Дмитрий Осипов

Delphi XE2

- Визуальная библиотека компонентов
- Сетевые приложения на базе именованных каналов и механизма сокетов
- Клиент-серверные приложения на основе технологии COM
- Управление процессами, потоками и службами
- Многоуровневые приложения DataSnap
- Технология живого связывания LiveBindings
- Введение в кроссплатформенную разработку на основе библиотеки FireMonkey

+ Материалы на www.bhv.ru

Наиболее полное руководство

В ПОДЛИННИКЕ®

УДК 681.3.068+800.92Delphi XE2
ББК 32.973.26-018.1
О-74

Осипов Д. Л.

О-74 Delphi XE2. — СПб.: БХВ-Петербург, 2012. — 912 с.: ил. — (В подлиннике)

ISBN 978-5-9775-0825-4

Книга посвящена одному из самых совершенных языков программирования Delphi XE2. В ней излагаются основы программирования на языке Delphi XE2, подробно рассматривается визуальная библиотека компонентов (VCL), описывается порядок разработки программного обеспечения для 32- и 64-разрядных версий Windows с использованием функций Win API, предоставляется обзор новейшей кроссплатформенной библиотеки FireMonkey, позволяющей создавать программное обеспечение не только для ОС Microsoft Windows, но и для Mac OS X. Примеры проектов из книги размещены на сайте издательства.

Для программистов и студентов

УДК 681.3.068+800.92Delphi XE2
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 05.03.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 73,53.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0825-4

Оглавление

ГЛАВА 1

Знакомство

Часть I книги призвана научить читателя основам программирования на языке Delphi — потомке знаменитого языка Pascal, созданного известным швейцарским ученым, профессором Цюрихской высшей технической школы, обладателем премии Тьюринга (высшей награды для специалистов по информационным технологиям) Никлаусом Виртом (Niklaus Wirth).

Появившийся на свет в начале 1970-х годов язык Pascal предназначался для обучения студентов основам структурного программирования. За более чем четыре десятилетия своего развития Pascal не просто доказал свою жизнеспособность, он стал одним из ведущих языков разработки программного обеспечения. Во многом своему первому успеху язык Pascal обязан компании Borland, в стенах которой в 1992 г. появилась среда разработки Borland Pascal with Objects 7.0, и немного позднее, в 1995 г., — знаменитая Borland Delphi 1.0 для разработки программного обеспечения (ПО) для 16-разрядной ОС Microsoft Windows 3.*x*.

> ***Замечание***
>
> Имя Pascal было выбрано не случайно — язык был назван в честь французского математика Блеза Паскаля.

Успех первой версии Delphi и ее *библиотеки визуальных компонентов* (Visual Components Library, VCL) был столь оглушителен, что несколько лет спустя, опираясь на библиотеки Delphi, компания Borland создала еще один шедевр — среду C++Builder (на этот раз с базовым языком Си). Так, уже почти 20 лет, Delphi и C++Builder идут вместе. Эти два языка функционируют в одной среде разработки, обладают практически идентичными возможностями и отличаются лишь особенностями построения синтаксических конструкций, поэтому программист Delphi легко адаптируется к программам, написанным на C++Builder, и, наоборот, разработчик, имеющий опыт работы в среде C++Builder, поймет программу на Delphi.

Сегодня права на язык Delphi принадлежат компании Embarcadero Technologies. Новые владельцы Delphi не просто воспользовались успешными наработками программистов Borland, но и внесли в язык и среду разработки многочисленные улучшения и усовершенствования. В частности, современная Delphi позволяет разрабатывать 32- и 64-разрядные приложения для Windows на базе популярной библиотеки VCL и способна создавать кросс-платформенные приложения (для Windows, Mac OS X и iOS) на основе новейшей библиотеки FireMonkey. На момент выхода в свет Delphi XE2 осенью 2011 г. подобными возможностями не обладала ни одна среда разработки!

В настоящее время существуют две базовые разновидности Delphi:

- Delphi XE2 (16-я версия языка), предназначенная для создания программного обеспечения для Microsoft Windows, Mac OS X и iOS;
- Delphi Prism XE2 — язык разработки для платформы .NET (Microsoft Windows).

