БИБЛИЯ Delphi

МИХАИЛ ФЛЕНОВ

- Программирование в Delphi от А до Я
- Программирование звука и графики с помощью OpenGL
- Динамические библиотеки
- Создание локальных, клиент-серверных и трехуровневых баз данных
- Практические рекомендации и примеры

bhv

Михаил Фленов

Библия Delphi

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.01
Ф69

Фленов М. Е.

Ф69 Библия Delphi. — СПб.: БХВ-Петербург, 2004. — 880 с.: ил.

ISBN 5-94157-456-8

Цель книги — научить читателя строить логику программы и алгоритмы различных вычислений. Уметь программировать еще не достаточно, надо знать, как применять полученные знания на практике. Для этого подробно описывается логика выполнения каждого участка кода, чтобы читатель смог использовать эти знания при решении собственных задач. Книга содержит большое количество примеров практического программирования; некоторые из них вынесены в качестве дополнительной информации на прилагаемый компакт-диск. Электронная версия книги была размещена в Internet в 2003 году. Автор собрал все замечания и предложения по дополнению книги и написал совершенно новый вариант, который вы сейчас держите в руках. Таким образом, книга прошла массовое тестирование и теперь отражает потребности множества как начинающих, так и опытных программистов.

Для программистов

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.01

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Владимир Красильников*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.04.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 70,95.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-456-8

Содержание

Введение

Данная книга посвящена одному из наиболее популярных в нашей стране и перспективному во всем мире языку программирования Delphi. Она предназначена для программистов всех уровней, от начинающего до опытного. Как показывает практика, большинство людей научились программированию по книгам. Однако далеко не все из этих книг объясняют принципиальные основы работы Windows и компьютера в целом. Отсутствие базовых знаний в этой области не позволяет писать эффективные программы.

Я решил восполнить этот пробел. Я постараюсь написать так, чтобы, прочитав мой труд, любой человек смог стать настоящим программистом, а главное, вы должны понимать, что и для чего вы делаете. Несмотря на это, я не гарантирую, что именно вы сможете стать профессионалом.

Как показывает практика, из всех обучающихся программированию только 30% становятся настоящими программистами и только к ним можно применить понятие профессионал. Я обучил достаточно много людей и у меня этот показатель свыше 70%. Оставшиеся 30% смогли научиться писать программы, смогли понять основы, но почему-то не смогли сформировать у себя способность самостоятельно мыслить в данной области. У них постоянно возникают вопросы, ответы на которые можно получить, затратив всего лишь небольшие усилия. Надо просто немного подумать. Причиной такого положения дел может быть лень, а может просто человеку неинтересно самостоятельно мыслить. Чаще всего тут действует лень, особенно когда рядом есть люди, у которых можно спросить. В этой связи сразу хочу вас предостеречь — только самостоятельно найденный ответ на вопрос добавит новые знания.

Эта книга может научить многому. Однако без стремления совершенствоваться в данной области вы не сможете самостоятельно писать "хорошие" программы. На протяжении всей книги будут рассматриваться различные методы, некоторые шаблоны и приемы программирования на языке Delphi, однако описать абсолютно все, как вы понимаете, здесь просто невозможно. Программирование — это такая область, в которой требуется постоянное обучение. В связи с этим нельзя останавливаться на достигнутом, прочитав только одну книгу. Нужно постоянно совершенствоваться и обучаться.

Прежде чем приступить к изучению самой книги, необходимо сделать несколько замечаний. Первое из них касается терминологии. В тексте часто будет использоваться выражение "Язык программирования Delphi". Многие утверждают, что Delphi — это среда разработки, которая использует язык

программирования Pascal (Паскаль). В принципе, здесь не утверждается, что это ошибка. И все же, в Delphi от старого Паскаля осталось очень мало, поэтому я считаю, что это не просто среда разработки, а самостоятельный язык программирования. Это лично мое мнение как автора, и вы можете с ним соглашаться или нет.

Теперь о содержимом книги. В ней сделана попытка представить изучаемый материал таким образом, чтобы было понятно даже человеку, который только недавно познакомился с компьютером. Возможно, опытным программистам начальную часть книги читать будет скучно. Но даже здесь будут описываться достаточно специфичные вещи, среди которых можно найти для себя довольно много полезного. Поверьте, это действительно так и связано с тем, что большинство книг по данной проблематике упускают из виду некоторые очень важные тонкости, которые желательно знать для понимания принципа работы программ. Без этого понимания тяжело двигаться дальше и любые новые технологии будут казаться тяжелыми и сложными.

Прежде чем приступить к чтению книги, учтите один совет. Книгу желательно читать полностью, от начала и до конца, потому что материал излагается постепенно и некоторые вещи могут быть непонятны, если что-то пропустить вначале. Как только вы почувствуете, что набрали достаточно знаний и способны самостоятельно писать хотя бы простейшие программы, можете сделать единственный скачок на *гл. 24*. В ней дается материал, касающийся отладки приложений, потому что при самостоятельном написании программ всегда появляются ошибки или опечатки. Эта глава объясняет, как находить такие ошибки. В ней вы также узнаете некоторые приемы по работе с редактором кода, которые могут пригодиться в будущем при программировании собственных приложений, да и при работе с примерами, которые представлены в этой книге.

После прочтения этой главы можно вернуться к той, на которой вы остановились ранее, и продолжить чтение книги уже без каких-либо скачков. Иначе какой-то важный момент может быть упущен, и нагнать упущенное потом будет очень тяжело, потому что вы можете не заметить, что что-то упустили.

Если вы читали вариант книги, представленный в Internet (**www.vr-online.ru/books/index.htm**), вам также будет полезно прочитать эту книгу, потому что данный вариант переработан полностью от начала и до самого конца.

