную userdata.
3. Третья строка выводит на экран диалоговое окно с запросом Сколько раз ваш компьютер зависал сегодня?; чуть ниже отображается значение по умолчанию, равное 98. Любые введенные пользователем данные после щелчка на кнопке OK присваиваются переменной userdata.
4. Четвертая строка выводит на экран сообщение Пользователь ввел следующие данные =, после чего указывается значение переменной userdata.
5. Пятая строка сообщает компьютеру о завершении программы, написанной на JavaScript.

Разбираемся с функциями

Вместо того чтобы создавать одну массивную программу на JavaScript, вы можете создать подпрограммы, называемые *функциями*.

Функция состоит из следующих четырех частей.

- ✓ **Ключевое слово function.** Эта часть определяет функцию как действительную подпрограмму JavaScript.
- ✓ Имя функции. Эта часть используется JavaScript при "вызове" функции для ее запуска. Ниже приведен пример функции square:

```
function square(число) {
 return значение число * число
}
```

- ✓ **Список аргументов (данных), необходимых для работы функции.** В качестве данных могут выступать числа или строки, но все эти элементы должны разделяться запятыми. В приведенном выше примере используется всего один аргумент — число.
- ✓ **Инструкции функции в фигурных скобках.** Заключенные в фигурные скобки инструкции определяют, как именно должна работать функция. Например, приведенная выше функция получает число, умножает его само на себя, после чего возвращает результат подобного умножения.

Типичная функция выглядит следующим образом:

```
function имя_функции(данные) {
  // одна или несколько инструкций
}
```

Для создания функции вы должны определить имя функции, необходимые для ее работы данные, а также выполняемые ею инструкции. Например, приведенная ниже функция возводит в куб переданное ей число:

```
function square (число) {
  return" значение число * число * число
}
```

Если вы не хотите, чтобы функция возвращала результат, опустите ключевое слово return.

Для того чтобы увидеть все эти инструкции в настоящей программе на JavaScript, введите следующий код в любом текстовом редакторе (например, Блокнот из состава Windows), и сохраните его в виде файла с расширением .htm или .html:

```
<html>
<body>
<script language = "JavaScript">
<!--
function square (number) {
  return number * number
}
function printbig (headlevel , headtext) {
 , document.write("<H", htadlevel, ">, headtext,
                  "</H",headlevel, ">")
}
var myvalue, longstring
myvalue = prompt("Сколько раз ваш компьютер зависал сегодня?", 98)
Longstring = "Компьютер завис столько-то раз = " + square(myvalue)
printbig (2, longstring)
//-->
</script>
</body>
</html> -
```

Настоящая программа, написанная на JavaScript, начиная с дескриптора <script> приказывает компьютеру выполнить следующее.

1. Первая строка сообщает компьютеру о том, что все, что заключено между дескрипторами <script> и </script>, является программой, написанной на JavaScript.
2. Вторая программа определяет всю программу на JavaScript как один большой комментарий для старых версий броузеров, которые не поддерживают JavaScript.
3. Третья строка определяет функцию square, которая принимает данные, сохраненные программой в виде переменной number,
4. Четвертая строка приказывает функции square умножить значение переменной number само на себя, после чего возвратить результат умножения основной программе JavaScript.
5. Пятая строка обозначает окончание функции square.

6. Шестая строка определяет функцию printbig, которая принимает данные, сохраненные программой в виде переменных headlevel и headtext.
7. Седьмая стока создает дескриптор HTML для определения уровня заголовка и отображения соответствующего текста заголовка.
8. Восьмая строка обозначает окончание функции printbig.
9. Девятая **строка** создает **две переменные** myvalue и longstring.
10. Десятая строка выводит на экран диалоговое окно с запросом Сколько раз ваш компьютер зависал сегодня?; чуть ниже отображается значение по умолчанию, равное 98. Любые введенные пользователем данные после щелчка на кнопке OK присваиваются переменной myvalue.
11. Одиннадцатая строка вызывает функцию square, используя значение переменной myvalue. Полученный результат добавляется к строке Компьютер завис столько-то раз. Вся полученная строка присваивается переменной longstring.
12. Двенадцатая строка вызывает функцию printbig и передает ей число 2 и значение переменной longstring. В данном случае функция printbig создает заголовок второго уровня и отображает на экране значение переменной longstring.

Открываем и закрываем окна

Хотя ваш броузер отображает в каждый момент времени только одно окно, вы можете открыть несколько окон для отображения различных Web-страниц. (Очень часто при просмотре Web-узлов открываются дополнительные окна, содержащие рекламные сообщения.)

Открытие окна

Для открытия окна между дескрипторами <script> и </script> помешается команда open, как показано в следующем примере:

```
<script language = "JavaScript" SRC="program.js">
<!--
  Имя_окна = window.open(адрес или имя Web-страницы)
//-->
</script>
```

Вы также можете использовать команду window.open, назначив ее определенному элементу интерфейса пользователя, например кнопке. О совместном использовании команд и элементов интерфейса пользователя мы говорили в главе 23, "Забавы с HTML".

Вы должны обязательно определить параметр Имя_окна, в качестве которого может выступать любое имя окна. Вы также должны определить, что именно должно отображаться в окне. Если вы хотите, чтобы в окне отображалась определенная Web-страница, вы должны указать ее имя файла, как показано в следующем примере:

```
MyWindow = window.open("index.html")
```

Эта команда открывает новое окно MyWindow и отображает в нем Web-страницу, сохраненную в виде файла index.html. Если вы хотите, чтобы в окне отображался определенный Web-узел, вы должны указать его полный адрес, как показано в следующем примере:

```
MyWindow = window.open("http://www.dialektika.com")
```

Как выглядит окно

Для того чтобы предоставить вам больший контроль над внешним видом окна, вы можете не только изменять его размер, но и отображать или скрывать определенные элементы. Если вы определяете внешний вид, вы должны определить его второе имя, за которым следуют все необходимые атрибуты, как показано ниже:

```
MyWindow = window.open("index.html", "второе_имя",
                       "toolbar=no, resizable=yes")
```

В результате откроется окно, содержащее Web-страницу index.html. В этом окне не будет панели инструментов, но вы сможете изменять его размеры. При обращении к этому окну из другого окна вам необходимо использовать его второе имя.

Атрибуты изменяют внешний вид окна, например добавляют панель инструментов и строку меню. Назначение различных атрибутов, которые вы можете определить для окна, приведено ниже.

- ✓ toolbar [=yes|no] | [=1|0]. Отображение панели инструментов в верхней части окна вместе с такими кнопками, как Назад, Вперед, Остановить и др.
- ✓ location [=yes|no] I [=1|0]. Отображение текстового поля, содержащего адрес Web-страницы.
- ✓ directories [=yes|no] I [=1|0]. **Отображение кнопок для перемещения по каталогам в верхней части окна.**
- ✓ status [=yes|no] 1 [=1|0]. Отображение строки состояния в нижней части окна.
- ✓ menubar [=yes|no] I [=1|0]. **Отображение меню в верхней части окна.**
- ✓ scrollbars [=yes|no]] [=1|0]. Отображение вертикальной и горизонтальной полос прокруток, если размер документа превышает размер экрана.
- ✓ resizeble [=yes|no] | [=1|0]. Позволяет пользователю изменять размеры окна.
- ✓ width=pixels. **Определение ширины окна в пикселях.**
- f ✓ height pixels. **Определение высоты окна в пикселях.**

Например, если вы хотите открыть окно, но при этом скрыть панель инструментов, вы можете присвоить атрибуту toolbar значение по или 0, как показано ниже.

С использованием варианта yes/no:

```
MyWindow = window.open(("index.html", "второе_имя", "toolbar=no")
```

Или **с использованием варианта** 1/0:

```
: MyWindow = window.open(("index.html", "второе_имя", "toolbar=0")
```

Закрытие окна

После того как вы открыли окно, вы наверняка захотите его закрыть. Для того чтобы закрыть окно, вы должны использовать команду close:

```
Имя_окна.close()
```

Эта команда закрывает окно с именем Имя_окна. Например, если вы хотите закрыть окно с именем Реклама, воспользуйтесь следующей командой:

```
Реклама.close()
```

Вместе с командой close используется то же самое имя окна, которое вы использовали при его открытии, Если вы открываете окно с помощью следующей команды:

```
WeirdStuff = window.open("index.html")
```

то закрываете с помощью следующей команды:

```
WeirdStuff.close()
```

JavaScript — это полноценный язык профаммирования, поэтому полностью рассмотреть его возможности в рамках одной главы просто невозможно. Более подробные сведения JavaScript вы найдете в книге *JavaScript "для чайников". 2-е издание*, написанной Эмили А. Вандер Вер (выпушена издательством "Диалектика").

Глава 25

Использование Java-аплетов на Web-страницах

В этой главе...

- Как работают Java-аплеты
- Добавление Java-аплета к Web-странице
- Поиск бесплатных Java-аплетов

Язык программирования JavaScript позволяет создавать программы двух типов — полноценные приложения (текстовые процессоры или Web-броузеры) и небольшие аплеты, которые выполняют только в окне Web-броузера. (В настоящей главе мы будем говорить об использовании Java для создания аплетов, а не полноценных приложений.)

Когда вы пишете аплеты на Java, вы получаете возможность добавлять на Web-страницы более изысканные средства. (Если вы не хотите писать собственные Java-аплеты, вы всегда можете использовать аплеты, написанные кем-то другим.)

Написание Java-аплетов оказывается достаточно трудным занятием и отнимает много времени, что зависит, прежде того, от того, насколько сложным будет создаваемый вами аплет. Если вам необходимы подробные инструкции по написанию программ на Java, приобретите книгу *Java Programming For Dummies* (Donald J. Koosis and David Koosis) издательства *IDG Books Worldwide, Inc.*

Как работают Java-аплеты

Java-аплет — это миниатюрная программа, написанная на языке программирования Java. В отличие от программ, написанных на C++ (он компилирует их в машинные коды), Java-аплеты преобразуются из исходного кода в специальный формат — байт-код.

Байт-код — это специальный формат файлов, уникальный для программ, написанных на Java. Некоторые компиляторы Java, такие как Visual Café компании *WebGain*, способны преобразовать исходный код программ в байт-код или непосредственно в машинные коды.

Исходный код программ, написанных на Java, сохраняется в ASCII-файле с расширением `.java`, например `Virus.java`. При компиляции Java-программы в байт-код вы создаете отдельный файл с расширением `.class`, например `Virus.class`.

Java и JavaScript

Оба языка программирования, Java и JavaScript, позволяют создавать интерактивные Web-страницы, чего не позволяет делать обычный язык HTML. Но в каких ситуациях вам следует использовать Java, а в каких ситуациях лучше отдать предпочтение JavaScript?

JavaScript намного проще Java в изучении и в использовании, поэтому позволяет быстрее приступить к созданию интерактивных Web-страниц. С другой стороны, Java более мощный и гибкий язык программирования, позволяющий создавать средства, которые очень сложно, вернее — почти невозможно, создать с помощью JavaScript.

Вообще, JavaScript следует использовать при выполнении коротких, простых "заданий, а Java — более сложных. Конечно, вы можете использовать оба языка программирования для создания элементов одной Web-страницы, чтобы получить самые лучшие результаты.