Эта книга посвящена наиболее новой версии языка — Delphi XE2, однако более половины примеров из книги сохранят работоспособность и в более ранних версиях среды разработки.

Структура консольного приложения

Для начала следует обратиться к фундаментальным основам языка Delphi, поэтому чтобы не "оглушить" читателя обилием пока непонятных терминов, мы, насколько возможно, постараемся абстрагироваться от сложной для начинающего программиста визуальной библиотеки компонентов (VCL). Девизом этой части книги можно считать разумный минимализм. Именно поэтому первые шаги изучения языка разработки мы сделаем в элементарных консольных приложениях, где нет отвлекающих новичка элементов управления и код максимально прост и линеен.

Для создания заготовки консольного приложения Delphi в главном меню среды разработки выберите пункт меню **File | New | Other** (Файл | Создать | Другое). Если все сделано верно, то перед нами появится окно **New Items** (Новый элемент) с открытым набором проектов **Delphi Projects**. Найдите на этой странице значок **Console Application** (Консольное приложение) и щелкните по кнопке **OK** (рис. 1.1). За этот труд Delphi отблагодарит нас заготовкой для самого простейшего приложения — консольного окна Windows.

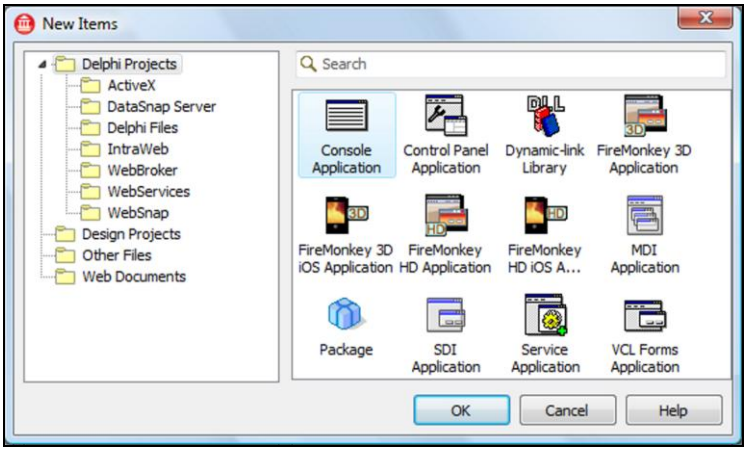

Рис. 1.1. Окно выбора типа проекта Delphi XE2

Сразу приучим себя к одному из ключевых правил программистов — исходный код программы должен регулярно сохраняться на жестком диске компьютера. Поэтому, прежде чем мы приступим к изучению заготовки консольного приложения, немедленно сохраним только что созданный проект. Для этого находим пункт главного меню среды разработки **File | Save All** (Файл | Сохранить все), в появившемся диалоге сохранения выбираем целевую (желательно пустую) папку, в которую отправятся файлы проекта. Среда проектирования обязательно спросит, под каким именем следует сохранить проект. Присвоим своему пер-

вому проекту имя (для первого раза вполне подойдет название по умолчанию — Project1.dproj). В завершение нажмем кнопку **OK**.

Обязательно откройте целевую папку в любом файловом менеджере, например в Проводнике Windows, и посмотрите, какие файлы были созданы Delphi во время сохранения нашего проекта:

- Project1.dproj — файл в формате расширяемого языка разметки (eXtensible Markup Language, XML), содержащий основные конфигурационные и справочные сведения о нашем проекте;
- Project1.dpr — головной файл с кодом проекта на языке Delphi;
- Project1.res — файл с ресурсами приложения.

После компиляции проекта появится еще ряд файлов, в том числе исполняемый файл с расширением exe. Но сейчас для нас наибольшую ценность представляет головной файл проекта — Project1.dpr. Именно в нем находится самое важное — исходный код будущего приложения. Этот же исходный код отобразится в редакторе Delphi (рис. 1.2).