Внимание

Я не читал ни одной книги по Delphi на русском языке. Единственная книга, которую я видел (я даже не прочитал ее полностью, а просмотрел несколько глав), была по Borland Pascal. Это было в 1994 году, поэтому я даже не помню ее названия. Именно поэтому некоторые мои термины могут отличаться от таких же в другой литературе.

И последнее, некоторые термины, встречающиеся в книге, могут отличаться от аналогичных, которые используются в другой технической литературе, относящейся к данному вопросу. Это связано с особенностями перевода англоязычного текста на русский язык. В любом случае, терминология, которая приводится в книге, делает ее намного проще и понятней как начинающим, так и опытным программистам.

Структура книги

В этом месте принято описывать содержание глав книги. Это поможет вам легко найти интересующую главу или, наоборот, узнать какие главы вы уже хорошо знаете и читать не стоит.

Глава 1 "Основные принципы работы компьютера". Посвящена рассмотрению принципов работы компьютера. В ней рассказывается о том, как компьютер производит расчеты и выполняет различные команды. В этой главе даются основы, без которых невозможно понимание самого принципа программирования. Конечно же, можно обойтись и без нее, потому что и обезьяну можно научить кидать гранату. Но только с помощью знаний, полученных при изучении данной главы, можно понять, что и зачем вы пишете в своей программе.

В принципе, эту главу можно и опустить, потому что научиться программированию можно и без этого. Однако только с пониманием работы "железа" можно стать настоящим программистом.

Глава 2 "Машинная математика". В этой главе рассматриваются основы кодирования данных в компьютере или машинная математика. Машинная математика — это основа программирования. Здесь вы познакомитесь с логикой выполнения программ и сами научитесь формировать логическую структуру будущей программы.

Мы познакомимся с гениальным изобретением всех времен — *"блок-схемами"* программ. Они очень хорошо помогают начинающим программистам в понимании работы логики компьютера. Конечно, в будущем можно научиться писать программы и без использования блок-схем, но на начальном этапе это очень удобный инструмент, как для определения логики работы программы, так и для обучения программированию в целом.

Глава 3 "Начальные сведения о Delphi". В этой главе излагается процесс установки Delphi 7, а также рассказывается о входящих в поставку утилитах. После этого мы запустим оболочку Delphi 7 и рассмотрим, из чего она состоит. В главе также будут рассмотрены начальные сведения о Delphi. Ее могут пропустить те, кто уже знаком с этим языком программирования. Хотя в конце главы будет идти речь о настройках оболочки, поэтому желательно ее все же прочесть.

Глава 4 "Визуальная модель Delphi". В этой главе речь пойдет о визуальной модели Delphi. Это то, на чем построена вся теория программирования в среде этой оболочки. Кроме того, здесь будут затронуты вопросы теории объектно-ориентированного программирования, без понимания которых невозможно движения дальше.

Глава 5 "Основы языка программирования в Delphi". В этой главе мы познакомимся с типами данных, используемыми Delphi, и напишем нашу первую программу. Здесь мы познакомимся со всеми "внутренностями" этой программы и узнаем, из чего состоит ее "скелет".

Глава 6 "Работа с компонентами". Работа в среде Delphi предполагает активное использование компонентов среды. В этой главе рассматриваются основы работы с этими компонентами, описываются основные их свойства, а также дается характеристика событийной модели Windows и основных событий главной формы.

Глава 7 "Палитра компонентов *Standard* ". В этой главе мы познакомимся с вкладной **Standard** палитры компонентов Delphi. Здесь будут определены все компоненты данной вкладки, а также рассказано, для чего они предназначены и как их использовать. Изложение материала главы сопровождается рассмотрением большого количества примеров с использованием этих компонентов.

Глава 8 "Учимся программировать". Здесь подробно рассказывается про циклы, логические операции, работу со строками и многое другое. Это последняя глава, которая посвящена изучению основ программирования. Те, кто уже имеет опыт программирования в Delphi, могут эту главу пропустить.

Глава 9 "Создание рабочих приложений". Сейчас уже трудно себе представить программу, состоящую только из одного главного окна. Большинство приложений состоит хотя бы из нескольких окон, а некоторые даже из сотен. В этой главе дается понятие многооконных приложениий. Здесь также рассказывается, как создавать главное меню программы.

Глава 10 "Основные приемы кодинга". В этой главе, на первый взгляд, рассматривается совокупность не связаных между собой понятий и объектов программирования. Тут и работа с массивами, файлами, реестром, преобразование данных, структуры и указатели. Все это собрано под одной крышей с одной целью, чтобы читатель мог изучать все последовательно, по мере надобности. Меня просто бесит литература, в которой сначала описывают разные функции и бесполезные примеры (которые в жизни не пригодятся), и только к концу книги находишь что-то действительно полезное. Таким образом, благодаря материалу этой главы полезная информация дается своевременно и изучение программирования не становится рутинным и скучным.

Глава 11 "Обзор дополнительных компонентов Delphi". После рассмотрения основных приемов программирования можно перейти к изучению остальных компонентов Delphi. Данная глава посвящена рассмотрению именно этого вопроса. В ней приводятся достаточно полезные в будущем примеры. Если бы это было сделано раньше, то ничего интересного в качестве примеров указать было бы просто не возможно.

Глава 12 "Графические возможности Delphi". Здесь рассказывается обо всем, что касается графики. Показывается, как можно рисовать встроенными средствами в Delphi различные фигуры и как работать с изображениями разного формата.

Глава 13 "Печать Delphi". Эта глава полностью посвящена печати, и только печати. В ней рассказывается, как выводить на принтер текст и графику, как учитывать разрешение принтера и многое другое.

Глава 14 "Delphi и базы данных". Многие слышали или знают, на примере более ранних версий, что на Delphi очень легко писать базы данных, потому что в структуру его среды программирования встроены мощные средства, предназначенные именно для этого. Данная глава позволит вам в этом убедиться. Здесь будет показано, как работать с локальными базами MS Access, а также приводится множество полезных примеров.

