В. Зиборов

MS Visual C++ 2010 в среде .NET

- Более сотни примеров и алгоритмов
- Экранная форма и элементы управления
- Программное заполнение веб-форм
- Обработка баз данных SQL Server и MS Access

ПИТЕР

ББК 32.973.233-018.2
УДК 004.658
359

Зиборов В. В.
359 MS Visual C++ 2010 в среде .NET. Библиотека программиста. — СПб.: Питер, 2012. — 320 с.: ил.

ISBN 978-5-459-00786-2

Книга посвящена программированию в среде Visual Studio 2010 на языке программирования C++/CLI. Автор выделил наиболее типичные и актуальные задачи, которые обычно стоят перед программистами, и представил их готовые решения. Разобрано более сотни конкретных примеров и алгоритмов.

Рассмотрены программы с экранной формой и элементами управления в форме. Приведены примеры чтения и записи файлов в долговременную память. Описана работа с графикой и буфером обмена. Приведено несколько подходов к выводу диаграмм. Рассмотрены манипуляции табличными данными, в том числе организация связанных таблиц. Показан принцип использования элемента управления WebBrowser для отображения различных данных, а также для программного заполнения веб-форм. Обсуждены примеры программирования с применением функций объектных библиотек систем MS Excel, MS Word, AutoCAD и MATLAB. Описано создание PDF-файлов. Разобраны вопросы обработки баз данных SQL Server и MS Access с помощью технологии ADO.NET. Представлено много различных авторских оригинальных решений задач программирования, которых читатель не сможет найти в Интернете.

Издание предназначено для начинающих программистов, программистов среднего уровня, а также для программистов, имеющих опыт разработки на других языках и желающих ускоренными темпами освоить новый для себя язык MS Visual C++/CLI.

ББК 32.973.233-018.2
УДК 004.658

ISBN 978-5-459-00786-2

Краткое оглавление

Оглавление

Введение

Система разработки программного обеспечения Microsoft Visual Studio 2010 является продуктом номер один на рынке программного обеспечения. Используя эту систему, можно «малой кровью» и очень быстро написать, почти сконструировать, *как в детском конструкторе*, довольно-таки функционально сложные как настольные приложения (в виде exe-файлов), так и приложения, исполняемые в браузере. В центре системы Visual Studio 2010 находится среда программирования или платформа .NET Framework — это встроенный компонент Windows, который поддерживает создание и выполнение приложений нового поколения и веб-служб. Основными компонентами .NET Framework являются общеязыковая среда выполнения (CLR) и *библиотека классов* .NET Framework, включающая ADO.NET, ASP.NET, Windows Forms и Windows Presentation Foundation (WPF). Платформа .NET Framework предоставляет среду управляемого выполнения, возможности упрощения разработки и развертывания, а также возможности интеграции со многими языками программирования.

Среда разработки программного обеспечения Visual Studio 2010 включает в себя языки программирования Visual Basic, Visual C#, Visual C++ и Visual F#. Используя эти языки программирования, можно подключаться к библиотекам классов и тем самым *иметь все преимущества* ускоренной разработки приложений. Если читатель уже имел опыт разработки на языке C++ различных реализаций (Borland, GNU, Intel), то он может очень быстро освоить MS Visual C++ для среды .NET. Данное подмножество языка C++ еще называют *C++/CLI*. При этом вы будете приятно удивлены тому, что здесь программный код пишется проще, легче читается, а конечный продукт получается очень быстро.

Существенный положительный эффект достигается при групповой разработке какого-либо проекта. Используя Visual Studio, над одним проектом могут работать программисты на C#, на Visual Basic и на C++, при этом среда .NET обеспечит совместимость программных частей, написанных на разных языках.

Целью этой книги является, прежде всего, популяризация программирования. Для достижения этой цели автор выбрал форму демонстрации *на примерах* решения задач от самых простых, элементарных до более сложных.

Так, рассмотрены примеры программ с экранной формой и элементами управления в форме, такими как текстовое поле, метка, кнопка и др. Написанные программы *управляются событиями*, в частности событиями мыши и клавиатуры. Поскольку большинство существующих программ взаимодействуют с дисковой памятью, в книге приведены примеры чтения и записи файлов в долговременную память. Описаны решения *самых типичных задач*, которые встречаются в практике программирования, таких как работа с графикой и буфером обмена. Приведено

несколько подходов к выводу диаграмм (графиков). Рассмотрены манипуляции табличными данными, в том числе организация связанных таблиц. Показан принцип использования элемента управления WebBrowser для отображения различных данных, а также для программного заполнения веб-форм. Обсуждены примеры программирования с применением функций (методов) объектных библиотек систем MS Excel, MS Word, AutoCAD и MATLAB. Представлено несколько выразительных примеров создания PDF-файла. Разобраны вопросы обработки баз данных SQL Server и MS Access с помощью технологии ADO.NET. Рассмотрены методы обработки различных источников данных с использованием технологии LINQ, хотя эту технологию *именно* для *C++/CLI компания* Microsoft разработала еще недостаточно. Представлено много различных авторских оригинальных решений задач программирования, которых читатель не сможет найти в Интернете.

Несколько слов об особенностях книги. Спросите у любого программиста, *как он работает* (творит...) над очередной поставленной ему задачей. Он вам скажет, что всю задачу он мысленно *разбивает на фрагменты* и вспоминает, в каких ранее решенных им задачах он *уже сталкивался с подобной ситуацией*. Далее он просто копирует фрагменты отлаженного программного кода и *вставляет их в новую задачу*. Сборник таких фрагментов (более 100 примеров) содержит данная книга. Автор пытался выделить *наиболее типичные*, актуальные задачи и решить их, с одной стороны, максимально эффективно, а с другой стороны, *кратко* и *выразительно*. С сайта издательства «Питер» (www.piter.com) вы можете скачать архив с рассмотренными в книге примерами.

Самая серьезная проблема, возникающая в процессе создания больших, сложных программ, — это *сложность*, *запутанность текстов*. Из-за запутанности программ появляются ошибки, нестыковки и т. п. Как следствие, страдает производительность процесса создания программ *и их сопровождение*. Решение этой проблемы состоит в *структуризации* программ. Появление объектно-ориентированного программирования связано в большой степени со структуризацией программирования. Мероприятия для обеспечения большей структуризации — это проектирование программы как *иерархической структуры*, отдельные процедуры, входящие в программу, не должны быть *слишком длинными*, не должны использоваться операторы перехода goto и т. п. Кроме того, современные системы программирования разрешают в названиях переменных, методов, свойств, событий, классов, объектов *использовать кириллические символы*. Между тем, современные программисты, как правило, *не используют* данную возможность, хотя появление среди англоязычного текста русских слов *вносит большую выразительность* в текст программы и, как следствие, большую структуризацию. Программный код начинает от этого лучше читаться, *восприниматься человеком* (транслятору, компилятору — все равно).

Данная книга предназначена для начинающих программистов, программистов среднего уровня, а также для программистов, имеющих навыки разработки программ *на других языках и желающих ускоренными темпами освоить новый для себя язык MS Visual C++/CLI*. Как пользоваться этой книгой? Эффективно пользоваться книгой можно, последовательно решая примеры в том в порядке, в котором

они представлены в книге, поскольку примеры расположены *от простого к более сложному*. Это будет способствовать постепенному совершенствованию ваших навыков разработки на данном языке программирования. А для программистов среднего уровня можно посоветовать решать *выборочно* именно те задачи, которые возникли у них при написании их текущих программ. Если вы программируете на C# или Visual Basic, то вам также будет полезна данная книга, поскольку и C#, и Visual Basic «питаются» *все той же средой* .NET, следовательно, названия соответствующих классов, методов, свойств и событий одинаковы, а программные коды различных языков отличаются всего лишь синтаксисом.

Надеюсь, что читатель получит одновременно *интеллектуальное удовольствие* и *прок от использования данной книги в своей работе и творчестве*. Свои впечатления о данной книге присылайте по адресу ziborov@ukr.net, я с удовольствием их почитаю.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На web-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Что такое «хороший стиль программирования»?

Отвечая на этот вопрос одной строчкой, можно сказать, что хороший стиль программирования подразумевает наличие *структуры* у написанной программы. Соответственно, если программа содержит много строк программного кода в одной процедуре, не разделена на смысловые блоки, напоминает «спагетти», то есть имеет множество переходов управления, содержит мало комментариев и, как следствие, выглядит запутанной, громоздкой, *трудно модифицируемой и управляемой* — это это и есть «плохой стиль программирования».

Отсюда появилось понятие структурное программирование. Появилось оно еще в 60-е годы, когда весь мир переживал кризис программного обеспечения. То есть разработка больших автоматизированных систем затягивалась, программисты не укладывалось в сроки, а в готовых программах обнаруживалось множество ошибок. С этими двумя проблемами (низкая производительность разработки программ и ошибки в них) программисты ведут неустанную борьбу. В 1968 году голландский математик-программист Эдсгер Вибе Дейкстра назвал (предположил) причину возникновения проблем: в больших программах зачастую отсутствует четкая логическая структура, они *неоправданно сложны*. Причем эту «неоправданную» сложность вносит команда goto. Программу с множеством goto называют «спагетти». Представьте, что вы читаете роман: сначала две страницы назад, потом четыре страницы вперед и т. д. Если программа содержит много goto, то человеку проследить передачу управления (последовательность управления) весьма затруднительно (компьютер затруднений не испытывает). Вместо операторов goto Дейкстра предложил использовать всего три типа управляющих структур. Первый тип — это простая последовательность (следование), когда операторы выполняются друг за другом слева направо и сверху вниз. Второй тип — это альтернатива (ветвление), выбор по условию (if — else), множественный выбор (switch — case). И третий тип управляющей структуры — это цикл, то есть повторение одного или нескольких операторов до выполнения условия выхода из цикла.

Так вот, основная идея Дейкстры заключается в том, что, используя эти три структуры (отсюда и название «структурное программирование»), можно обходиться без операторов goto. Он утверждал, что отсутствие наглядности — это и есть основной источник ошибок. Сначала программисты лишь посмеивались над идеями Дейкстры, тем более что он был всего лишь теоретиком программирования. Однако смеяться над чем-то новым — это опрометчиво. Один русский прославленный генерал, когда ему впервые показали только что изобретенный пулемет,

снисходительно похлопал изобретателя по плечу и сказал, что если бы в реальном бою нужно было убить одного человека несколько раз, то его изобретение было бы очень кстати. Поэтому не следует спешить высмеивать новое, непонятное и непривычное.

Так случилось и с идеями Дейкстры, фирма IBM весьма успешно применила принципы структурного программирования для создания базы данных газеты «Нью-Йорк таймс», и это стало весомым аргументом в пользу такого подхода. Со временем оператор goto программисты стали называть «позорным» и использовать его лишь в очень крайних случаях. И с тех пор концепция структурного программирования оказывает заметное влияние на теорию и практику программного обеспечения всех рангов.

Сегодня структурное программирование как альтернатива «интуитивному», «рефлекторному» подходу — это методология разработки программного обеспечения, в основе которой лежат не только три типа управляющих структур, но и многое другое, что помогает представлять программу в виде иерархической структуры и тем самым «прятать сложность». Все тело программы стараются делить на логически целостные вычислительные блоки, которые оформляются в виде подпрограмм, функций и классов. Эти вычислительные блоки представляют собой иерархическую структуру, причем наименее значимые «подробности» программы прячут в самые дальние ветви иерархии, чтобы они не мешали пониманию основной логики программы. Пряча те или иные смысловые блоки подальше от глаз, мы тем самым скрываем сложность программы для того, чтобы ее (программы) логика была максимально понятой. Причем «понятность», читабельность нужно обеспечить не только для автора программы, но и для других разработчиков, которые, возможно, будут работать с данной программой. Можно утверждать, что любую самую сложную программу можно сделать доступной для понимания «широким слоям разработчиков», скрывая сложность в смысловых блоках.

Каждый смысловой вычислительный блок должен содержать как можно больше комментариев, а в его начало необходимо вставлять преамбулу, в которой указывать, что этот блок делает и какие технологии и приемы при этом используются. Концепция объектно-ориентированного программирования — это одна из ступеней к большей структуризации программ. Современные системы программирования стараются обеспечить максимальную читабельность наших программ; обратите внимание на отступы и разноцветность программного кода, это придает программам бóльшую выразительность и понятность.

Следует всегда учитывать простой факт: суть программы, которую вы писали неделю назад, очень быстро забывается. А если вы посмотрите на программу, написанную вами месяц назад? Вы удивитесь, но у вас создастся впечатление, что это чужая программа, написанная кем-то другим, но никак не вами. Не стоит удивляться, это особенность человеческого организма — отбрасывать все на его (организма) взгляд ненужное!

В ходе написания программного кода следует давать как можно более точные названия объектам, переменным, процедурам, тогда смысл написанных операторов будет более «прозрачен». Автор в своих примерах старался присваивать соответ-

ствующие названия на русском языке, что обеспечило еще большую структурированность написанных программ. Вообще, следует помнить, что антонимом слова «структура» является «хаос».

Таким образом, большая структуризация программы — это путь повышения производительности разработки, уменьшения количества ошибок, это и есть «хороший стиль программирования».

1 Простейшие программы с экранной формой и элементами управления

Пример 1. Форма, кнопка, метка и диалоговое окно

После инсталляции системы программирования Visual Studio 2010, включающей в себя Visual C++ 2010, загрузочный модуль системы devenv.exe будет, скорее всего, расположен в папке: C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE.

После запуска системы мы увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

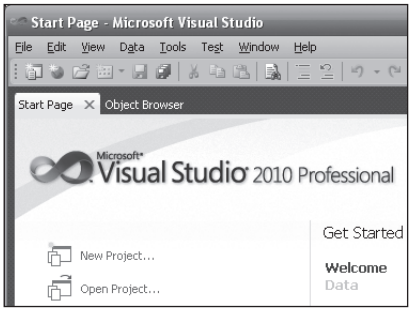

Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio 2010

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню File выполнить команду New Project. В появившемся окне New Project в левой колонке находится список установленных шаблонов (Installed Templates). Среди них — шаблоны языков программирования, встроенных в Visual Studio, в том числе Visual Basic, Visual C#, Visual C++, Visual F# и др. Нам нужен язык Visual C++. В узле Visual C++ области типов проектов выберем среду CLR, а затем в области шаблонов (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++. Теперь

введем имя проекта (Name) First и щелкнем на кнопке OK, в результате увидим окно, представленное на рис. 1.2.

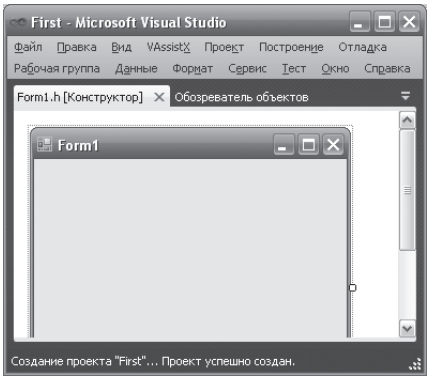

Рис. 1.2. Окно для проектирования пользовательского интерфейса

В этом окне изображена *экранная форма* — Form1, в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, элементы управления. Это поля для ввода текста TextBox, командные кнопки Button, строчки текста в форме — метки Label, которые не могут быть отредактированы пользователем, и прочие элементы управления. Следует отметить, что здесь используется самое современное, так называемое *визуальное программирование*, предполагающее простое перетаскивание элементов с помощью мыши из панели элементов Toolbox, где расположены всевозможные элементы управления, в форму. Это помогает свести к минимуму непосредственное написание программного кода.

Ваша первая программа будет отображать такую экранную форму, в которой будет что-либо написано, например «Microsoft Visual C++ 2010», также в форме будет расположена командная кнопка с надписью «Нажми меня». При нажатии кнопки будет появляться диалоговое окно с сообщением «Всем привет!»

Написать такую программку — вопрос 2–3 минут. Но вначале я хотел бы буквально двумя словами объяснить современный объектно-ориентированный подход к программированию. Подход заключается в том, что в программе все, что может быть названо именем существительным, называют *объектом*. Так в нашей программе мы имеем четыре объекта: форму Form, надпись на форме Label, кнопку Button и диалоговое окно MessageBox с текстом «Всем привет!» (окно с приветом).

Теперь давайте добавим в форму названные элементы управления. Для этого нам понадобится панель элементов управления Toolbox. Если в данный момент вы не видите панель элементов, то ее можно добавить, например, с помощью комбинации клавиш Ctrl+Alt+x или View ▸ Toolbox. Итак, добавьте метку Label и кнопку Button в форму, дважды щелкая на этих элементах на панели Toolbox. А затем следует расположить их примерно так, как показано на рис. 1.3.

Рис. 1.3. Форма первого проекта

Любой такой объект можно создавать самому, а можно воспользоваться готовыми объектами. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно перетаскивать с помощью мыши из панели элементов управления Toolbox. В этой задаче нам следует помнить, что каждый объект имеет свойства (properties). Например, свойствами кнопки являются (рис. 1.4): имя кнопки (Name) — button1, надпись на кнопке (Text), расположение кнопки (Location) в системе координат формы X, Y, размер кнопки Size и т. д. Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду Properties, при этом появится панель свойств Properties (рис. 1.4).

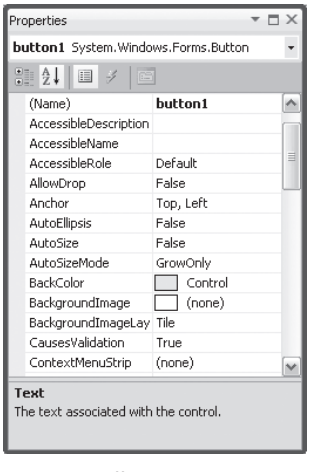

Рис. 1.4. Свойства кнопки button1

Указывая мышью на другие элементы управления в форме, можно просмотреть их свойства: формы Form1 и надписи в форме — метки label1.

Вернемся к нашей задаче. Для объекта label1 выберем свойство Text и напишем напротив этого поля «Microsoft Visual C++ 2010» (вместо текста label1). Для объекта button1 также в свойстве Text напишем «Нажми меня».

Следует помнить, что объекты не только имеют свойства, но и обрабатываются событиями. Событием, например, является щелчок на кнопке, щелчок в пределах формы, загрузка (Load) формы в оперативную память при старте программы и пр. Управляют тем или иным событием посредством написания процедуры обработки события в программном коде. Для этого вначале нужно получить *«пустой» обработчик события*. В нашей задаче единственным событием, которым мы управляем, является щелчок на командной кнопке. Для получения пустого обработчика этого события следует в свойствах кнопки button1 (см. рис. 1.4) щелкнуть на значке молнии Events (события) и в списке всех возможных событий кнопки button1 выбрать двойным щелчком событие Click. После этого мы попадаем на вкладку программного кода Form1.h (см. рис. 1.5).

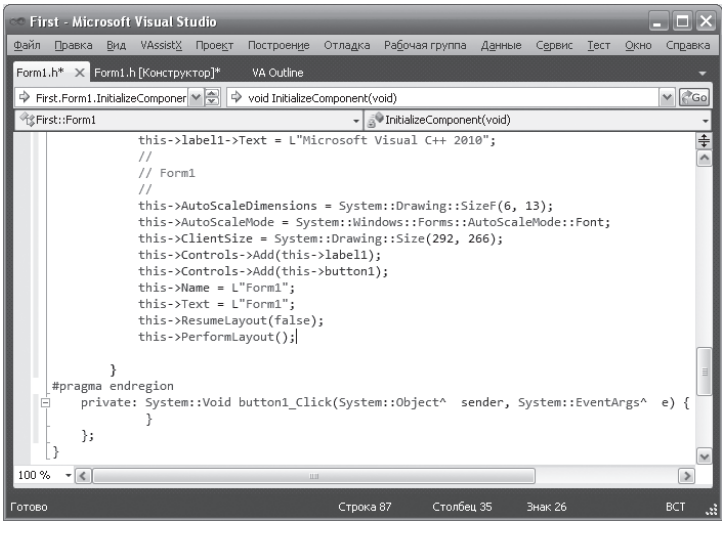

Рис 1.5. Вкладка программного кода

На вкладке Form1.h мы увидим, что управляющая среда Visual C++ 2010 сгенерировала довольно-таки много строк программного кода. Не стоит ужасаться такому обилию непонятного кода, постепенно многое прояснится. Во всяком случае, в этом тексте вы уже можете найти те присваивания, которые сделали в панели свойств Properties. Например, для свойства Text кнопки Button управляющая среда назначила строку «Нажми меня»:

```
this->button1->Text = L"Нажми меня";
```

На рис. 1.5. показан только фрагмент программного кода, причем только его заключительная часть, где мы имеем уже упомянутый нами пустой обработчик события button1_Click:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) { }
```

Здесь в фигурных скобках мы можем написать команды, подлежащие выполнению после щелчка на кнопке. Вы видите, что у нас теперь две вкладки: Form1.h и Form1.h [Design], то есть вкладка программного кода и вкладка визуального проекта программы (другое название этой вкладки — дизайнер формы). Переключаться между ними можно с помощью мыши или с помощью комбинации клавиш Ctrl+Tab, как это принято обычно при переключении между вкладками в Windows, а также функциональной клавишей F7.

Напомню, что по условию задачи после щелчка на кнопке должно появиться диалоговое окно, в котором написано: «Всем привет!» Поэтому в фигурных скобках обработчика события напишем:

```
MessageBox::Show("Всем привет!");
```

Здесь вызывается метод (программа) Show объекта MessageBox с текстом «Всем привет!» Оператор разрешения области действия (::) указывает системе найти метод Show среди методов объекта Messagebox. Таким образом, я здесь «нечаянно» проговорился о том, что объекты кроме свойств имеют также и *методы*, то есть программы, которые обрабатывают объекты. Кстати, после каждого оператора в С-программах ставят точку с запятой. Это вместе с фигурными скобками по форме отличает С-программу от программных кодов на других языках программирования.

Таким образом, мы написали *процедуру обработки события* щелчка (Click) на кнопке button1. Теперь нажмем клавишу F5 и проверим работоспособность программы (рис. 1.6).

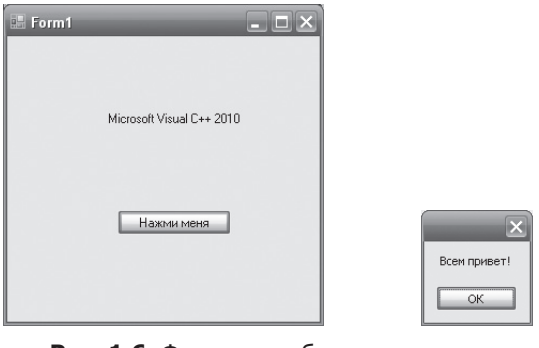

Рис. 1.6. Фрагмент работы программы

Поздравляю, вы написали свою первую программу на MS Visual C++!

Убедиться в работоспособности программы можно, открыв решение First.sln в папке First.

Пример 2. Событие MouseHover

Немного усложним задачу из предыдущего примера. Добавим еще одну обработку события MouseHover мыши для объекта label1. Событие MouseHover наступает тогда, когда пользователь указателем мыши «зависает» над каким-либо объектом,

причем именно «зависает»», а не просто перемещает мышь над объектом (от англ. *hover* — реять, парить). Можно сказать, что событие MouseHover происходит, когда указатель мыши наведен на элемент. Есть еще событие MouseEnter (Войти), когда указатель мыши *входит в пределы области элемента управления (в данном случае метки* label1), но в данном примере будем использовать именно событие MouseHover.

Таким образом, программа в данном примере должна содержать на экранной форме текстовую метку Label и кнопку Button. Метка должна отображать текст «Microsoft Visual C++ 2010»; при щелчке на командной кнопке, на которой попрежнему будет написано «Нажми меня», появится диалоговое окно с сообщением «Всем привет!» Кроме того, когда указатель мыши наведен на текстовую метку (то самое событие MouseHover), должно появиться диалоговое окно с текстом «Событие Hover».

Для решения этой задачи запустим Visual Studio 2010, щелкнем на пункте меню New Project. В появившемся окне New Project в левой колонке в узле Visual C++ выберем среду CLR, а затем в области шаблоны (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем имя Hover и щелкнем на кнопке OK.

В дизайнере формы из панели Toolbox перетащим на форму метку Label и кнопку Button, а затем немного уменьшим размеры формы на свое усмотрение. Теперь добавим три обработчика событий в программный код. Для этого в панели Properties следует щелкнуть на значке молнии (Events) и двойным щелчком последовательно выбрать событие загрузки формы Form_Load, событие «щелчок на кнопке button1_Click» и событие label1_MouseHover.

При этом осуществится переход на вкладку программного кода Form1.h, и среда Visual Studio 2010 сгенерирует три пустых обработчика события. Например, обработчик последнего события будет иметь вид:

```
private:System::Void lab el1_MouseHover(System::Object^ sender,
System::EventArgs^ e){}
```

Между фигурными скобками вставим вызов диалогового окна:

```
MessageBox::Show("Событие Hover!");
```

Теперь проверим возможности программы: нажимаем клавишу F5, «зависаем» указателем мыши над label1, щелкаем на кнопке button1. Все работает!

А теперь я буду немного противоречить сам себе. Я говорил про визуальную технику программирования, направленную на минимизацию написания программного кода. А сейчас хочу сказать про *наглядность*, *оперативность*, *технологичность* работы программиста. Посмотрите на свойства каждого объекта в панели Properties. Как много строчек! Если вы меняете какое-либо свойство, то оно выделяется жирным шрифтом. Удобно! Но еще удобнее свойства объектов *назначать (устанавливать) в программном коде*. Почему?

Каждый программист имеет в своем арсенале множество уже отлаженных фрагментов, которые он использует в своей очередной новой программе. Программисту стоит лишь вспомнить, где он уже программировал ту или иную ситуацию. Программа, которую написал программист, имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали три месяца назад, то поймете, что

многое забыли; если прошел год, то вы смотрите на написанную вами программу, как на чужую. Поэтому при написании программ на первое место выходят *понятность*, *ясность*, *очевидность* написанного программного кода. Для этого каждая система программирования имеет определенные средства. Кроме того, сам программист должен придерживаться некоторых правил, помогающих ему работать *производительно* и *эффективно*.

Назначать свойства объектов в программном коде удобно или сразу после инициализации компонентов формы (после вызова процедуры InitializeComponent), или при обработке события Form1_Load, то есть события загрузки формы в оперативную память при старте программы. Для того чтобы получить простой обработчик этого события, как и в предыдущих случаях, можно выбрать нужное событие в панели свойств объекта, а можно поступить еще проще: дважды щелкнуть в пределах проектируемой формы на вкладке Form1.h [Design]. В любом случае получаем пустой обработчик события на вкладке программного кода. Заметим, что для формы таким умалчиваемым событием, для которого можно получить пустой обработчик двойным щелчком, является событие загрузки формы Form1_Load, для командной кнопки Button и метки Label таким событием является одиночный щелчок мышью на этих элементах управления. То есть если дважды щелкнуть в дизайнере формы по кнопке, то получим пустой обработчик button1_Click в программном коде, аналогично — для метки Label.

Итак, вернемся к событию загрузки формы, для него управляющая среда сгенерировала пустой обработчик:

```
private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e) { }
```

В фигурных скобках обычно помещают свойства различных объектов и даже часто пишут много строчек программного кода. Здесь мы назначим свойству Text объекта label1 значение «Microsoft Visual C++ 2010»:

```
label1->Text = "Microsoft  Visual  C++ 2010";
```

Аналогично для объекта button1:

```
button1->Text = "Нажми меня";
```

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать «la», уже это вызовет появление раскрывающегося меню, где вы сможете выбрать нужные для данного контекста ключевые слова. Это очень мощное и полезное современное средство, называемое IntelliSense (его иногда называют суфлером), для редактирования программного кода! Если вы от Visual Studio 2010 перешли в другую систему программирования, в которой отсутствует подобный сервис, то будете ощущать сильный дискомфорт. Возможно, вам досталась такая версия Visual Studio 2010, в которой функция IntelliSense недоступна, в этом случае система в левом нижнем углу пишет: «Intellisense: Unavailable for C++/CLI» (функция IntelliSense недоступна для C++/CLI). В таком случае я посоветую использовать плагин Visual Assist X для Microsoft Visual Studio разработанный компанией Whole Tomato; привожу адрес их сайта, где этот плагин можно скачать и инсталлировать: http://www.wholetomato.com.

Пользуясь функцией IntelliSense, очень удобно после ввода оператора разрешения области действия (::), оператора-точки (.) или оператора-стрелки (->) получать список допустимых вариантов дальнейшего ввода. Можно выделить элемент и нажать клавишу Tab или Enter или дважды щелкнуть на элементе, чтобы вставить его в код. Как видите, не следует пугаться слишком длинных ключевых слов, длинных названий объектов, свойств, методов, имен переменных. Система подсказок современных систем программирования значительно облегчает всю нетворческую работу. Вот почему в современных программах можно встретить такие длинные имена ключевых слов, переменных и т. п. Я призываю вас, уважаемые читатели, также использовать в своих программах для названий переменных и объектов наиболее ясные, полные имена, причем можно на вашем родном русском языке. Потому что на первое место выходят ясность, прозрачность программирования, а громоздкость названий с лихвой компенсируется системой подсказок.

Далее хотелось бы, чтобы слева вверху формы на синем фоне (в так называемой строке заголовка) была не надпись «Form1», а что-либо осмысленное. Например, слово «Приветствие». Для этого ниже присваиваем эту строку свойству Text формы. Поскольку мы изменяем свойство объекта Form1 внутри подпрограммы обработки события, связанного с формой, следует к форме обращаться или через ссылку this или используя имя объекта Form1:

```
this->Text = "Приветствие"; или  Form1::Text = "Приветствие";
```

После написанная последней строчки кода мы должны увидеть на экране программный код, представленный в листинге 1.1, где приведен лишь фрагмент с нашими корректировками.

Листинг 1.1. Фрагмент файла Form1.h, содержащего программный код с тремя обработчиками событий

```
            // ..............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Данная программа управляется тремя событиями. Событие загрузки формы
        // Form1_Load инициализирует надписи заголовка формы, текстовой метки
        // и кнопки. Событие щелчок на кнопке button1_Click вызывает появление
        // диалогового окна с текстом "Всем привет!". Событие, когда указатель
        // мыши наведен на метку, вызывает появление диалогового окна с текстом
        // "Событие Hover".
    private: System::Void Form1_Load(System::Object^  sender,
                                                      System::EventArgs^  e)
             { // Обработка события загрузки формы:
                 this->Text = "Приветствие";
                 // или Form1::Text = "Приветствие";
```

```
            label1->Text = "Microsoft  Visual  C++ 2010";
            button1->Text = "Нажми меня";
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        { // Обработка события щелчок на кнопке:
            MessageBox::Show("Всем привет!");
        }
    private: System::Void label1_MouseHover(System::Object^  sender,
                                        System::EventArgs^  e)
        { // Обработка события, когда указатель мыши наведен на метку:
            MessageBox::Show("Событие Hover!");
        }
    };
}
```

Комментарии, поясняющие работу программы, в окне редактора кода будут показаны зеленым цветом, что в тексте выделяет их среди прочих элементов программы. В языках C комментарий пишут после двух слэшей (///) или внутри пар /* */. Уважаемые читатели, даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *напишите комментарий*. Как показывает опыт, даже самый очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным. Кроме того, даже если текст программы вполне ясен, в начале программы должны быть описаны ее назначение и способ использования, то есть как бы «преамбула» программы. Далее в последующих примерах мы будем следовать этому правилу.

На рис. 1.7 приведен фрагмент работы программы.

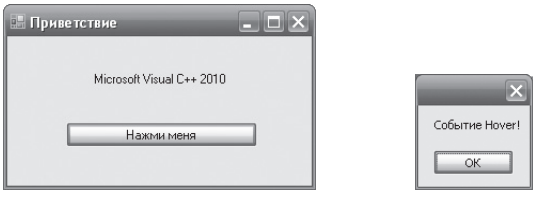

Рис. 1.7. Фрагмент работы программы

Обычно в редакторах программного кода используется моноширинный шрифт, поскольку все символы такого шрифта имеют одинаковую ширину, в том числе и точка, и прописная русская буква «Ш». По умолчанию в редакторе программного кода C++ 2010 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то его настройку можно изменить, выбрав меню Tools ▸ Options ▸ Environment ▸ Fonts and Colors.

Теперь закроем проект (File ▸ Close Project). Система предложит нам сохранить проект, сохраним проект под именем Hover. Теперь программный код этой программы можно посмотреть, открыв решение Hover.sln, в папке Hover.

Пример 3. Выбор нужной даты

Задача состоит том, чтобы, например, при заказе железнодорожных билетов или при регистрации отдыхающего в санатории (ситуаций может быть много) при щелчке на соответствующем элементе управления появлялся календарь, где можно выбрать необходимую дату. Это может быть дата отправления поезда или дата окончания отдыха в санатории или другие ситуации, связанные с выбором нужной даты.

Для решения этой задачи запустим систему программирования Visual Studio 2010, щелкнем на пункте New Project. В появившемся окне New Project в узле Visual C++ выберем среду CLR, а затем в области шаблоны (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем имя ВыборДаты и щелкнем на кнопке OK.

В дизайнере формы из панели Toolbox перетащим на форму командную кнопку Button, метку Label и элемент DateTimePicker. Этот элемент непосредственно выполняет функцию выбора даты. Если щелкнуть на стрелке элемента DateTimePicker, то раскроется календарь для выбора даты, как видно из рис. 1.8. Также мы хотим, чтобы календарь элемента DateTimePicker раскрывался при нажатии кнопки Button. В итоге выбранная дата должна отображаться в текстовой метке Label.

С помощью указателя мыши немного уменьшим размеры экранной формы и расположим элементы управления, как показано на рис. 1.8. Двойной щелчок в пределах формы вызовет переход на вкладку программного кода Form1.h. При этом среда Visual Studio сгенерирует пустой обработчик события загрузки формы Form_Load (листинг 1.2). Таким же способом, то есть двойным щелчком на командной кнопке в дизайнере формы, мы получим пустой обработчик события «щелчок на кнопке». Нам еще понадобится обработчик события ValueChanged. Чтобы его получить, перейдем на вкладку конструктора формы Form1.h[Design] и в панели свойств, щелкнув на значке молнии (Events), для элемента dateTimePicker1 двойным щелчком выберем событие ValueChanged. Это приведет к созданию пустого обработчика этого события на вкладке Form1.h. На вкладке программного кода добавим недостающие команды (см. листинг 1.2).

Листинг 1.2. Фрагмент файла Form1.h, содержащего программный код выбора необходимой даты

```
            // ...............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа выбора нужной даты
    private: System::Void Form1_Load(System::Object^  sender,
                 System::EventArgs^  e)
```

```
            { // Обработка события загрузки формы:
                this->Text = "Средство выбора даты";
                dateTimePicker1->Format = DateTimePickerFormat::Custom;
                dateTimePicker1->CustomFormat = "ddd, dd MMM, yyyy";
                button1->Text = "Выбрать дату:";
                label1->Text = String::Format("Сегодня: {0}",
                                              dateTimePicker1->Text);
            }
    private: System::Void dateTimePicker1_ValueChanged(System::Object^ sender,
                                                       System::EventArgs^ e)
            { // Обработка события изменения даты:
                label1->Text = String::Format("Выбранная дата: {0}",
                    dateTimePicker1->Text);
            }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
            { // Обработка события "щелчок на кнопке"
                // Передаем фокус на элемент управления dateTimePicker1:
                dateTimePicker1->Focus();
                // Имитируем нажатие клавиши <F4>:
                SendKeys::Send("{F4}");
            }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы задается нужный формат отображения даты: вначале (первые три буквы ddd) задаем вывод дня недели в краткой форме; затем (dd MMM) — дня и названия месяца, также в краткой форме, и наконец, вывод года (yyyy). При обработке события изменения даты ValueChanged текстовой метке label1 присваиваем выбранное значение даты. При этом пользуемся методом String::Format. Использованный формат «Выбранная дата: {0}» означает: взять нулевой выводимый элемент, то есть свойство Text объекта dataTimePicker1, и записать его вместо фигурных скобок.

При обработке события «щелчок на кнопке» вначале передаем фокус на элемент управления dateTimePicker1. Теперь можем использовать нажатие клавиши F4 для раскрытия календаря. Функциональная клавиша F4 обеспечивает раскрытие списка в приложениях Windows. Например, в интернет-браузерах, если передать фокус (щелчком мыши) на адресную строку, после нажатия F4 раскроется список уже посещавшихся вами веб-страниц, и вам останется лишь выбрать в этом списке нужный ресурс. В данном случае мы имитируем нажатие F4. Метод Send посылает сообщение активному приложению о нажатии соответствующей клавиши.

Фрагмент работы программы показан на рис. 1.8.

Заметим, что подобный пример использования раскрывающегося календаря приведен в документации Visual Studio на сайте Microsoft www.msdn.com. Автором внесены лишь некоторые изменения.

Убедиться в работоспособности программы можно, открыв решение ВыборДаты.sln в папке ВыборДаты.

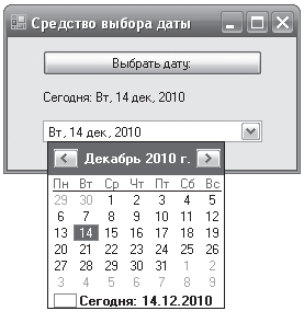

Рис. 1.8. Выбор нужной даты из раскрывающегося календаря

Пример 4. Ввод данных через текстовое поле TextBox с проверкой типа методом TryParse

При работе с формой очень часто ввод данных организуют через элемент управления текстовое поле TextBox. Напишем типичную программу, которая вводит через текстовое поле число, при нажатии командной кнопки извлекает из него квадратный корень и выводит результат на метку Label. В случае ввода не числа сообщает пользователю об этом.

Решая сформулированную задачу, запускаем Visual Studio, выбираем пункт меню File ▸ New ▸ Project. В окне New Project в узле Visual C++ выберем среду CLR, а затем в области шаблоны выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем имя Корень и щелкнем на кнопке OK. Далее из панели элементов управления Toolbox (если в данный момент вы не видите панель элементов управления, то ее можно добавить, например, с помощью комбинации клавиш Ctrl+Alt+x или меню View ▸ Toolbox) в форму с помощью указателя мыши перетаскиваем текстовое поле TextBox, метку Label и командную кнопку Button. Получить названные элементы на проектируемой экранной форме можно, также дважды щелкая указателем мыши на каждом элементе в панели Tools. Таким образом, в форме будут находиться три элемента управления. Расположим их на экранной форме так, как показано на рис. 1.9.

Теперь нам следует изменить некоторые свойства элементов управления. Это уместно сделать после инициализации компонентов формы в программном коде файла Form1.h, то есть сразу после вызова процедуры InitializeComponent. Но это неудобно с точки зрения изложения, нам проще присвоить новые свойства при обработке загрузки формы. Чтобы получить пустой обработчик загрузки формы, дважды щелкнем по проектируемой экранной форме. Сразу после этого мы попадаем на вкладку программного кода Form1.h. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки this), кнопкам button1 и текстового поля textBox1, метке label1 следующие значения:

```
this->Text  = "Извлечение квадратного корня";
button1->Text  = "Извлечь корень";
textBox1->Clear(); //  Очистка текстового поля
label1->Text  = nullptr; // или = String::Empty;
```

Нажмем клавишу F5 для выявления возможных опечаток, то есть синтаксических ошибок и предварительного просмотра дизайна будущей программы.

Далее программируем событие button1_Click — «щелчок мышью на кнопке Извлечь корень». Создать пустой обработчик этого события удобно, дважды щелкнув мышью на этой кнопке. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа Single и непосредственное извлечение корня (листинг 1.3).

Листинг 1.3. Фрагмент программы извлечения корня с проверкой типа методом TryParse

```
            // ...............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа вводит через текстовое поле число, при щелчке на командной
        // кнопке извлекает из него квадратный корень и выводит результат
        // на метку label1. В случае ввода не числа сообщает пользователю об
        // этом, выводя красным цветом предупреждение также на метку label1.
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 button1->Text = "Извлечь корень";
                 label1->Text = String::Empty;
                 // или label1->Text = nullptr;
                 this->Text = "Извлечение квадратного корня";
                 textBox1->Clear(); // Очистка текстового поля
                 textBox1->TabIndex = 0; // Установка фокуса в текстовом поле
             }
    private: System:: // Обработка щелчка на кнопке "Извлечь корень":
                Void button1_Click(System::Object^  sender, System::EventArgs^  e)
             {
                 Single X; // - из этого числа будем извлекать корень
                 // Преобразование из строковой переменной в Single:
                 bool Число_ли = Single::TryParse(textBox1->Text,
                     System::Globalization::NumberStyles::Number,
                     System::Globalization::NumberFormatInfo::CurrentInfo, X);
                 // Второй параметр - это разрешенный стиль числа (Integer,
                 // шестнадцатеричное число, экспоненциальный вид числа и прочее).
                 // Третий параметр форматирует значения на основе текущего языка
                 // и региональных параметров из Панели управления - Язык и
                 // региональные стандарты - число допустимого формата; метод
                 // возвращает значение в переменную X
                 if (Число_ли == false)
                 { // Если пользователь ввел не число:
                     label1->Text = "Следует вводить числа";
```

продолжение ⇩

Листинг 1.3 (продолжение)

```
                    label1->ForeColor = Color::Red; // - цвет текста на метке
                    return; // - выход из процедуры
                }
                Single Y = (Single)Math::Sqrt(X); // - извлечение корня
                label1->ForeColor = Color::Black; // - черный цвет текста
                на метке
                label1->Text = String::Format("Корень из {0} равен {1:F5}", X, Y);
            }
    };
}
```

Здесь при обработке события «щелчок мышью на кнопке Извлечь корень» проверяется, введено ли число в текстовом поле. Проверка осуществляется с помощью функции TryParse. Первым параметром метода TryParse является анализируемое поле textBox1->Text. Второй параметр — это разрешаемый для преобразования стиль числа, он может быть целого типа (Integer), шестнадцатеричным (HexNumber), представленным в экспоненциальном виде и пр. Третий параметр указывает, на какой основе формируется допустимый формат, в нашем случае мы использовали CurrentInfo, то есть на основе текущего языка и региональных параметров. По умолчанию при инсталляции русифицированной версии Windows XP разделителем целой и дробной части числа является запятая. Однако эту установку можно изменить, если в Панели управления выбрать значок Язык и региональные стандарты, затем на вкладке Региональные параметры щелкнуть на кнопке Настройка и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей точку вместо запятой. В обоих случаях (и для запятой, и для точки) метод TryParse будет работать так, как указано на вкладке Региональные параметры.

Четвертый параметр метода TryParse возвращает результат преобразования. Кроме того, функция TryParse возвращает булеву переменную true или false, которая сообщает, успешно ли выполнено преобразование. Как видно из текста программы, если пользователь ввел не число (например, введены буквы), то на метку label1 выводится красным цветом текст: «Следует вводить числа». Далее, поскольку ввод неправильный, организован выход из программы обработки события button1_Click с помощью оператора return. На рис. 1.9 показан фрагмент работы программы.

Как видно из рисунка, функция TryParse не восприняла введенные символы «2,3» как число, поскольку автор специально для демонстрации данного примера указал на вкладке Региональные параметры точку в качестве разделителя целой и дробной частей.

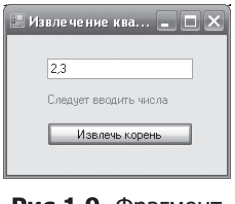

Рис 1.9. Фрагмент работы программы

Если пользователь ввел все-таки число, то будет выполняться следующий оператор извлечения квадратного корня Math::Sqrt(X). Математические функции Visual Studio 2010 являются методами класса Math. Их можно увидеть, набрав Math и введя так называемый *оператор разрешения области действия* (::). В раскрывающемся списке вы увидите множество математических функций: Abs, Sin, Cos, Min и т. д. и два свойства — две константы E = 2,718... (основание на-

туральных логарифмов) и PI = 3,14... (число диаметров, уложенных вдоль окружности). Функция Math::Sqrt(X)возвращает значение типа double (двойной точности с плавающей запятой), которое мы *приводим* с помощью неявного преобразования (Single) к переменной одинарной точности.

Последней строчкой обработки события button1_Click является формирование строки label1->Text с использованием метода String::Format. Использованный формат «Корень из {0} равен {1:F5}» означает: взять нулевой выводимый элемент, то есть переменную X, и записать эту переменную вместо фигурных скобок; после чего взять первый выводимый элемент, то есть Y, и записать его вместо вторых фигурных скобок в формате с фиксированной точкой и пятью десятичными знаками после запятой.