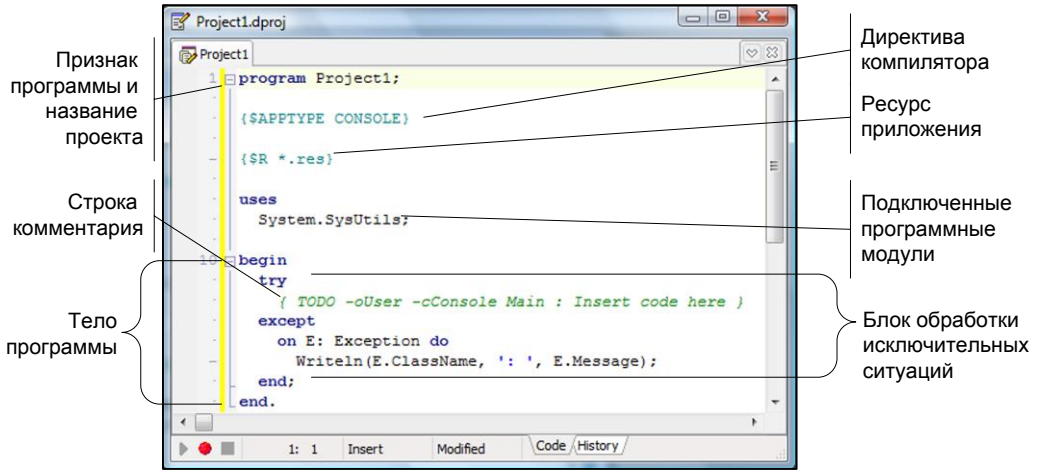

Рис. 1.2. Код консольного приложения Delphi XE2

Простейшее консольное приложение содержит немногим больше десяти строк (см. рис. 1.2). Листинг начинается с зарезервированного слова `program`. Это признак того, что перед нами листинг будущей программы. За словом `program` следует имя программы (в нашем случае `Project1`). Строкой ниже вы найдете директиву, определяющую порядок компиляции проекта. Еще ниже, после ключевого слова `uses`, приводится перечень подключаемых к проекту дополнительных программных модулей: в примере это всего лишь один модуль системных утилит — `SysUtils`, принадлежащий *пространству имен* `System`. Основной код программы располагается между ключевыми словами — `begin` и `end`. Пока здесь мы обнаружим только секцию обработки ошибок `try..except` и строку комментария.

Комментарии

Ни один, даже очень профессиональный программист, при разработке программного обеспечения не может обойтись без услуг комментариев. *Комментарий* — это не что иное, как краткое пояснение того, что происходит в тех или иных строках листинга. Человеческая

память не идеальна, и нередко нас подводит. А теперь представьте себе, что вы решили доработать код годовалой давности. Попробуйте быстро сообразить, для чего предназначалась переменная с "мудреным" названием `X` или что произойдет после вызова процедуры `MySuperProcedure()`?

Другими словами, в комментариях разработчик кратко поясняет смысл рожденных в его голове команд. В результате листинг программы становится более понятным и доступным для изучения.

Для того чтобы при компиляции программы текст комментариев не воспринимался Delphi в качестве команд и не служил источником ошибок, приняты следующие соглашения. Комментарием считается (листинг 1.1):

- отдельная строка, начинающаяся с двух наклонных черт `//`;
- весь текст, заключенный в фигурные `{ }` или в круглые скобки с символами звездочек `(* *)`.

Листинг 1.1. Примеры комментариев

```
//Одна строка комментария

{Текст
 многострочного
 комментария}

(*Это также
комментарий*)
```

Будьте внимательны! Если внутри фигурных скобок на первой позиции окажется символ `$`, то это не строка комментария, а директива компилятора. В шаблоне только что созданного нами приложения (см. рис. 1.2) такая директива имеется — `{$APPTYPE CONSOLE}`. Это означает, что наш проект должен стать консольным приложением.

> ***Внимание!***
>
> Директивы компилятора начинаются с символа открывающейся фигурной скобки и символа доллара `{$...}`. Начинающему программисту ни в коем случае не следует удалять, редактировать или изменять местоположение директив компилятора, в противном случае вы рискуете привести свой проект в негодность.