Глава 15 "Создание отчетности". В этой главе рассматриваются вопросы, касающиеся экспорта данных из таблиц в среду Excel, а также вопросы, связанные с подготовкой к печати документов любой сложности.

Глава 16 "Работа с DBF, Paradox, XML и клиент-серверными базами данных". В главе рассказывается о том, как работать с другими таблицами, отличными от Access. Здесь описывается технология доступа к данным через драйвер BDE, который предоставляет фирма Borland для доступа к таблицам DBF и Paradox.

Глава 17 "Потоки". Windows — многозадачная операционная система, позволяющая выполнять многопоточные приложения, в которых операции выполняются параллельно. В этой главе вы познакомитесь с понятием многопоточности, с реализацией ее в программах, также здесь будут рассмотрены несколько примеров реализации многопоточных приложений.

Глава 18 "Динамические библиотеки". Здесь рассказывается все необходимое о динамических библиотеках. Вы узнаете, как создавать библиотеки с математическими процедурами и функциями, как хранить окна в библиотеках и увидите реальные примеры их использования.

Глава 19 "Разработка собственных компонентов". В этой главе речь пойдет о том, как создавать свои VCL (Visual Component Library — библиотека визуальных компонентов) компоненты, как устанавливать чужие разработки в Delphi и как работать с пакетами компонентов.

Глава 20 "Мультимедиа". Эта глава полностью посвящена принципам программирования звука и видео. В ней показано, как создавать приложения для работы со звуком, используя встроенные в Delphi компоненты, или без них.

Глава 21 "Графика OpenGL". Есть две достаточно перспективные разработки для профессиональной работы с компьютерной графикой — OpenGL и DirectX. В данной главе книги достаточно подробно описывается только OpenGL. Информацию по DirectX вы сможете найти на компакт-диске, прилагаемом к книге, в папке / Документация.

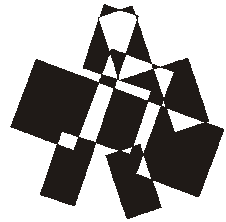
Глава 22 "LE, COM, ActiveX". В этой главе будут описаны основные принципы реализации технологий OLE, COM и ActiveX. Все эти термины взаимосвязаны и должны описываться вместе. Не все программисты любят использовать эти технологии, но описать их необходимо, потому что иногда работать с ними все же приходится. Возможно, что и вы когда-нибудь столкнетесь с этой технологией.

Глава 23 "Буфер обмена". Кнопки **Копировать** и **Вставить** есть практически в любом полноценном приложении. Можно надеяться, что вы также захотите использовать такую возможность в своей программе. В этой главе дается максимум полезной теоретической и практической информации для того, чтобы вы смогли сделать свои программы более привлекательными, добавив возможность переноса данных между приложениями.

Глава 24 "Дополнительная информация". Данная глава единственная, которую можно прочитать вне очереди. Как только вы почувствуете, что полученных знаний достаточно для написания собственных небольших приложений, можете перескочить на эту главу. Здесь описываются некоторые приемы работы с оболочкой Delphi, которые смогут вам помочь при разработке собственных приложений, а также принципы тестирования и отладки программ.

Глава 25 "Сплошная практика". Эта глава является завершающей, поэтому в ней приводится несколько интересных программ. Это делается для того, чтобы вы могли увидеть некоторые приемы программирования, которые могут пригодиться в будущем. Данную главу можно рассматривать как дополнительный материал ко всему сказанному выше.

Глава 1



Основные принципы работы компьютера

Прежде чем начинать программировать, необходимо понять, как работает компьютер. Как говорил какой — то полководец: "Нужно хорошо изучить своего врага!!!" Возможно, это говорил и не полководец, но это неважно. Программирование — это постоянная борьба с машиной. Нужно заставлять ее делать то, что вам нужно. Поэтому любой программист просто обязан знать структуру компьютера.

1.1. Основы работы персонального компьютера

В общем случае компьютер состоит из следующих основных компонентов: процессор, память, видеокарта, винчестер (жесткий диск) и различные разъемы для подключения дополнительных устройств. Все эти компоненты связаны между собой с помощью шлейфов и шин.

Вся информация в компьютере хранится на винчестере. Когда запускается на выполнение какая-либо программа, она сначала загружается во внутреннюю или оперативную память (ОП) и только потом процессор начинает выполнять содержащиеся в ней инструкции. Чем больше программа, тем дольше она загружается, и это желательно учитывать всегда. Если у вас получается слишком большой программный код, то для ускорения выполнения программы можно в основном файле задействовать только основные возможности. Все дополнительные возможности, которые используются не всегда, можно вынести в отдельные файлы, например, динамически подключаемые библиотеки DLL (Dynamic Link Library), о которых будет отдельный разговор.

Результат работы программы выводится на экран через видеокарту. На любой видеокарте есть чип памяти, в котором отображается все содержимое экрана. Когда вам нужно вывести что-то на экран, вы просто копируете эти данные в видеопамять, и видеокарта автоматически выводит их

на монитор. Именно такой способ использовался в среде ОС MS-DOS (Операционная система MS-DOS). В Windows запрещен прямой доступ к видеопамяти (не считая DirectX), и вывод работает здесь немного по-другому. Как именно, мы будем рассматривать, когда придет время знакомиться с графикой в Delphi.

Это пока все, что необходимо знать о работе компьютера. В следующих разделах некоторые, наиболее важные, моменты будут рассмотрены более подробно. Сейчас же нас будет интересовать работа процессора, поэтому ему уделяется здесь особое внимание.

1.2. Двоичная система работы процессора

Компьютеры изобрели достаточно давно. В те далекие времена об электронике даже и не упоминалось. Первые компьютеры были ламповыми и занимали очень много места. Для того чтобы управлять такой махиной, нужно было очень много обслуживающего персонала.