Поскольку компьютеры понимают только машинный код, ни один из них не знает, как выполнить Java-программу, сохраненную в формате байт-кода. Если вы хотите выполнить Java-программу, сохраненную в формате байт-кода, вам следует использовать специальную (причем бесплатную) программу, которая называется виртуальной машиной Java (Java Virtual Machine — Java VM). (Вам очень повезло, если на вашем компьютере установлена последняя версия Web-броузера Microsoft Internet Explorer или Netscape Navigator.)

Java-программы могут выполняться на любом компьютере, на котором установлена виртуальная машина Java, включая компьютеры, работающие под управлением Windows, Mac OS или UNIX. Если вы откомпилировали Java-программу в байт-код, вы сможете выполнять ее на компьютере любого типа, работающем под управлением любой операционной системы.

Для того чтобы показать вам, как выглядит Java-аплет (не стоит беспокоиться, что вам непонятно назначение той или иной команды), я привожу пример такого аплета, который отображает сообщение Не смотри на меня! на экране:

```
Import java.awt.*ж
import java.applet.Applet;
public class TrivialApplet extends Applet
{
  Font f = new Font("TimesRoman", Font.BOLD, 24);
  public void init() {
  repaint();
  }
  public viod paint ( Graphics g ) {
  g.setFont(f)
  g.setColor(Color.blue);
  g.drawString ("Не смотри на меня!" , 15, 75 );
      }
}
```

Результат выполнения программы приведен на рис. **25.1.**

Ограничение возможностей Java-аплетов

Java-аплеты — это миниатюрные программы, т.е. существует возможность удалить данные на жестком диске или повредить даже аппаратное обеспечение вирусом или "троянским конем", написанными на Java не очень порядочным программистом. Для предотвращения атак на ваш компьютер со стороны подобных программ Java просто запрещает им выполнять определенные дей-

ствия в системе, например удаление файлов, чтение содержимого жестких дисков, переименование файлов, создание каталогов, а также запуск любых внешних программ.

Конечно же, с момента представления Java люди нашли массу "дыр" в его системе безопасности. Правда, ни одно из подобных слабых мест не позволило хакерам нанести урон, но их наличие указывает, что Java все-таки предоставляет возможности для атак на компьютеры, а серьезность наносимого урона всецело зависит от профессионализма и творческого подхода программиста.

Рис. 25.1. Исходный код Java-аплета, выполненный в окне Web-броузера

Если язык программирования Java бесплатен, зачем платить за компилятор Java?

Стремясь обеспечить как можно большие распространенность и универсальность Java, компания Sun *Microsystems* предлагает бесплатный компилятор Java, исходный код и некоторые инструментальные средства, которые вы можете совершенно бесплатно загрузить с Web-узла компании (`java.sun.com`). Но если у вас есть бесплатный компилятор Java, зачем вам приобретать еще один?

Во-первых, компилятор от компании Sun *Microsystems* предоставляет только базовые возможности для написания и компилирования программ. Использование компилятора от компании *Sun Microsystems* во многом напоминает прогулку пешком из Санкт-Петербурга в Одессу: теоретически вы сможете это сделать, однако полет на самолете окажется намного удобнее, хотя за него придется платить.

Во-вторых, коммерческие компиляторы Java предлагают дополнительные средства по сравнению с бесплатным компилятором от компании *Sun Microsystems.* Например, JBuilder (от компании *Borland)* позволяет проектировать интерфейс пользователя Java-программы в графическом виде, что избавляет от необходимости самостоятельно писать соответствующий программный код. Visual Café от компании *WebGain* предлагает настоящий компилятор Java, который способен преобразовать Java-программы или в байт-код (для запуска на компьютерах различных типов) или в машинные коды (для создания программ, оптимизированных для выполнения в среде Microsoft Windows).

Вы можете набирать текст программы в любом текстовом редакторе (редакторе Liberty BASIC или Блокнот из состава Windows). Для преобразования Java-программы в байт-код вам следует использовать компилятор Java, например бесплатный компилятор от компании *Sun Microsystems* (java.sun.com) или коммерческий компилятор JBuilder от компании *Borland* (www.borland.com) либо CodeWarrior от компании *Metrowerks* (www.metrowerks.com).

Добавление Java-аплета tut Web-страницу

Завершив преобразование Java-аплета в байт-код, вы готовы к использованию дескрипторов HTML для запуска Java-аплета на Web-странице. Для добавления Java-аплета на Web-страницу используются два специальных дескриптора, помещаемых между дескрипторами <BODY> и </BODY>:

```
APPLET. CODE = "Имя_Java-аплета">
Сообщение, отображаемое, когда аплет нельзя выполнить •
</APPLET>
```

Значение переменной Имя_Java-аплета соответствует названию настоящего Java-аплета, который вы хотите выполнить. Например, если вы хотите выполнить аплет Message.class, соответствующие дескрипторы будут выглядеть следующим образом:

```
<APPLET CODE = "Message.class">
Сообщение, отображаемое, когда аплет нельзя выполнить
</APPLET>
```

Между дескрипторами <APPLET> и </APPLET> можно поместить одну или несколько строк текста. Этот текст будет отображаться только в том случае, если Web-броузер не сможет выполнить соответствующий Java-аплет. Поэтому вместо того, чтобы отображать на экране пустую Web-страницу, броузер отобразит сообщение, которое подскажет пользователю, что Java-аплет нельзя выполнить.

Определение размеров окна Java-аплета

Помимо всего прочего, вы можете задать размеры окна, в котором будет отображаться выполняемый Java-аплет, используя команды WIDTH и HEIGHT, как показано ниже:

```
<APPLET CODE = "Message.class"> WIDTH = 250 HEIGHT = 100> :
Сообщение, отображаемое, когда аплет нельзя выполнить
</APPLET>
```

В данном случае Java-аплет будет выполняться в окне размером 250x100 пикселей.

Определение расположения окна Java-аплета

При использовании Java-аплета на Web-странице вы можете применить команду ALIGN, определяющую расположение текста Web-страницы по отношению к окну аплета. Вам доступны три варианта команды ALIGN.

- ✓ ALIGN=TOP. Текст выравнивается по верхнему краю окна аплета.
- ✓ ALIGN=MIDDLE. Текст выравнивается по середине окна аплета.
- ✓ ALIGN=BOTTOM. Текст выравнивается по нижнему краю окна аплета.

Все три способа выравнивания текста показаны на рис. 25.2.

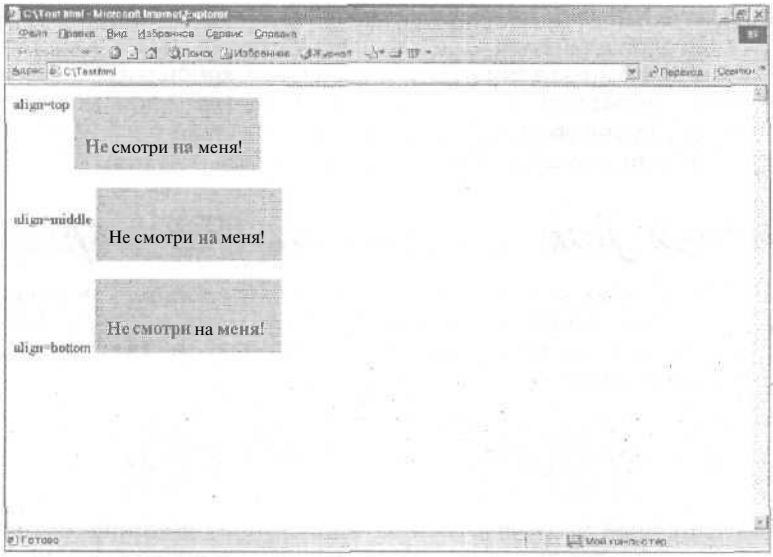

Рис. 25.2. Три различных способа выравнивания текста

Добавление пространства вокруг окна Java-аплета

Чтобы текст не располагался слишком близко к Java-аплету, определите расстояние от текста до аплета по горизонтали и вертикали, воспользовавшись командами HSPACE и VSPACE соответственно:

```
<APPLET CODE = "Message.class"> WIDTH = 250 HEIGHT = 100> VSPACE = 25
        HSPACE = 15>
Сообщение, отображаемое, когда аплет нельзя выполнить
</APPLET>
```

В данном случае расстояние между текстом и аплетом по вертикали равно 25 пикселей, а по горизонтали — 15 пикселей. Еще один пример использования команд HSPACE и VSPACE приведен на рис. 25.3.

Поиск бесплатных Java-аплетов

Если вы хотите создавать собственные Java-аплеты, вам придется потратить время на изучение языка программирования Java, а он покажется достаточно сложным для новичков. Пока вы не освоите программирование на Java (или вообще вместо этого), лучше использовать Java-аплеты, написанные другими людьми.

Вы можете добавлять на свои Web-страницы готовые Java-аплеты, написанные другими, более опытными программистами. Или, если доступен исходный код, вы можете внести в аплет необходимые изменения и снова его откомпилировать; это часто удобно для того, чтобы быстрее разобраться в тонкостях Java.

Для поиска Web-узлов, предлагающих Java-аплеты, **посетите свою любимую поисковую** машину в Internet, **например Hotbot** (www.hotbot.com) **или Excite** (www.excite.com), и введите **критерий поиска Java applet. Или посетите поисковую** машину **Yahoo!** (www.yahoo.com) и последовательно пользуйтесь **ссылками** Computers and Internet, Programming Languages, Java **и, наконец,** Applets. **Вы**

получите список десятков Web-узлов, предлагающих массу разнообразных Java-аплетов, а также их исходных кодов, доступных для загрузки.

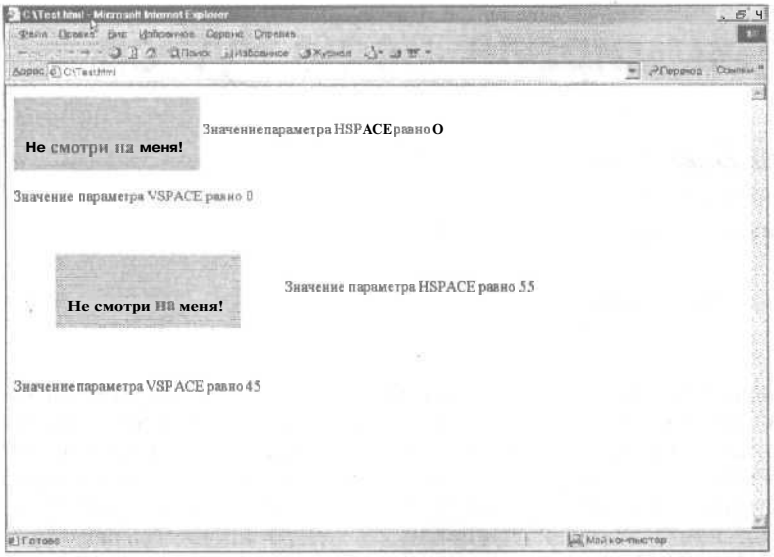

Рис. 25.3. Команды HSPACE и VSPACE позволяют добавлять расстояние между текстом и аплетом на Web-странице

Глава 26

Программирование на Python

В этой главе...