Перевод листинга программы в машинные коды

Для того чтобы листинги программы превратились в полноценное приложение, необходимо сделать их понятными для центрального процессора компьютера. Процессор не понимает языка Delphi (как, впрочем, и любых других языков высокого уровня), вместо этого процессор работает с машинными командами.

Чтобы листинг программы превратился в машинные коды, необходима помощь трех специальных подпрограмм: *препроцессора*, *компилятора* и *компоновщика*.

Препроцессор подготавливает программу к компиляции. Для этого он изучает и выполняет имеющиеся в программе директивы.

Компиляция — это процесс преобразования команд языка программирования в машинный код, понятный процессору компьютера. Во время компиляции программа-компилятор просматривает исходный код программы на предмет синтаксических ошибок и выполняет смысловой анализ кода.

На финальной стадии в дело вступает *компоновщик*. Он подключает к нашей программе все необходимые для ее работы модули, в первую очередь модули дополнительных библиотек.

В результате работы препроцессора, компилятора и компоновщика Delphi получается исполняемый файл с расширением exe. Полученный файл можно переносить на другой компьютер и использовать как самостоятельную программу для Windows.

> ***Замечание***
>
> Далее в книге процесс перевода листинга программы в машинные коды мы станем называть *компиляцией*.

Компилятор Delphi является высокотехнологичной программой и представляет собой ядро всей среды разработки. Для того чтобы полученная в результате компиляции программа стала высокопроизводительной, профессиональные программисты *оптимизируют параметры компиляции*. Для вызова окна с опциями компилятора следует воспользоваться пунктом меню **Project | Options** (Проект | Параметры) и в появившемся окне настроить параметры компилятора, выбрав элемент **Delphi Compiler** (рис. 1.3).

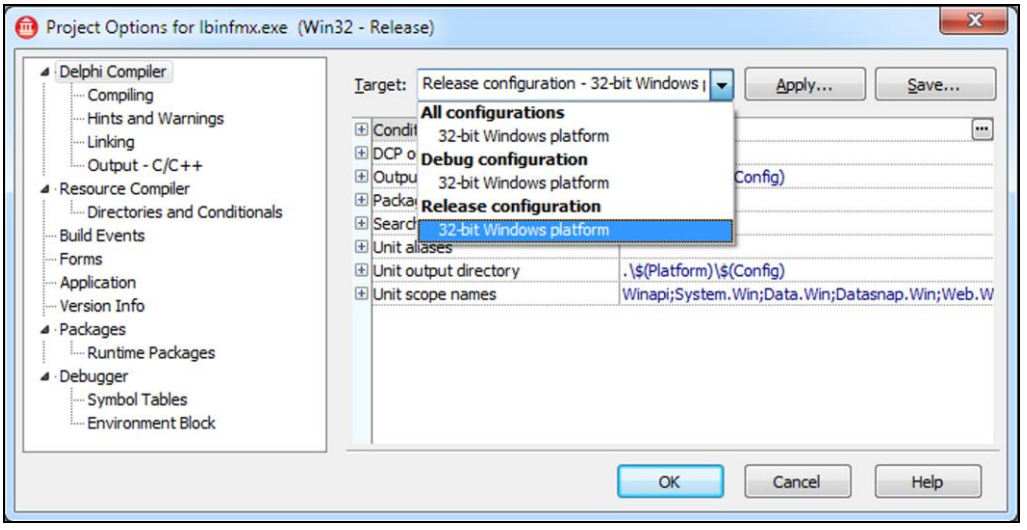

Рис. 1.3. Настройка параметров компиляции проекта

Без особой на то причины начинающему программисту не стоит экспериментировать с параметрами компилятора, однако необходимо знать следующее. По умолчанию компилятор настроен на работу в *режиме отладки* (debug). Этот режим используется во время проектирования приложения Delphi и позволяет эффективно выявлять и устранять ошибки в исходном коде. После окончательной победы программиста над всеми ошибками отладка программы завершается, самая последняя компиляция проекта осуществляется в режиме *выпуска конечного продукта* (release). Заказчику программного обеспечения ваши программы должны передаваться откомпилированными в режиме конечного продукта. Для этого следует изменить конфигурацию компилятора (**Build Configuration**), выбрав в раскрывающемся списке (см. рис. 1.3) вариант **Release configuration**.