Однако именно тогда были заложены основные принципы работы компьютера, которые действует до сих пор. Суть их заключается в следующем. Данные передаются с помощью какого-то сигнала (для нас не имеет значения какого, потому что мы не электронщики) методом "есть сигнал или нет" или, по-другому, "включен или выключен". Так появился "бит" (bit). Бит это единица информации, которая может принимать значение 0 или 1, т. е. "включен или выключен ". Восемь бит объединяются в байт, один байт равен 8 битам. Почему именно 8? Да потому, что первые компьютеры были восьмиразрядными и могли работать одновременно только с 8 битами, например, 010000111.

Все первые нули можно удалять, поэтому число 010000111 можно записать как — 10000111. Это то же самое, что и в привычной для нас десятичной системе счисления, где каждый разряд может принимать значения от 0 до 9. Здесь так же никто не будет писать число 5743 как 0005743.

В один байт можно записать любое число от 0 до 255. Почему? Об этом немного позже. Указанный диапазон чисел довольно мал. Поэтому чаще используют более крупные градации:

- два байта = слово;
- два слова = двойное слово.

Итак, компьютер стал работать в двоичной системе счисления. Но как же тогда записать число 135, если у нас единица информации может принимать значения 0 или 1? Это можно сделать в двоичной системе. Давайте разберемся, как это работает.

Для начала вспомним, как действует десятичная система счисления, к которой мы привыкли. Для этого рассмотрим число 519 578 246. Я специально

выбрал такое число, чтобы оно состояло из восьми разрядов (цифр). Теперь запишем его, как показано на рис. 1.1.

Номер разряда	7	6	5	4	3	2	1	0
	1	9	5	7	8	2	4	6

Рис. 1.1. Число 519 578 246 в десятичной системе исчисления

Как видите, я пронумеровал разряды, начиная с нуля до восьми, и справа налево. Теперь представьте себе, что это не целое число, а просто набор разрядов. 1, 9, 5, 7, 8, 2, 4 и 6. Как из этих разрядов получить целое число? Наверно некоторые скажут, что надо просто записать их подряд. А если я спрошу, почему? Вот тут появляется математика. Нужно каждый разряд умножить на 10 (степень счисления, возведенную в степень номера разряда). Непонятно? Попробую оформить сказанное в виде формулы, показанной на рис. 1.2.

$$\text{Разряд 0} * (10^{\text{Номер разряда}}) + \text{Разряд 1} * (10^{\text{Номер разряда}}) + \ldots$$

Рис. 1.2. Работа с десятичными числами

Давайте посчитаем значение числа по этой формуле, начиная с нулевого разряда. Получается, что 6 нужно умножить на 10 в нулевой степени $6 \times 10^0 = 6$. Потом прибавить 4×10 в 1 степени, или $4 \times 10^1 = 40$ (и того уже 46). Потом 2×10 во второй степени, $2 \times 10^2 = 200$ (итого 246). Потом 8×10 в 3 степени, $8 \times 10^3 = 8000$ (и того 8246) и т. д. В итоге получится число 519 578 246.

А теперь рассмотрим двоичную систему счисления. Здесь каждый разряд может принимать значение 0 или 1 (2 состояния). Кстати, в десятичной системе у нас каждый разряд мог принимать значения от 0 до 9, т. е. имел десять состояний. Давайте рассмотрим следующий байт — 010000111. Запишем его на листке бумаги так, как показано на рис. 1.3.

Номер разряда	8	7	6	5	4	3	2	1	0
Биты		0	1	0	0	0	0	1	1

Рис. 1.3. Двоичное число

Здесь действует та же самая формула, только нужно возводить в степень не 10, а число 2. Опять же произведем расчет, начиная с нулевого разряда, т. е. справа налево. Получается, что первую 1 мы должны умножить на 2 в нулевой степени ($1 \times 2^0 = 1$). Следующую единицу нужно умножить на 2^1, получается 2 (итого 2 + 1 = 3) и т. д. Вот как это будет выглядеть полностью:

$$(1 \times 2^0) + (1 \times 2^1) + (1 \times 2^2) + (0 \times 2^3) + (0 \times 2^4) + (0 \times 2^5) + \\ + (0 \times 2^6) + (1 \times 2^7) + (0 \times 2^8) = 135$$

Вот так, оказывается, выглядит в двоичной системе счисления число 135. Давайте теперь научимся пересчитывать числа из десятичной системы в двоичную. Для этого нужно число 135 разделить на 2. Получается 67 и остаток 1 (запомним 1, т. к. она определяет первый двоичный разряд искомого числа). Теперь 67 снова делим на 2, получается 33 и остаток 1 (таким образом получено уже две двоичные единицы, т. е. 11). 33 делим на 2, получаем 16 и остаток 1 (в результате получаем три двоичные единицы, 111). 16 делим на 2, получаем 8 и остаток 0 (результат 0111). И наконец, 8 делим на 2 = 4, остаток от деления при этом будет 0 (получаем 00111); 4 делим на 2 = 2, остаток 0 (получаем 000111). 2 делим на 2 = 1, остаток 0 (итого 0000111). Оставшаяся 1 частного на 2 не делится, значит, это последний, самый старший разряд искомого числа. Просто дописываем ее к ранее сформированным разрядам и получаем окончательный ответ — 10000111. Получилось первоначальное число.

Вот так происходит преобразование чисел в двоичную систему счисления. Таким же образом можно перевести число в любую другую систему (двоичную, восьмеричную, шестнадцатеричную). Для более полного закрепления материала в табл. 1.1 показано соответствие десятичных чисел двоичным. Попробуйте сами перевести пару чисел из одной системы счисления в другую.