- Знакомство с Python
- Управляющие структуры
- Циклы
- Модульное программирование

Официально *Python* — это интерпретируемый, интерактивный, объектно-ориентированный язык программирования с достаточно странным названием, он получил свое имя в честь популярного в Британии комедийного телевизионного шоу. Python — это язык общего назначения, который особенно хорошо подходит для управления текстом, хотя программисты часто используют его в качестве "связующего" языка, который объединяет различные программные пакеты, благодаря чему они могут обмениваться данными при совместной работе над проектами.

Python достаточно популярен, поскольку прост в изучении и *переносим*, что означает, что если вы напишете программы на Python, то сможете выполнять их на различных типах компьютеров, работающих под управлением таких операционных систем, как Linux, Windows или Macintosh. Несмотря на то, что Python часто используется для создания Internet-приложений, например программ, получающих данные от Web-страниц и сохраняющих их в базе данных, его можно использовать и для создания практически любых других приложений.

Например, в Университете Калифорнии (г. Ирвин, США) создана программа для управления учебным процессом, написанная исключительно на Python. Даже поисковое средство Google (`www.google.com`) при работе полагается на Python. Поэтому, когда вы будете решать, какой язык программирования выбрать для написания следующей программы, не стоит забывать и о Python.

Знакомство с Python

Python работает на самых различных платформах, поэтому вам для работы потребуется определенная версия интерпретатора, предназначенная именно для той платформы, с которой вы работаете. Вы найдете необходимую версию интерпретатора, если посетите официальный Web-узел Python по адресу `www.python.org`. Установив интерпретатор Python на компьютере, вы готовы приступать к программированию на этом языке. На рис. 26.1 показан внешний вид окна интерпретатора Python для платформы Windows.

Поскольку Python, как и BASIC, интерпретируемый язык программирования, вы можете сразу приступать к вводу текста программ и сразу видеть, как они работают. Python может похвастаться многими возможностями C/C++, но он проще в изучении, поэтому попробуйте изучить этот язык программирования как подготовительный этап к освоению сложного синтаксиса C/C++.

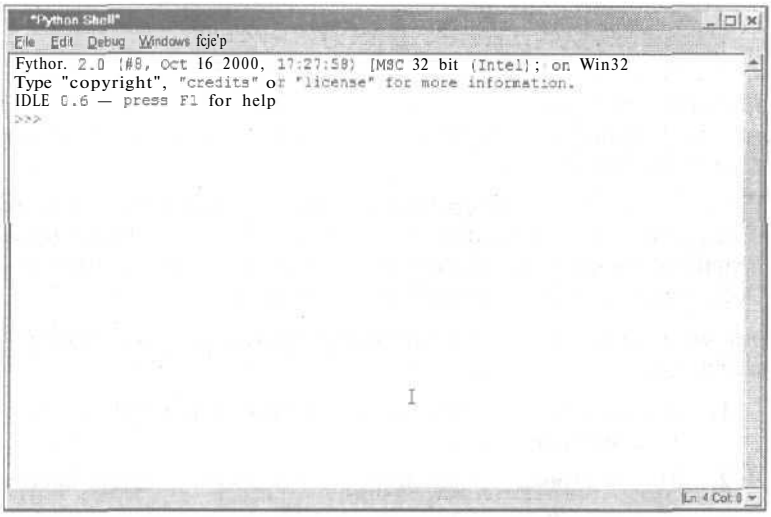

Рис. 26.1. Внешний вид окна интерпретатора Python для платформы Windows

Не стоит особо беспокоиться о технических подробностях использования Python. Просто просмотрите материал настоящей главы, чтобы получить общее представление об этом языке программирования и понять, чем он отличается от Liberty BASIC, а также какими преимуществами и недостатками обладает каждый из них.

Работа с данными

Во многих языках программирования, в частности C/C++ и Java, вы должны определить тип данных, которые будут в той или иной переменной — `string`, `real` и т.д. Основная идея состоит в том, что исключается возможность по ошибке присвоить числовое значение переменной типа `string`, что приводит к зависанию программы.

Создатели Python придерживались противоположной точки зрения. Вместо того чтобы заставлять вас объявлять все необходимые переменные и четко указывать допустимый тип данных для каждой из них, Python позволяет присваивать переменным практически любые значения. Для того чтобы присвоить переменной какое-то значение, используется простой знак равенства:

```
Имя_переменной = число
```

Или же это можно сделать следующим образом:

```
Имя_переменной = "строка"
```

Если вы хотите, чтобы переменная содержала строку, обязательно заключайте эту строку в двойные кавычки.

Ниже приведен пример типичного сеанса работы с Python.

```
X» cheese4sale = 25
>>> print 'Этот магазин предлагает следующее количество сортов сыра -',
cheese4sale
Этот магазин предлагает следующее количество сортов сыра - 25
>>> cheese4sale = 'швейцарского сыра'
>>> print 'Извините, но мы только что продали последнюю головку ',
cheese4sale
```

```
Извините, но мы только что продали последнюю головку швейцарского сыра
>>>
```

Строка приглашения Python имеет следующий вид: >>>. Любая строка, в которой эта последовательность отсутствует, означает, что соответствующие данные отображаются компьютером на экране.

Python — это язык программирования, чувствительный к регистру используемых символов, поэтому команда PRINT не будет распознана как правильная (правильный вариант — print), а переменные cheese4sale и Cheese4sale — это совершенно разные вещи.

Эта написанная на Python программа приказывает компьютеру выполнить следующее.

1. Первая строка приказывает создать переменную cheese4sale и присвоить ей значение 25.
2. Вторая строка приказывает отобразить на экране сообщение: Этот магазин предлагает следующее количество сортов сыра -, а затем **указать значение переменной** cheese4sale, **равное** 25.
3. **Третья строка** представляет **собой то, что компьютер** вывел на **экран, — сообщение:** Этот магазин предлагает следующее количество сортов сыра - 25.
4. **Четвертая строка приказывает создать переменную** cheese4sale **и присвоить** ей значение швейцарского сыра.
5. **Пятая строка приказывает отобразить на** экране **сообщение:** Извините, но мы только что продали последнюю головку, **а затем** указать Значение **переменной** cheese4sale, **равное** швейцарского сыра.
6. **Шестая строка представляет собой то, что компьютер вывел на экран, — сообщение:** Извините, но мы только что продали последнюю головку швейцарского сыра.
7. Седьмая строка представляет собой обычную командную строку Python, готовую к получению новых инструкций.

Вместо того чтобы вводить команды строка за строкой в интерпретаторе Python, вы можете написать программу целиком, используя обычный текстовый редактор, такой как Блокнот из состава Windows, сохранить ее, после чего загрузить в интерпретатор Python весь файл целиком.

Конечно же, Python может выполнять простейшие математические операции над значениями, как показано ниже:

```
>>>X = (4+5)*7
>>> print X
63
```

Вы также можете манипулировать строками, используя знак "плюс" для *конкатенации* строк (за этим странным термином скрывается обычное объединение строк), как показано ниже:

```
>>> X - 'Bo'
>>>Y - 'the'
>>> z - 'Cat.'
```

```
>>> print X+Y+Z
Bo the Cat.
```

Структуры данных

Одно из основных преимуществ Python состоит в том, что он поддерживает встроенные структуры данных высокого уровня, которые программистам обычно приходится создавать самостоятельно. Поэтому, вместо того чтобы заставлять вас тратить время на написание инструкций для создания сложной структуры данных, например списка (см. главу 18, "Связанные списки и указатели"), Python создает их самостоятельно, позволяя вам больше внимания уделить написанию собственно программы.

Python поддерживает две наиболее мощные структуры данных — *списки* и *словари.* Списки во многом напоминают массивы. (Подробно о массивах мы говорили в главе 16, "Сохранение информации в массивах".) Но если массивы позволяют хранить данные только одного типа, то списки позволяют хранить и числа, и строки.

Списки также намного удобнее при сохранении или удалении элемента в середине списка. Сохранение или удаление элемента в середине массива приводит или к появлению "пропусков" (а значит, к трате впустую такой ценной памяти компьютера) или к переупорядочению всего массива при сохранении или удалении элемента (а значит, к трате времени).

Для создания списка на Python вам необходимо создать переменную и перечислить все нужные элементы в квадратных скобках, не забывая помещать в кавычки строковые элементы, как показано ниже:

```
Переменная_список = [элемент1, элемент2, элемент3, элемент4]
```

Изучите следующий пример сеанса работы с Python и попробуйте выполнить соответствующие команды на своем компьютере:

```
>>> randomthoughts = ['girls', 3.1415, 'money', 900, 3.2, 'dogs']
>>> print randomthoughts
['girls', 3.1415, 'money', 900, 3.2, 'dogs']
```

Списки представляют собой достаточно гибкие средства для хранения данных различных типов, поэтому Python также предлагает встроенные команды для манипулирования списками, как показано ниже.

- ✓ `append(x)`. Добавление элемента в конец списка, где x — число или строка.
- ✓ `extend(x)`. Добавление элемента в конец списка, где x — еще один список, например [45, "Greetings", 9.09].
- ✓ `insert(i,x)`. Добавление элемента в список, где x представляет этот элемент, а i — его расположение в списке. В Python первому элементу соответствует номер 0, второму элементу — номер 1 и т. д.
- ✓ `remove(x)`. Удаление элемента из списка, где x представляет этот элемент.
- ✓ `pop([i])`. Удаление элемента, расположенного на позиции i, из списка.
- ✓ `index(x)`. Возвращает **номер элемента x (его *индекс)* в списке.**
- ✓ `count(x)`. Возвращает число, равное тому, сколько раз элемент x встречается в списке.
- ✓ `sort()`. Сортировка списка. Python сортирует числа по возрастанию, а строки — по алфавиту.

- ✓ `reverse()`. Сортировка списка в обратном порядке, т.е. эта команда выполняет действие, обратное действию команды `sort()`.

Для того чтобы дать вам понять, как работают эти команды, я привожу следующий пример сеанса работы с интерпретатором Python:

```
>>> cats = ['Bo', 'Scraps', 'Tasha', 'Nuit']
>>> cats.insert (1, "Thomas")
>>> print cats
['Bo', 'Thomas', 'Scraps', 'Tasha', 'Nuit']
>>> cats.reverse ()
>>> print cats
['Nuit', 'Tasha', 'Scraps', 'Thomas', 'Bo']
>>> cats.sort()
>>> print cats
['Bo', 'Nuit', 'Scraps', 'Tasha', 'Thomas']
>>> cats.remove('Thomas')
>>> print cats
['Bo', 'Nuit', 'Scraps', 'Tasha']
>>>
```

Второй мощной (и сложной) структурой данных, поддерживаемой Python, является словарь, который позволяет сохранять и получать данные, используя уникальные ключи. Ниже приведен пример работы Python со словарями.

```
>>> dict = {}
>>> dict['government'] = "Здесь встречаются белые воротнички криминального мира."
>>>dict['prison'] = "Здесь встречаются синие воротнички криминального мира."
>>> print dict['government']
Здесь встречаются белые воротнички криминального мира.
```

Эта написанная на Python программа приказывает компьютеру выполнить следующее.