Нажав клавишу F5, проверяем, как работает программа.

Результат работающей программы представлен на рис. 1.10.

Если появились ошибки, то работу программы следует проверить отладчиком — клавиши F10 или F11. В этом случае управление останавливается на каждом операторе, и вы можете проверить значение каждой переменной, наводя указатель мыши на переменные. Можно выполнить программу до определенной программистом точки (*точки останова*), используя, например, клавишу F9 или оператор Stop, и в этой точке проверить значения необходимых переменных.

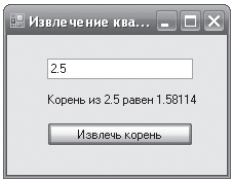

Рис. 1.10. Извлечение квадратного корня

Убедиться в работоспособности этой программы можно, открыв соответствующее решение в папке Корень.

Пример 5. Ввод пароля в текстовое поле и изменение шрифта

Это очень маленькая программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, «находящийся за спиной пользователя», увидит только звездочки. Программа состоит из формы, текстового поля TextBox, метки Label, куда для демонстрации возможностей мы будет копировать пароль (паспорт, то есть секретные слова), и командной кнопки Button — Покажи паспорт.

После запуска Visual Studio и выбора приложения в среде CLR шаблона Windows Forms Application Visual C++ перемещаем в форму все названные элементы управления. Текст программы приведен в листинге 1.4.

Листинг 1.4. Фрагмент программы, организующей ввод пароля

```
        // ..............
        // Программный код, расположенный выше, создан средой  Visual  Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
        this->PerformLayout();
    }
```

продолжение ⇩

Листинг 1.4 (продолжение)

```
#pragma endregion
        // Программа для ввода пароля в текстовое поле, причем при вводе вместо
        // вводимых символов некто, "находящийся за спиной пользователя", увидит
        // только звездочки
    private: System:: // Обработка события "загрузка формы":
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 Form1::Text = "Введи пароль";
                 textBox1->Text = nullptr; textBox1->TabIndex = 0;
                 textBox1->PasswordChar = '*';
                 textBox1->Font = gcnew System::Drawing::Font("Courier New",
                 9.0F);
                 // или textBox1->Font = gcnew System::Drawing::Font(FontFamily::
                 //                                        GenericMonospace,9.0F);
                 label1->Text = String::Empty;
                 label1->Font = gcnew System::Drawing::Font("Courier New", 9.0F);
                 button1->Text = "Покажи паспорт";
             }
    private: System:: // Обработка события "щелчок на кнопке":
                Void button1_Click(System::Object^  sender, System::EventArgs^  e)
             {
                 label1->Text = textBox1->Text;
             }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы мы очищаем текстовое поле и делаем его «защищенным от посторонних глаз» с помощью свойства textBox1->PasswordChar; каждый введенный пользователем символ маскируется символом звездочки (*). Далее мы хотели бы для большей выразительности и читабельности программы сделать так, чтобы вводимые звездочки и результирующий текст имели одинаковую длину. Все символы шрифта Courier New имеют одинаковую ширину, поэтому его называют моноширинным шрифтом. Кстати, используя именно этот шрифт, благодаря одинаковой ширине букв удобно программировать таблицу. Еще одним широко используемым моноширинным шрифтом является шрифт Consola. Задаем шрифт, используя свойство Font обоих объектов: textBox1 и label1. Число 9.0 означает размер шрифта. Свойство текстового поля TabIndex = 0 обеспечивает передачу фокуса при старте программы именно в текстовое поле.

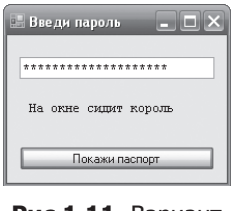

Рис 1.11. Вариант работы программы

Осталось обработать событие button1_Click — щелчок на кнопке. Здесь — банальное присваивание текста из поля тексту метки. Программа написана, нажимаем клавишу F5. На рис. 1.11 приведен вариант работы данной программы.

При необходимости используйте отладчик (клавиша F11 или F10) для *пошагового выполнения программы* и выяснения всех промежуточных значений переменных путем «зависания» указателя мыши над переменными.

Убедиться в работоспособности программы можно, открыв решение Паспорт.sln в папке Паспорт.

Пример 6. Управление стилем шрифта с помощью элемента управления CheckBox

Кнопка CheckBox (Флажок) также находится на панели элементов управления Toolbox. Флажок может быть *либо установлен (содержит* «галочку»), либо сброшен (пустой). Напишем программу, которая управляет стилем шрифта текста, выведенного на метку Label. Управлять стилем будем посредством флажка CheckBox.

Запустим Visual Studio 2010 и выберем приложение в среде CLR шаблона Windows Forms Application Visual C++. Используя панель элементов Toolbox, в форму поместим метку Label и флажок CheckBox. В листинге 1.5 приведен текст программы управления этими объектами.

Листинг 1.5. Фрагмент программы, управляющей стилем шрифта

```
            // ...............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа управляет стилем шрифта текста, выведенного на метку Label,
        // посредством флажка CheckBox
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Флажок CheckBox";
                checkBox1->Text = "Полужирный"; checkBox1->Focus();
                label1->Text = "Выбери стиль шрифта";
                label1->TextAlign = ContentAlignment::MiddleCenter;
                label1->Font = gcnew System::Drawing::Font("Courier New",
                14.0F);
             }
    private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
                                                   System::EventArgs^  e)
             { // Изменение состояния флажка на противоположное
                if (checkBox1->Checked == true) label1->Font = gcnew System::
                            Drawing::Font("Courier New", 14.0F,
                            FontStyle::Bold);
                if (checkBox1->Checked == false) label1->Font = gcnew System::
                          Drawing::Font("Courier New", 14.0F,
                          FontStyle::Regular);
             }
    };
}
```

При обработке события загрузки формы задаем начальные значения некоторых свойств объектов Form1 (посредством ссылки this), label1 и checkBox1. Так, тексту флажка, выводимого с правой стороны, присваиваем значение «Полужирный». Кроме того, при старте программы фокус должен находиться на флажке (checkBox1.Focus();), в этом случае пользователь может изменять установку флажка даже клавишей Пробел.

Текст метки — «Выбери стиль шрифта», выравнивание метки TextAlign задаем посередине и по центру (MiddleCenter) относительно всего того места, которое предназначено для метки. Задаем шрифт метки Courier New (в этом шрифте все буквы имеют одинаковую ширину) размером 14 пунктов.

Изменение состояния флажка соответствует событию CheckedChanged. Чтобы получить пустой обработчик события CheckedChanged, следует дважды щелкнуть на элементе checkBox1 вкладки Form1.h [Design]. Между соответствующими строчками следует записать (см. текст программы): если флажок установлен (то есть содержит «галочку») Checked = true, то для метки label1 устанавливается тот же шрифт Courier New, 14 пунктов, но Bold, то есть полужирный.

Далее — следующая строчка кода: если флажок не установлен, то есть checkBox1.Checked = false, то шрифт устанавливается Regular, то есть обычный. Очень часто эту ситуацию программируют, используя ключевое слово else (иначе), однако это выражение будет выглядеть более выразительно и понятно так, как написали его мы.

Программа написана, нажмите клавишу F5. Проверьте работоспособность программы. В рабочем состоянии она должна работать примерно так, как показано на рис. 1.12.

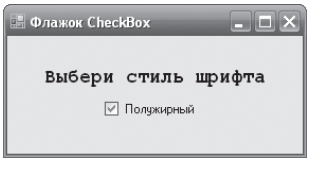

Рис. 1.12. Фрагмент работы программы управления стилем шрифта

Убедиться в работоспособности программы можно, открыв решение Флажок1.sln в папке Флажок1.

Пример 7. Побитовый оператор «исключающее ИЛИ»

Несколько изменим предыдущую программу в части обработки события CheckedChanged (изменение состояния флажка). Вместо двух условий if() напишем один оператор:

```
label1->Font = gcnew System::Drawing::Font(
          "Courier New", 14.0F, label1->Font->Style ^ FontStyle::Bold);
```

Здесь каждый раз при изменении состояния флажка значение параметра label1->Font->Style сравнивается с одним и тем же значением FontStyle::Bold. Поскольку между ними стоит побитовый оператор (^) (исключающее ИЛИ), он будет назначать «Bold», если текущее состояние label1->Font->Style «не Bold». А если label1->Font->Style пребывает в состоянии «Bold», то оператор (^) будет назначать состояние «не Bold». Этот оператор еще называют логическим XOR.

Таблица истинности логического XOR такова:

```
A Xor B = C
0 Xor 0 = 0
1 Xor 0 = 1
0 Xor 1 = 1
1 Xor 1 = 0
```

В нашем случае мы имеем всегда B = 1 (FontStyle::Bold), а A (label1->Font->Style) попеременно то Bold, то Regular (то есть «не Bold»). Таким образом, оператор (^) всегда будет назначать противоположное тому, что записано в label1->Font->Style.

Как видно, применение побитового оператора привело к существенному уменьшению количества программного кода. Использование побитовых операторов может значительно упростить написание программ со сложной логикой.

Посмотрите, как работает программа, нажав клавишу F5.

Теперь добавим в форму еще один элемент управления CheckBox. Мы собираемся управлять стилем шрифта FontStyle двумя флажками. Один, как и прежде, задает полужирный стиль Bold или обычный Regular, а второй задает наклонный Italic или возвращает в Regular. Фрагмент новой программы приведен в листинге 1.6.

Листинг 1.6. Усовершенствованный программный код

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа использует побитовый оператор "^" - исключающее ИЛИ
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Флажок CheckBox";
                checkBox1->Text = "Полужирный"; checkBox2->Text = "Наклонный";
                label1->Text = "Выбери стиль шрифта";
                label1->TextAlign = ContentAlignment::MiddleCenter;
                label1->Font = gcnew System::Drawing::Font("Courier New",
                14.0F);
             }
    private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
                                                   System::EventArgs^  e)
```

продолжение ⇗

Листинг 1.6 (продолжение)

```
            {
                label1->Font = gcnew System::Drawing::Font(
                    "Courier New", 14.0F, label1->Font->Style ^
                    FontStyle::Bold);
            }
    private: System::Void checkBox2_CheckedChanged(System::Object^  sender,
                                                   System::EventArgs^  e)
            {
                label1->Font = gcnew System::Drawing::Font(
                   "Courier New", 14.0F, label1->Font->Style ^
                   FontStyle::Italic);
            }
    };
}
```

Как вы видите, здесь ничего принципиально нового нет, только лишь добавлена обработка события изменения состояния флажка CheckedChanged для CheckBox2. Фрагмент работы программы можно увидеть на рис. 1.13.

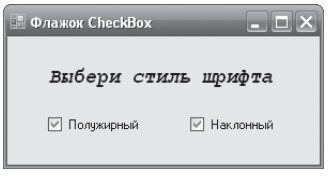

Рис. 1.13. Фрагмент работы усовершенствованной программы

Убедиться в работоспособности программы можно, открыв решение Флажок2.sln в папке Флажок2.

Пример 8. Вкладки TabControl и переключатели RadioButton

Вкладки программируют для организации управления и оптимального использования экранного пространства. Выразительным примером использования вкладок является диалоговое окно Internet Explorer Свойства обозревателя. То есть если требуется отобразить большое количество управляемой информации, то весьма уместно использовать вкладки TabControl.

Поставим задачу написать программу, позволяющую выбрать текст из двух вариантов, задать цвет и размер шрифта этого текста на трех вкладках TabControl с использованием переключателей RadioButton. Фрагмент работы программы приведен на рис. 1.14.

Программируя поставленную задачу, запустим Visual Studio и выберем приложение в среде CLR шаблона Windows Forms Application Visual C++. Назовем этот проект Вкладки. Используя панель элементов Toolbox, в форму перетащим с помощью мыши

элемент управления TabControl. Как видно, по умолчанию имеем две вкладки, а по условию задачи, как показано на рис. 1.14, три вкладки. Добавить третью вкладку можно в конструкторе формы, а можно программно.

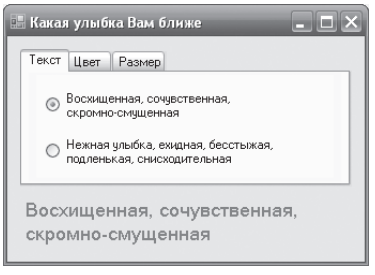

Рис. 1.14. Программа с переключателями и вкладками

Рассмотрим, как добавить третью вкладку в конструкторе. Для этого в свойствах (окно Properties) элемента управления tabControl1 выбираем свойство TabPages, в результате попадаем в диалоговое окно TabPage Collection Edit, где добавляем (кнопка Add) третью вкладку (первые две присутствуют по умолчанию). Эти вкладки нумеруются от нуля, то есть третья вкладка будет распознаваться как TabPages(2). Название каждой вкладки будем указывать в программном коде.

Для того чтобы вы могли в большей степени управлять всем процессом, рассмотрим, как добавить третью вкладку не в конструкторе, а в программном коде при обработке события загрузки формы (листинг 1.7). Однако прежде чем перейти на вкладку программного кода, для каждой вкладки выбираем из панели Toolbox по два переключателя RadioButton, а в форму перетаскиваем метку Label. Теперь с помощью щелчка правой кнопкой мыши в пределах формы переключаемся на редактирование программного кода.

Листинг 1.7. Фрагмент программы, управляющей вкладками и переключателями

```
            // ..............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа, позволяющая выбрать текст из двух вариантов, задать цвет
        // и размер шрифта для этого текста на трех вкладках TabControl
        // с использованием переключателей RadioButton
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 // Создание третьей вкладки "программно":
                 auto tabPage3 = gcnew System::Windows::Forms::TabPage();
                 tabPage3->UseVisualStyleBackColor = true;
```

продолжение ⇨

Листинг 1.7 (продолжение)

```
            // Добавление третьей вкладки в существующий набор
            // вкладок tabControl1:
            this->tabControl1->Controls->Add(tabPage3);
            // Добавление переключателей 5 и 6 на третью вкладку:
            tabPage3->Controls->Add(this->radioButton5);
            tabPage3->Controls->Add(this->radioButton6);
            // Расположение переключателей 5 и 6:
            this->radioButton5->Location = System::Drawing::Point(20, 15);
            this->radioButton6->Location = System::Drawing::Point(20, 58);
            this->Text = "Какая улыбка вам ближе";
            // Задаем названия вкладок:
            tabControl1->TabPages[0]->Text = "Текст";
            tabControl1->TabPages[1]->Text = "Цвет";
            tabControl1->TabPages[2]->Text = "Размер";
            // Эта пара переключателей изменяет текст:
            radioButton1->Text =
                "Восхищенная, сочувственная,\nскромно-смущенная";
            radioButton2->Text = "Нежная улыбка, ехидная, бес" +
                "стыжая,\nподленькая, снисходительная";
            // или
            // radioButton2->Text = "Нежная улыбка, бесстыжая," +
            // Environment::NewLine + "подленькая, снисходительная";
            // Эта пара переключателей изменяет цвет текста:
            radioButton3->Text = "Красный";
            radioButton4->Text = "Синий";
            // Эта пара переключателей изменяет размет шрифта:
            radioButton5->Text = "11 пунктов";
            radioButton6->Text = "13 пунктов";
            label1->Text = radioButton1->Text;
        }
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
        {  label1->Text = radioButton1->Text;  }
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
        {  label1->Text = radioButton2->Text;  }
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
        {  label1->ForeColor = Color::Red;  }
private: System::Void radioButton4_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
        {  label1->ForeColor = Color::Blue;  }
private: System::Void radioButton5_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
        {  label1->Font = gcnew System::Drawing::
                                        Font(label1->Font->Name, 11);  }
private: System::Void radioButton6_CheckedChanged(System::Object^  sender,
                                                  System::EventArgs^  e)
```

```
            {  label1->Font = gcnew System::Drawing::
                                        Font(label1->Font->Name, 13);  }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы Form1_Load (этот участок программного кода можно было бы задать сразу после вызова процедуры InitializeComponent) создаем «программно» третью вкладку. Заметьте, что мы ее объявили как auto, то есть тип переменной tabPage3 выводится из выражения инициализации (как объявление var в C#) — это новая возможность в Visual C++ 2010. Далее добавляем новую вкладку tabPage3 в набор вкладок tabControl1, созданный в конструкторе. Затем «привязываем» пятый и шестой переключатели к третьей вкладке.

Дальнейшие установки очевидны и не требуют дополнительных комментариев. Заметим, что каждая пара переключателей, расположенных на каком-либо элементе управления (в данном случае на различных вкладках), «отрицают» друг друга, то есть если пользователь выбрал один, то другой переходит в противоположное состояние. Отслеживать изменения состояния переключателей удобно с помощью обработки событий переключателей CheckChanged (см. листинг 1.7). Чтобы получить пустой обработчик этого события в конструкторе формы, следует дважды щелкнуть на соответствующем переключателе и таким образом запрограммировать изменения состояния переключателей.

Убедиться в работоспособности программы можно, открыв решение Вкладки.sln в папке Вкладки.

Пример 9. Свойство Visible и всплывающая подсказка ToolTip в стиле Balloon

Продемонстрируем возможности свойства Visible (Видимый). Программа пишет в метку Label некоторый текст, а пользователь с помощью командной кнопки делает этот текст невидимым, а затем опять видимым и т. д. При зависании мыши над кнопкой появляется подсказка «Нажми меня».

Для программирования этой задачи запускаем Visual Studio 2010, далее, щелкая мышью на пунктах меню File ▸ New ▸ Project, выберем приложение в среде CLR шаблона Windows Forms Application Visual C++. Назовем этот проект Visible. Затем из панели элементов управления Toolbox в форму перетаскиваем метку Label, кнопку Button и всплывающую подсказку ToolTip. Только в этом случае каждый элемент управления в форме (включая форму) получает свойство ToolTip on Tip. Убедитесь в этом, посмотрев свойства (окно Properties) элементов управления.

Для кнопки button1 напротив свойства ToolTip on Tip мы могли бы написать «Нажми меня». Однако я предлагаю написать это непосредственно в программном коде. В этом случае программист не будет долго искать *соответствующее свойство*, когда станет применять данный фрагмент в своей новой программе!

Перейдем на вкладку программного кода — щелчок правой кнопкой мыши в пределах формы и выбор команды View Code. Окончательный текст программы представлен в листинге 1.8.

Листинг 1.8. Свойство Visible и всплывающая подсказка ToolTip

```
            // ..............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа пишет в метку Label некоторый текст, а пользователь
        // с помощью командной кнопки делает этот текст либо видимым, либо
        // невидимым. Здесь использовано свойство Visible. При зависании мыши
        // над кнопкой появляется подсказка "Нажми меня" в стиле Balloon
    private: System:: // Обработка события "загрузка формы":
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 Form1::Text = "Житейская мудрость";
                 label1->Text = "Сколько ребенка не учи хорошим манерам,\n" +
                                "он будет поступать так, как папа с мамой";
                 label1->TextAlign = ContentAlignment::MiddleCenter;
                 button1->Text = "Кнопка";
                 toolTip1->SetToolTip(button1, "Кнопка\r\nсчастья");
                 // Должна ли всплывающая подсказка использовать всплывающее
                 // окно:
                 toolTip1->IsBalloon = true;
                 // Если IsBalloon = false, то используется стандартное
                 // прямоугольное окно
             }
    private: System:: // Обработка события "щелчок на кнопке":
                Void button1_Click(System::Object^  sender, System::EventArgs^  e)
             {
                 // Можно программировать так:
                 // if (label1->Visible == true) label1->Visible = false;
                 // else label1->Visible = true;
                 // или так:
                 // label1->Visible = label1->Visible ^ true;
                 // здесь "^" - логическое исключающее ИЛИ,
                 // или совсем просто:
                 label1->Visible = !label1->Visible;
             }
    };
}
```

При обработке события загрузки формы свойству Text метки присваиваем некоторый текст, «склеивая» его с помощью знака «плюс» (+) из отдельных фрагментов. Использование в текстовой строке символов «\n» (или «\r\n») означает перенос

текста на новую строку[1] (это так называемый *перевод строки*). Можно переводить строку с помощью конструкции Environment::NewLine. В перечислении Environment можно выбрать и другие управляющие символы. Свойство метки TextAlign располагает текст метки по центру и посередине (MiddleCenter). Выражение, содержащее toolTip1, устанавливает (Set) текст всплывающей подсказки для кнопки button1 при «зависании» над ней указателя мыши (см. рис. 1.14). По умолчанию свойство IsBalloon пребывает в состоянии false, и при этом во всплывающей подсказке используется стандартное прямоугольное окно, установка IsBalloon = true переводит подсказку в стиль комиксов (в стиль Balloon) (рис. 1.15).

Чтобы в программном коде получить пустой обработчик события «щелчок мышью на кнопке», следует в дизайнере (конструкторе) формы (то есть на вкладке Form1.cs[Designer]) дважды щелкнуть на кнопке button1. При обработке этого события, как видно, закомментированы пять строчек, в которых записана логика включения видимости метки или ее выключение. Логика абсолютно понятна: если свойство видимости (Visible) *включено* (true), то его следует *выключить* (false); иначе (else) — включить.

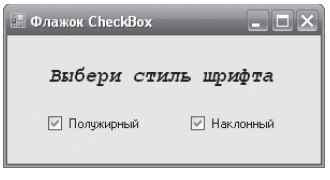

Рис. 1.15. Фрагмент работы программы

Несколько путано, но разобраться можно. И все работает. Проверьте! Кнопку можно нажимать мышью, клавишей Enter и клавишей Пробел.

Однако можно пойти другим путем. Именно поэтому пять строчек этой сложной логики переведены в комментарий. Мы уже встречались с побитовым оператором ^ (исключающее ИЛИ). Напоминаю, что этот оператор, говоря кратко, выбирает «да» (true), сравнивая «нет» и «да», и выбирает «нет» (false), сравнивая «да» и «да». Однако можно еще более упростить написание программного кода:

```
label1->Visible  = ! label1->Visible;
```

То есть при очередной передаче управления на эту строчку свойство label1->Visible будет принимать противоположное значение. Вы убедились, что можно по-разному программировать подобные ситуации.

---

[1] Строго говоря, *перевод строки* осуществляется управляющей последовательностью "\n", а символ "\r" осуществляет *возврат каретки* без перевода строки. На большинстве терминалов перевод строки автоматически влечет за собой и возврат каретки, поэтому обычно достаточно использовать только последовательность "\n". Но надежнее для перехода на новую строку использовать последовательность "\r\n" Использование последовательности "\r" приведет к возврату курсора в начало строки без перехода на новую строку (но опять же не на всех терминалах). — *Примеч. ред.*

Как видно, мы использовали много закомментированных строчек программного кода. Очень удобно комментировать строки программного кода в редакторе Visual Studio 2010, вначале выделяя их, а затем использовать комбинацию клавиш Ctrl+K ▸ C (Comment). Чтобы убрать с нескольких строк знак комментария, можно точно так же вначале отметить их, а затем воспользоваться другой комбинацией клавиш Ctrl+K ▸ U (Uncomment).

Текст программы можно посмотреть, открыв решение Visible.sln в папке Visible.

Пример 10. Калькулятор на основе комбинированного списка ComboBox

Элемент управления ComboBox служит для отображения вариантов выбора в раскрывающемся списке. Продемонстрируем работу этого элемента управления на примере программы, реализующей функции калькулятора. Здесь для отображения вариантов выбора арифметических операций используется комбинированный список ComboBox.

Запустим Visual Studio 2010, выберем приложение в среде CLR шаблона Windows Forms Application Visual C++. Назовем этот проект Комби. Из панели Toolbox перетащим в форму два текстовых поля TextBox, кнопку Button, метку Label и комбинированный список ComboBox.

Текст программы представлен в листинге 1.9.

Листинг 1.9. Суперкалькулятор

```
            // ..............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа, реализующая функции калькулятора. Здесь для отображения
        // вариантов выбора арифметических действий используется комбинированный
        // список ComboBox
    private: System:: // Обработка события загрузки формы:
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Суперкалькулятор"; label1->Text = "Равно: ";
                button1->Text = "Выбери операцию";
                comboBox1->Text = "Выбери операцию";
                // Заполение (инициализация) раскрывающегося списка:
                array<String^>^ Операции = {"Прибавить", "Отнять",
                    "Умножить", "Разделить", "Очистить"};
                // Инициализировать массив можно также таким образом:
                // auto Операции = gcnew array<String^>{"Прибавить", "Отнять",
```

```
            //                          "Умножить", "Разделить", "Очистить"};
            comboBox1->Items->AddRange(Операции);
            comboBox1->TabIndex = 2;
            textBox1->Clear(); textBox1->TabIndex = 0;
            textBox2->Clear(); textBox2->TabIndex = 1;
        }
    private: System::Void comboBox1_SelectedIndexChanged(System::Object^ sender,
                                                         System::EventArgs^  e)
        { // Обработка события изменения индекса выбранного элемента
            label1->Text = "Равно: ";
            // Преобразование из строковой переменной в Single:
            Single X, Y, Z = 0;
            bool Число_ли1 = Single::TryParse(textBox1->Text,
                System::Globalization::NumberStyles::Number,
                System::Globalization::NumberFormatInfo::CurrentInfo, X);
            bool Число_ли2 = Single::TryParse(textBox2->Text,
                System::Globalization::NumberStyles::Number,
                System::Globalization::NumberFormatInfo::CurrentInfo, Y);
            if (Число_ли1 == false || Число_ли2 == false)
            {
                MessageBox::Show("Следует вводить числа!", "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Error);
                return;
            }
            // Оператор множественного выбора:
            switch (comboBox1->SelectedIndex)
            { // Выбор арифметической операции:
            case 0:   // Выбрали "Прибавить":
                Z = X + Y; break;
            case 1:   // Выбрали "Отнять":
                Z = X - Y; break;
            case 2:   // Выбрали "Умножить":
                Z = X * Y; break;
            case 3:   // Выбрали "Разделить":
                Z = X / Y; break;
            case 4:   // Выбрали "Очистить":
                textBox1->Clear(); textBox2->Clear();
                label1->Text = "Равно: "; return;
            }
            label1->Text = String::Format("Равно {0:F5}", Z);
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        { // Обработка события "щелчок на кнопке":
            // "Принудительно" раскрываем список:
            comboBox1->DroppedDown = true;
        }
    };
}
```

В данной программе сразу при обработке события загрузки формы присваиваем начальные значения некоторым свойствам, в том числе задаем коллекцию элементов комбинированного списка: «Прибавить», «Отнять» и т. д. Здесь также задаем табличные индексы TabIndex для текстовых полей и комбинированного списка. Табличный индекс определяет *порядок обхода элементов*. Так, при старте программы фокус будет находиться в первом текстовом поле, поскольку мы назначили textBox1->TabIndex = 0. Далее при нажатии пользователем клавиши Tab будет происходить переход от элемента к элементу соответственно табличным индексам (рис. 1.16).

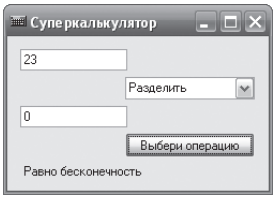

Рис. 1.16. Переход от одного текстового поля к другому

При обработке события «изменение индекса выбранного элемента» comboBox1_SelectedIndexChanged с помощью функции TryParse проверяем, можно ли текстовые поля преобразовать в число. Первым параметром метода TryParse является анализируемое поле. Второй параметр — это разрешаемый для преобразования стиль числа, в данном случае тип Number, то есть десятичное число, которое имеет целую и дробную части. Третий параметр указывает, на какой основе формируется допустимый формат, в нашем случае мы использовали CurrentInfo, то есть на основе текущего языка и региональных параметров. По умолчанию при инсталляции русифицированной версии Windows разделителем целой и дробной частей числа является запятая. Однако эту установку можно изменить, если в Панели управления выбрать значок Язык и региональные стандарты, а затем на вкладке Региональные параметры щелкнуть на кнопке Настройка и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей точку вместо запятой. В обоих случаях (и для запятой, и для точки) метод TryParse будет работать так, как указано на вкладке Региональные параметры.

Четвертый параметр метода TryParse возвращает результат преобразования. Кроме того, функция TryParse возвращает булеву переменную true или false, которая сообщает, успешно ли выполнено преобразование. Как видно из текста программы, если хотя бы одно поле невозможно преобразовать в число, то программируем сообщение «Следует вводить числа!» и выход из процедуры обработки события с помощью оператора return.

Далее оператор switch осуществляет множественный выбор арифметической операции в зависимости от индекса выбранного элемента списка SelectedIndex. Оператор switch передает управление той или иной метке case. Причем, как говорят программисты, оператор множественного выбора в C++ (C#, Turbo C) в отличие от других языков, например Basic, «проваливается», то есть управление переходит

на следующую метку case, поэтому приходиться использовать оператор break для выхода из switch.

Последний оператор в процедуре обработки события изменения индекса выбранного элемента осуществляет формирование строки с помощью метода String::Format для вывода ее на метку label1. Формат «{0:F5}» означает, что значение переменной Z следует выводить по фиксированному формату *с пятью знаками после запятой (или точки)*. Отметим, что в этом примере даже не пришлось программировать событие деления на ноль. Система Visual Studio сделала это за нас (см. рис. 1.16).

Последняя процедура обработки события «щелчок на кнопке» обеспечивает раскрытие комбинированного списка через нажатие на кнопку, а не на стрелку в элементе ComboBox. Эта возможность реализована с помощью метода DroppedDown объекта comboBox1.

Убедиться в работоспособности программы можно, открыв решение Комби.sln в папке Комби.

Пример 11. Вывод греческих букв, символов математических операторов. Кодовая таблица Unicode

Немного ликбеза. Хранение текстовых данных в памяти ЭВМ *предполагает кодирование символов по какому-либо принципу*. Таких кодировок несколько. Каждой кодировке соответствует своя таблица символов. В этой таблице каждой ячейке соответствуют номер в таблице и символ. Мы упомянем такие кодовые таблицы: ASCII, ANSI Cyrillic (другое название этой таблицы — Windows 1251), а также Unicode.

Первые две таблицы являются однобайтовыми, то есть каждому символу соответствует 1 байт данных. Поскольку в 1 байте — 8 битов, байт может принимать $2^8 = 256$ различных состояний, этим состояниям можно поставить в соответствие 256 разных символов. Так в таблице ASCII от 0 до 127 — базовая таблица — есть английские буквы, цифры, знаки препинания, управляющие символы. От 128 до 255 — это расширенная таблица, в ней находятся русские буквы и символы псевдографики. Некоторые из этих символов соответствуют клавишам IBM-совместимых компьютеров. Еще эту таблицу называют «ДОСовской» по имени операционной системы MS-DOS, где она применяется. Эта кодировка встречается также иногда у некоторых веб-страниц.

В операционной системе Windows используется преимущественно ANSI (Windows 1251). Базовые символы с кодами от 0 до 127 в этих таблицах совпадают, а расширенные — нет. То есть русские буквы в этих таблицах находятся в разных местах таблицы. Из-за этого бывают недоразумения. В ANSI нет символов псевдографики. ANSI Cyrillic — другое название кодовой таблицы Windows 1251.

Существует также двухбайтовый стандарт Unicode. Здесь один символ кодируется двумя байтами. Размер такой таблицы кодирования — $2^{16} = 65\,536$ ячеек.

Кодовая таблица Unicode включает в себя практически все современные шрифты. Когда в текстовом редакторе MS Word мы выполняем команду Вставка ▸ Символ, то вставляем символ из таблицы Unicode. В Блокноте также можно сохранять файлы в кодировке Unicode: Сохранить как ▸ Кодировка Юникод. В этом случае в Блокноте будут, например, греческие буквы, математические операторы Π, Δ, Σ и пр. Кстати, греческой букве Σ и математическому оператору Σ соответствуют разные коды в таблице Unicode. Размер файла при сохранении в Блокноте будет ровно в два раза больше.

Напишем программу, которая приглашает пользователя ввести радиус R, чтобы вычислить длину окружности. При программировании этой задачи длину окружности в метке Label назовем греческой буквой β, приведем формулу для вычислений с греческой буквой π = 3,14. Результат вычислений выведем в диалоговое окно MessageBox также с греческой буквой.

После традиционного запуска Visual Studio 2010 и выбора в среде CLR шаблона Windows Forms Application C++ перетащим в форму две метки Label, текстовое поле TextBox и командную кнопку Button. Посмотрите на рис. 1.17, так должна выглядеть форма после программирования этой задачи.

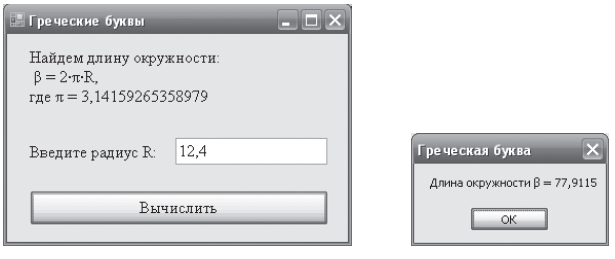

Рис. 1.17. Фрагмент работы программы, использующей символы Unicode

Вывод греческих букв на метку label1 и в диалоговое окно MessageBox можно осуществить, например, таким путем: в текст программы через буфер обмена вставляем греческие буквы из текстового редактора MS Word. Поскольку по умолчанию Visual Studio 2010 сохраняет h-файлы в формате Unicode, в принципе таким образом можно программировать вывод греческих букв и других символов на форму и на другие элементы управления.

С точки зрения технологии правильнее было бы пойти другим путем, а именно вставлять подобные символы с помощью функции Convert::ToChar, а на вход этой функции подавать номер символа в таблице Unicode. В этом случае, даже если h-файл будет сохранен в традиционной для Блокнота кодировке ANSI, программа будет работать корректно. Номер символа в таблице Unicode легко выяснить, выбрав в редакторе MS Word пункты меню Вставка ▸ Символ. Здесь в таблице следует найти этот символ и соответствующий ему код знака в шестнадцатеричном представлении. Чтобы перевести шестнадцатеричное представление в десятичное, следует перед шестнадцатеричным числом поставить 0x. Например, после выполнения оператора n = 0x3B2 в переменной n будет записано десятичное число 946. На этом месте в таблице Unicode расположена греческая буква β. Именно таким образом запрограммирована данная задача (листинг 1.10).

Листинг 1.10. Использование символов Unicode

```
            // ..............
            // Программный код, расположенный выше, создан средой  Visual  Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа демонстрирует возможность вывода в текстовую метку, а также
        // в диалоговое окно MessageBox греческих букв. Программа приглашает
        // пользователя ввести радиус R, чтобы вычислить длину окружности
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 this->Font = gcnew System::Drawing::
                                          Font("Times New Roman", 12.0F);
                 this->Text = "Греческие буквы"; button1->Text = "Вычислить";
                 // бета = 2 x Пи x R
                 label1->Text = String::Format(
                   "Найдем длину окружности:\n {0} = 2{1}{2}{1}R,\nгде {2} = {3}",
                    Convert::ToChar(0x3B2), Convert::ToChar(0x2219),
                    //           0=бета             1 - точка
                    Convert::ToChar(0x3C0), Math::PI);
                    //           2 - Пи   3 - число Пи
                 label2->Text = "Введите радиус R:";
                 textBox1->Clear();
             }
    private: System::
                Void button1_Click(System::Object^  sender, System::EventArgs^  e)
             {
                 // Проверка - число ли введено:
                 Single R; // - радиус
                 bool Число_ли = Single::TryParse(textBox1->Text,
                         System::Globalization::NumberStyles::Number,
                         System::Globalization::NumberFormatInfo::CurrentInfo, R);
                 if (Число_ли == false)
                 {
                     MessageBox::Show("Следует вводить числа!", "Ошибка",
                     MessageBoxButtons::OK, MessageBoxIcon::Error); return;
                 }
                 Single beta = 2 * (Single)Math::PI * R;
                 // 0x3B2 - греческая буква бета
                 MessageBox::Show(String::Format("Длина окружности {0} = {1:F4}",
                    Convert::ToChar(0x3B2), beta), "Греческая буква");
             }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы мы задали шрифт Times New Roman, 12 пунктов для формы, *этот шрифт будет*

распространяться на все элементы управления на форме, то есть на текстовое поле, метку и командную кнопку. Далее, используя метод String::Format, инициализировали свойство Text метки label1. Различные шестнадцатеричные номера соответствуют греческим буквам и арифметической операции «умножить», в инициализации строки участвует также константа π = 3,14. Ее *более* точное значение получаем из Math::PI. Escape-последовательность "\n" используем для переноса текста на новую строку. Перевод строки можно осуществить также с помощью строки NewLine из перечисления Environment.

Обрабатывая событие button1_Click (щелчок на кнопке), мы проверяем с помощью метода TryParse, число ли введено в текстовое поле. Если пользователь ввел число (true), то метод TryParse возвращает значение радиуса R. При вычислении длины окружности beta приводим значение константы Math::PI из типа Double к типу Single посредством неявного преобразования.

После вычисления длины окружности beta выводим ее значение вместе с греческой буквой β — Convert::ToChar(0x3B2) в диалоговое окно MessageBox. Здесь используем метод String::Format. Выражение «{0:F4}» означает, что значение переменной beta следует выводить по фиксированному формату с четырьмя знаками после запятой.

Данная программа будет корректно отображать греческие буквы, даже если открыть файл Form1.h текстовым редактором Блокнот и сохранить его в кодировке ANSI. Убедиться в работоспособности программы можно, открыв решение Unico.sln в папке Unico.

2

Программирование консольных приложений

Пример 12. Ввод и вывод в консольном приложении

Иногда, например для научных расчетов, требуется организовать какой-нибудь самый простой ввод данных, выполнить весьма сложную математическую обработку введенных данных и оперативно вывести на экран результат вычислений. Такая же ситуация возникает тогда, когда большая программа отлаживается по частям. И для отладки вычислительной части совершенно не важен сервис при вводе данных.

Можно по-разному организовать такую программу, в том числе программируя так называемое *консольное приложение* (от англ. *console* — пульт управления). Под консолью обычно подразумевают экран компьютера и клавиатуру.

Для примера напишем консольное приложение, которое приглашает пользователя ввести два числа, складывает их и выводит результат вычислений на консоль.

Для этого запускаем Visual C++ 2010, далее создаем новый проект (New Project), в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, задаем имя решения (Name) — Сумма. После щелчка на кнопке OK попадаем сразу на вкладку программного кода (рис. 2.1).

Как видите, здесь управляющая среда Visual Studio 2010 приготовила несколько строк программного кода. Это вполне работоспособная программа, по ее операторам можно «пройтись» отладчиком, последовательно нажимая клавишу F10. При запуске консольного или Windows-приложения C++ метод Main() является первым вызываемым методом. В фигурные скобки после Main() мы вставим собственный программный код (листинг 2.1).

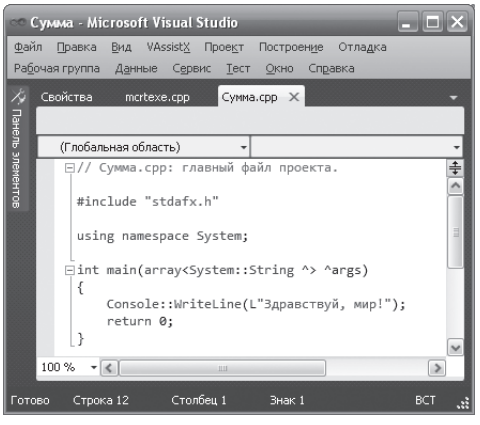

Рис. 2.1. Вкладка программного кода

Листинг 2.1. Ввод и вывод данных в консольном приложении

```
// Сумма.cpp: главный файл проекта.
// Программа организует ввод двух чисел, их сложение и вывод суммы на консоль
#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
{
    // Задаем строку заголовка консоли:
    Console::Title = "Складываю два числа:";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    // Ввод первого слагаемого:
    Console::WriteLine("Введите первое слагаемое:");
    String^ Строка = Console::ReadLine();
    Single X, Y, Z;
    // Преобразование строковой переменной в число:
    X = Single::Parse(Строка);
    // Ввод второго слагаемого:
    Console::WriteLine("Введите второе слагаемое:");
    Строка = Console::ReadLine();
    Y = Single::Parse(Строка);
    Z = X + Y;
    Console::WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z);
    // Звуковой сигнал частотой 1000 Гц и длительностью 0.5 секунды:
    Console::Beep(1000, 500);
    // Приостановить выполнение программы до нажатия какой-нибудь клавиши:
    Console::ReadKey();
    return 0;
}
```

Итак, в данной программе Main() — это стартовая точка, с которой начинается ее выполнение. Обычно консольное приложение выполняется в окне на черном

фоне. Чтобы как-то украсить традиционно черное окно консольного приложения, установим цвет фона окна BackgroundColor сине-зеленым (Cyan), а цвет символов, выводимых на консоль, черным (Black). Выводим строки в окно консоли методом WriteLine, а для считывания строки символов, вводимых пользователем, используем метод ReadLine. Далее объявляем три переменных типа Single для соответственно первого числа, второго и значения суммы. Тип данных Single применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части. Переменная типа Single занимает 4 байта. Для преобразования строки символов, введенных пользователем в числовое значение, используем метод Parse.

После вычисления суммы необходимо вывести результат вычислений из оперативной памяти на экран. Для этого воспользуемся форматированным выводом в фигурных скобках метода WriteLine объекта Console:

```
Console::WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z)
```

Затем выдаем звуковой сигнал Beep, символизирующий об окончании процедуры и выводе на экран результатов вычислений. Последняя строка в программе Console::ReadKey(); предназначена для приостановки выполнения программы до нажатия какой-нибудь клавиши. Если не добавить эту строку, окно с командной строкой сразу исчезнет, и пользователь не сможет увидеть вывод результатов выполнения. Программа написана. Нажмите клавишу F5, чтобы увидеть результат. Фрагмент работы данной программы представлен на рис. 2.2.

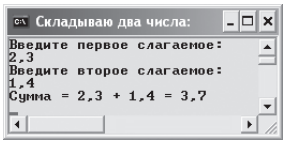

Рис. 2.2. Фрагмент работы консольного приложения

При организации научных расчетов или в ситуации, когда необходимо отладить расчетную часть большой программы, когда сервис при вводе данных вообще не имеет значения, можно просто присваивать значения переменным при их объявлении. Очень технологичным является вариант, когда данные записываются в текстовый файл с помощью, например, Блокнота (notepad.exe), а в программе предусмотрено чтение текстового файла в оперативную память.

Убедиться в работоспособности программы можно, открыв решение Сумма.sln в папке Сумма.

Пример 13. Вывод на консоль таблицы чисел с помощью форматирования строк

Пользуясь тем, что шрифт вывода на консоль является моноширинным, то есть каждый выводимый символ (например, точка и заглавная буква «Ш») имеет одинаковую ширину, в данном примере покажем, как путем форматирования строк

можно вывести таблицу в окно консоли. В этой программе выведем в таблицу извлеченный квадратный корень в цикле от нуля до десяти. Для этого запустим Visual Studio 2010, закажем новый проект (New Project), в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, задаем имя решения — ТаблКорней и на вкладке программного кода введем программный код, представленный в листинге 2.2.