Таблица 1.1. *Таблица соответствия десятичных и двоичных чисел*

Десятичное	Двоичное	Десятичное	Двоичное
0	0	6	110
1	1	7	111
2	10	8	1000
3	11	9	1001
4	100	10	1010
5	101		

Например, попробуйте перевести число, состоящее из 8 бит (1 байт), у которого все биты состоят из единиц, в десятичную систему. Вы должны по-

лучить 255. Это максимальное число, которое можно записать в одном байте, потому что все его биты равны 1. Вот так и получается, что в 8 битах можно записать числа от 0 до 255. В 16 битах (2 байта или слово) можно записать число от 0 до 65 535. В 32 битах (двойное слово) можно уже записать число от 0 до 4 294 967 295.

В компьютере принято вести расчет в двоичной или шестнадцатеричной системе. Вторая вошла в обиход, когда компьютеры стали 16-разрядными.

Шестнадцатеричная система выглядит немного по-другому. Каждый разряд содержит уже не 2 состояния (как в двоичной системе) или десять (как в десятичной системе), а шестнадцать. Поэтому один разряд может принимать значения: 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Буква "A" соответствует цифре 10 в десятичной системе "B" соответствует 11 и т. д. Например, число 1A в шестнадцатеричной системе, равно 26 в десятичной. Почему? Да в соответствии все с той же формулой. Только здесь нужно возводить уже 16 в степень номера разряда. "A" — десять, нужно умножить на 16^0. В результате получится 10. 1 — первый разряд нужно умножить на 16^1, получится значение 16. Затем полученные результаты складываются и определяется искомое число — 10 + 16 = 26. В результате, как показано в табл. 1.2, можно установить соответствие между числами, записанными в различных системах счисления.

Таблица 1.2. *Таблица соответствия десятичных, двоичных и шестнадцатеричных чисел*

Десятичное	Двоичное	Шестнадцатеричное
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C

Таблица 1.2 *(окончание)*

Десятичное	Двоичное	Шестнадцатеричное
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14

На протяжении всей книги мы будем иногда встречаться с шестнадцатеричной системой счисления (без этого никуда не денешься). В этом случае, когда нужно будет показать, что число шестнадцатеричное, перед ним будет ставиться знак решетки #, например, #13. В других языках, например Assembler или C++, принято ставить в конце числа букву h (13h). Но эта книга о Delphi, поэтому здесь будем писать так, как принято в этой среде разработки, чтобы потом не возникало никаких вопросов.

До сих пор рассматривались целые числа. С числами с плавающей точкой совершенно другая история. Если заранее предусмотрено, что число может быть отрицательным, то его длина сокращается ровно на один бит (этот бит отводится под знак числа). Так, неотрицательное целое число может быть 8-битным, тогда как число со знаком будет 7-битным. Первый бит будет означать знак. Если первый бит равен 1, то число отрицательное, иначе положительное.

В дробных числах один байт может быть отведен для целой части и один для дробной. Никогда не смешивают целую и дробную части в одно целое. За счет этого дробные числа всегда будут занимать больше памяти, и операции с ними будут проходить намного дольше.

На первый взгляд перевод чисел очень сложный процесс, но вручную им пользоваться необязательно. Человек уже давно придумал для себя хорошего помощника — калькулятор. С его помощью без проблем можно перевести число в любую систему счисления.

Запустите встроенный в Windows калькулятор (**Пуск | Программы | Стандартные | Калькулятор**). Теперь выберите из меню **Вид** пункт **Инженерный**. На рис. 1.4 показано окно, которое вы должны увидеть.

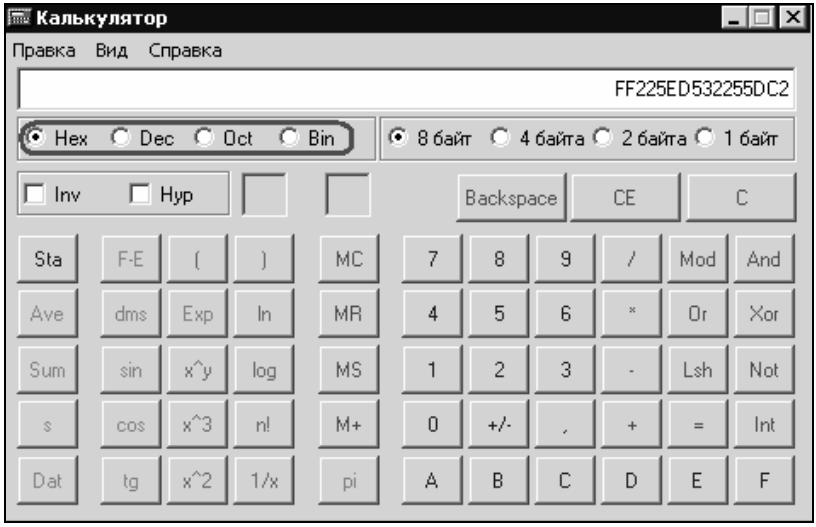


Рис. 1.4. Внешний вид калькулятора

Для перевода числа в другую систему просто наберите его в окне ввода калькулятора и потом выберите нужную систему счисления. На рисунке красным цветом выделены кнопки переключения из одной системы счисления в другую:

- **Hex** — шестнадцатеричная;
- **Dec** — десятичная;
- **Oct** — восьмеричная;
- **Bin** — двоичная.

Возникает вопрос — зачем здесь так долго рассказывалось о преобразованиях чисел, когда так легко можно воспользоваться калькулятором? Ответ прост — это нужно знать. Поверьте мне. Если вы будете понимать, как происходит преобразование, то вам потом легче будет работать с этими числами.

Никогда не полагайтесь только на технику. Всегда полезно знать, как и зачем она что-то делает. Если вы разберетесь с шестнадцатеричным представлением данных, то сможете простые преобразования делать в уме. Ну, а если вы еще собираетесь стать и *хакером*, то вам просто необходимо научиться хорошо оперировать разными системами счисления.