1. Первая строка приказывает создать пустую структуру данных типа словарь.
2. **Вторая** строка **приказывает компьютеру сохранить строку:** Здесь встречаются белые воротнички криминального мира **в словаре, который** идентифицируется по ключу `'government'`.
3. **Третья строка приказывает компьютеру сохранить строку:** Здесь встречаются синие воротнички криминального мира **в словаре, который** идентифицируется **по ключу** `'prison'`.
4. Четвертая строка приказывает ввести данные, содержащиеся в словаре, который идентифицируется по ключу `'government'`.
5. **Пятая строка** содержит **выведенное на экране** сообщение: Здесь встречаются белые воротнички криминального мира, что является реакцией **Python на команду, указанную в четвертой строке.**

Не стоит особо беспокоиться о технических подробностях, касающихся списков и словарей. Просто помните о том, что Python предлагает собственные встроенные структуры данных, которые вы можете немедленно использовать в своих программах, в то время как другие языки программирования, например C/C++ или BASIC, не позволяют этого делать. Если вы используете язык программирования, который не предлагает структуры данных (списки и словари), вам придется создавать их самостоятельно, что требует много времени на написание и отладку, делая программирование более сложным занятием, чем оно должно быть.

Комментарии

Одно из наиболее важных свойств любого языка программирования — возможность создавать комментарии к программам. Это позволяет другим людям (а часто и вам самим через некоторое время) лучше разобраться в том, как же работает программа.

В BASIC для создания комментариев используется один из следующих двух символов.

- ✓ **REM**
- ✓ **' (апостроф)**

В Python при создании комментариев используется символ #. Ниже приведен пример типичного комментария на Python:

```
# Пример типичного комментария на Python
```

Интерпретатор Python игнорирует любые данные, содержащиеся справа от символа #. Поэтому вы можете размешать комментарии как на отдельных строках, так и на одной строке с работающим кодом программы.

Использование управляющих структур

Управляющие структуры позволяют вашей программе принимать решение о том, какой набор инструкций необходимо выполнить, в зависимости от значений некоторых переменных. Чаще других используется управляющая структура — инструкция if, которая как бы говорит компьютеру: если выполняется определенное условие, выполни такие-то инструкции. Инструкция if в случае Python выглядит следующим образом:

```
>>> if (условие):
Первая инструкция
 Последняя инструкция
```

В качестве условия выступает булево выражение x > 45 или total = 5, значение которого может быть равно true или false. Если условие не выполняется (т. е. равно false), компьютер просто игнорирует соответствующую инструкцию if. Если условие выполняется (т. е. равно true), компьютер выполняет соответствующую инструкцию if, как показано в следующем примере:

```
>>> age=56
>>> if age < 65:
print 'Пенсия вам еще не положена.'
 Пенсия вам еще не положена.
>>>
```

В языке BASIC вы могли создавать инструкцию IF THEN ELSE (подробно об этом мы говорили в главе 9, "Принятие решений с помощью управляющих операторов"), как показано ниже:

```
IF (условие) THEN
Инструкции, выполняемые, если условие выполняется -
ELSE
Инструкции, выполняемые, если условие не выполняется
END IF
```

Инструкция IF THEN ELSE гарантирует, что компьютер обязательно выполнит определенную последовательность инструкций, независимо от того, выполняется условие или нет. В Python инструкция if else выглядит следующим образом:

```
if age < 65:
print 'Пенсия вам еще не положена.'
else:
print 'Лучше откладывайте деньги, пока государственная система не
обанкротилась.'
```

В данном случае инструкции, указанные после инструкции else, выполняются без предварительной проверки каких-либо условий. Если вам необходимо, чтобы перед выполнением инструкций обязательно проверялось определенное условие, следует использовать инструкцию if elif, как показано ниже:

```
if (условие1):
Инструкции, выполняемые, если условие1 выполняется
elif (условие2):
Инструкции, выполняемые, если условие2 выполняется
```

Вы также можете использовать инструкцию if elif else:

```
if (условие1):
Инструкции, выполняемые, если условие1 выполняется
elif (условие2):
Инструкции, выполняемые, если условие2 выполняется
else:
Инструкции, выполняемые, если ни одно условие из перечисленных выше
не выполняется
```

Можно при необходимости использовать несколько инструкций elif, как показано ниже:

```
if (условие1):
Инструкции, выполняемые, если условие1 выполняется
fcel-if (условие2):
Инструкции, выполняемые, если условие2 выполняется
- if (условие3): .
Инструкции, выполняемые, если условие3 выполняется
elif (условиеX):
Инструкции, выполняемые, если условиеX выполняется

else:
Инструкции, выполняемые, если ни одно условие из перечисленных выше
не выполняется
```

Использование циклов

Очень важна для любого языка программирования возможность многократно выполнять одну или несколько инструкций, избавляя вас от ввода несколько раз подряд одних и тех же инструкций. Чаще всего при создании циклов в Python используются такие инструкции, как while и for. В следующих подразделах подробно рассмотрена каждая из них.

Инструкция while

Инструкция while языка Python функционирует практически так же, как и инструкция WHILE WEND языка BASIC, повторяя одну или несколько инструкции при выполнении определенного условия:

```
while (условие):
- Инструкции, выполняемые, если условие выполняется
```

При использовании инструкции while обязательно убедитесь в том, что в определенной ситуации цикл изменит значение условия с true на false. В противном случае вы создадите бесконечный цикл, который приведет к зависанию программы.

Для того чтобы понять, как же работает инструкция while, изучите следующий пример:

```
>>> age = 1
>>> while age < 5:
print age
age = age + 1
1
2
3
4
>>>
```

Эта написанная на Python программа приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру создать переменную age и присвоить ей значение 1.
2. Вторая строка начинает выполнение инструкции while и приказывает компьютеру выполнять инструкции, указанные в строках третьей и четвертой, до тех пор, пока выполняется булево выражение age < 5.
3. **Третья строка приказывает компьютеру вывести на экран значение переменной age.**
4. Четвертая строка приказывает компьютеру увеличить значение переменной age на 1.
5. Строки с пятой по восьмую показывают результат выполнения инструкции while языком Python четыре раза.
6. Девятая строка представляет собой обычную командную строку Python, готовую к получению новых инструкций.

Инструкция for

Инструкция for в языке Python функционирует несколько не так, как инструкция FOR NEXT в BASIC. В BASIC (и многих других языках) инструкция FOR NEXT позволяет выполнять блок инструкций строго определенное число раз:

```
FOR I = 1 TO 5
Инструкции
NEXT I
```

В Python инструкция for не определяет, сколько инструкций должно быть выполнено. Вместо этого она использует количество элементов списка для того, чтобы определить, сколько раз должны быть выполнены инструкции цикла. Например, если вы хотите, чтобы инструкции цикла выполнялись пять раз, вам необходимо предоставить инструкции for список, содержащий пять элементов, как показано ниже:

```
>>> mylist = Peats', 58.3, 3, 'dogs', 'five and twenty']
>>> for x in mylist:
print x
cats
58.3
3
dogs
five and twenty
>>>
```

Эта написанная на Python программа приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру создать список, содержащий следующие пять переменных: 'cats', 58.3, 3, 'dogs', 'five and twenty', и сохранить его в качестве переменной mylist.
2. Вторая строка начинает выполнение инструкции for и приказывает компьютеру выполнять инструкции для каждого элемента списка, представляемого переменной mylist.
3. Третья строка приказывает компьютеру вывести на экран элемент, представляемый переменной mylist.
4. Строки с четвертой по седьмую показывают результат выполнения инструкции for языком Python пять раз.
5. Девятая строка представляет собой обычную командную строку Python, готовую к получению новых инструкций.

Написание подпрограмм на Python

Написать одну большую программу, которая работала бы безошибочно, практически нельзя, поэтому большинство языков программирования, в том числе и Python, позволяют программистам писать подпрограммы, выполняющие определенные задачи. Таким образом, вы получите одну большую программу, состоящую из нескольких небольших, которые в Python называются *функциями*.

Для определения функции необходимо использовать ключевое слово def, за которым должно следовать имя функции:

```
def имя_функции (список параметров):
инструкции
```

Если вам необходимо создать функцию, которая должна принимать строковое значение и выводить сообщение на экран, например, это можно сделать следующим образом:

```
def moronspotter (mytarget):
mytarget + ' - это имя звучит достаточно глупо!'
```

Для того чтобы выполнить эту функцию, вам необходимо *вызвать* ее по имени, обязательно передав необходимые данные:

```
>>> moronspotter ('Bill McPherson') -
Bill McPherson - это имя звучит достаточно глупо!
>>>
```

Этот фрагмент программы на Python приказывает компьютеру выполнить следующее.

1. Первая строка приказывает компьютеру выполнить функцию moronspotter **и передать ей строку** 'Bill McPherson'.
2. **Вторая строка показывает результат передачи строки** 'Bill McPherson' **функции** moronspotter. **На экране** отображается **сообщение** Bill McPherson - это имя звучит достаточно глупо!
3. Третья строка представляет собой обычную командную строку Python, готовую к получению новых инструкций.

Python также позволяет вам создавать стандартный список параметров, используемый в том случае, если при вызове функции не передается никаких параметров. Например, в следующем примере функция moronspotter использует стандартный параметр 'Bobby Lee':

```
def moronspotter (mytarget = 'Bobby Lee'):
mytarget + '- это имя звучит достаточно глупо!'
```

Если вы вызовете эту функцию, передав ей определенные данные, результат будет иметь следующий вид:

```
>>> moronspotter ('Von Markey')
Jon Markey - это имя звучит достаточно глупо!
```

Но если вы не передадите ей никаких данных, результат будет иметь уже такой вид:

```
>>> moronspotter ()
Bobby Lee- это имя звучит достаточно глупо!
>>>
```

В перечисленных выше примерах все, что вы вводите в командной строке Python, отображается после последовательности символов >>>. Если же компьютер реагирует на ваши команды, он выводит сообщение в отдельной строке без последовательности символов >>>.

Часть VII

Великолепные десятки

В этой части

После того как вы изучите основы программирования, перед вами встанет вопрос: что же делать с приобретенными знаниями? В настоящей книге я предлагаю некоторые варианты применения приобретенных знаний.

Будучи программистом, вы обладаете уникальными знаниями. Если вы будете выполнять простую неинтересную работу в маленькой компании, ваша жизнь постепенно станет такой же скучной. Ознакомившись с настоящей частью книги, вы поймете, что вы можете выполнять сложную работу, приносящую вам немало удовольствия.

Глава 27

Десять вариантов работы для программиста

В этой главе...

- Создание компьютерных игр
- Анимация
- Шифрование данных
- Программирования для Internet
- Борьба с компьютерными вирусами
- Хакерство
- Работа над проектом с открытым кодом
- Программирование для специальных рынков
- Обучение других пользователей
- Продажа собственного программного обеспечения

Если БЫ спросите у руководителей высших учебных заведений, какие специалисты ценятся больше всего, они посоветуют вам стать программистом-аналитиком или заняться обслуживанием баз данных. В настоящей главе рассмотрены возможные варианты работы несколько шире. Лучше заняться работой, которая приносит удовольствие!

Создание компьютерных игр для собственного удовольствия и получения прибыли

Изо всех занятий, которым может посвятить себя программист, самое популярное – создание игр. Однако, кроме проектирования игр (и получения за это денег), создатели компьютерных игр вынуждены заниматься другим, что не всегда приносит удовольствие.

Основная часть игр создается группой. Одни группы программистов создают правила игры, другие — программируют игры. Есть группы программистов, которые занимаются созданием графического фона и анимации. И наконец, есть люди, которые получают большие деньги за то, что играют в старые игры и ищут в них ошибки. Они делают это для того, чтобы понять, как сделать игры более захватывающими (и следовательно, лучше продаваемыми).