Для старта компиляции проекта Delphi достаточно выбрать пункт меню **Run | Run** или (что еще проще) нажать клавишу <F9>. Если компилятор работает в режиме настроек по умолчанию, то в результате этих действий в папке с проектом появится дочерняя папка Debug, "порывшись" в которой вы обнаружите готовую к работе программу с расширением exe.

Первая программа

Настал тот час, когда мы готовы написать свою первую консольную программу на языке Delphi. Создайте новый проект консольного приложения и сохраните его в отдельную папку под любым именем (я выбрал имя Hello.dproj).

По давно сложившейся традиции (у ее истоков стояли известные программисты Брайан Керниган (Brian Kernighan) и Деннис Ритчи (Dennis Ritchie)) самой первой программой, открывающей книгу по программированию, станет программа-приветствие "Hello, world!", ее код представлен в листинге 1.2.

Листинг 1.2. Программа "Hello, world!"

```
program Hello;

{$APPTYPE CONSOLE}//это директива компилятора, которую мы не трогаем!

uses System.SysUtils;  (*строка подключения внешних библиотек подпрограмм, хотя,
между нами говоря, в этой программе внешние модули нам не нужны*)

begin
WriteLn('Hello, world!'); //выводим текст приветствия
ReadLn; //ожидаем ввод — нажатие любой клавиши завершит работу консоли
end.
```

Введя код программы, обратитесь к главному меню Delphi и выберите пункт **Run | Run** или воспользуйтесь клавишей <F9>. Если вы были внимательны и переписали строки листинга, не совершив ни одной ошибки, то за считанные доли секунды на экране появятся плоды нашего коллективного творчества — консольное окно со строкой "Hello, world!". Если же была допущена ошибка, то компилятор просигнализирует о ней, выделив в листинге подозрительную строку или следующую за ней строку.

Для вывода текстовой строки на экран компьютера нам понадобились услуги процедуры `WriteLn()`. Эта процедура еще не раз встретится нам в других примерах к этой книге, пока вам достаточно запомнить то, что `WriteLn()` позволяет отобразить в окне консольной программы текст, но при условии, что символы текста заключены в одинарные кавычки.

Завершая вводную главу, попрошу запомнить вас еще одну особенность синтаксиса языка Delphi — каждое выражение должно завершаться *точкой с запятой*, а признаком окончания всей программы является ключевое слово `end` *с точкой на конце*.

ГЛАВА 2

Типы данных Delphi

Функциональное назначение любой программы, будь это простейшее консольное приложение, выводящее на экран приветствие "Hello, world", или сложный математический пакет, моделирующий термоядерную реакцию, заключается в получении, обработке и возврате пользователю определенных результатов. Обслуживаемые программой данные размещаются в памяти компьютера в виде разнообразных структур, в простейшем случае это обычная последовательность байтов, а в более сложных ситуациях структура представляет собой весьма неординарную конструкцию, которая способна не просто хранить, но и управлять своими данными.

Сложность программы состоит в прямой зависимости от способов представления данных в памяти и задействованных для обслуживания этих данных алгоритмов. Николаусу Вирту принадлежит меткое определение программы:

Программа = структуры + алгоритмы.

Заметьте, что в определении профессора Вирта на первом месте стоят структуры, а алгоритмы только на втором. Это не случайность. Дело в том, что даже самый совершенный и глубоко проработанный алгоритм окажется малоэффективным, если подлежащие обработке данные предоставляются алгоритму в ненадлежащем виде. Здесь все как в автомобиле, никакой даже лучший на свете двигатель (читай — алгоритм) не в состоянии сдвинуть автомобиль с места, если к нему не прикручены колеса.

В этой главе мы начнем обсуждение первого ингредиента программы — структур. Здесь мы рассмотрим простейшие способы представления данных в программах Delphi, более сложные решения будут вынесены в следующую главу книги.

Переменные

Наиболее распространенный способ хранения данных реализуется при посредничестве *переменных*. Переменные позволяют размещать в памяти значения различных типов (целые и вещественные числа, символы, текстовые строки). Термин "переменная" (variable) подсказывает, что в ходе выполнения программы содержимое памяти может изменяться.