1.3. Машинный язык

Данные на диске хранятся в двоичном виде. Даже текстовые файлы на диске выглядят в виде нулей и единиц. Точно так же выглядит и любая про-

грамма, только ее называют *машинным кодом*. Давайте познакомимся с ним немного поближе.

Любая программа представляет собой последовательность команд. Эти команды называются *процессорными инструкциями*. В точном соответствии этим инструкциям процессор определяет, что и как ему нужно делать. Когда вы запускаете программу, компьютер загружает ее машинный код в оперативную память и начинает выполнять команду за командой. Наша задача как программистов, написать эти инструкции таким образом, чтобы компьютер понял, что мы от него хотим.

Реальная программа, которую выполняет компьютер, представляет собой последовательность единиц и нулей. Такую последовательность называют машинным языком. Человек не способен эффективно думать единицами и нулями. Для нас легче воспринимается осмысленный текст, а не сумасшедшие числа в двоичной системе счисления, с которой мы не привыкли работать. Например, *команда сложения* двух регистров выглядит так: #03C3. Это мало о чем говорит, и запомнить такую команду очень тяжело. На много проще написать "сложить число1 и число2 ".

Первое время программисты писали программы в машинных кодах, пока кому-то не пришла в голову идея: "Почему бы не писать текст программы на понятном языке, а потом заставлять компьютер переводить этот текст в машинный код?" Идея действительно заслуживала внимания. Так появился первый компилятор — программа, которая переводила текст программ в машинный код. Таким образом, пользователи стали писать программы более осмысленно, а всю рутинную работу по переводу текста программы в машинный код возложили на компьютер.

Здесь настало время сделать паузу и рассказать вам небольшую историю языков программирования. Она достаточно интересна и поучительна. Ну, а потом мы продолжим изучение принципов работы компьютера и познакомимся со структурой процессора и его работой при выполнении программы.

1.4. История языков программирования

Как мы уже выяснили, компьютер — примитивное существо, которое мыслит нулями и единицами, из которых складываются числа. Основное его устройство — процессор. Все, что может делать процессор, это оперировать двоичными числами. Программы — это тоже числа, которые воспринимаются процессором как некоторая совокупность команд процессору с целью выполнения им определенных действий.

Мы также выяснили, что первые программисты писали программы в машинных кодах. Тогда еще не было компиляторов и приходилось все писать числами. Вы даже представить себе не можете, какой это адский труд. Постоянно держать в памяти таблицу машинных кодов (это не таблица умно-

жения). Например, вам понятно шестнадцатеричное число 8BC3? Нет? А это обычная команда копирования между двумя ячейками регистров. Это просто пример, потому что тогда регистры были другие и процессоры были намного проще.

Со временем компьютер стал умнеть, но самое главное, он все также оперировал двоичными числами, однако делал это намного быстрее. Программист — это человек, а не железка, и ему очень тяжело создавать логику в числах. Намного легче работать с привычными словами. Например, все ту же команду пересылки удобней записать словами типа "скопировать ebx в eax". Но что делать, если компьютер не понимает слов, а только числа? Выход есть — написать такую программу, которая будет превращать текст в машинные коды. Пусть компьютер сам создает *байт*-код. Программу, выполняющую эти действия, назвали компилятором, а язык, на котором писался исходный текст программы, назвали языком программирования.

```
MainUnit.pas.131: InitViewGrid;
00527F7C 8BC3            mov eax,ebx
00527F7E E84D010000      call TSborDataForm.InitViewGrid
MainUnit.pas.132: InitProgramm;
00527F83 8BC3            mov eax,ebx
00527F85 E8B6030000      call TSborDataForm.InitProgramm
MainUnit.pas.133: if lDeviceNum>=0 then
00527F8A 83BB1004000000  cmp dword ptr [ebx+$00000410],$00
00527F91 7C26            jl +$26
MainUnit.pas.135: ScanPortThr := TScanPortThread.Create(true);
00527F93 B101            mov cl,$01
00527F95 B201            mov dl,$01
00527F97 A1D4A45200      mov eax,[$0052a4d4]
00527F9C E8EB94EFFF      call TThread.Create
00527FA1 8BF0            mov esi,eax
00527FA3 89B314040000    mov [ebx+$00000414],esi
MainUnit.pas.136: ScanPortThr.lDeviceNum:=lDeviceNum;
00527FA9 8B8310040000    mov eax,[ebx+$00000410]
00527FAF 89464C          mov [esi+$4c],eax
MainUnit.pas.137: ScanPortThr.Resume;
00527FB2 8BC6            mov eax,esi
00527FB4 E81F98EFFF      call TThread.Resume
MainUnit.pas.141: try
00527FB9 33C0            xor eax,eax
00527FBB 55              push ebp
00527FBC 680B805200      push $0052800b
00527FC1 64FF30          push dword ptr fs:[eax]
```

Рис. 1.5. Программа в машинных кодах и Assembler

И вот был разработан первый компилятор. Эту программу назвали *Assembler*, что переводится, как "сборщик". Писать на нем практически так же сложно, как и в машинных кодах, однако теперь уже использовались не числа, а по-

нятные, как показано на рис.1.5, человеку слова. Например, все та же команда копирования регистров теперь выглядела так: `mov eax, ebx`. То есть цифры заменились на понятные слова.

Вроде все прекрасно и удобно, но почему-то среди программистов возникли споры и разногласия. Кто-то воспринял новый метод с удовольствием. А кто-то говорил, что машинные коды лучше. Любители языка Assembler хвалили компилятор за то, что программировать стало проще и быстрее, а противники утверждали, что программа, написанная в кодах, работает быстрей. Говорят, что эти споры доходили до драк и иногда лучшие друзья становились врагами. В принципе, и те, и другие были правы. На языке Assembler действительно программу писать легче и быстрей, но программа, написанная в машинных кодах, работала быстрее.