Для того чтобы войти в семью профессиональных создателей игр, вы обязательно должны любить играть. Если вы собираетесь написать компьютерную игру, необходимо изучить языки программирования C/C++ и язык ассемблера, потому что компьютерные игры должны занимать небольшой объем и работать как можно быстрее. Компания *Metrowerks* (`www.metrowerks.com`) продает специальную версию компилирующей про-

граммы CodeWarrior, которая используется для написания компьютерных игр на языках программирования C/C++ для игровых консолей Sony PlayStation и Nintendo.

Если вы хотите усовершенствовать свои художественные навыки, вам придется изучить анимацию. А это предполагает изучение математики (она используется для расчета наилучшего способа создания движущихся на экране объектов).

Для того чтобы начать создание собственных игр, обдумайте, какой игровой движок вы будете использовать. Игровой движок — это программа, которая говорит компьютеру, как анимированные объекты должны перемещаться по экрану. Создавая свою игру, вы же не будете тратить время на то, чтобы разбираться, как персонажи игры перемешаются.

Для того чтобы загрузить бесплатный игровой движок игры Crystal Space, который можно запустить на компьютерах, работающих под управлением операционных систем Windows, Linux или MacOS, посетите официальный Web-узел Crystal Space (`crystal.linuxgames.com`). Используя этот игровой движок (как и игровые движки других игр), можно создавать трехмерные фигуры, а также прозрачные или полупрозрачные текстуры, с помощью которых создают прозрачную воду или окно.

Если что-либо из перечисленного выше вас заинтересовало, попытайтесь создать свою игру. Параллельно вы сможете изучить все необходимые для этого элементы и работу самой программы. Именно поэтому многие используют игровые движки для того, чтобы научиться создавать собственные игры. Без игрового движка создание игры вызовет большие сложности.

Для того чтобы больше узнать о создании игр, посетите один из следующих Web-узлов (и начните карьеру профессионального создателя игр).

- ✓ **International Game Developers Association** (`www.cgda.org`). Это ведущая организация, занимающаяся компьютерными играми. Она поддерживает индустрию компьютерных игр, а также проводит конференции по повышению профессионального уровня создателей игр.
- ✓ **Game Developer** (`www.gdmad.com`). Специальный электронный журнал, посвященный исключительно новейшим технологиям для создания компьютерных игр и новостям этой отрасли.
- ✓ **Game Programmer** (`gameprodrammer.com`). Web-узел, на котором приведено большое количество ссылок на ресурсы Internet, касающиеся программирования игр.
- ✓ **DigiPen** (`www.digipen.edu`). Узел одной из первых школ, в которой готовят профессиональных создателей игр. Выпускникам этой школы присваивается степень бакалавра.
- ✓ **GameJobs** (`www.gamejobs.com`). Узел, на котором приведены информация, советы и адреса, которые помогут найти работу в отрасли компьютерных игр.

Анимация

Компьютерная анимация — это не просто создание анимированных образов для видеоигр. Анимация также используется для создания виртуальной реальности, симуляторов и специальных эффектов в фильмах (разрушить здание, созданное с помощью компьютера, намного проще, чем реальное).

Компьютерная анимация помогает создавать различные объекты в телевидении и кино. Ее можно использовать и для создания эффектных презентаций. Если вы любите рисовать, но хотите стать кем-то большим, чем простой художник, объедините ваши способности и знание программирования. Вы сможете создавать новые программы для анимации, симуляторы или спецэффекты для очередного шедевра, снятого в Голливуде.

Для того чтобы больше узнать о прекрасном мире компьютерной анимации, посетите следующие Web-узлы.

- ✓ **Pixar Animation Studios** (www.pixar.com). Ведущая анимационная студия Голливуда, в которой сняты такие фильмы, как *Toy Story (История игрушек)* и *A Bug's Life (Из жизни насекомых)*.
- ✓ **МП Animation and Graphics Club** (www.mit.edu/magc/www). Клуб Массачусетсского технологического института (Massachusetts Institute of Technologies — MIT), посвященный созданию коротких фильмов с использованием компьютерных и традиционных анимационных технологий.
- ✓ **International Animated Film Society** (www.asifa-hollywood.org). Присуждает премию за лучший короткий фильм, в котором используется компьютерная анимация.
- ✓ **Animation Magazine** (www.animag.com). Содержит новости и информацию обо всей анимационной индустрии.
- ✓ **National Centre for Computer Animation** (ncca.bournemouth.ac.uk). Ведущая организация Великобритании по обучению компьютерной анимации и работе с цифровой аппаратурой.
- ✓ **Computer Graphics World Online** (http://cgw.pennnet.com/home/home.cfm). Журнал, посвященный инструментам, новостям и сведениям, которые необходимы для профессионального художника, работающего на компьютере.

Шифрование и дешифрование

Каждый раз, как одно государство начинало войну с другим и посылало своих воинов на поле битвы, удовлетворяя свои политические амбиции, для связи с войсками использовались различные способы шифрования. Благодаря этому враги не могли прочитать передаваемую информацию.

Несмотря на то, что войны рано или поздно заканчиваются, страны все равно уделяют большое внимание развитию техники шифрования и дешифрования. И даже такому известному шпиону, как Джеймс Бонд, пришлось освоить способы шифрования.

Шифрование — это искусство преобразования простой текстовой информации в текст, который ни один непосвященный человек не прочитает. Только используя секретный пароль или ключ, можно прочитать зашифрованное сообщение.

Искусство шифрования данных основывается на выполнении математических операций (обычно основное внимание уделяется простым числам). Если вы собираетесь стать дешифровщиком, вам придется получить математическое образование и улучшить навыки программирования на языке C/C++. После этого вы сможете получить работу в армии, у оборонного подрядчика, а также в агентстве безопасности.

Если вы овладели всеми премудростями шифрования, однако не хотите помогать подготовке вашей страны к войне, примените приобретенные вами знания в финансовой или банковской сфере. Здесь шифрование используется для защиты электронного перевода денег из одного банка в другой.

Для того чтобы получить более подробную информацию о шифровании, посетите один из следующих Web-узлов. На них можно найти исходные коды программ, написанных на C/C++, с помощью которых вы повысите и свой программистский уровень, и знания в области шифрования.

- **CypherNet** (www.cypher.net). Это общественная организация, помогающая пользователям освоить шифрование для защиты от собственного правительства.
- **Central Intelligence Agency** (www.cia.gov). ЦРУ — это самая известная разведывательная служба, занимающаяся шпионажем во многих странах.
- **North American Cryptography Archives** (www.cryptography.org). Содержит большое количество программ для шифрования и дешифрования, исходный код которых поможет вам больше узнать о возможностях шифрования.
- **International PGP Home Page** (www.pgpi.com). Это набор алгоритмов и программ для высоконадежного шифрования сообщений с использованием открытых ключей (Pretty Good Privacy— PGP).
- RSA (www.rsasecurity.com). Это первая компания, предлагающая технологии шифрования для многих программ, доступ к которым осуществляется через Internet.

Программирование для Internet

Хотя компании, работающие с Internet, нанимают многих программистов, другие компании, работающие по старинке, нуждаются в услугах программистов, которые помогут им в создании интерактивных Web-страниц. Учитывая все возрастающее значение Internet, повышение спроса на программистов для Internet нельзя назвать неожиданностью.

Для того чтобы стать профессионалом в этой области, вам придется потратить некоторое время на то, чтобы разобраться с языком HTML, который используется для создания Web-страниц. (Неплохо было бы познакомиться с графическим дизайном и верстkой.)

С помощью языка HTML можно создавать великолепные Web-страницы. Но так как компании предпочитают эксплуатировать Internet для продажи своей продукции, для создания интерактивных Web-узлов программисты используют различные языки программирования: Java, XML, JavaScript, VBScript, Perl, C# и Python.

Для того чтобы приступить к программированию для Internet, изучите язык HTML. Стоит хоть немного познакомиться с одним из языков программирования для Internet (например, JavaScript). Постарайтесь повысить свои знания об операционных системах Windows 2000, Linux или Unix, узнать больше о работе с базами данных с помощью SQL. Поэкспериментируйте с Web-серверами, такими как Apache (который часто бесплатно поставляется с операционной системой Linux).

Посетите один из следующих Web-узлов для того, чтобы понять, насколько быстро можно найти работу (и заработать много денег), став программистом для Internet.

- **CoolWeb Jobs** (www.coolwebjobs.co.uk). Содержит список различных вакансий для программистов для Internet, что поможет быстро найти работу.
- **GeerFinder** (www.geekfinder.com). Предлагает очень много вакансий в различных компаниях по всему миру.
- **Web Jobs USA** (www.webjobsusa.com). Создан для того, чтобы помочь профессионалам найти применение своим знаниям в создании Web-страниц и программировании для Internet.
- **Java Jobs** (javajobs.com). Предлагает учебные пособия и руководства по программированию, а также список вакансий для программистов на языке программирования Java.

Борьба с компьютерными вирусами и червями

Каждый месяц появляются сотни новых компьютерных вирусов. К счастью, большинство компьютерных вирусов содержат ошибки, которые не позволяют им правильно работать: они не хотят распространяться, их легко обнаружить, они ничего не делают, а только занимают часть дискового пространства, и т.д.

Но каждый год несколько новых компьютерных вирусов доставляют кучу проблем пользователям по всему миру. К наиболее известным можно отнести вирусы Michelangelo, CIH, Melissa и червь LoveBug.

Несмотря на то, что создатели вирусов в основном занимаются этим для своего собственного удовольствия, некоторые программисты пишут вирусы-разрушители для того, чтобы бросить вызов. Таких нечистоплотных программистов достаточно много, поэтому люди, умеющие создавать антивирусные программы, всегда найдут работу.

Для того чтобы больше узнать о компьютерных вирусах, изучите язык ассемблера и язык программирования VBA (Visual Basic for Applications — Visual Basic для прикладных программ). Этот язык программирования используется в офисном пакете компании *Microsoft*. Компьютерные вирусы представляют собой маленькие, быстро работающие программы, написанные на языке ассемблера. Они получают доступ к аппаратному обеспечению компьютера. Для того чтобы больше узнать о компаниях, создающих антивирусные программы, посетите следующие Web-узлы.

- ✓ **Network** Associates (`www.nai.com`). Здесь вы найдете популярную антивирусную программу VirusScan.
- ✓ **Sophos** (`www.sophos.com`). Вы найдете популярную антивирусную программу Sophos AntiVirus.
- ✓ **Symantec** (`www.symantec.com`). Вы найдете популярную антивирусную программу Norton AntiVirus.
- ✓ **Trend Micro** (`www.trend.com`). Здесь вы найдете популярную антивирусную программу PC-cillin AntiVirus.
- ✓ **F-Secure** (`www.datafellows.com`). Вы найдете хорошо зарекомендовавшую себя антивирусную программу F-Prot.

Хакерство

Хакеры часто бывают высокообразованными людьми, которые получают удовольствие от взлома чужих компьютеров. Несмотря на то, что взлом компьютеров — формально незаконная операция, как только вы научитесь взламывать компьютеры, вырастут и ваши шансы на получение хорошей работы.