Листинг 2.2. Вывод таблицы извлечения квадратного корня на консоль

```
// ТаблКорней.cpp: главный файл проекта.
// Консольное приложение задает цвета и заголовок консоли, а затем выводит
// таблицу извлечения квадратного корня от нуля до десяти
#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
{
    Console::BackgroundColor = ConsoleColor::Cyan;
    Console::ForegroundColor = ConsoleColor::Black;
    Console::Title = "Таблица"; Console::Clear();
    Console::WriteLine("Число  Корень");
    for (double i = 0; i <= 10; i++)
    {
        Console::WriteLine("{0,4} {1,8:F4}", i, Math::Sqrt(i));
    }
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, в пошаговом цикле for для вывода строк используется метод WriteLine, на вход которого подают строку в определенном формате. Причем форматирование производится по тем же правилам, что и в очень часто используемом методе String::Format, предназначенном для обработки строк. Использование формата «{0,4} {1,8:F4}» означает: взять нулевой выводимый элемент, то есть счетчик цикла i, и записать его с выравниванием вправо в четырех столбцах; после чего взять первый выводимый элемент, то есть значение квадратного корня из i, и записать его с выравниванием вправо в следующих восьми столбцах в формате с фиксированной точкой и четырьмя десятичными знаками после запятой. В результате работы данной программы получим таблицу извлечения квадратного корня (рис. 2.3).

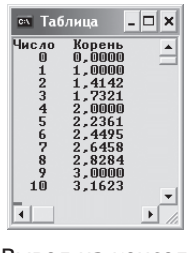

Рис. 2.3. Вывод на консоль таблицы

Подобным форматированием мы будет обрабатывать строки в последующих примерах данной книги, пользуясь методом String::Format.

Убедиться в работоспособности программы можно, открыв решение ТаблКорней.sln в папке ТаблКорней.

Пример 14. Вызов метода MessageBox::Show в консольном приложении. Формат даты и времени

Программирование консольного приложения возвращает нас в конец 1980-х годов, когда появились первые персональные компьютеры с очень слабой производительностью и небольшим объемом памяти. Для вывода данных Visual C++ имеет удобное средство MessageBox::Show, однако его используют, когда работают с экранными формами, а не с консолью. Но в консольном приложении его вызвать все-таки можно. Покажем на следующем примере, как на практике это можно сделать.

Итак, в данном примере следует в консольном приложении вывести в окно MessageBox текущую дату и время в различных форматах, чтобы попутно продемонстрировать читателю некоторые возможности форматирования строк с помощью метода String::Format. Запустим Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, задаем имя решения — ConsoleMessageBox.

Далее необходимо в проект *добавить ссылку на библиотеку*, содержащую объект MessageBox. Для этого в пункте меню Project выберем команду References и в новом окне щелкнем на кнопке Add New Reference, а на вкладке .NET дважды щелкнем на ссылке System.Windows.Forms. Теперь в окне Properties в области References (Ссылки) увидим добавленную в наш проект выбранную ссылку. Это ссылка на библиотеку (файл) System.Windows.Forms.dll, которая расположена в папке Program Files, где инсталлирована среда .NET Framework.

Чтобы сделать выражения в программном коде более компактными, вставим строку:

```
using namespace System::Windows::Forms;
```

Директива using используется для импортирования пространства имен, которое содержит класс MessageBox (листинг 2.3).

Листинг 2.3. Вызов MessageBox.Show в консольном приложении

```
// ConsoleMessageBox.cpp: главный файл проекта.
// Консольное приложение выводит в окно MessageBox текущую дату
// и время в различных форматах, используя String::Format
#include "stdafx.h"
using namespace System;
using namespace System::Windows::Forms;
int main(array<System::String ^> ^args)
```

продолжение ⇨

Листинг 2.3 (продолжение)

```
{
    String^ ДатаВремя = String::Format(
        "(d) - это формат \"короткой\" даты: . . . . . {0:d}\n" +
        "(D) - это формат \"полной\" даты:. . . . . .{0:D}\n" +
        "(t) - это формат \"короткого\" времени: . . . . . {0:t}\n" +
        "(T) - это формат \"длинного\" времени:. . . . . {0:T}\n" +
        "(f) - выводится \"полная\" дата и \"короткое\" время: {0:f}\n" +
        "(F) - выводится \"полная\" дата и \"длинное\" время:. {0:F}\n" +
        "(g) General - короткая дата и короткое время: . . {0:g}\n" +
        "(G) General - \"общий\" формат: . . . . . {0:G}\n" +
        "Пустой формат - такой же, как формат (G). . . . . {0}\n" +
        "(M) - выводится только месяц и число: . . . . . {0:M}\n" +
        "(U) Universal full date/time - время по Гринвичу. . {0:U}\n" +
        "(Y) - по этому формату выводится только год. . . . . {0:Y}\n",
        DateTime::Now);
    MessageBox::Show(ДатаВремя, "Время и дата в различных форматах");
    return 0;
}
```

Запустим наше приложение, нажав клавишу F5. В результате появится черное окно консоли, а под ним — диалоговое окно MessageBox (рис. 2.4).

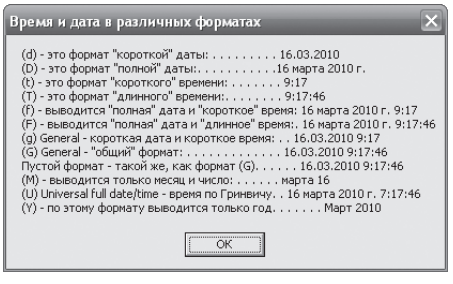

Рис. 2.4. Вывод текущего времени и даты в различных форматах

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ConsoleMessageBox.

Пример 15. Вызов функций Visual Basic из программы C++

Покажем, как можно воспользоваться в вашей C++-программе методами другой программной среды, в частности функциями Visual Basic. Решим следующую задачу: программа приглашает пользователя ввести одно число, затем другое, анализирует, ввел ли пользователь именно числа, а не другие символы, и выводит результат суммирования в диалоговое окно. Программировать будем в консольном

приложении C++ среды CLR, при этом воспользуемся функцией ввода InputBox, функцией вывода MsgBox, а также функцией IsNumeric, которая определяет, число ли подано на вход этой функции. Все *эти три функции были еще* в Visual Basic 6.0.

Для решения этой задачи запустим Visual Studio 2010, выберем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, задаем имя решения — СсылкаНаVisualBasic.

Далее необходимо в проект *добавить ссылку на библиотеку* Microsoft.VisualBasic.dll. Для этого в пункте меню Project выберем команду Properties и в новом окне щелкнем на кнопке Add New Reference, а на вкладке .NET дважды щелкнем на ссылке Microsoft.VisualBasic. Теперь в окне Properties в области References (Ссылки) увидим добавленную в наш проект выбранную ссылку. Затем перейдем на вкладку программного кода и введем текст, представленный в листинге 2.4.

Листинг 2.4. Программный код с вызовом функций Visual Basic

```
// СсылкаНаVisualBasic.cpp: главный файл проекта.
// В данном консольном приложении Visual C++ используем функции Visual Basic.
// Приложение приглашает пользователя ввести два числа, анализирует, числа ли
// ввел пользователь, и выводит результат суммирования на экран. При этом
// используем функции Visual Basic: InputBox, IsNumeric (для контроля,
// число ли ввел пользователь) и MsgBox
#include "stdafx.h"
using namespace System;
// Добавляем пространство имен для более коротких выражений:
using namespace Microsoft::VisualBasic;
// Для вызова функций из Visual Basic добавим ссылку в текущий проект.
// Для этого в пункте меню Project щелкнем мышью команду References
// и в появившемся окне щелкнем кнопку Add Reference, а в диалоговом окне
// на вкладке .NET выберем элемент Microsoft.VisualBasic
int main(array<System::String ^> ^args)
{
    String^ Строка;
    // Бесконечный цикл, пока пользователь не введет именно число:
    for (; ; )
    {
        // Первые два параметра Inputbox - очевидны, третий - это строка,
        // которая окажется в текстовом поле, если пользователь ничего не введет,
        // последние два - это координаты левого верхнего угла диалогового окна:
        Строка = Interaction::
            InputBox("Введите первое число:", "Складываю два числа", "", 100,
            100);
        if (Information::IsNumeric(Строка) == true) break;
    }
    // - преобразование строковой переменной в число:
    Single X = Single::Parse(Строка);
    // Ввод второго числа:
    for (; ; )
```

продолжение ⇩

Листинг 2.4 (продолжение)

```
    {
        Строка = Interaction::
            InputBox("Введите второе число:", "Складываю два числа", "", 100,
            100);
        // Если строка,введенная пользователем,оказалась числом,то выход
        из цикла:
        if (Information::IsNumeric(Строка) == true) break;
    }
    Single Y = Single::Parse(Строка);
    Single Z = X + Y;
    Строка = String::Format("Сумма = {0} + {1} = {2}", X, Y, Z);
    // Вывод результата вычислений на экран:
    Interaction::MsgBox(Строка, MsgBoxStyle::Information,
                                                "Результат суммирования");
    return 0;
}
```

Ввод первого числа организуем с помощью бесконечного цикла for(;;){}. Функция InputBox возвращает строку, введенную пользователем в диалоговом окне (рис. 2.5).

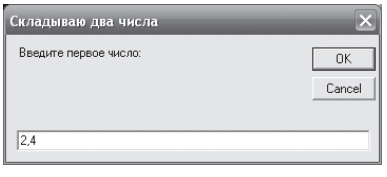

Рис. 2.5. Диалоговое окно ввода данных

Эта строка анализируется функцией IsNumeric, и если в строке записано число, то происходит выход break из вечного цикла. Аналогично организован ввод второго числа. Для преобразования строки символов, введенных пользователем, в числовое значение используем метод Parse. Кстати, для преобразования строки символов в числовое значение платформа .NET Framework содержит метод TryParse, который более эффективно решает подобные задачи по проверке типа введенных данных и преобразованию их в числовое значение.

После вычисления суммы необходимо вывести результат вычислений из оперативной памяти в диалоговое окно MsgBox. Для этого сначала подготовим строку вывода с помощью метода String.Format. Выражение «Сумма = {0} + {1} = {2}» означает: взять нулевой выводимый элемент, то есть переменную X, и записать ее вместо первых фигурных скобок, после чего взять первый выводимый элемент, то есть переменную Y, и записать ее вместо вторых фигурных скобок, аналогично — для третьего выводимого элемента.

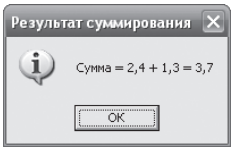

Рис. 2.6. Диалоговое окно вывода данных

Результат работы программы представлен на рис. 2.6.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке СсылкаНаVisualBasic.

Пример 16. Замечательной структурой данных является словарь Dictionary

Словарь данных Dictionary представляет собой совокупность (коллекцию) ключей и значений. То есть каждый элемент (запись), добавляемый в словарь, состоит из значения Value и связанного с ним ключа Key. Извлечение значения по его ключу происходит очень быстро, поскольку класс Dictionary<Key, Value> реализован как хэш-таблица. Каждый ключ в словаре Dictionary<Key, Value> должен быть уникальным, то есть единственным в своем роде, эксклюзивным. При добавлении в коллекцию Dictionary очередного элемента так называемый компаратор проверяет на равенство уникальность нового ключа. Ключ не может быть пустым (null), а значение может, если тип значения Value является ссылочным типом. Возможно создание словаря, в котором не различается регистр символов. Использование словаря Dictionary может существенно повлиять на эффективность алгоритма, на простоту его понимания и легкость программной реализации.

Задача, рассматриваемая в данном примере, состоит в том, чтобы продемонстрировать удобство и технологичность манипулирования месяцами года для какого-либо анализа с помощью словаря данных Dictionary. В данном случае помещаем в словарь данных названия месяцев, они будут являться ключом словаря, а также количество дней в данном месяце, которое будет представлять значение. В таком случае очень удобно манипулировать этими парами ключ-значение. Из этого словаря выведем на консоль названия месяцев, в которых количество дней равно 30.

Для решения этой задачи запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — Месяцы. Ниже в листинге 2.5 приведем программный код данного консольного приложения.

Листинг 2.5. Использование словаря данных для манипулирования месяцами года

```
// ___Месяцы.cpp: главный файл проекта.
// Программа создает словарь данных типа Dictionary и записывает в этот
// словарь названия месяцев и количество дней в каждом месяце. Ключом
// словаря является название месяца, а значением - количество дней.
// Используя цикл for each, программа выводит на консоль только те месяцы,
// количество дней в которых равно 30
#include "stdafx.h"
using namespace System;
// Добавим эту директиву для краткости выражений:
using namespace System::Collections::Generic;
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    Console::Title = "Использование словаря данных";
    // Создаем словарь данных с полями типа String^ и int:
    auto Месяцы = gcnew Dictionary<String^, int>();
```

продолжение ⇩

Листинг 2.4 (продолжение)

```
    // Инициализация словаря Месяцы:
    Месяцы["Январь"] = 31;    Месяцы["Июль"] = 31;
    Месяцы["Февраль"] = 28;   Месяцы["Август"] = 31;
    Месяцы["Март"] = 31;      Месяцы["Сентябрь"] = 30;
    Месяцы["Апрель"] = 30;    Месяцы["Октябрь"] = 31;
    Месяцы["Май"] = 31;       Месяцы["Ноябрь"] = 30;
    Месяцы["Июнь"] = 30;      Месяцы["Декабрь"] = 31;
    Console::WriteLine("Месяцы с 30 днями: \n");
    // Поиск в словаре месяцев, содержащих 30 дней:
    for each(KeyValuePair<String^, int> Месяц in Месяцы )
        if ( Месяц.Value == 30 )
        Console::WriteLine("{0} - {1} дней", Месяц.Key, Месяц.Value);
    // Ждем от пользователя нажатия какой-либо клавиши:
    Console::ReadKey();
    return 0;
}
```

В программном коде вначале создаем объект класса Dictionary, то есть словарь данных с полями типа String^ — для названий месяцев и типа int — для количества дней. Инициализация словаря, то есть присвоение начальных значений, здесь очень удобная: название месяца выступает в роли «индекса», если использовать терминологию массивов. Поиск в словаре месяцев, содержащий 30 дней, мы организовали с помощью цикла for each. Здесь использована системная структура KeyValuePair, предназначенная именно для манипуляций со словарем, хотя ее можно использовать и в других случаях.

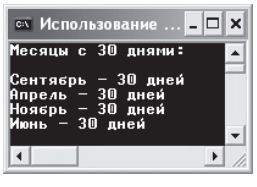

Рис. 2.7. Вывод на консоль месяцев, содержащих 30 дней

Результат работы программы показан на рис. 2.7.

Убедиться в работоспособности программы можно, открыв решение Месяцы.sln в папке Месяцы.

3

Инициирование и обработка событий мыши и клавиатуры

Пример 17. Координаты курсора мыши относительно экрана и элемента управления

Напишем программу, которую мы условно называем *мониторингом положения мыши*. Имеем форму, список элементов ListBox и два текстовых поля. Они расположены в форме так, как показано на рис. 3.1.

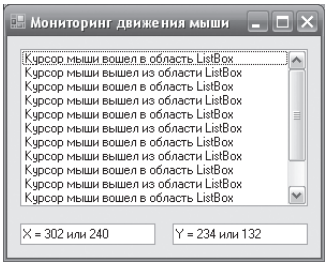

Рис. 3.1. Фрагмент работы программы определения координат курсора мыши

Программа заполняет список ListBox данными о местоположении и изменении положения курсора мыши. Кроме того, в текстовых полях отображаются координаты положения курсора мыши относительно экрана, а также относительно элемента управления ListBox.

Для программирования этой задачи после запуска Visual Studio 2010 и выбора приложения в среде CLR шаблона Windows Forms Application Visual C++ из панели элементов управления Toolbox перетащим в форму элемент управления ListBox и два текстовых поля.

В данной программе нам понадобится обработать событие загрузки формы, а также три события, относящиеся к объекту listBox1. Получить пустой обработчик события загрузки формы просто — достаточно дважды щелкнуть на проектируе-

мой экранной форме. А чтобы получить три соответствующих пустых обработчика для объекта listBox1, следует в конструкторе формы в панели свойств (Properties) щелкнуть на пиктограмме молнии и в появившемся списке возможных событий для объекта listBox1 выбрать следующие три события: MouseEnter, MouseLeave и MouseMove. Соответственно получим четыре пустых обработчика событий (листинг 3.1).

Листинг 3.1. Координаты курсора мыши относительно экрана и элемента управления

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа отображает координаты курсора мыши относительно экрана
        // и элемента управления. Программа содержит форму, список элементов
        // ListBox и два текстовых поля. Программа заполняет список ListBox
        // данными о местоположении и изменении положения курсора мыши. Кроме
        // того, в текстовых полях отображаются координаты положения курсора мыши
        // относительно экрана и элемента управления ListBox
    private: // Процедура обработки события загрузки формы:
        System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
        {
            Form1::Text = "Мониторинг движения мыши";
        }
        // Процедура обработки события, когда указатель мыши оказывается
        // на элементе управления ListBox:
    private: System::Void listBox1_MouseEnter(System::Object^  sender,
                                              System::EventArgs^  e)
             {
                 // Добавляем в список элементов новую запись
                 listBox1->Items->Add("Курсор мыши вошел в область ListBox");
             }
             // Процедура обработки события, когда указатель мыши покидает
             // элемент управления ListBox:
    private: System::Void listBox1_MouseLeave(System::Object^  sender,
                                              System::EventArgs^  e)
             {
                 listBox1->Items->Add("Курсор мыши вышел из области ListBox");
             }
             // Процедура обработки события, происходящего при перемещении
             // указателя мыши по элементу управления ListBox:
    private: System::Void listBox1_MouseMove(System::Object^  sender,
                                  System::Windows::Forms::MouseEventArgs^  e)
             {
                 // Свойство объекта Control MousePosition возвращает точку,
                 // соответствующую текущему положению мыши относительно
```

```
                // левого верхнего угла монитора
                textBox1->Text = String::Format("X = {0} или {1}",
                                                this->MousePosition.X, e->X);
                textBox2->Text = String::Format("Y = {0} или {1}",
                                                this->MousePosition.Y, e->Y);
            }
    };
}
```

Вы видите, что при обработке события мыши MouseEnter, когда курсор мыши входит в границы элемента управления, в список ListBox1 *добавляется* (метод Add) *запись* «Курсор мыши вошел в область ListBox». При обработке события мыши MouseLeave, когда курсор мыши выходит за пределы элемента управления, в список ListBox добавляется запись «Курсор мыши вышел из области ListBox». Таким образом, отслеживая поведение мыши, мы заполняем список ListBox1.

При обработке события MouseMove, когда курсор мыши перемещается в пределах элемента управления ListBox1, в текстовые поля записываем координаты X и Y курсора мыши, используя свойство объекта Control MousePosition. Здесь мы получаем координаты положения курсора мыши в системе координат экрана, когда начало координат расположено в левом верхнем углу экрана, ось x направлена вправо, а ось y — вниз.

Заметим, что аргументы события мыши e также содержат текущие координаты курсора мыши, но в системе координат элемента управления, в данном случае listBox1. Начало координат этой системы расположено в левом верхнем углу элемента управления listBox1, ось x также направлена вправо, ось y — вниз. Эти координаты получаем из аргументов события e->X и e->Y и выводим их в текстовое поле, отделяя от предыдущих координат словом «или».

Таким образом, добиваемся *контроля положения курсора мыши*, обрабатывая события мыши. Убедиться в работоспособности программы можно, открыв решение Мониторинг.sln папки Мониторинг.

Пример 18. Создание элемента управления Button «программным» способом и подключение события для него

Мы знаем, как, используя панель элементов управления, мышью перенести в форму нужный элемент. Чтобы сделать разработку программы более управляемой, в данной программе научимся создавать элементы управления в форме «программным» способом, то есть с помощью написания программного кода, не используя при этом панель элементов управления Toolbox. Понятно, что назвать способ «программным» можно весьма условно, поскольку в описываемой нами среде трудно назвать что-либо, что «программным» не является.

Итак, данная программа создаст командную кнопку в форме «программным» способом, задаст свойства кнопки: ее видимость, размеры, положение, надпись на кнопке и подключит событие «щелчок на кнопке».

Для этого создаем новый проект с формой. При этом, как обычно, запускаем Visual Studio 2010, в окне New Project выбираем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Чтобы к программному коду добавить пустой обработчик события загрузки формы, дважды щелкнем на проектируемой экранной форме. Далее вводим программный код, представленный в листинге 3.2.