Тогда никто не мог себе представить, чем же все может закончиться. Но время показало свое. С помощью Assembler программы писались быстрее, а это один из основных факторов успеха любой программы на рынке. Люди начинают пользоваться тем продуктом, который выходит на рынок первым. Даже если более поздний вариант лучше, человека трудно переубедить перейти на другую версию. Здесь начинает играть большую роль фактор привычки. К тому же, когда программист напишет свою первую версию программы в машинных кодах, программист на языке Assembler выпустит уже пару новых версий.

Вот так и получилось, что те, кто программировал на языке Assembler, превратились в убегающих вперед, а те, кто программировал в машинных кодах превратились в постоянно догоняющих. В конце концов, первые убежали настолько "далеко", что вторые не смогли догнать и вынуждены были или перейти на Assembler, или отойти от программирования совсем.

С этого момента начался бум. Языки программирования стали появляться один за другим. Так появились *C*, *ADA*, *FoxPro*, *Fortran*, *Basic*, *Pascal* и др. Некоторые из них были предназначены только для школьников или обучения, другие были ориентированы на профессиональных программистов. И тут споры перенеслись в другую плоскость — какой язык лучше. Некоторые говорили, что это Pascal, другие утверждали что C, ну, а кое-кто утверждал, что это Visual Basic. Этот спор длится уже около 30 лет, и конца ему не видно. При этом все спорные вопросы разделились на две части:

1. Какой язык самый лучший?
2. Что лучше — язык высокого уровня или низкого?

Спор по первому пункту не может закончиться до сих пор. Каждый пытается доказать, что его язык программирования самый мощный, удобный и создает самый быстрый программный код. Мне кажется, что этот спор не закончится никогда. В принципе, такое положение дел устраивает всех, потому что это своеобразная конкуренция. Благодаря ей происходит развитие языков программирования, и мы быстро продвигаемся вперед.

Так все же, какой язык лучше? На этот вопрос можно дать ответ, но об этом немного позже.

Наиболее интересным был спор: "Что лучше — язык высокого уровня или низкого?" Язык низкого уровня это тот, который ориентирован на команды процессора, т. е. Assembler. К языкам высокого уровня относят C, Pascal, Basic и др. Они ориентированы на людей и создают им максимум удобства при написании программ. Этот спор проходил в той же манере, как и спор между любителями Assembler и любителями программирования в машинных кодах. Только теперь приверженцы языка Assembler утверждали, что их код самый быстрый, а любители языков высокого уровня утверждали, что они напишут программу быстрей, чем самый лучший программист на языке Assembler.

Спор продолжался достаточно долгое время. И опять победила скорость разработки и "удобство" языка программирования. Любителям Assembler пришлось отступить, потому что теперь они превратились в " догоняющих".

Конечно же, нельзя сказать, что машинные коды и Assembler окончательно ушли из нашей жизни. Они используются до сих пор, но в очень ограниченном количестве. Язык Assembler используется только в качестве вставок для языков высокого уровня, а машинные коды используются для написания того, что не может сделать компилятор (да и для написания самого компилятора они могут потребоваться). Ушедшие технологии живут и будут жить, но рядовой программист очень редко встречается с ними.

Следующей ступенью стало объектно-ориентированное программирование. Язык C превратился в *C++*, Pascal превратился в *Object Pascal* и т. д. И снова борьба. И снова скорость разработки против скорости выполнения программного кода. Опять споры, драки и оскорбления.

Война длилась несколько лет. Сколько времени было потрачено в спорах, сколько волос было вырвано на голове в процессе доказательств силы именно его программного кода. А в результате победила скорость и удобство разработки, т. е. объектно-ориентированное программирование (ООП).

Последней крупной революцией, происходящей в программировании, считется переход на визуальное программирование. Этот переход происходит прямо на наших глазах. Визуальность дает нам еще более удобные средства разработки для более быстрого написания кода, но проигрывает ООП по быстроте работы. Вот почему многие начинающие программисты стоят на перекрестке, не зная, какой язык выбрать.

Лидером в разработке визуальных языков программирования является Borland, а приверженцем ООП остается Microsoft. Конечно же, Билл Гейтс пытается встроить в свои языки визуальность, но она примитивна по сравнению с такими гигантами, как *Delphi*, *Kylix* или *C++ Builder*. Это связано с изначальной неправильной разработкой MFC (Microsoft Foundation Classes —

Основные классы Microsoft), которая не может работать визуально. Нужна глобальная переработка кода, которую почему-то не хотят делать. Вот народ и стоит на двух атомных бомбах, ожидая взрыва одной из них. Как вы думаете, какая бомба рванет? Что победит — скорость разработки или скорость кода? Я не буду отвечать на этот вопрос. История говорит сама за себя, а мы подождем подтверждение этому.

Считается, что информационные процессы не будут стоять на одном месте и переход на новые технологии программирования рано или поздно состоится. Поэтому я уже перешел на Delphi. Если вы хотите успеть за прогрессом, то тоже обязаны вступить в партию любителей Borland. Выбирайте любой из его компиляторов, и вы не ошибетесь. Для вас есть все, что угодно: Delphi, JBuilder, Kylix или C++ Builder. Как видите, у корпорации Borland есть визуальные варианты всех языков, и они действительно лучшие.

Как ранее уже говорилось, самая лучшая технология программирования это та, в которой поддерживается визуальность. Ваша среда разработки просто обязана быть визуальной, потому что за этим наше будущее. Если вы хотите получить визуальность плюс мощь разработки — ваша среда от Borland. С помощью языков этой фирмы можно сделать абсолютно все. Так что с этим мы покончили. Вердикт окончательный и обжалованию не подлежит.