Вместо того чтобы заниматься незаконной деятельностью и постоянно рисковать попасть в тюрьму, найдите другое применение вашим наклонностям. Попытайтесь обмануть самого себя. Став экспертом в компьютерной безопасности, вы можете найти работу в правительстве, защищая компьютеры от тех же хакеров. Можно работать и в корпорациях, защищая их компьютеры от проникновения.

Работая "добрым" хакером, вы сможете удовлетворить все свои интеллектуальные амбиции, не рискуя попасть за решетку. Кроме того, вы можете работать в правоохранительных органах, выслеживая хакеров по всему миру.

Для того чтобы больше узнать о том, как направить свои хакерские наклонности в законное русло, посетите следующие Web-узлы.

- ✓ **Ethical Hackers Against Pedophilia** (www.ehap.org). Общественная организация, которая использует хакерские знания для выслеживания и закрытия Web-узлов, посвященных педофилии.
- ✓ **AntiOnline** (www.antionline.com). Предлагает различные инструменты для хакеров, благодаря котором они могут законно и без риска использовать свои знания.
- ✓ **2600** (www.2600.com). Ежеквартальный хакерский журнал, в котором содержатся статьи на хакерские темы, а также различная информация.
- ✓ **Hacker** News **Network** (www.hackernews.com). Предлагает последние новости о хакерах.

Работа над проектом с открытым кодом

Для того чтобы получить работу, вы должны обладать опытом работы. Однако нельзя получить этакий опыт, не работая. Для того чтобы разрешить этот парадокс, можно продемонстрировать свои способности, работая бесплатно.

Для того чтобы получить опыт работы программиста, который так ценится в больших компаниях, попрактикуйтесь в так называемых проектах с открытым кодом. Основная идея таких проектов — содействие развитию отдельных проектов, таких как операционная система Linux или создание интерфейса пользователя GNOME для Linux.

Работа над проектом с открытым кодом не только позволяет получить что-то полезное, но также гарантирует опыт работы в реальном программистском проекте. В то время как другие программисты работают на низших иерархических ступенях в скучных проектах, не рассчитывая на какой-либо карьерный рост, вы сможете гордиться тем, что участвовали в интересном проекте.

Участие в проекте с открытым кодом поможет в будущем найти престижную высокооплачиваемую работу, а также может обернуться интересным хобби. Участие в проектах с открытым кодом позволит показать всему миру, на что вы способны в действительности.

Для того чтобы подробнее познакомиться с проектами с открытым кодом, посетите следующие Web-узлы.

- ✓ **Open Source** (www.opensource.org). Содержит новости и информацию о важности различных проектов с открытым кодом.
- ✓ **Free Software Foundation** (www.fsf.org). Предлагает информацию о проектах с открытым кодом в общем и о проекте GNU C в частности.
- ✓ **Perl** (www.perl.com). Начальная страница языка программирования Perl, который становится популярным языком программирования для Internet.
- ✓ **Apple Open Source** (www.publicsource.apple.com). Здесь содержится информация о проектах с открытым кодом, касающихся компьютеров, созданных компанией *Apple*.
- ✓ **GNOME project** (www.gnome.org). Данный узел посвящен разработке пользовательского интерфейса GNOME для операционной системы Linux.
- ✓ **Mozilla** (www.mozilla.org). Проект с открытым кодом для Netscape Navigator, второго по популярности Web-броузера.
- ✓ **Linux** (www.linux.org). Посвящен известной разновидности операционной системы Unix.

Программирование для специальных рынков

Очень часто в специализированных учебных заведениях обучают тому, как писать программы, при этом ничего не говоря о том, где же их использовать.

В большинстве компаний используются компьютеры. Таким образом, неплохо бы применить знание программирования и в прикладных задачах.

Например, кто, как не программист, обладающий знаниями в медицине (или медик, умеюший хорошо программировать), сможет написать программное обеспечение для медицинских клиник. Любители спорта объединяют программирование со спортом, создавая программы-тренажеры для различных видов спорта. Профессионалы здравоохранения создают программы по правильному питанию и диетам. Юристы создают юридическое программное обеспечение.

Практически в каждой сфере простейшее узконаправленное программное обеспечение (например, электронные таблицы или базы данных) не решает все проблемы. Поэтому многие компании принимают на работу высокопрофессиональных создателей программного обеспечения.

Лучше всего сначала применить ваши знания в торговле, а не пытаться покорить такие вершины, как крупные компании, которые уничтожат все ваши розовые надежды. Кроме того, вы сможете написать программы для торговой сферы, обладая минимальными познаниями в программировании. Многие ведущие программисты приобретали опыт работы именно в торговле или управлении гостиницами. Это означает, что вы получите прекрасную возможность усовершенствовать свои навыки программирования.

Если вы хотите заниматься своей основной работой и, вместе с тем, применять свои знания программиста, попробуйте создавать программы для разрешения отдельных проблем в сфере торговли. И кто знает? Может быть, применив свои знания, вы найдете новые пути решения проблем, которые казались тупиковыми.

Обучение других пользователей

Став экспертом во многих областях, вы сможете поделиться своими знаниями с другими. Кроме того, можно преподавать в школах, обучая пользователей работе с такими популярными программами, как Microsoft Word, Lotus Notes, а также программированию на языке C++ для расчета рентабельности предприятия.

Такие преподаватели-профессионалы ездят по всему миру, организуя семинары на тех предприятиях, руководители которых хотят повысить профессиональный уровень своих сотрудников, обучив их работе с различными программами. Будучи таким преподавателем, вы посетите очень много стран, встретитесь с различными людьми и увидите разные достопримечательности.

Если вы любите путешествовать, общаться с людьми и передавать свои знания другим, эта работа именно для вас.

fljtocfOJtca собственного программного обеспечения

Как мне кажется, наиболее выгодное занятие — это создание и продажа собственного программного обеспечения. В отличие от ресторанного бизнеса или книжного магазина, вам не нужна большая площадь для работы. Вы просто создаете программы и продаете их прямо через Internet.

Для опробования программ лучше всего подходит их условно-бесплатное распространение. При этом вы предоставляете пользователям копию своей программы и предлагаете прислать вам деньги, если эта программа им подойдет. Для того чтобы пользователи захотели прислать деньги, созданная вами программа должна работать надежно и быть полезной.

Несмотря на то, что такой способ распространения продукции кажется несколько экзотическим, многие создатели профамм заработали таким образом не один десяток долларов в год (а некоторые и несколько миллионов). (Одним из наиболее удачных проектов является распространение программы WinZip, которую можно зафузить с Web-узла www.winzip.com.) Распространяя программы таким образом, можно заработать достаточно денег, вложив в дело минимум средств.

Если вы хотите начать свое дело, но не хотите брать в банке ссуду, продажа условно-бесплатного профаммного обеспечения — самый простой и реальный способ заработать первый капитал. Имея хорошую идею, навыки программирования и небольшие навыки продажи, вы можете открыть свое дело.

Если созданная вами программа не интересна основной части пользователей, попытайтесь продать ее через магазины. Кроме магазинов, занимающихся продажей различных продуктов, есть магазины, которые нацелены исключительно на компьютерную продукцию, например на продажу карманных компьютеров Palm или PocketPC.

Направьте ваши знания программирования на развитие своего собственного дела, создавая специализированные программы. Если вы любите профаммировать, разве есть что-то лучше, чем сидеть целыми днями дома, ифая на компьютере и зарабатывая при этом большие деньги?

Глава 28

Десять дополнительных ресурсов для программиста

В этой главе...

- Коммерческие компиляторы
- Поиск бесплатных и условно-бесплатных компиляторов
- Использование собственных языков программирования
- Поиск исходных кодов
- Присоединение к группе пользователей
- Использование групп новостей Usenet

Если Liberty BASIC — это первый язык программирования, с которым вы познакомились, входя в восхитительный мир программирования, у вас наверняка возникает вопрос, а куда же двигаться дальше. Хотя вы и можете продолжать, практикуясь с Liberty BASIC, и даже использовать этот язык программирования для создания коммерческих программ, вы наверняка захотите изучить еще какой-нибудь язык программирования.

Если вы всерьез задумываетесь о карьере программиста, следующим логичным шагом будет изучение таких языков программирования, как C/C++, C# или Java. Конечно же, этот шаг подразумевает изучение запутанного синтаксиса команд C/C++, C# или Java, поэтому вам предлагается альтернатива — изучение более простого (но все равно достаточного мощного) языка программирования Visual Basic.

И вообще, зачем ограничивать себя C/C++, C#, Java или версиями BASIC, если вы можете выбрать из сотен других языков программирования: Modula-2, LISP, LOGO, Scheme, Prolog, ICON, APL, COBOL, FORTRAN, Ada или Perl?

Программирование часто обескураживает, поэтому в настоящей главе собран ряд тем, которые помогут вам получить помощь от настоящих профи, например групп пользователей компьютеров или групп новостей.

Если вы устали заниматься "серьезным" программированием, я расскажу о некоторых играх программистов, которые позволят вам значительно поднять свой профессиональный уровень, но при этом еще и поиграть.

Просто запомните, что какой бы язык программирования вы ни выбрали и какую бы помощь вам ни оказывали, только ваш собственный уровень и опыт в программировании определят, сможете ли вы завершить написание программы вовремя и будет ли она работать должным образом или же она будет настолько ненадежной, что пользователи постараются поскорее отказаться от нее.

Коммерческие компиляторы

Самый важный инструмент для любого программиста — *компилятор*. (О том, что такое компилятор и зачем он нужен, я подробно говорил в главе 4, "Инструменты настоящего программиста".) Несмотря на то, что можно найти много замечательных бесплатных компиляторов, большинство программистов все-таки при работе полага-

ются на коммерческие программы, которые обеспечиваются поддержкой и обновлениями. Версии коммерческих компиляторов чаще всего стоят несколько сотен долларов (что не играет никакой роли, если за программные продукты платит ваша компания), но вы можете найти специальные версии компиляторов, предназначенные для начинающих программистов или обеспечивающие только базовые функции, которые стоят немного (от 50 до 150 долларов).

Программирование для Windows

Нравится вам это или нет, но Microsoft Windows доминирует как операционная система на планете (хотя популярность Linux постоянно растет). Если вы планируете писать программы, а при этом еще и продавать их, знайте, что самый большой — именно рынок Windows-программ.

Стандартом для написания Windows-программ считается язык программирования Visual C++, автор которого вездесущая компания *Microsoft* (www.microsoft.com). Несмотря на присутствие в названии слова visual, Visual C++ — это достаточно сложная среда программирования, проблемы с освоением которой возникают даже у профессиональных программистов. Но, если вы хотите писать программы для платформы Windows, вы не ошибетесь, приобретя копию Visual C++.

Поскольку очень немногие согласны потратить полжизни на изучение зашифрованной структуры C/C++ или не менее загадочных внутренних функций операционной системы Windows, программисты выбирают второй по популярности инструмент для создания программ — Visual Basic.

В отличие от Visual C++, Visual Basic действительно прост в изучении и позволяет проектировать интерфейс программы полностью в графическом виде. Используя Visual Basic, вы сможете сначала определить внешний вид окон, кнопок и меню, а затем написать код программы на BASIC. Если вы разобрались с Liberty BASIC, то следующим логичным шагом в покорении мира программирования будет изучение Visual Basic.