Листинг 3.2. Создание кнопки «программным» способом

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа создает командную кнопку в форме «программным» способом,
        // т.е. с помощью написания непосредственно программного кода, не
        // используя при этом панель элементов управления Toolbox. Программа
        // задает свойства кнопки: ее видимость, размеры, положение, надпись
        // на кнопке и подключает событие "щелчок на кнопке"
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                // Создание кнопки без панели элементов управления:
                Button^ button1 = gcnew Button();
                // Задаем свойства кнопки:
                button1->Visible = true;
                // Ширина и высота кнопки:
                button1->Size = Drawing::Size(100, 30);
                // Расположение кнопки в системе координат формы:
                button1->Location = Drawing::Point(100, 80);
                button1->Text = "Новая кнопка";
                // Добавление кнопки в коллекцию элементов управления
                this->Controls->Add(button1);
                // Подписку на событие Click для кнопки можно делать "вручную".
                // Связываем событие Click с процедурой обработки этого события:
                button1->Click += gcnew EventHandler(this,
                                                     &Form1::ЩелчокНаКнопке);
             }
    private: System::
               Void ЩелчокНаКнопке(System::Object^  sender, System::EventArgs^  e)
             {
                MessageBox::Show("Нажата новая кнопка");
             }
    };
}
```

Мы видим, что при обработке события загрузки формы создаем новый объект button1 стандартного класса кнопок. Задаем свойства кнопки: ее видимость (Visible), размеры (Size), положение (Location) относительно левого нижнего угла формы, надпись на кнопке — «Новая кнопка».

Далее необходимо организовать корректную работу с событием «щелчок на созданной нами командной кнопке». В предыдущих примерах мы для этой цели в конструкторе формы дважды щелкали на проектируемой кнопке, и исполняемая среда автоматически генерировала пустой обработчик этого события в программном коде. Или опять же в конструкторе формы в панели свойств проектируемой кнопки щелкали мышью на значке молнии (Events) и в появившемся списке всех событий выбирали необходимое событие. Однако согласно условию задачи мы должны организовать обработку события «программным» способом без использования конструктора формы. Для этого в программном коде сразу после добавления командной кнопки в коллекцию элементов управления поставим оператор стрелки (->) после имени кнопки button1 и в раскрывающемся списке выберем необходимое событие Click. Затем, как приведено в листинге 3.2, осуществляем так называемую «подписку» на данное событие, то есть с помощью ключевого слова EventHandler связываем событие Click с процедурой обработки события. Мы его назвали «Щелчок-НаКнопке». Теперь создадим обработчик события «щелчок на кнопке», как показано в листинге 3.2. В этой процедуре предусматриваем вывод сообщения «Нажата новая кнопка». На рис. 3.2 приведен фрагмент работы программы.

Рис. 3.2. Создание кнопки программным способом

В заключение отметим, что в случае создания пустого обработчика события в конструкторе формы строка подписки на событие формируется автоматически в методе InitializeComponent в файле Form1.h проекта. Убедиться в работоспособности программы можно, открыв решение NewButton.sln папки NewButton.

Пример 19. Обработка нескольких событий одной процедурой

Для того чтобы события от нескольких элементов управления обрабатывались одной процедурой обработки события, в некоторых языках, например в VB6, было предусмотрено создание массива элементов управления. Однако в современных языках Visual Studio (C++, C#, F# и Visual Basic) элементы *не могут быть сгруппированы в массивы*. Но можно организовать обработку нескольких событий одной

процедурой путем подписки этих событий на одну и ту же процедуру их обработки. Как мы убедились в предыдущем разделе, синтаксис языка позволяет называть процедуру обработки события как угодно, даже по-русски.

Как это сделать, покажем на примере, когда в форме присутствуют две командные кнопки, и щелчок на любой из них обрабатывается одной процедурой. При этом, используя параметр процедуры sender, будем определять, на какой из двух кнопок щелкнули мышью.

Итак, запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов перенесем в форму две командные кнопки и текстовую метку. Далее через двойной щелчок мышью в пределах проектируемой формы создадим пустой обработчик загрузки формы и перейдем к вкладке программного кода (листинг 3.3).

Листинг 3.3. Связывание двух событий с одной процедурой обработки

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // В форме имеем две командные кнопки, и при нажатии указателем мыши
        // любой из них получаем номер нажатой кнопки. При этом в программе
        // предусмотрена только одна процедура обработки событий
   private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                Form1::Text = "Щелкните на кнопке"; label1->Text = nullptr;
                // Связываем события Click от обеих кнопок с одной процедурой
                КЛИК:
                button1->Click += gcnew EventHandler(this, &Form1::КЛИК);
                button2->Click += gcnew EventHandler(this, &Form1::КЛИК);
                // Подпиской на событие называют связывание названия события
                // с названием процедуры обработки события посредством
                EventHandler
            }
    private: System::Void КЛИК(System::Object^  sender, System::EventArgs^  e)
            {
                // String S = Convert.ToString(sender);
                // получить текст, отображаемый на кнопке, можно таким образом:
                Button^ Кнопка = (Button^)sender;
               // или String^ НадписьНаКнопке = ((Button^)sender)->Text;
                label1->Text = "Нажата кнопка " + Кнопка->Text; // или
                Кнопка->Name
            }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы мы осуществляем так называемую подписку на событие, то есть связываем название

события с названием процедуры обработки события КЛИК посредством метода (делегата) EventHandler. Этот метод делегирует (передает полномочия) обработку события button1->Click процедуре КЛИК. Заметим, что события Click от обеих кнопок мы связали с одной и той же процедурой КЛИК.

Далее создаем процедуру обработки события КЛИК, ее параметр sender содержит *ссылку на объект-источник события*, то есть кнопку, нажатую пользователем. С помощью неявного преобразования можно конвертировать параметр sender в экземпляр класса Button и, таким образом, выяснить все свойства кнопки, которая инициировала событие.

На рис. 3.3 приведен пример работы написанной программы.

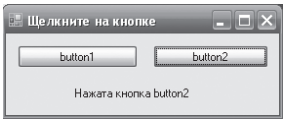

Рис. 3.3. Фрагмент работы программы, определяющей нажатую кнопку

Мы убедились в этом разделе, что сведения об объекте, который создал событие, находятся в объектной переменной sender. Работу этой программы можно исследовать, открыв соответствующее решение в папке ДваСобытияОднаПроц.

Пример 20. Калькулятор

Обработка *нескольких событий от разных объектов одной процедурой оказывается весьма полезной при программировании данного приложения. Напишем программу* Калькулятор с кнопками-цифрами, выполняющую только арифметические операции, причем управление Калькулятором возможно только мышью.

Запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox перетащим в форму 16 командных кнопок для ввода цифр, арифметических операций, знака «равно» (=) и операции Очистить, а также текстовое поле. Вкладка Form1.h [Design] будет иметь примерно такой вид, как показано на рис. 3.4.

Рис. 3.4. Вкладка конструктора формы

В листинге 3.4 приведен программный код данного приложения.

Листинг 3.4. Калькулятор

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа Калькулятор с кнопками цифр. Управление калькулятором
        // возможно только мышью. Данный калькулятор выполняет лишь арифметические
        // операции
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Объявляем внешние переменные, видимые из всех процедур класса Form1:
        String^ Znak;      // - знак арифметической операции;
        bool Начало_Ввода; // - ожидание ввода нового числа;
        Double Число1, Число2; // - первое и второе числа, вводимые пользователем
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Калькулятор"; Начало_Ввода = true;
                Znak = nullptr;
                button1->Text = "1";  button2->Text = "2";  button3->Text = "3";
                button4->Text = "4";  button5->Text = "5";  button6->Text = "6";
                button7->Text = "7";  button8->Text = "8";  button9->Text = "9";
                button10->Text = "0"; button11->Text = "="; button12->Text =
                "+";
                button13->Text = "-"; button14->Text = "*"; button15->Text =
                "/";
                button16->Text = "Очистить";
                textBox1->Text = "0";
                textBox1->TextAlign = HorizontalAlignment::Right;
                // Связываем все события "щелчок на кнопках-цифрах"
                // с обработчиком ЦИФРА:
                button1->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button2->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button3->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button4->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button5->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button6->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button7->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button8->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button9->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button10->Click += gcnew EventHandler(this, &Form1::ЦИФРА);
                button12->Click += gcnew EventHandler(this, &Form1::ОПЕРАЦИЯ);
                button13->Click += gcnew EventHandler(this, &Form1::ОПЕРАЦИЯ);
                button14->Click += gcnew EventHandler(this, &Form1::ОПЕРАЦИЯ);
                button15->Click += gcnew EventHandler(this, &Form1::ОПЕРАЦИЯ);
```

```
            button11->Click += gcnew EventHandler(this, &Form1::РАВНО);
            button16->Click += gcnew EventHandler(this, &Form1::ОЧИСТИТЬ);
        }
    private: System::Void ЦИФРА(System::Object^ sender, System::EventArgs^ e)
        {
            // Обработка события нажатия кнопки-цифры.
            // Получить текст, отображаемый на кнопке, можно таким образом:
            Button^ Кнопка = (Button^)sender;
            String^ Digit = Кнопка->Text;
            if (Начало_Ввода == true)
            {  // Ввод первой цифры числа:
                textBox1->Text = Digit;
                Начало_Ввода = false; return;
            }
            // "Сцепливаем" полученные цифры в новое число:
            if (Начало_Ввода == false)
                textBox1->Text = textBox1->Text + Digit;
        }
    private: System::Void ОПЕРАЦИЯ(System::Object^ sender, System::EventArgs^  e)
        {
            // Обработка события нажатия кнопки арифметической операции:
            Число1 = Double::Parse(textBox1->Text);
            // Получить текст, отображаемый на кнопке, можно таким образом:
            Button^ Кнопка = (Button^)sender;
            Znak = Кнопка->Text;
            Начало_Ввода = true; // ожидаем ввод нового числа
        }
    private: System::Void РАВНО(System::Object^ sender, System::EventArgs^ e)
        {
            // Обработка нажатия клавиши "равно"
            double Результат = 0;
            Число2 = Double::Parse(textBox1->Text);
            if (Znak == "+") Результат = Число1 + Число2;
            if (Znak == "-") Результат = Число1 - Число2;
            if (Znak == "*") Результат = Число1 * Число2;
            if (Znak == "/") Результат = Число1 / Число2;
            Znak = nullptr;
            // Отображаем результат в текстовом поле:
            textBox1->Text = Результат.ToString();
            Число1 = Результат; Начало_Ввода = true;
        }
    private: System::Void ОЧИСТИТЬ(System::Object^ sender, System::EventArgs^ e)
        {
            // Обработка нажатия клавиши "Очистить"
            textBox1->Text = "0"; Znak = nullptr; Начало_Ввода = true;
        }
    };
}
```

В этой книге мы принципиально отказались от задания свойств элементов управления и формы в окне Properties, чтобы не потеряться в огромном количестве этих свойств и не забывать, какое свойство мы изменили, а какое нет. Исходя из этих соображений, автор задал все свойства объектов в процедуре обработки события загрузки формы Form1_Load. Именно здесь заданы надписи на кнопках, ноль в текстовом поле, причем этот ноль прижат к правому краю поля: textBox1-> TextAlign = HorizontalAlignment::Right.

Далее связываем все события Click от кнопок-цифр с одной процедурой обработки этих событий ЦИФРА. Аналогично все события Click от кнопок арифметических операций связываем с одной процедурой ОПЕРАЦИЯ.

В процедуре обработки события щелчком на любой из кнопок-цифр ЦИФРА в строковую переменную Digit копируем цифру, изображенную на кнопке из свойства Text так, как мы это делали в предыдущем примере, когда отлавливали нажатие пользователем одной из двух кнопок. Далее необходимо значение Digit присвоить свойству textBox1->Text, но здесь изначально записан ноль. Если пользователь вводит первую цифру, то вместо нуля нужно записать эту цифру, а если пользователь вводит последующие цифры, то их надо «сцепить» вместе. Для управления такой ситуацией мы ввели булеву (логическую) переменную Начало_Ввода. Мы сознательно назвали эту переменную по-русски, *чтобы она выделялась среди прочих переменных*, ведь она играет ключевую роль в программе и участвует в обработке практически *всех событий*. Поскольку мы ввели ее в начале программы, область действия этой переменной — весь класс Form1, то есть эта переменная *«видна» в процедурах обработки всех событий*.

То есть различаем начало ввода числа Начало_Ввода = true, когда ноль следует менять на вводимую цифру, и последующий ввод Начало_Ввода = false, когда очередную цифру следует добавлять справа. Таким образом, если это уже не первая нажатая пользователем кнопка-цифра (Начало_Ввода = false), то «сцепляем» полученную цифру с предыдущими введенными цифрами, иначе — просто запоминаем первую цифру в текстовом поле textBox1.

При обработке событий «щелчок указателем мыши по кнопкам» арифметических операций +, -, *, / в процедуре ОПЕРАЦИЯ преобразуем первое введенное пользователем число из текстового поля в переменную value1 типа Double. Строковой переменной Znak присваивается символьное представление арифметической операции. Поскольку пользователь нажал кнопку арифметической операции, ожидаем, что следующим действием пользователя будет ввод очередного числа, поэтому присваиваем булевой переменной Начало_Ввода значение true. Заметьте, что обрабатывая два других события: нажатие кнопки «равно» и нажатие кнопки Очистить, мы также устанавливаем логическую переменную Начало_Ввода в состояние true (то есть начинаем ввод числа).

В процедуре обработки события нажатия кнопки «равно» преобразуем второе введенное пользователем число в переменную типа Double. Теперь, поскольку знак арифметической операции нам известен, известны также оба числа, мы можем выполнить непосредственно арифметическую операцию. После того как пользователь получит результат, например result = value1 + value2, возможно, он захочет с этим результатом выполнить еще какое-либо действие, поэтому этот результат

записываем в первую переменную value1. Заметьте, что в этой программе мы сознательно не предусмотрели обработку исключительной ситуации *деления на ноль*, поскольку среда Visual Studio 2010 (впрочем, как и ее предыдущие версии) взяла на себя обработку этой ситуации. Когда в строковую переменную попадает очень большое число, в эту переменную система пишет слово «бесконечность» (рис. 3.5).

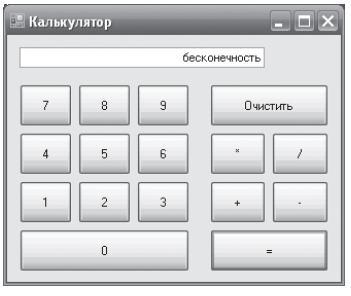

Рис. 3.5. Фрагмент работы калькулятора

Убедиться в работоспособности программы можно, открыв решение Калькулятор.sln в папке Калькулятор.

Пример 21. Ссылка на другие ресурсы LinkLabel

Элемент управления LinkLabel позволяет *создавать в форме ссылки на веб-страницы, подобно гиперссылкам в* HTML-документах, ссылки на открытие файлов какими-либо программами, ссылки на просмотр содержания логических дисков, папок и пр.

Напишем программу, которая с помощью элемента управления LinkLabel обеспечит ссылку для посещения почтового сервера www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот.

Для программирования этой задачи запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели Toolbox перетащим в форму три элемента управления LinkLabel. Равномерно разместим их в форме. Далее, следуя нашим традициям, не будем задавать никаких свойств этим элементам в окне Properties. Все начальные значения свойств укажем в программном коде при обработке процедуры загрузки формы (листинг 3.5).

Листинг 3.5. Ссылки на ресурсы

```
        // ..............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
        this->PerformLayout();
    }
```

продолжение ⇗

Листинг 3.5 (продолжение)

```
#pragma endregion
        // Программа обеспечивает ссылку для посещения почтового сервера
        // www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для
        // запуска текстового редактора Блокнот с помощью элемента управления
        // LinkLabel
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             { // Обработка процедуры загрузки формы:
                this->Text = "Щелкните по ссылке:";
                linkLabel1->Text = "www.mail.ru";
                linkLabel2->Text = "Папка C:\\Windows\\";
                linkLabel3->Text = "Вызвать \"Блокнот\"";
                this->Font = gcnew System::Drawing::Font("Consolas", 12.0F);
                linkLabel1->LinkVisited = true;
                linkLabel2->LinkVisited = true;
                linkLabel3->LinkVisited = true;
                // Подписка на события: все три события обрабатываются
                // одной процедурой:
                linkLabel1->LinkClicked += gcnew
                    LinkLabelLinkClickedEventHandler(this, &Form1::ССЫЛКА);
                linkLabel2->LinkClicked += gcnew
                    LinkLabelLinkClickedEventHandler(this, &Form1::ССЫЛКА);
                linkLabel3->LinkClicked += gcnew
                    LinkLabelLinkClickedEventHandler(this, &Form1::ССЫЛКА);
             }
    private: System::Void ССЫЛКА(System::Object^ sender,
                                 LinkLabelLinkClickedEventArgs^ e)
             { // Обработка щелчка на любой из ссылок:
                LinkLabel^ ссылка = (LinkLabel^)sender;
                String^ Имя = ссылка->Name;
                // Выбор ссылки по девятому элементу в массиве Имя:
                switch (Имя[9])
                {
                case '1': //"linkLabel1":
                    Diagnostics::Process::Start(
                        "IExplore.exe", "http://www.mail.ru"); break;
                case '2': // "linkLabel2":
                    Diagnostics::Process::Start("C:\\Windows\\"); break;
                case '3': //"linkLabel3":
                    Diagnostics::Process::Start("Notepad", "text.txt");
                    break;
                }
             }
    };
}
```

Как видно из программного кода, в свойстве Text каждой ссылки LinkLabel задаем текст, который поясняет пользователю назначение данной ссылки. В задании свойства Text ссылки LinkLabel3 для того, чтобы слово «Блокнот» было в двойных

кавычках, используем escape-последовательность (\»). Для большей выразительности задаем шрифт Consolas, 12 пунктов. Это шрифт моноширинный, кстати, по умолчанию редактор в Visual Studio 2010 имеет также шрифт Consolas. Поскольку свойство LinkVisited = true, то соответствующая ссылка отображается как уже посещавшаяся (изменяется цвет).

Так же как и в предыдущих разделах, организуем обработку всех трех событий Click по каждой из ссылок одной процедурой обработки ССЫЛКА. В этой процедуре, так же как и в программе о трех кнопках и калькуляторе, в зависимости от имени объекта (ссылки), создающего события (linkLabel1, linkLabel2, linkLabel3), мы вызываем одну из трех программ: либо Internet Explorer, либо Windows Explorer, либо Блокнот. Информация об объекте, создающем событие Click, записана в объектную переменную sender. Она позволяет распознавать объекты (ссылки), создающие события. Чтобы «вытащить» эту информацию из sender, объявим переменную ссылка типа LinkLabel и с помощью неявного преобразования выполним конвертирование параметра sender в экземпляр класса LinkLabel. В этом случае переменная ссылка будет содержать все свойства объекта-источника события, в том числе свойство name, с помощью которого мы сможем распознавать выбранную ссылку. Идентифицируя по свойству name каждую из ссылок, с помощью метода Start вызываем либо Internet Explorer, либо Windows Explorer, либо Блокнот. Вторым параметром метода Start является имя ресурса, подлежащего открытию. Именем ресурса может быть или название веб-страницы, или имя текстового файла.

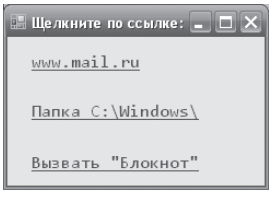

Рис. 3.6. Ссылки на ресурсы

Фрагмент работы обсуждаемой программы приведен на рис. 3.6. Убедиться в ее работоспособности можно, открыв соответствующее решение в папке ССылкиLinkLabel.

Пример 22. Обработка событий клавиатуры

События клавиатуры (клавишные события) создаются в момент нажатия или отпускания ее клавиш. Различают событие KeyPress, которое *генерируется в момент нажатия клавиши*. При удержании клавиши в нажатом состоянии оно генерируется непрерывно с некоторой частотой. С помощью этого события можно распознать нажатую клавишу, если только она не является так называемой *модифицирующей*, то есть Alt, Shift и Ctrl. А вот для того чтобы распознать, нажата ли модифицирующая клавиша Alt, Shift или Ctrl, следует обработать либо событие KeyDown, либо событие KeyUp. Событие KeyDown генерируется в *первоначальный момент нажатия клавиши, а событие* KeyUp — в момент *отпускания* клавиши.

Напишем программу, информирующую пользователя о тех клавишах и комбинациях клавиш, которые тот нажал. Запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели Toolbox перетащим в форму две текстовых метки Label. Далее, поскольку нам потребуются клавишные события формы: KeyPress, KeyDown,

KeyUp, уже привычным способом получим пустые обработчики этих событий. То есть в панели Properties щелкнем на пиктограмме молнии (Events), а затем в списке всех возможных событий выберем каждое из названных событий клавиатуры. Программный код приведен в листинге 3.6.

Листинг 3.6. Обработка событий клавиатуры

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа, информирующая пользователя о тех клавишах
        // и комбинациях клавиш, которые тот нажал
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                // Устанавливаем шрифт с фиксированной шириной (моноширинный):
                Form1::Font = gcnew Drawing::
                    Font(FontFamily::GenericMonospace, 14.0F);
                // Поскольку мы задали этот шрифт увеличенным (от 8 по умолчанию
                // до 14), форма окажется пропорционально увеличенной
                Form1::Text = "Какие клавиши нажаты сейчас:";
                label1->Text = String::Empty; label2->Text = String::Empty;
            }
    private: System::Void Form1_KeyPress(System::Object^  sender,
                System::Windows::Forms::KeyPressEventArgs^  e)
            {
                // Здесь событие нажатия клавиши: при удержании
                // клавиши генерируется непрерывно
                label1->Text = "Нажатая клавиша: " + e->KeyChar;
            }
    private: System::Void Form1_KeyDown(System::Object^  sender,
                System::Windows::Forms::KeyEventArgs^  e)
            {
                // Здесь обрабатываем мгновенное событие первоначального
                // нажатия клавиши
                label2->Text = String::Empty;
                // Если нажата клавиша Alt
                if (e->Alt == true) label2->Text += "Alt: Yes\n";
                else                label2->Text += "Alt: No\n";

                // Если нажата клавиша Shift
                if (e->Shift == true) label2->Text += "Shift: Yes\n";
                else                  label2->Text += "Shift: No\n";

                // Если нажата клавиша Ctrl
                if (e->Control == true) label2->Text += "Ctrl: Yes\n";
```

```
            else                    label2->Text += "Ctrl: No\n";

            label2->Text += String::Format(
                "Код клавиши: {0} \nKeyData: {1} \nKeyValue: {2}",
                                e->KeyCode, e->KeyData, e->KeyValue);
        }
    private: System::Void Form1_KeyUp(System::Object^  sender,
                                System::Windows::Forms::KeyEventArgs^  e)
        {
            // Очистка меток при освобождении клавиши
            label1->Text = String::Empty; label2->Text = String::Empty;
        }
};
}
```

В первую метку label1 записываем сведения о нажатой обычной (то есть не модифицирующей и не функциональной) клавише при обработке события KeyPress. Во вторую метку из аргумента события e (e->Alt, e->Shift и e->Control) получаем сведения, была ли нажата какая-либо модифицирующая клавиша (либо их комбинация). Обработчик события KeyUp очищает обе метки при освобождении клавиш.

На рис. 3.7 приведен фрагмент работы программы.

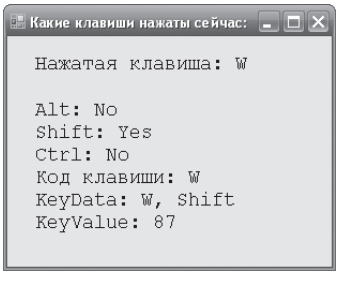

Рис. 3.7. Фрагмент работы программы, определяющей нажатую клавишу

Убедиться в работоспособности программы можно, открыв соответствующий sln-файл в папке СобытияКлавиатуры.

Пример 23. Разрешаем вводить в текстовое поле только цифры

Обычно для диагностики вводимых числовых данных мы пользовались функцией TryParse. Эта функция возвращает true, если на ее вход подаются числовые данные, и false в противном случае. Покажем, как можно совершенно по-другому решить задачу контроля вводимых пользователем данных. Можно вообще *не давать возможность пользователю вводить нечисловые данные*. Обратите внимание, как происходит ввод числовых данных в программе Калькулятор системы Windows, эта

программа просто не дает возможность пользователю ввести нечисловой символ. Продемонстрируем и мы такое решение на следующем примере.

Данная программа анализирует каждый символ, вводимый пользователем в текстовое поле формы. Если символ не является числовым, то текстовое поле получает запрет на ввод такого символа. Таким образом, программа *не дает* пользователю ввести нечисловые данные.

Запустим систему Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. На панели элементов Toolbox найдем текстовое поле TextBox и перетащим его в форму. Текст программы показан в листинге 3.7.

Листинг 3.7. Контроль вводимых пользователем числовых данных (вариант 1)

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа анализирует каждый символ, вводимый пользователем
        // в текстовое поле формы. Если символ не является цифрой или Backspace,
        // то текстовое поле получает запрет на ввод такого символа
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Введите число"; textBox1->Clear();
             }
    private: System::Void textBox1_KeyPress(System::Object^ sender,
                System::Windows::Forms::KeyPressEventArgs^ e)
             {
                // Разрешаем ввод только десятичных цифр и Backspace:
                if (Char::IsDigit(e->KeyChar) == true) return;
                if (e->KeyChar == (char)Keys::Back) return;
                // Запрет на ввод других вводимых символов:
                e->Handled = true;
             }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы мы задаем текст строки заголовка «Введите число» и очищаем текстовое поле TextBox1.Clear(). Самое интересное начинается при обработке события нажатия клавиши в текстовом поле textBox1_KeyPress. Пустой обработчик этого события мы получаем так же, как и в предыдущих программах, то есть на вкладке конструктора формы в панели свойств Properties, щелкнув на символе молнии (Events), выбираем в списке всех возможных событий событие KeyPress для текстового поля. Управляющая среда Visual Studio 2010 генерирует при этом пустую процедуру обработки данного события.

В этой процедуре можно легко определить, какую клавишу нажал пользователь, из аргумента события e. Символ, соответствующий нажатой клавише, содержится в свойстве аргумента e->KeyChar. На вход функции IsDigital подаем это свойство (то есть исследуемый символ), а на выходе получаем заключение, является ли исследуемый символ цифрой (true или false). Аргумент события e имеет замечательное свойство Handled, которое либо запрещает получение данного события текстовым полем (true), либо разрешает (false). Всякий раз при очередном выполнении процедуры textBox1_KeyPress изначально свойство e->Handled = false, то есть получение данного события текстовым полем разрешено. Последней строкой в процедуре запрещаем ввод символов в текстовое поле, но если пользователь вводит цифру или нажал клавишу Backspace, то этот запрет мы обходим с помощью оператора return. Таким образом, мы добиваемся игнорирования текстовым полем не цифровых символов.

Интерфейс рассматриваемого приложения показан на рис. 3.8.

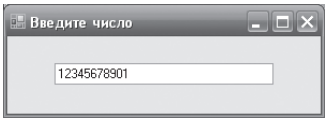

Рис. 3.8. Контроль введенных данных

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТолькоЦифры.

Пример 24. Разрешаем вводить в текстовое поле цифры, а также разделитель целой и дробной части числа

Мы совсем забыли, уважаемые читатели, что число, вводимое пользователем, может иметь дробную часть после точки или запятой. Причем выяснить, что именно установлено в вашей системе — точка или запятая в качестве разделителя целой и дробной частей числа, можно, например, запустив Калькулятор Windows. Здесь среди экранных кнопок увидим кнопку либо с десятичной точкой, либо с десятичной запятой. Очень легко поменять данную установку системы (обычно по умолчанию в русифицированной версии Windows — запятая). Для этого следует в Панели управления выбрать значок Язык и региональные стандарты, затем на вкладке Региональные параметры щелкнуть на кнопке Настройка и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей либо точку, либо запятую. Нам нужно добиться того, чтобы текстовое поле разрешало ввод только того разделителя, который указан на вкладке Региональные параметры.

Для решения данной задачи запустим систему Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. На панели элементов Toolbox найдем текстовое поле TextBox и перетащим его в форму. Текст программы представлен в листинге 3.8.

Листинг 3.8. Контроль вводимых пользователем числовых данных (вариант 2)