Осталось только ответить на вопрос: "Какой язык программирования лучше?" Я уже несколько лет пытаюсь ответить на этот вопрос, но окончательного решения вынести не могу. Даже у того же *Visual C++* от Microsoft есть свои плюсы. Как это ни странно, но положительные стороны есть у всех. Вопрос остается только за тем, какие программы будут создаваться? Здесь можно дать примерно такую градацию:

1. Если вы будете писать базы данных, программы общего значения или утилиты, то ваш язык Delphi или C++ Builder.
2. Если это игры, то желательно Visual C++ или Watcome C плюс знание Assembler. Но это не значит, что нельзя использовать Delphi или C++ Builder. В этих средах вы потеряете не намного больше в скорости работы, поэтому в большинстве игр можно не обращать внимания на эту потерю. Если правильно использовать свои знания, то и на самом медленном и слабом языке программирования можно создать шедевр.
3. Если это будут драйверы и работа с "железом", где критичен размер файла, ваш язык — чистый C или Assembler.

И все же большую массу программ занимают утилиты и базы данных. А тут визуальность необходима, если вы хотите оказаться впереди. Визуальные языки будут жить и за ними будущее. На протяжении всей этой книги будет рассматриваться самый лучший (это на мой взгляд, и он может отличаться от других) язык программирования — Delphi.

Корпорация Microsoft тоже движется в сторону визуальности и простоты разработки программных продуктов. Об этом говорит их новая разработка *C#*. Но этот язык я считаю самой неудачной их разработкой и самым наглым маркетинговым ходом. Поэтому хочу вас предостеречь от бесполезной траты времени на него.

Совет

Я не могу описывать здесь как сам этот язык программирования, так и его проблемы. Это выходит за рамки данной книги. Однако советую вам прочитать мою статью о технологии .NET, которая опубликована на сайте (или на диске к этой книге) в разделе "Без рамки".

1.5. Исполнение машинных инструкций

Прежде чем переходить к дальнейшему изучению материала, необходимо рассмотреть ряд понятий.

Сегмент — это область внутренней (оперативной) памяти компьютера (внешняя память представлена накопителями на магнитных дисках и лентах). Раньше, когда операционные системы были 16-битными, процессор не мог работать с памятью размером более 64 килобайт (это максимальный размер области памяти, который можно адресовать, используя в этих целях адрес длиной в два байта). Поэтому память делилась на сегменты по размеру и по назначению. На данный момент мы используем 32-разрядную ОС, которая может адресовать до 4 Гбайт оперативной памяти (длина адреса — 32 бита или 4 байта). Поэтому можно сказать, что память стала сплошной. Однако деление ее по назначению все-таки осталось. Существуют следующие сегменты памяти:

- *сегмент кода* — в эту область памяти загружается машинный код, который будет потом выполняться процессором;
- *сегмент данных* — это область памяти для хранения данных;
- *сегмент стека* — область памяти для хранения временных (локальных) данных и адресов возврата из процедур.

Каждой запущенной программе отводится свой сегмент кода, данных и стека. Поэтому данные одной программы не могут пересекаться с данными или кодом другой программы, если, конечно же, не произошел сбой с ошибкой доступа краспределенной области оперативной памяти.

Регистр — ячейка памяти в процессоре. Размер ячейки зависит от ее разрядности. В 32-разрядных процессорах ячейки 32-битные. Мы будем говорить о 32-разрядных процессорах, а значит и о 32-битных регистрах. Таких регистров у процессора несколько и каждый из них предназначен для определенных целей.

Когда компьютер был 16-битным, все регистры были тоже 16-битными. С появлением 32-разрядной платформы размер ячейки регистра тоже увеличился до 32 бит. Для совместимости со старыми программами такие регистры как бы делятся на две части (две ячейки по 16 бит). Первая — определяет старый регистр, а вторая — дополнительные 16 бит. На словах не совсем понятно, поэтому давайте посмотрим на рис. 1.6.

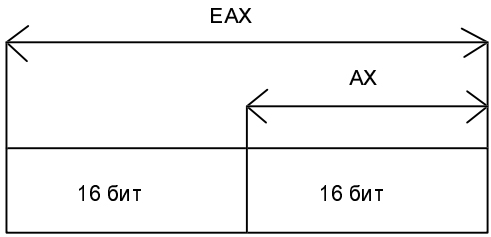


Рис. 1.6. Программа в машинных кодах и Assembler

На этом рисунке показан регистр EAX. Полная его длинна 32 бита, но младшая половина — это регистр AX (16-битный вариант регистра). То есть если мы "попросим" процессор показать нам содержимое регистра AX, то мы увидим половину регистра EAX. Иногда это очень удобно, особенно когда требуется прочитать только половину числа из регистра.

Рассмотрим пример. Пусть в регистре EAX находится шестнадцатеричное число #21CD52B, тогда в регистре AX будут находиться последние 16 бит, а именно число #D52B.

Теперь рассмотрим описание регистров процессора. При этом будем иметь в виду следующее: если имя регистра будет начинаться с буквы "E", это значит, что он 32-битный и для него существует и 16-битный вариант без буквы " E". Все регистры можно разделить на несколько групп.

- *Сегментные регистры* — *CS*, *DS*, *SS* и *ES* (в данную группу входит еще ряд регистров, но нас пока интересуют только эти).
  - *Регистр CS* — регистр сегмента кода, в нем хранится начальный адрес сегмента кода.
  - *Регистр DS* — регистр сегмента данных, в нем хранится начальный адрес сегмента данных. В этом сегменте располагаются глобальные переменные.
  - *Регистр SS* — регистр сегмента стека, в нем хранится начальный адрес сегмента стека. Здесь располагаются локальные переменные.
  - *Регистр ES* — регистр дополнительного сегмента. Применяется при использовании в программе дополнительного сегмента.