Если вы хотите упростить создание интерфейса пользователя, но при этом не терять мощь C++, подумайте об использовании нового языка программирования от компании *Microsoft* — C#. Новый язык программирования C# — это несколько упрощенная версия C/C++, которая легче для изучения и позволяет проектировать интерфейс пользователя в графическом виде практически так же легко, как Visual Basic.

В связи со все возрастающей популярностью языка программирования Java подумайте об использовании Visual Cafe от компании *WebGain*(www.webgain.com), одной из наиболее популярных сред разработки Java-программ для Windows. В отличие от своих конкурентов, Visual Café предлагает полноценный компилятор для преобразования исходного кода программ в машинные коды, что позволит им намного быстрее выполняться в среде Windows.

Очень близок к Visual Cafe продукт JBuilder от компании *Borland* (*www.borland.com*), которая может похвастаться многолетней историей выпуска качественных инструментов программирования. Вы найдете намного больше книг, посвященных JBuilder, чем Visual Café.

Компания *Borland* также предлагает и два других популярных инструмента быстрой разработки приложений — C++ Builder и Delphi. В отличие от Visual C++ от компании *Microsoft,* C++ Builder действительно инструмент быстрой разработки приложений, который позволяет проектировать интерфейс пользователя программы так же легко, как Visual Basic. Завершив создание интерфейса пользователя, вы можете просто написать соответствующий код на C++, который будет выполнять действительную работу программы.

Инструмент быстрой разработки приложений RAD (Rapid Application Development) позволяет очень быстро создавать интерфейс пользователя программы. (Более подробно об инструментах быстрой разработки приложений RAD мы говорили в главе 2, "Кое-что о языках программирования".)

Если вы нашли C++ слишком сложным для изучения, но хотите использовать инструмент, по возможностям превосходящий Visual Basic, подумайте об использовании еще одного программного продукта от компании *Borland* — Delphi. Как и Visual Basic и C++ Builder, Delphi позволяет так же легко проектировать интерфейс пользователя программ, но для обеспечения работоспособности программы используются команды Pascal.

Несмотря на второстепенное положение Delphi (связанное с утратой языком программирования популярности у программистов), он позволяет так же быстро, как и Visual Basic, создавать программы, которые при этом работают практически так же быстро и эффективно, как и программы, написанные на Visual C++. (Если вы уже об этом забыли, напомню, что программы, написанные на Visual Basic, работают очень медленно, а программы, написанные на Visual C++, очень трудно создавать.)

Если у вас есть какой-то интерес к созданию кросс-платформенных приложений (программ, которые работали бы под управлением Windows, Unix или Mac OS), подумайте об использовании такого инструмента, как CodeWarrior от компании *Metrowerks* (`www.metrowerks.com`). В отличие от своих многочисленных конкурентов, Visual C++ или C++ Builder, CodeWarrior поддерживает такие платформы, как Windows, Solaris, Linux и Macintosh, поэтому вы можете (по крайне мере, теоретически) скопировать исходный код, написанный на версии CodeWarrior для Windows, в версию CodeWarrior для Linux и откомпилировать ее для этой операционной системы, обойдясь без внесения каких-либо изменений.

Для того чтобы упростить ваш выбор, я перечислил в табл. 27.1 наиболее популярные компиляторы для платформы Windows.

Таблица 27.1. Наиболее популярные компиляторы для платформы Windows

Название компилятора	Используемый язык программирования	Web-узел
Visual C++	C, C++	www.microsoft.com
Visual Basic	BASIC	www.microsoft.com
Visual Café	Java	www.webgain.com
CodeWarrior	C, C++, Java	www.metrowerks.com
JBuilder	Java	www.borland.com
C++ Builder	C, C++	www.borland.com
Delphi	Pascal	www.borland.com

Программирование для Macintosh и Palm OS

Компьютеры Macintosh постоянно подтверждают свою репутацию самых простых в использовании компьютеров в мире, но создавать для них программы чрезвычайно сложно. К счастью, с появлением последних версий инструментов программирования эта задача значительно упрощается.

Наилучшим инструментом для написания программ для платформы Macintosh считается CodeWarrior, единственный надежный инструмент. CodeWarrior, разработанный компанией *Metrowerks* (`www.metrowerks.com`), позволяет создавать программы на таких языках программирования, как C, C++ и Java. Поэтому, вместо того чтобы приобретать несколько отдельных компиляторов, вы получаете все необходимые функции вместе.

Кроме того, компания *Metrowerks* предлагает специальные версии CodeWarrior, позволяющие создавать программы для платформ Windows (Windows 95/98/Me/NT/2000 и Windows CE), Solaris, Linux, игровых консолей Sony PlayStation и Nintendo, а также для самого популярного карманного компьютера в мире — Palm. Если вы планируете

писать программы для Macintosh, карманного компьютера Palm или игровых консолей Sony PlayStation и Nintendo, CodeWarrior станет вашим первым (и, возможно) единственным любимцем.

Написание программ для карманного компьютера

Компания *Microsoft* выпустила усеченную версию операционной системы Windows, называемую Windows CE для управления карманными компьютерами (часто называемыми компьютерами PocketPC). К сожалению, программы, написанные вами для Windows 95/98/Me/NT/2000, под управлением Windows CE не работают. Поэтому, если вы хотите создавать программы для этой платформы, вам придется использовать специальный компилятор. Компания разработала операционную систему Windows CE, поэтому не вызывает удивления то, что она разработала и специальные инструментальные средства, позволяющие создавать программы для платформы Windows CE/PocketPC, используя языки программирования Visual Basic или Visual C++.

Еще два языка программирования позволяют создавать программы для платформы Windows CE/PocketPC — PocketC (www.orbworks.com) и NSBAS1C (www.nsbasic.com). PocketC базируется на усеченной версии языка программирования C, а NSBASIC базируется на усеченной версии языка программирования BASIC. PocketC и NSBASIC не обеспечивают такими широкими возможностями, как Visual Basic или Visual C++, но они отлично справляются с основной поставленной перед ними задачей — созданием коммерческих программ для платформы Windows CE/PocketPC.

Существуют версии PocketC и NSBASIC для работы под управлением операционной системы Palm OS, поэтому вы получаете возможность создавать программы для карманных компьютеров Palm. Так как платформы Windows CE/PocketPC и Palm значительно отличаются между собой, вы не сможете запускать программы, написанные для Windows CE/PocketPC, на компьютере Palm, и наоборот, без внесения значительных изменений.

Конечно же, CodeWarrior не поддерживает язык программирования BASIC, поэтому если вы хотите написать программу для Macintosh, используя BASIC, у вас остаются два варианта выбора — Future Basic и REALBasic.

Future Basic (доступен на Web-узле www.stazsoftware.com) очень похож на Liberty BASIC, но предназначен исключительно *для* платформы Macintosh.

REALBasic (доступный на Web-узле www.realbasic.com) очень похож на Visual Basic; этот язык программирования точно так же позволяет в графическом виде проектировать интерфейс пользователя, а затем просто писать команды, выполняющие действительную работу программы.

Однако REALBasic на этом не останавливается и позволяет преобразовать исходный код программ, написанных на Visual Basic, для их запуска на платформе Macintosh. Если у вас хорошие программы, написанные на Visual Basic, и вам необходимо как можно быстрее запустить их на платформе Macintosh, достаточно воспользоваться языком программирования REALBasic.

Конечно же, преобразование программ, написанных на Visual Basic, к формату REALBasic не всегда оказывается полностью корректным, поэтому необходимо вносить в исходный код программы небольшие изменения. Поэтому, если вам нужно создать программу, которая работала бы на платформах Windows и Macintosh, напишите ее на REALBasic, после чего позвольте откомпилировать ее для обеих платформ.

Самые популярные компиляторы для платформы Macintosh перечислены в табл. 27.2.

Таблица 27.2. Самые популярные компиляторы для платформы Macintosh

Название компилятора	Используемый язык программирования	Web-узел
CodeWarrior	C, C++, Java	www.metrowerks.com
RealBasic	BASIC	www.realbasic.com
Future Basic	BASIC	www.stazsoftware.com

Программирование для Linux

Если какая-нибудь операционная система и способна поколебать позиции операционных систем компании *Microsoft* на рынке персональных компьютеров, то наиболее вероятным кандидатом на эту роль кажется Linux. Операционная система Linux стремительно набирает популярность, в связи с чем многие компании и программисты быстро преобразуют свои программы для их использования в среде Linux.

Ряд компаний, выпускающих коммерческие программы, уже выпустили Linux-версии своих компиляторов (такие как CodeWarrior, JBuilder и Delphi); вас ожидает приятный сюрприз — существует целый ряд других компиляторов, доступных совершенно бесплатно.

В зависимости от выбранной вами версии Linux (RedHat, Caldera или Black Cat, например), вы получите в свое распоряжение компилятор, такой как GNU C (компилятор для языка программирования C) или EGCS (компилятор для языка программирования C++).

Несмотря на то, что для платформы Linux не написано так много приложений, как для Windows или Macintosh, существуют Linux-версии компиляторов для таких языков программирования, как Ada, Pascal, FORTRAN и BASIC. Для поиска необходимого компилятора для Linux обязательно посетите Web-узел Linux Programming по адресу www.linuxprogramming.com.

Поиск бесплатных и ifсловно-бесплатных компиляторов

Выбор языка программирования часто оказывается настолько же эмоциональным и субъективным, как и выбор будущего спутника жизни. Вместо того чтобы сразу приобретать несколько коммерческих компиляторов с единственной целью выбрать среди них наиболее понравившийся, потратьте некоторое время на загрузку нескольких бесплатных и условно-бесплатных компиляторов.

Испытывая бесплатные и условно-бесплатные компиляторы, вы попрактикуетесь в использовании различных языков программирования, в частности C++ или Java. Если вы найдете язык программирования, который вам действительно понравится, подумайте о приобретении коммерческой или условно-бесплатной версии соответствующего компилятора. Список распространяемых бесплатно компиляторов для различных языков программирования вы найдете на Web-узле Catalog of Free Compilers and Interpreters по адресу www.idiom.com/free-compilers.

Компиляторы BASIC

В качестве компилятора BASIC, позволяющего создавать программы для операционных систем MS DOS или Windows, используйте продукты компании *PowerBasic* (www.powerbasic.com). Для создания программ для MS DOS эта компания предлагает свои условно-бесплатные компиляторы PowerBasic и FirstBasic. Для создания программ для Windows предназначен PowerBasic for Windows.

Если вам необходим интерпретатор (не компилятор) для Macintosh, загрузите бесплатный Chipmunk Basic с Web-узла www.rahul.net/rhn/cbas.page.html.

Одна из самых сложных задач, стоящих перед программистом, — написание трехмерных компьютерных игр. Хотя большинство программистов используют C/C++ для решения таких задач, воспользуйтесь собственными знаниями BASIC при написании компьютерной игры для Windows, используя специальный язык программирования DarkBASIC (доступен с Web-узла www.darkbasic.co.uk), предназначенный для создания игр.

Компиляторы C/C++ и Java

C и C++ — это мощные языки программирования, но их стоимость окажется не по карману очень многим. Вместо того чтобы тратить огромную сумму на приобретение коммерческого компилятора C/C++, сначала поэкспериментируйте с бесплатными и условно-бесплатными версиями таких компиляторов.

Самый популярный компилятор C для Linux — это GNU C, который перенесен и на платформу Windows, получив при этом название Cygwin (доступен на Web-узле http://sources.redhat.com/cygwin).