Объявление переменной включает упоминание слова `var` и указание имени и типа переменной.

```
var имя_переменной : тип данных;
```

Если необходимо объявить несколько однотипных переменных, то разрешается перечислить их, разделяя имена запятыми так, как представлено в листинге 2.1.

Листинг 2.1. Объявление переменных

```
var a: integer;  //переменная для хранения целого числа
    b: boolean;  //переменная для хранения логического значения
    c, d : real; //переменные для обслуживания действительных чисел
```

Более подробно о типах данных мы поговорим немного позднее, а пока достаточно понимать, что тип данных определяет особенность хранения и представления значений в переменных. Если необходимо оперировать целыми числами, то, скорее всего, вам пригодится тип данных `integer`, для обслуживания вещественных чисел воспользуйтесь типом `real`, для манипуляций отдельными символами понадобится `Char`.

> ***Примечание***
>
> Пока, при встрече с термином "тип данных", нам достаточно понимания того, что тип данных определяет число битов памяти, отдаваемых в распоряжение переменной. Чем больше битов, тем данные большего размера могут быть переданы в переменную. Добравшись до последних страниц главы, читатель увидит, что тип данных представляет собой более сложное понятие.

Современные версии Delphi позволяют проинициализировать содержимое переменной в момент ее объявления (листинг 2.2).

Листинг 2.2. Инициализация переменной во время объявления

```
var i: integer=10;    //целочисленная переменная i содержит значение 10
    b: boolean=false; //логическая переменная b содержит false
    c: char='!';      //символьная
```

Унаследованный от языка Pascal классический стиль программирования строго устанавливает области кода, в которых допускается объявление переменных. В консольных приложениях переменные должны объявляться перед телом программы, которое, в свою очередь, заключается внутрь составного оператора `begin..end`. При описании функции (процедуры, метода) все объявления локальных переменных должны быть сосредоточены между заголовком процедуры и началом ее реализации (листинг 2.3).

Листинг 2.3. Объявление локальной переменной в функции

```
function MyFunction: integer;
var i: integer; //локальная переменная i
begin
  i:=0; //переменная i будет доступна только внутри функции
  ...
end;
```

В зависимости от места объявления переменной можно говорить о ее локальном или глобальном статусе. *Глобальные переменные* проекта доступны или, как говорят программисты, видимы из всех областей программы. К такой переменной можно обратиться из любой процедуры или функции. В противовес глобальным переменным, область видимости их локальных "коллег" существенно у́же. Например, переменная, объявленная внутри функции (см. листинг 2.3), может быть задействована только в границах этой функции. Попытка

прочитать содержимое локальной переменной извне области ее видимости обречена на неудачу.

Константы

В отличие от переменных *константы* (constants) предназначены для хранения заранее известных и неизменяющихся во время выполнения программы значений. Объявление константы начинается с ключевого слова `const` и включает имя, необязательное указание типа и значение константы.

const *имя_константы*: [*необязательный тип*] = *значение*;

Пример объявления констант предложен в листинге 2.4. Обратите внимание на то, что для определения значения константы могут применяться выражения, например константа `c` станет хранить результат операции деления 5 на 3.

Листинг 2.4. Порядок объявления констант

```
const a = 3.14;
      b = 65535;
      c : extended = 5/3;
```

Так же как и в случае с переменными, в Delphi константа должна быть объявлена в строго отведенном для этого месте программы (идентичном местам объявления переменных). По аналогии с переменными константы обладают областью видимости, поэтому различают *глобальные* и *локальные константы*.

Строки-ресурсы

К одной из разновидности констант стоит отнести *строки-ресурсы* (resource strings). Строки-ресурсы состоят из текстовой информации и специальных символов форматирования (применяемых в функциях `Format()` и `FormatDateTime()`), вместо которых позднее можно вставить дополнительную информацию.

Листинг 2.5 демонстрирует один из вариантов применения строк-ресурсов.