```
            // ..............
            // Программный код расположенный выше создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа разрешает ввод в текстовое поле только цифровых символов,
        // а также разделитель целой и дробной частей числа (то есть точки или
        запятой)
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Разделитель целой и дробной частей числа может быть
        // точкой "." или запятой "," в зависимости от
        // установок в пункте Язык и региональные стандарты
        // Панели управления ОС Windows:
        String^ ТчкИлиЗпт;
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Введите число";
                // Выясняем, что установлено на данном ПК в качестве
                // разделителя целой и дробной частей: точка или запятая
                ТчкИлиЗпт = Globalization::
                    NumberFormatInfo::CurrentInfo->NumberDecimalSeparator;
             }
    private: System::Void textBox1_KeyPress(System::Object^  sender,
                System::Windows::Forms::KeyPressEventArgs^  e)
             {
                bool ТчкИлиЗптНАЙДЕНА = false;
                // Разрешаю ввод десятичных цифр:
                if (Char::IsDigit(e->KeyChar) == true) return;
                // Разрешаю ввод <Backspace>:
                if (e->KeyChar == (char)Keys::Back) return;
                // Поиск ТчкИлиЗпт в textBox, если IndexOf() == -1, то не
                найдена:
                if (textBox1->Text->IndexOf(ТчкИлиЗпт) != -1)
                    ТчкИлиЗптНАЙДЕНА = true;
                // Если ТчкИлиЗпт уже есть в textBox, то запрещаем вводить и ее,
                // и любые другие символы:
                if (ТчкИлиЗптНАЙДЕНА == true) { e->Handled = true; return; }
                // Если ТчкИлиЗпт еще нет в textBox, то разрешаем ее ввод:
                if (e->KeyChar.ToString() == ТчкИлиЗпт) return;
                // В других случаях - запрет на ввод:
                e->Handled = true;
             }
    };
}
```

Как видно из текста программы, мы вначале выясняем, что установлено в данной системе в качестве разделителя целой дробной части — точка или запятая. Этот разделитель записываем в строковую переменную ТчкИлиЗпт, которая видна из всех процедур данной программы, поскольку объявлена вне всех процедур. Далее, как и в предыдущем примере, в процедуре обработки события KeyPress разрешаем ввод десятичных цифр и нажатие клавиши Backspace путем обхода с помощью return последнего оператора процедуры e->Handled = true, запрещающего ввод символа в текстовое поле.

В данной задаче мы имеем некоторую сложность с разрешением ввода разделителя целой и дробной частей, поскольку разрешить его ввод мы можем только один раз, но при этом надо помнить, что пользователь может его удалить и ввести в другом месте числовой строки. Эту проблему мы решили следующим образом: каждый раз при очередном нажатии клавиши, разрешив ввод десятичных цифр, в текстовом поле ищем искомый разделитель. Если он найден, то запрещаем ввод любых нецифровых символов, включая злосчастный разделитель. *А если не найден*, то разрешаем его ввод.

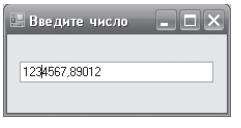

Рис. 3.9. Кроме цифр программа разрешает ввод десятичной запятой

На рис. 3.8 показан фрагмент работы программы.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТолькоЦифры+ТчкОrЗпт.

Пример 25. Программно вызываем событие «щелчок на кнопке»

Щелкая мышью на кнопке, мы вызываем это событие и можем обрабатывать его в соответствующей процедуре. В данной программе мы хотим показать, как можно программно вызвать это событие без щелчка на этой кнопке. Пусть у нас будет две командных кнопки на экранной форме. Щелкая на первой кнопке, мы программируем появление окна с сообщением о произошедшем событии нажатия первой кнопки. При этом щелкая на второй кнопке, мы имитируем нажатие первой кнопки путем программного вызова события нажатия первой кнопки.

Для решения этой задачи запустим систему Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. На панели элементов Toolbox нам понадобятся две командные кнопки. Текст программы представлен в листинге 3.9.

Листинг 3.9. Программный вызов события нажатия кнопки

```
        // ...............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->Text = L"Form1";
        this->ResumeLayout(false);
    }
```

продолжение ⇗

Листинг 3.9 (продолжение)

```
#pragma endregion
        // На экранной форме имеем две кнопки. Щелчок на первой кнопке вызывает
        // появление окна с сообщением о произошедшем событии нажатия первой
        // кнопки. Щелкая на второй кнопке, мы имитируем нажатие первой кнопки
        // путем программного вызова события нажатия первой кнопки
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 MessageBox::Show("Произошло событие \"щелчок на первой
                 кнопке\"");
             }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 // Здесь программно вызываем событие "щелчок на первой кнопке",
                 // хотя щелкнули на второй кнопке:
                 button1->PerformClick();
                 // То же самое можно сделать так:
                 // button1_Click(nullptr, nullptr);
                 // или так:
                 // button1_Click(button1, EventArgs::Empty);
             }
    };
}
```

При обработке события «щелчок на второй кнопке» мы вызвали событие «щелчок на первой кнопке». В комментарии указали варианты вызова этого события другими способами. Это открывает возможность использования приведенной методики и для других элементов управления, не имеющих прямого метода вызова соответствующего события наподобие PerformClick.

На рис. 3.10 приведен фрагмент работы программы.

Рис. 3.10. Щелкая на второй кнопке, вызываем событие нажатия первой кнопки

Убедиться в работоспособности программы можно, открыв соответствующее решение (sln-файл) в папке Пгм-ноВызватьКликКнопки.

4

Чтение, запись текстовых и бинарных файлов, текстовый редактор

Пример 26. Чтение/запись текстового файла в кодировке Unicode. Обработка исключений try...catch

Очень распространенной задачей является сохранение данных на диске в текстовом формате (не в двоичном). Понятно, что такое деление на текстовые и двоичные форматы условно, поскольку и текстовые, и не текстовые файлы на самом деле являются двоичными. Но если не текстовый файл открыть, например, Блокнотом, мы увидим то, что называют «нечитаемым месивом»; отсюда такая классификация. Часто данные сохраняют на диск именно в текстовом формате, поскольку в этом случае сохраненные данные можно читать, редактировать любым текстовым редактором, например Блокнотом или TextEdit. Также следует уметь читать текстовые данные в своей пользовательской программе.

Казалось бы, это — очень простая задача. Например, чтение текстового файла сводится буквально к нескольким строчкам:

```
// Создание экземпляра StreamReader для чтения из файла
IO::StreamReader^ Reader = gcnew IO::StreamReader("C:\\Text1.txt");
// Считывание содержимого текстового файла в строку
String^ Stroka = Reader->ReadToEnd();
Reader->Close();
```

Однако есть некоторые серьезные нюансы. Напишем программу, содержащую на экранной форме текстовое поле и две командные кнопки. При щелчке мышью на первой кнопке происходит чтение текстового файла в текстовое поле в кодировке Unicode. При щелчке на второй кнопке отредактированный пользователем текст в текстовом поле сохраняется в файл на диске.

Запустим систему Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Далее в форму из панели Toolbox перенесем текстовое поле и две командные копки. Для текстового поля в окне Properties сразу укажем для свойства Multiline значение True, чтобы текстовое поле имело не одну строку, а столько, сколько поместится в растянутом указателем мыши поле. Одна кнопка предназначена для открытия файла, а другая — для сохранения файла на машинном носителе. В листинге 4.1 приведен текст данной программы.

Листинг 4.1. Чтение/запись текстового файла в кодировке Unicode

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа для чтения/записи текстового файла в кодировке Unicode
        String ^ filename;
        // Объявляем filename здесь, чтобы эта переменная была "видна"
        // в процедурах обработки обоих событий.
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 // Установка начальных значений:
                 textBox1->Multiline = true; textBox1->Clear();
                 textBox1->Size = Drawing::Size(268, 112);
                 button1->Text = "Открыть"; button1->TabIndex = 0;
                 button2->Text = "Сохранить";
                 Form1::Text = "Здесь кодировка Unicode";
                 filename = "C:\\Text1.txt";
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 // Щелчок на кнопке Открыть.
                 // Русские буквы будут корректно читаться,
                 // если открыть файл в кодировке UNICODE:
                 try
                 {
                     // Создание объекта StreamReader для чтения из файла:
                     auto Читатель = gcnew IO::StreamReader(filename);
                     // Непосредственное чтение всего файла в текстовое поле:
                     textBox1->Text = Читатель->ReadToEnd();
                     Читатель->Close(); // закрытие файла
                     // Читать текстовый файл в кодировке UNICODE в массив строк
                     // можно также таким образом (без Open и Close):
                     //  array <String^>^ МассивСтрок =
```

```
                //      IO::File::ReadAllLines("C:\\Text1.txt");
            }
            catch (IO::FileNotFoundException^ Ситуация)
            {   // Обработка исключительной ситуации:
                MessageBox::Show(Ситуация->Message + «\nНет такого файла»,
                                 "Ошибка", MessageBoxButtons::OK,
                                           MessageBoxIcon::Exclamation);
            }
            catch (Exception^ Ситуация)
            {
                // Отчет о других ошибках:
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Щелчок на кнопке Сохранить:
            try
            {
                // Создание объекта StreamWriter для записи в файл:
                auto Писатель = gcnew
                                  IO::StreamWriter(filename, false);
                Писатель->Write(textBox1->Text);
                Писатель->Close();
                // Сохранить текстовый файл можно также таким образом
                // (без Close), причем, если файл уже существует,
                // то он будет заменен:
                // IO::File::WriteAllText("C:\\tmp.tmp", textBox1->Text);
            }
            catch (Exception^ Ситуация)
            {
                // Отчет обо всех возможных ошибках:
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    };
}
```

Несколько слов о блоках try, которые, как мы видим, используются в данном программном коде. Логика использования try следующая: *попытаться* (try) выполнить некоторую задачу, например прочитать файл. Если задача решена некорректно (например, файл не найден), то *«перехватить»* (catch) управление и *обработать* возникшую (*исключительную*, Exception) ситуацию. Как видно из текста программы, обработка исключительной ситуации свелась к информированию пользователя о недоразумении.

При обработке события «щелчок на кнопке Открыть» организован ввод файла C:\Text1.txt. Обычно в этой ситуации пользуются элементом управления OpenFileDialog для выбора файла. Мы не стали использовать этот элемент управления для того, чтобы *не «заговорить»* проблему, а также свести к минимуму программный код.

Далее создаем объект (поток) Читатель для чтения из файла. Для большей выразительности операций с данным объектом мы назвали его русскими буквами. Затем следует чтение файла filename методом ReadToEnd() в текстовое поле textBox1.Text и закрытие файла методом Close().

При обработке события «щелчок на кнопке Сохранить» организована запись файла на диск аналогично через объект Писатель. При создании объекта Писатель первым аргументом является filename, а второй аргумент false указывает, что данные следует *не добавить* (append) к содержимому файла (если он уже существует), а *перезаписать* (overwrite). Запись на диск производится с помощью метода Write() из свойства Text элемента управления textBox1. На рис. 4.1 приведен фрагмент работы программы.

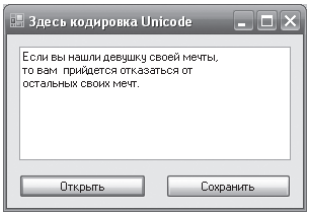

Рис. 4.1. Чтение/запись текстового файла в кодировке Unicode

Сделаем очень важное примечание. Запись текстового файла с помощью данной программы будет происходить *в формате (кодировке) Unicode*, как и чтение из файла. То есть вы сможете читать эти файлы Блокнотом, редактировать их, но каждый раз при сохранении файлов следить, чтобы кодировка была (оставалась) Unicode.

Однако по умолчанию в редакторах обычно используется кодировка ANSI. Кодировку ANSI с русскими буквами называют Windows 1251. Некоторые редакторы, например RPad («русский» Блокнот), вообще не работают с Unicode. Таким образом, на сегодняшний день пока что записывать в кодировке Unicode — это как бы «эксклюзив». Возможно, даже наверняка, в ближайшее время многое изменится в пользу Unicode, поскольку информационные технологии меняются очень быстро.

Если в Блокноте подготовить текстовый файл в обычной кодировке ANSI, то прочитать русские буквы данной программой не получится, хотя английские буквы отобразятся в текстовом поле *без проблем*. Почему? Дело в том, что приведенная в тексте данной программы технология описана в учебных пособиях по Visual C++, в MSDN, на сайтах, подобных http://msdn.microsoft.com/library/rus/. Однако чаще это все англоязычные источники, а для английских текстов переход от одной кодировки к другой оказывается практически незаметным. Например, английские буквы

кодировок Windows 1251, ASCII и Unicode совпадают, а с русскими буквами всегда возникают недоразумения. Эта проблема не раз обсуждалась программистами на различных форумах в Интернете.

Подобные недоразумения вам следует уметь учитывать. Разрешению *этого недоразумения* посвящена программа в следующем разделе.

Убедиться в работоспособности данной программы можно, открыв решение TxtUnicode.sln в папке TxtUnicode.

Пример 27. Чтение/запись текстового файла в кодировке Windows 1251

В данном примере также на экранной форме имеем текстовое поле и две командные кнопки, назначение этих элементов управления такое же, как и в предыдущем примере. Однако чтение и запись текстового файла в этом примере происходит в кодировке Windows 1251 (ANSI). Поскольку структура данной программы аналогична структуре предыдущей, сразу обратимся к ее коду в листинге 4.2.

Листинг 4.2. Чтение/запись текстового файла в кодировке Windows 1251

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа для чтения/записи текстового файла в кодировке Windows 1251
        String^ filename;
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 textBox1->Multiline = true; textBox1->Clear();
                 textBox1->Size = Drawing::Size(268, 112);
                 button1->Text = "Открыть"; button1->TabIndex = 0;
                 button2->Text = "Сохранить";
                 this->Text = "Здесь кодировка Windows 1251";
                 filename = "C:\\Text2.txt";
             }
    private: System::Void button1_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 // Щелчок на кнопке Открыть
                 try
                 {
                     // Чтобы русские буквы читались корректно, объявляем
```

продолжение ⇩

Листинг 4.2 (продолжение)

```
                    // объект Кодировка:
                    System::Text::Encoding^ Кодировка =
                        System::Text::Encoding::GetEncoding(1251);
                    // Создание экземпляра StreamReader для чтения из файла
                    IO::StreamReader^ Читатель = gcnew
                        IO::StreamReader(filename, Кодировка);
                    textBox1->Text = Читатель->ReadToEnd();
                    Читатель->Close();
                    // Читать текстовый файл в кодировке Windows 1251
                    // в массив строк
                    // можно также таким образом (без Open и Close):
                    // array <String^>^ МассивСтрок =
                    //    IO::File::ReadAllLines("C:\\Text2.txt", Кодировка);
                }
                catch (IO::FileNotFoundException^ Ситуация)
                {  // Обработка исключительной ситуации:
                    MessageBox::Show(Ситуация->Message + «\nНет такого файла»,
                        "Ошибка", MessageBoxButtons::OK,
                        MessageBoxIcon::Exclamation);
                }
                catch (Exception^ Ситуация)
                {  // Отчет о других ошибках
                    MessageBox::Show(Ситуация->Message, "Ошибка",
                        MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
                }
            }
    private: System::Void button2_Click(System::Object^  sender,
                System::EventArgs^  e)
            {
                // Щелчок на кнопке Сохранить:
                try
                {
                    auto Кодировка =
                        System::Text::Encoding::GetEncoding(1251);
                    // Создание экземпляра StreamWriter для записи в файл:
                    auto Писатель = gcnew
                        IO::StreamWriter(filename, false, Кодировка);
                    Писатель->Write(textBox1->Text);
                    Писатель->Close();
                    // Сохранить текстовый файл можно также таким образом (без
                    // Close), причем если файл уже существует, то он будет
                    // заменен:
                    // IO::File::WriteAllText("C:\\tmp.tmp",
                    //                        textBox1->Text, Кодировка);
                }
                catch (System::Exception^ Ситуация)
                {
                    // Отчет обо всех возможных ошибках:
```

```
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    };
}
```

Этот текст программы отличается от предыдущего лишь тем, что здесь введен новый объект — Кодировка. Метод GetEncoding(1251) устанавливает кодовую страницу Windows 1251 для объекта Кодировка. Можно убедиться в этом, если распечатать свойство Кодировка->HeaderName.

При создании объекта Читатель используются уже два аргумента: *имя файла* filename и объект Кодировка, указывающий, в какой кодировке (для какой кодовой страницы) читать данные из текстового файла. А при создании объекта Писатель используются три аргумента: имя файла filename, установка false (для случая, если файл уже существует, нужно будет не добавлять новые данные, а создавать новый файл) и объект Кодировка, указывающий, в какой кодировке писать данные в файл.

Заметьте, что при обработке события «щелчок на второй кнопке» мы объявили переменные Кодировка и Писатель как auto, то есть типы этих переменных выводятся из выражения инициализации (как объявление var в C#) — это новая возможность в Visual C++ 2010.

Убедиться в работоспособности программы можно, открыв решение TXT_1251.sln в папке TXT_1251.

Пример 28. Простой текстовый редактор. Открытие и сохранение файла. Событие формы Closing

Итак, мы уже знаем, что существуют технологии чтения/записи текстового файла для нужной кодовой страницы. Таким образом, мы имеем основные компоненты *для написания текстового редактора*.

Запустим систему Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели элементов управления Toolbox перенесем в форму текстовое поле TextBox и меню MenuStrip. Используя элемент управления MenuStrip, создадим один пункт меню Файл и три пункта подменю: Открыть, Сохранить как и Выход (рис. 4.2).

Из панели Toolbox нам понадобятся еще элементы управления OpenFileDialog и SaveFileDialog, также перенесем их в форму, хотя их очень легко объявить и использовать непосредственно в программном коде. Чтобы растянуть текстовое поле на всю форму, в свойстве Multiline укажем True (разрешим введение множества строк).

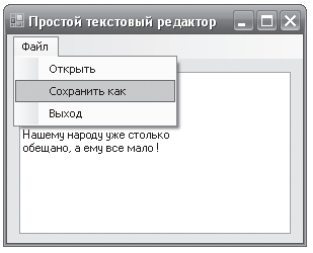

Рис. 4.2. Простой текстовый редактор

Итоговый текст программы «Простой текстовый редактор» представлен в листинге 4.3, и автор заранее приносит свои извинения за слишком длинный программный код, однако короче уже нельзя! Кстати, чтобы увеличить количество строчек программного кода, которое вы хотели бы одновременно видеть на экране, удобно пользоваться комбинацией клавиш Shift+Alt+Enter (Full Screen).

Листинг 4.3. Простой текстовый редактор

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Простой текстовый редактор
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                textBox1->Multiline = true; textBox1->Clear();
                textBox1->Size = Drawing::Size(268, 160);
                this->Text = "Простой текстовый редактор";
                openFileDialog1->FileName = "C:\\Text2.txt";
                openFileDialog1->Filter =
                    "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
                saveFileDialog1->Filter =
                    "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
             }
    private: System::Void открытьToolStripMenuItem_Click(System::Object^ sender,
                                                         System::EventArgs^  e)
             {
                // Вывести диалог открытия файла
                openFileDialog1->ShowDialog();
                if (openFileDialog1->FileName == nullptr) return;
                // Чтение текстового файла:
                try
                {  // Создание экземпляра StreamReader для чтения из файла
                    auto Читатель = gcnew
                        IO::StreamReader(openFileDialog1->FileName,
```

```
                    System::Text::Encoding::GetEncoding(1251));
                // - здесь заказ кодовой страницы Win1251 для русских букв
                textBox1->Text = Читатель->ReadToEnd();
                Читатель->Close();
            }
            catch (IO::FileNotFoundException^ Ситуация)
            {
                MessageBox::Show(Ситуация->Message + "\nНет такого файла",
                    "Ошибка", MessageBoxButtons::OK,
                              MessageBoxIcon::Exclamation);
            }
            catch (Exception^ Ситуация)
            {
                // Отчет о других ошибках
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK,
                    MessageBoxIcon::Exclamation);
            }
        }
private: System::Void сохранитьКакToolStripMenuItem_Click(
                     System::Object^ sender, System::EventArgs^ e)
        {
            // Пункт меню Сохранить как
            saveFileDialog1->FileName = openFileDialog1->FileName;
            if (saveFileDialog1->ShowDialog() ==
                     Windows::Forms::DialogResult::OK) Запись();
        }
        void Запись()
        {
            try
            {
                // Создание экземпляра StreamWriter для записи в файл:
                auto Писатель = gcnew
                    IO::StreamWriter(saveFileDialog1->FileName, false,
                    System::Text::Encoding::GetEncoding(1251));
                // - здесь заказ кодовой страницы Win1251 для русских букв
                Писатель->Write(textBox1->Text);
                Писатель->Close(); textBox1->Modified = false;
            }
            catch (Exception^ Ситуация)
            {
                // Отчет обо всех возможных ошибках
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK,
                    MessageBoxIcon::Exclamation);
            }
        }
private: System::Void выходToolStripMenuItem_Click(System::Object^ sender,
                                                   System::EventArgs^ e)
```

продолжение ⇩

Листинг 4.3 (продолжение)

```
            {
                this->Close();
            }
    private: System::Void Form1_FormClosing(System::Object^  sender,
                System::Windows::Forms::FormClosingEventArgs^  e)
            { // Обработка момента закрытия формы:
                if (textBox1->Modified == false) return;
                // Если текст модифицирован, то спросить, записывать ли файл?
                auto MBox = MessageBox::Show(
                    "Текст был изменен. \nСохранить изменения?",
                    "Простой редактор", MessageBoxButtons::YesNoCancel,
                                        MessageBoxIcon::Exclamation);
                // YES – диалог;  NO – выход;  CANCEL - редактировать
                if (MBox == Windows::Forms::DialogResult::No) return;
                if (MBox == Windows::Forms::DialogResult::Cancel) e->Cancel =
true;
                if (MBox == Windows::Forms::DialogResult::Yes)
                {
                    if (saveFileDialog1->ShowDialog() ==
                             Windows::Forms::DialogResult::OK)
                    {
                        Запись(); return;
                    }
                    else e->Cancel = true; // Передумал выходить из ПГМ
                } // DialogResult::Yes
            }
    };
}
```

Как видно их текста программы, при обработке события загрузки формы присваиваем начальные значения некоторым переменным. В частности, для открытия (Open) и сохранения (Save) файлов заказываем фильтр (Filter) для текстовых файлов *.txt, а также для всех файлов *.*.