Если вы хотите попрактиковаться в написании программ на Java, загрузите бесплатный набор инструментальных средств непосредственно с Web-узла компании *Sun Microsystems* (java.sun.com), создательницы этого языка программирования. Инструмент программирования на Java, обладающий всеми необходимыми базовыми возможностями, позволит вам лучше разобраться в премудростях Java. Затем, когда вы подготовитесь к созданию серьезных приложений, подумайте о приобретении коммерческих программ — JBuilder или Visual Café.

Компиляторы Pascal

Если вам необходим компилятор для MS DOS, Windows или Linux, загрузите Free Pascal с Web-узла www.freepascal.org. Если вы хотите глубоко разобраться в работе компилятора, посетите Web-узел Bloodshed Software (www.bloodshed.com), где вы сможете присоединиться к группе энтузиастов создания и разработки компилятора Pascal для Windows.

Компиляторы и интерпретаторы для других языков программирования

Никому не нравится следовать за толпой и изучать только традиционные языки программирования, в частности C/C++ или BASIC. Если вы придерживаетесь именно такой точки зрения, подумайте об изучении бесплатных компиляторов и интерпретаторов для несколько нестандартных языков программирования, которые пригодятся вам в той или иной ситуации.

Язык программирования Prolog снискал славу одного из самых популярных языков при исследованиях искусственного интеллекта. Если вы хотите разобраться с языками моделирования искусственного интеллекта вообще и с Prolog в частности, загрузите бесплатную копию Strawberry Prolog с Web-узла www, dobrev.com.

В начале 1980-х годов Министерство обороны США предприняло попытку сделать язык программирования Ada единым стандартом для разработки всех военных проектов. К сожалению для Пентагона, к моменту появления серьезных компиляторов Ada весь мир перешел к использованию C/C++, оставив Ada без особого внимания. Несмотря на это, у языка программирования Ada есть стойкая группа приверженцев, поэтому, если вы хотите поэкспериментировать с языком программирования, который должен был стать самым лучшим языком программирования в мире, загрузите копию GNAT Ada с Web-узла www.gnat.com.

Язык программирования BASIC предназначался для обучения новичков премудростям написания программ, а другой язык программирования — LOGO, разрабатывался для обучения тому же детей. Если вы хотите создавать программы для Windows, используя язык программирования LOGO, загрузите бесплатную копию MSW Logo от компании *Softronics*(www.softronix.com).

Для того чтобы познакомиться с действительно удивительными языками программирования, посетите Web-страницу dir.yahoo.com/Computers_and_Internet/Programming_Languages, предназначенную специально для программистов. Здесь вы найдете ссылки на практически все существующие в мире языки программирования.

Использование собственных языков программирования

Очень много книг, журналов, газет, исходных кодов, а также пользователей со всего мира помогут вам советом при решении определенной задачи с использованием таких языков программирования, как C/C++ или Java. К сожалению, популярные языки программирования проектировались для решения широкого круга разнообразных проблем, т.е. они не могут очень быстро и легко решить какую-то одну конкретную проблему.

Как альтернативу популярным языкам программирования можно рассматривать *собственные языки программирования*. Компании часто разрабатывают собственные языки программирования для решения строго определенных задач, таких как создание мультимедийных презентаций или программ искусственного интеллекта. Собственные языки программирования обладают следующими преимуществами.

- ✓ Собственные языки программирования обычно проще в изучении, чем популярные языки программирования.
- ✓ Программы, написанные на собственных языках программирования, обычно занимают меньший объем и создаются быстрее, поскольку такие языки предназначены для решения узкого круга задач.

Хотя собственные языки программирования проще в использовании и позволяют создавать замечательные приложения с минимальным объемом работы, у них есть целый ряд недостатков, о которых я расскажу ниже, что не позволит вам использовать эти языки при решении очень важных задач.

- ✓ Вы не получите поддержку со стороны (в виде книг или журналов) для собственных языков программирования, в отличие от популярных языков программирования.
- ✓ Собственные языки программирования создаются только для определенных операционных систем (что уменьшает до минимума возможность переноса программы на другую платформу).
- ✓ Вы можете получить поддержку только от одной компании. Если компания, создавшая используемый вами язык программирования, прекратит заниматься соответствующим родом деятельности, вам никто не поможет создать новую программу или модернизировать существующую.
- ✓ Собственные языки программирования оказываются намного дороже популярных.
- ✓ Программы, написанные на собственных языках программирования, обычно выполняются медленнее программ, для написания которых использовались языки программирования общего назначения.

KnowledgePro

Knowledge Pro — это смесь объектно-ориентированного программирования, гипертекста и искусственного интеллекта. Доступный в версиях для MS DOS и Windows, KnowledgePro позволяет вам использовать уникальный язык программирования для отображения окон, текста, кнопок и графики с минимальным числом команд.

После создания программы на KnowledgePro вы сможете преобразовать ее в код C++, после чего откомпилировать в машинные коды. Таким образом, вы получаете возможность быстро создать программу, используя KnowledgePro, а затем еще и заставить ее быстро выполняться.

Если вы еще не поняли, насколько полезен для вас язык программирования KnowledgePro, посетите Web-узел компании-разработчика (www.kgarden.com), чтобы загрузить бесплатную копию KnowledgePro или увидеть список пользователей и приложений, созданных ими с помощью KnowledgePro.

Clarion

Clarion — это специальный язык программирования для создания приложений баз данных. Так как программы, написанные на Clarion, компилируются в машинные коды, они работают намного быстрее, чем программы, созданные с помощью Microsoft Access или Visual FoxPro.

Лучше всего то, что Clarion — один из немногих языков программирования, который позволяет создавать приложения для Windows 95/98/Me/NT/2000 и World Wide Web без внесения каких-либо изменений в исходный код. Таким образом, рынок пользователей вашей программы значительно увеличится. Для получения более подробных сведений о Clarion посетите Web-узел http://www.softvelocity.com. Только помните о том, что Clarion не позволит вам создавать другие виды программ (например, игры) или программы для других операционных систем (т.е. для Mac OS или Linux).

PowerBuilder

Одним из самых популярных языков программирования для разработки баз данных считается *PowerBuilder,* который позволяет выполнять ряд операций в графическом режиме. Если вам необходимо переносить программы с мэйнфреймов на мини-компьютеры, вам также стоит подумать об использовании PowerBuilder. Для получения более подробных сведений о PowerBuilder посетите Web-узел http://www.sybase.com.

Поиск исходных кодов

Поскольку один из лучших способов обучения — консультации с более опытными коллегами, многие программисты добровольно распространяют исходные коды своих программ, чтобы другие могли воспользоваться результатами их работы. В качестве замечательного примера свободного распространения исходных кодов можно привести операционную систему Linux-

Получив исходный код какой-то программы, вы сможете включить ее элементы в свою собственную программу, значительно сэкономив время. Многие компании продают программы-заготовки (текстовые процессоры, электронные таблицы или программы для построения диаграмм), которые вам просто добавить в свои программы. Кроме того, такие компании очень часто вместе с программами продают еше и их исходные коды, которые вы можете изменить в соответствии со своими потребностями.

Исходные коды небольших программ легко найти в Internet, причем совершенно бесплатно. Такие небольшие программы предназначены для решения простых проблем, но одна из таких проблем может совпасть с поставленной перед вами задачей.

Для поиска исходных кодов воспользуйтесь поисковыми машинами в Internet. В результате поиска вы получите целые списки Web-узлов, содержащих интересующую вас информацию.

Для того чтобы сузить поиск исходных кодов, я предлагаю вам следующий список Web-узлов.

- **Code Guru** (www.codeguru.com). Здесь содержится масса исходных кодов программ, написанных на C/C++, Visual Basic, Java и других популярных языках программирования.
- **Planet Source Code** (www.planet-source-code.com). Здесь содержится масса исходных кодов программ, написанных на Visual Basic и Java.
- **Carl & Gary's Visual Basic Page** (www.cgvb.com). Здесь вы найдете огромное количество информации и исходных кодов программ, касающихся только Visual Basic.
- **The Delphi Source** (www.doit.com/delphi). Здесь вы найдете исходные коды программ, написанных на Delphi.
- **ABC: All BASIC Code** (www.basicguru/abc). Здесь вы найдете исходные коды программ, написанных на различных версиях языка программирования BASIC, включая Liberty BASIC, QBASIC, QuickBasic и Visual Basic.
- **The JavaScript Source** (javascript.internet.com). Здесь вы найдете исходные коды программ, написанных на языке программирования JavaScript.
- **The cprogramming.com** (www.cprogramming.com). Здесь вы найдете исходные коды программ, написанных на языке программирования C/C++.

Присоединение к группе пользователей

Изучать программирование, находясь в полной изоляции, достаточно сложно. Если вам повезло (или не повезло, в зависимости от вашей точки зрения) жить в большом городе, вы наверняка найдете местную группу людей, увлекающихся программированием.

Группы любителей встречаются достаточно регулярно — обычно один раз в неделю или в месяц, позволяя программистам обмениваться советами и полезными сведениями об определенных языках программирования, например C/C++, Java или Delphi.

Сведения о существующих группах пользователей можно найти на Web-узле компании — разработчика компилятора, например www.microsoft.com или www.borland.com.

Использование групп новостей Usenet

Помощь друзей при изучении чего-нибудь нового переоценить очень трудно. Если ваши ближайшие друзья не помогут вам при изучении программирования, обратитесь к неоценимому ресурсу — группам новостей Usenet.

Группы новостей — это электронные доски объявлений, на которых свои сообщения может оставить любой пользователь. На ваши вопросы ответят настоящие профессионалы. Если вы достаточно долго будете работать с группами новостей, вы обязательно найдете и вопросы других пользователей, на которые сможете ответить самостоятельно.

Группы новостей существуют для любых языков программирования. Ниже я привожу список самых популярных групп новостей, посвященных языкам программирования.

1. comp.lang. Группа новостей, посвященная общим вопросам программирования.

- ✓ сотр.land.basic. Группа новостей, объединяющая энтузиастов программирования на языке BASIC.
- ✓ comp.land.c. Группа новостей, объединяющая любителей программирования на языке C.
- ✓ сотр.land.c++. Группа новостей, посвященная программированию на языке C++.
- ✓ comp.land.delphi. Группа новостей, позволяющая вам общаться с программистами, использующими Delphi.
- ✓ сотр.land,java.help. Группа новостей, посвященная программированию на языке Java.
- ✓ сотр.land.pascal. Группа новостей, посвященная программированию на языке Pascal, а также предлагающая сведения об использовании Delphi.

Предметный указатель

М

О

П

С

Т

У

Ф

Х

Э

Я

Научно-популярное издание

Уоллес Вонг

Основы программирования для "чайников"

В издании использованы карикатуры американского художника Рича Теннанта

Литературный редактор *Л.Н. Важенина*
Верстка *О.В. Линник*
Художественный редактор *В.Г. Павлютин*
Технический редактор *Г.Н. Горобец*
Корректоры *Л.А. Гордиенко и О.В. Мишутина*

Издательский дом "Вильяме".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Изд. лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати.

Подписано в печать 16.07.02. Формат 70×100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 27. Уч.-изд. л. 18,5.
Доп. тираж 4000 экз. Заказ № 822.

Отпечатано с фотоформ в ФГУП **"Печатный** двор"
Министерства **РФ** по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.