Листинг 2.5. Порядок объявления констант

```
resourcestring ResDateTime = 'Дата d.mm.yyyy Время hh:mm';
var s:string;
begin
  S := FormatDateTime(ResDateTime, Now);
  WriteLn(S);
end.
```

В предложенном примере символы `d.mm.yyyy` замещаются текущей датой, а символы `hh:mm` — текущим временем.

Правила объявления идентификаторов

Все используемые в программах Delphi переменные, константы, процедуры, функции, объекты должны обладать уникальным именем — *идентификатором*.

В языке Delphi длина идентификатора не ограничена, но значащими являются только первые 255 символов. Идентификатор может содержать любые символы латинского алфавита, цифры и символ нижнего подчеркивания. Первый символ идентификатора обязан быть буквой. Само собой, в роли идентификаторов не допускается применять зарезервированные слова.

Различают *неквалифицированные* и *квалифицированные идентификаторы*. Синтаксис последних выглядит следующим образом:

```
идентификатор1.идентификатор2
```

Подобное обращение необходимо в тех ситуациях, когда требуемая переменная (функция, метод, свойство, объект и т. п.) принадлежит другому модулю (объекту, классу и т. д.). Таким образом, квалифицированный идентификатор знает всю цепочку своих владельцев.

Неквалифицированные идентификаторы не требуют явного указания своих владельцев. Это допускается в том случае, когда в принадлежности идентификатора сомнений не возникает.

> ***Замечание***
>
> В отличие от идентификаторов языка Си, идентификаторы языка Delphi *не чувствительны к регистру символов*, другими словами, объявив переменную `myvariable`, программисту разрешено обращаться к ней, не реагируя на регистр символов, например `MYVARIABLE` или `MyVariable`.

Типы данных

Отправной точкой процесса построения любой системы хранения и обработки данных по праву считается выбор *типа данных*. Вне зависимости от того, какой язык программирования выбран для разработки проекта, первым шагом становится рассмотрение типов данных, имеющихся в распоряжении системы.

В первую очередь тип данных определяет порядок хранения данных в памяти компьютера. Физическая концепция хранения данных зависит от особенностей конкретной платформы, на базе которой функционирует программное обеспечение. В первую очередь это утверждение относится к указателям. Например, в 32-разрядных ОС размер указателя равняется 4 байтам. В 64-битных системах это утверждение ошибочно, здесь для указателя требуется 8 байт.

> ***Внимание!***
>
> В Delphi XE2 появилась возможность создавать 64-разрядные приложения для Windows. Во всех предыдущих версиях Delphi эта опция отсутствовала.

Типизация хранимых значений не просто указывает на размер в байтах, которую должна выделить система для размещения в памяти того или иного значения. Преследуемая цель еще более значима. Типизация определяет, какие операции могут быть осуществлены с теми или иными данными. Delphi не позволит новичку передать результаты деления в целочисленную переменную, ведь в этом случае есть риск потерять дробную часть результата. Delphi станет отчаянно сопротивляться, если мы попробуем просуммировать символьный и вещественный типы данных, пусть даже в символьной переменной хранится числовое зна-

чение. В последнем случае перед проведением математической операции необходимо выполнить преобразование данных.

Программы функционируют в интересах человека, поэтому результаты их работы должны представляться в удобном для оператора виде. Во многом это достигается благодаря концепции типа данных, например при выводе на экран действительных чисел отображается целая и дробная части, разделенные запятой, а при выводе значений, специализирующихся на работе с датой и временем, дата и время форматируются так, как в принято в вашей стране.

Подытожим все вышесказанное. Понятие "тип данных" интегрирует в себе три компонента:

- ограничение множества значений, принадлежащих типу;
- дефиниция[1] набора операций, применяемых к типу;
- определение способа отображения (внешнего представления) значений типа.

Язык Delphi предоставляет в распоряжение программиста весьма широкий спектр *предопределенных типов данных* и, кроме того, разрешает конструировать собственные *пользовательские типы*. Вариант классификации фундаментальных типов данных Delphi представлен на рис. 2.1.

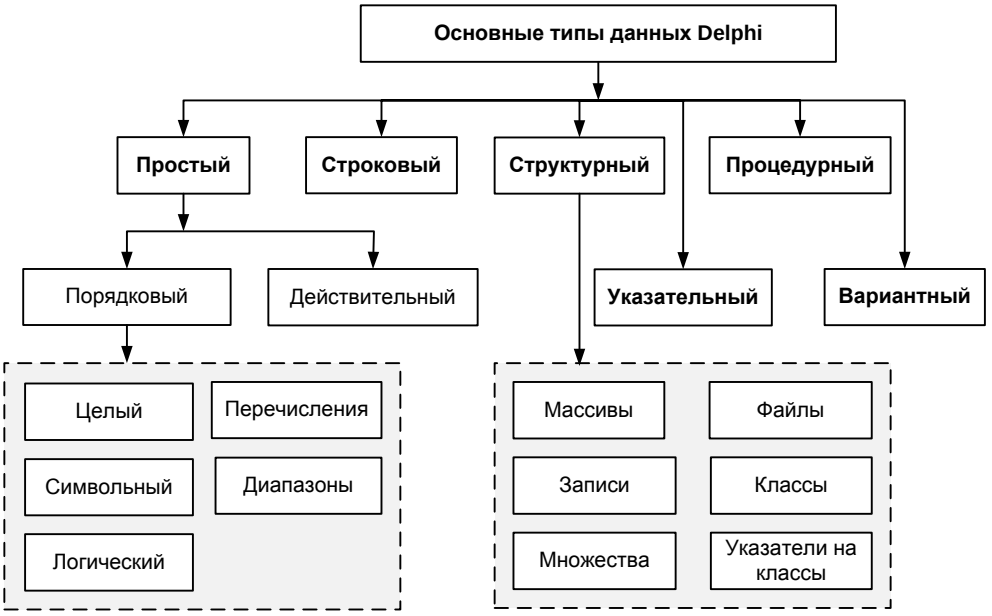

Рис. 2.1. Классификация типов данных Delphi

Все существующие в языке Delphi типы данных можно разделить на шесть ветвей:

- простые;
- строковые;
- структурные;
- указательные;
- процедурные;
- вариантные типы данных.

[1] Краткое определение какого-либо понятия, содержащее наиболее существенные его признаки.

В этой главе мы изучим простые, строковые и указательные типы данных, с оставшимися типами мы познакомимся в других главах книги.

Простые типы

Из представленных на рис. 2.1 типов данных наиболее востребованным является *простой*. Простой тип специализируется на обслуживании числовых данных и упорядоченных последовательностей, тип разделяется на два направления: *порядковые* и *действительные* типы.

Своим названием порядковый тип обязан тому, что между элементами этого типа существуют отношения порядка, другими словами, они могут расположиться в виде упорядоченной последовательности. К еще одной особенности порядковых типов данных стоит отнести то, что все без исключения простые типы по своей сути являются целыми числами.

Целые числа

Характерным представителем порядкового типа считаются целые числа, их характеристики вы найдете в табл. 2.1.

Таблица 2.1. *Целые числа*

Тип	Диапазон значений	Размер (байт)
`Shortint`, `Int8`	–128...127	1
`Smallint`, `Int16`	–32 768...32 767	2
`Longint`, `Integer`, `Int32`	–2 147 483 648...2 147 483 647	4
`Int64`	$-2^{63}...2^{63}-1$	8
`Byte`, `UInt8`	0...255	1
`Word`, `UInt16`	0...65 535	2
`Longword`, `UInt32`	0...4 294 967 295	4
`UInt64`	$0...2^{64}-1$	8

Символьный тип

Второй по счету представитель простых типов — символьный тип данных — специализируется на хранении кода символа. В современных версиях Delphi по умолчанию используется кодировка Unicode. Один символ в такой кодировке может занимать до 4 байт, что позволяет закодировать все символы национальных алфавитов. Вместе с тем, Delphi поддерживает и более старую однобайтовую кодировку ANSI (табл. 2.2).

Таблица 2.2. *Символьный тип данных*

Тип	Кодировка	Размер в байтах
`AnsiChar`	ANSI	1
`WideChar`, `Char`	Unicode	От 1 до 4