При обработке события «щелчок на пункте меню Открыть» выводим стандартный диалог открытия файлов OpenFileDialog, и если полученное в результате диалога имя файла не пусто (nullptr), то организуем чтение текстового файла. Эта процедура в точности соответствует процедуре чтения файла из предыдущего раздела, разве что упрощен заказ на чтение в кодировке Windows 1251.

Аналогично написана обработка события «щелчок на пункте меню Сохранить как» (см. рис. 4.2). Выводится стандартный диалог SaveFileDialog сохранения файла, и если пользователь щелкнул на кнопке OK (или Сохранить), то вызывается процедура Запись(). Если нет, то пользователь отправляется редактировать текст. Процедура Запись() также полностью соответствует процедуре записи текстового файла из предыдущего раздела. Как видно, в процедуре Запись() попытка (Try) записи заканчивается оператором textBox1->Modified = false. Свойство Modified отслеживает изменения в тестовом поле. Понятно, что сразу после записи в файл следует это свойство перевести в состояние false.

На мой взгляд, наибольший интерес представляет организация выхода из программы. Выйти из программы можно либо через пункт меню Выход (см. процедуру обработки события, начинающуюся со слова «выход» в листинге 3.3), либо закрывая программу традиционными методами Windows, то есть нажатием комбинации клавиш Alt+F4, кнопки Закрыть или кнопки системного меню (слева вверху) (обработка *события закрытия формы* FormClosing). Момент закрытия формы отслеживаем с помощью события формы FormClosing, которое происходит во время закрытия формы. Обратите внимание: выход по пункту меню Выход организован не через метод Application::Exit(), а через закрытие формы this->Close(). Почему? Потому что метод Application::Exit() не вызывает событие формы FormClosing.

Обычно выход из любого редактора (текстового, графического, табличного и т. д.) реализуется по следующему алгоритму:

1. Изменения в тестовом поле регистрируются свойством textBox1->Modified.
2. Если пользователь не сделал никаких изменений в редактируемом файле, то программа просто завершает работу.
3. Если в документе имеются несохраненные изменения (textBox1->Modified = true), а пользователь хочет выйти из программы, то выводится диалоговое окно (рис. 4.3).

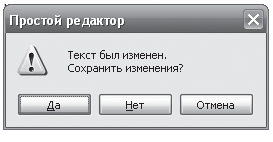

Рис. 4.3. Диалоговое окно при выходе из программы

В программе следует обработать каждый из возможных ответов пользователя по алгоритму, представленному на рис. 4.4.

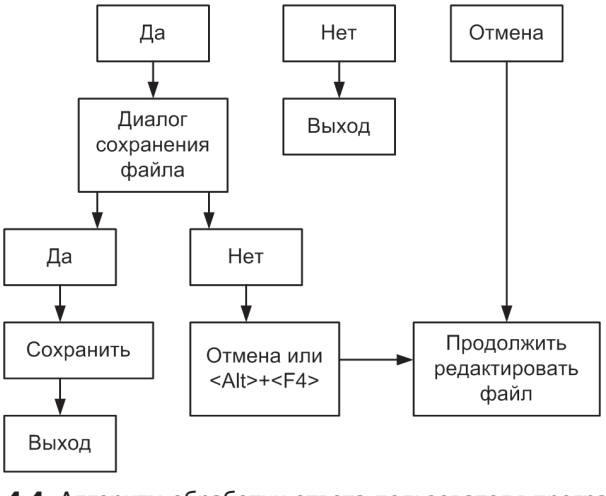

Рис. 4.4. Алгоритм обработки ответа пользователя программы

Обращаю внимание читателей *на ветвь алгоритма «Отмена»* (Cancel). Это случай, когда пользователь передумал выходить из программы и желает вернуться к редактированию файла. Для реализации этого случая (см. листинг 4.3) обработка события FormClosing предусматривает булево свойство e->Cancel, которому можно присвоить значение true, означающее отказ от закрытия программы (пользователь передумал), то есть в этом случае процедура Form1_FormClosing не закончится *выходом из программы*.

Аналогично, если пользователь согласился сохранить данные, то он попадает в стандартный диалог сохранения файла, и если при этом он передумал (диалог сохранения закрыт кнопкой Отмена или комбинацией клавиш Alt+F4), то следует предложить пользователю продолжить редактирование файла: e->Cancel = true. Как видно, в процедуре Form1_FormClosing, к сожалению, не удается избежать вложенных операторов условия.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТекстовыйРедактор.

Пример 29. Программа тестирования знаний студента по какому-либо предмету

В связи с внедрением в образование так называемого Болонского процесса — процесса сближения и гармонизации систем образования стран Европы с целью создания единого европейского пространства высшего образования — процедура проверки знаний студентов осуществляется, в том числе, с помощью тестирования по различным предметам преподавания. Причем тестированию уделяется все большее внимание. В данном примере создадим инструмент для тестирования студентов, напишем программу, которая читает заранее подготовленный преподавателем текстовый файл с вопросами по какому-либо предмету, выводит в экранную форму каждый вопрос с вариантами ответов. Студент выбирает правильный вариант ответа, а в конце тестирования программа подводит итоги проверки знаний, выставляет общую оценку и, в качестве обоснования поставленной оценки, показывает вопросы, на которые студент ответил неправильно.

Фрагмент такого текстового файла для проверки знаний по информатике представлен в листинге 4.4.

Листинг 4.4. Содержимое текстового файла для тестирования студента по информатике

```
Информатика и программирование
1/6. Основные компоненты вычислительной системы:
процессор, звуковая карта, видеокарта, монитор, клавиатура;
монитор, клавиатура, материнская плата, процессор
процессор, ОП, внешняя память, монитор, клавиатура
3
2/6. Во время исполнения прикладная программа хранится:
в ПЗУ
в процессоре
в оперативной памяти
```

```
3
3/6. Иерархию усложнения данных можно представить в виде:
Бит - Байт - Поле - Запись - Файл - База Данных
Запись - Файл - Бит - Байт - База Данных - Поле
База Данных - Байт - Поле - Запись - Бит - Файл
1
4/6. Укажите строку, содержащую неверное утверждение
1 Кбайт = 1024 байт; 1 Гбайт = 1024 Мбайт
1 Мбайт это примерно миллион байт; 1 байт = 8 бит
1 Гбайт это примерно миллион байт; 1 Мбайт = 1024 Кбайт
3
5/6. Экспоненциальное представление числа -1,84E-04 соответствует числу:
-0,000184
-0,00184
-18400
1
6/6. Текстовые данные кодируют с использованием:
таблиц размещения файлов FAT, NTFS и др.
таблиц символов Windows 1251, Unicode, ASCII и др.
структурированного языка запросов SQL
2
```

Структура этого файла следующая: первой строкой приведенного текстового файла является название предмета или темы, по которой проводится тестирование. Это название программа будет выводить в строке заголовка экранной формы. Для краткости изложения в данном тесте приводятся только шесть вопросов. На компакт-диске, прилагаемом к данной книге, представлен более полный файл test_полный.txt, с помощью которого автор тестирует знания своих студентов. Как видно из листинга 4.4, каждый вопрос имеет три варианта ответа. Строка с числом 1, 2 или 3 означает номер правильного ответа. Программа будет считывать это число и сравнивать с номером варианта, который выбрал тестируемый. Дробь, например 4/6, приведена для того, чтобы тестируемый понимал, в какой точке траектории в данный момент находится, то есть он отвечает на четвертый вопрос, а всего их шесть.

Таким образом, задача понятна, приступаем к ее программированию. Запустим систему Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов перенесем в форму две командные кнопки, текстовую метку и три переключателя RadioButton. Далее щелчком правой кнопкой мыши перейдем к вкладке программного кода (листинг 4.5).

Листинг 4.5. Программа тестирования знаний студента

```
        // ..............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
        this->PerformLayout();
    }
```

продолжение ⇨

Листинг 4.5 (продолжение)

```
#pragma endregion
        // Программа тестирует студента по какому-либо предмету обучения
        int СчетВопросов;    // Счет вопросов
        int ПравилОтветов;   // Количество правильных ответов
        int НеПравилОтветов; // Количество не правильных ответов
        // Массив вопросов, на которые даны неправильные ответы:
        array<String^>^ НеПравилОтветы;
        int НомерПравОтвета; // Номер правильного ответа
        int ВыбранОтвет;     // Номер ответа, выбранный студентом
        IO::StreamReader^ Читатель;
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 button1->Text = "Следующий вопрос";
                 button2->Text = "Выход";
                 // Подписка на событие "изменение состояния
                 // переключателей RadioButton:"
                 radioButton1->CheckedChanged +=
                     gcnew EventHandler(this, &Form1::ИзмСостПерекл);
                 radioButton2->CheckedChanged +=
                     gcnew EventHandler(this, &Form1::ИзмСостПерекл);
                 radioButton3->CheckedChanged +=
                     gcnew EventHandler(this, &Form1::ИзмСостПерекл);
                 НачалоТеста();
             }
             void НачалоТеста()
             {
                 System::Text::Encoding^ Кодировка =
                     System::Text::Encoding::GetEncoding(1251);
                 try
                 {  // Создание экземпляра StreamReader для чтения из файла
                     Читатель = gcnew IO::StreamReader(
                                     IO::Directory::GetCurrentDirectory()
                                                 + "\\test.txt", Кодировка);
                     this->Text = Читатель->ReadLine(); // Название предмета
                     // Обнуление всех счетчиков:
                     СчетВопросов = 0; ПравилОтветов = 0; НеПравилОтветов = 0;
                     НеПравилОтветы = gcnew array<String^>(100);
                 }
                 catch (Exception^ Ситуация)
                 {  // Отчет о всех ошибках
                     MessageBox::Show(Ситуация->Message, "Ошибка",
                         MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
                 }
                 ЧитатьСледВопрос();
             }
             void ЧитатьСледВопрос()
             {
                 label1->Text = Читатель->ReadLine();
```

```
            // Считывание вариантов ответа:
            radioButton1->Text = Читатель->ReadLine();
            radioButton2->Text = Читатель->ReadLine();
            radioButton3->Text = Читатель->ReadLine();
            // Выясняем, какой ответ - правильный:
            НомерПравОтвета = int::Parse(Читатель->ReadLine());
            // Переводим все переключатели в состояние "выключено":
            radioButton1->Checked = false; radioButton2->Checked = false;
            radioButton3->Checked = false;
            // Первую кнопку задаем не активной, пока студент не выберет
            // вариант ответа:
            button1->Enabled = false;
            СчетВопросов = СчетВопросов + 1;
            // Проверка, конец ли файла:
            if (Читатель->EndOfStream == true) button1->Text = "Завершить";
        }
private: Void ИзмСостПерекл(System::Object^  sender, System::EventArgs^  e)
        {  // Кнопка "Следующий вопрос" становится активной и ей
           // передаем фокус:
            button1->Enabled = true; button1->Focus();
            RadioButton^ Переключатель = (RadioButton^)sender;
            String^ tmp = Переключатель->Name;
            // Выясняем номер ответа, выбранный студентом:
            ВыбранОтвет = int::Parse(tmp->Substring(11));
        }
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
        {
            // Щелчок на кнопке
            // "Следующий вопрос/Завершить/Начать тестирование снач"
            // Счет правильных ответов:
            if (ВыбранОтвет == НомерПравОтвета)
                ПравилОтветов = ПравилОтветов + 1;
            if (ВыбранОтвет != НомерПравОтвета)
            {  // Счет неправильных ответов:
                НеПравилОтветов = НеПравилОтветов + 1;
               // Запоминаем вопросы с неправильными ответами:
                НеПравилОтветы[НеПравилОтветов] = label1->Text;
            }
            if (button1->Text == "Начать тестирование сначала")
            {
                button1->Text = "Следующий вопрос";
                // Переключатели становятся видимыми, доступными для выбора:
                radioButton1->Visible = true; radioButton2->Visible = true;
                radioButton3->Visible = true;
                // Переход к началу файла:
                НачалоТеста(); return;
            }
            if (button1->Text == "Завершить")
```

продолжение ⇘

Листинг 4.5 (продолжение)

```
                {  // Закрываем текстовый файл:
                    Читатель->Close();
                    // Переключатели делаем невидимыми:
                    radioButton1->Visible = false; radioButton2->Visible =
                    false;
                    radioButton3->Visible = false;
                    // Формируем оценку за тест:
                    label1->Text = String::Format("Тестирование завершено.\n" +
                        "Правильных ответов: {0} из {1}.\n" +
                        "Оценка в пятибальной системе: {2:F2}.", ПравилОтветов,
                        СчетВопросов, (ПравилОтветов * 5.0F) / СчетВопросов);
                    // 5F - это максимальная оценка
                    button1->Text = "Начать тестирование сначала";
                    // Вывод вопросов, на которые вы дали неправильный ответ
                    String^ Str = "СПИСОК ВОПРОСОВ, НА КОТОРЫЕ ВЫ ДАЛИ " +
                        "НЕПРАВИЛЬНЫЙ ОТВЕТ:\n\n";
                    for (int i = 1; i <= НеПравилОтветов; i++)
                        Str = Str + НеПравилОтветы[i] + "\n";
                    // Если есть неправильные ответы, то вывести через
                    // MessageBox список соответствующих вопросов:
                    if (НеПравилОтветов != 0)
                        MessageBox::Show(Str, "Тестирование завершено");
                }
                if (button1->Text == "Следующий вопрос") ЧитатьСледВопрос();
            }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
            {
                // Щелчок на кнопке "Выход"
                this->Close();
            }
    };
}
```

В программе есть несколько переменных, которые объявлены в начале вне всех процедур, чтобы они были «видны» из всех процедур класса Form1. В процедуре обработки загрузки формы организуем подписку на событие «изменение состояния переключателей» RadioButton одной процедурой ИзмСостПерекл. В данной программе изменение состояния любого из трех переключателей будем обрабатывать одной процедурой ИзмСостПерекл.

Далее в процедуре НачалоТеста открываем файл test.txt, в котором содержится непосредственно тест, и читаем первую строку с названием предмета или темы, подлежащей тестированию. При этом обнуляем счетчик всех вопросов и счетчики вопросов, на которые студент дал правильные и неправильные ответы. Затем вызываем процедуру ЧитатьСледВопрос, которая читает очередной вопрос, варианты ответов на него и номер варианта правильного ответа. Тут же проверяем, не достигнут ли конец читаемого программой файла. Если достигнут, то меняем надпись на первой кнопке на «Завершить». В данной программе надпись на первой кнопке

является как бы флагом, который указывает, по какой ветви в программе следует передавать управление.

Выбирая тот или иной вариант, сомневающийся студент может сколь угодно раз щелкать на разных переключателях, пока не выберет окончательно вариант ответа. Программа будет фиксировать выбранный вариант только на этапе щелчка на кнопке Следующий вопрос. В процедуре обработки события «изменение состояния переключателей» выясняем, какой из вариантов ответа выбрал студент, но делаем вывод, правильно ли ответил студент или нет, только при обработке события «щелчок на первой кнопке».

В процедуре обработки события «щелчок на первой кнопке» ведем счет правильных и неправильных ответов, а также запоминаем в строковый массив вопросы, на которые студент дал неверный ответ. Если достигнут конец файла и на кнопке появилась надпись «Завершить», то закрываем текстовый файл, все переключатели делаем невидимыми (уже выбирать нечего) и формируем оценку за прохождение теста, а также через MessageBox выводим список вопросов, на которые испытуемый дал неправильный ответ.

Фрагмент работы тестирующей программы представлен на рис. 4.5.

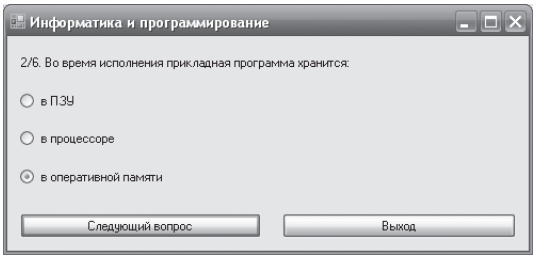

Рис. 4.5. Интерфейс тестирующей программы

На рис. 4.6 показан финальный фрагмент работы тестирующей программы, где выведено обоснование оценки тестирования со списком вопросов, на которые студент ответил неправильно.

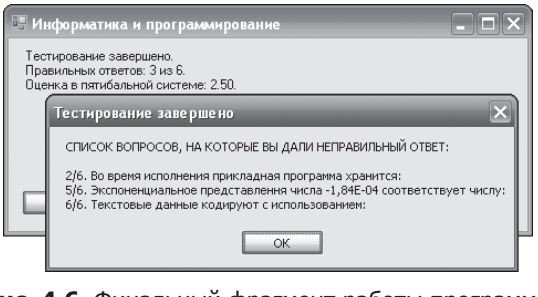

Рис. 4.6. Финальный фрагмент работы программы

Вы можете усовершенствовать данную программу, например, добавив элемент управления Timer, чтобы ограничить время сдачи теста. Можно в качестве исходных

данных для этой программы использовать не текстовый файл, который может обеспечить только текстовое представление, а файл HTML. Такой файл может содержать не только тексты, но и изображения для тестирования студентов.

Убедиться в работоспособности программы можно, открыв решение Тестирование.sln в папке Тестирование.

Пример 30. Простой RTF-редактор

Читателю, вероятно, известно, что Visual C++ 2010 имеет элемент управления RichTextBox (форматированное текстовое поле). Этот элемент управления позволяет осуществлять форматирование текста в *стандарте RTF* (один из форматов MS Word). В формате RTF в текст вводятся специальные коды форматирования, несущие информацию о гарнитуре, размерах шрифтов, стилях символов и абзацев, выравнивании и других возможностях форматирования.

Напишем очень простой RTF-редактор, который читает как RTF-файлы, так и обычные текстовые файлы в кодировке Windows 1251, но сохраняет файлы *на диск в любом случае в формате RTF. Для этой цели перенесем из панели элементов управления* Toolbox элементы управления RichTextBox, меню MenuStrip, SaveFileDialog и OpenFileDialog. В раскрывающемся меню Файл предусмотрим такие пункты меню, как показано на рис. 4.7: Открыть в формате RTF, Открыть в формате Win1251, Сохранить в формате RTF и Выход. Текст программы приведен в листинге 4.6.

Листинг 4.6. Простой RTF-редактор

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа простейшего RTF-редактора
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                Form1::Text = "Простой RTF-редактор"; richTextBox1->Clear();
                openFileDialog1->FileName = "c:\\Text2.txt";
                saveFileDialog1->Filter = "Файлы RTF (*.RTF)|*.RTF";
                открытьВФорматеRTFToolStripMenuItem->Click += gcnew
                    EventHandler(this, &Form1::ОТКРЫТЬ);
                открытьВФорматеWin1251ToolStripMenuItem->Click += gcnew
                    EventHandler(this, &Form1::ОТКРЫТЬ);
             }
    private: Void ОТКРЫТЬ(System::Object^ sender, System::EventArgs^ e)
             { // Процедура обработки событий открытия
               // файла в двух разных форматах.
                // Выясняем, в каком формате открыть файл:
                ToolStripMenuItem^ t = (ToolStripMenuItem^)sender;
```

```
            // Читаем надпись на пункте меню:
            String^ Формат = t->Text;
            try
            {  // Открыть в каком-либо формате:
                if (Формат == "Открыть в формате RTF")
                {
                    openFileDialog1->Filter =
                                    "Файлы RTF (*.RTF)|*.RTF";
                    openFileDialog1->ShowDialog();
                    if (openFileDialog1->FileName == nullptr) return;
                    richTextBox1->LoadFile(openFileDialog1->FileName);
                }
                if (Формат == "Открыть в формате Win1251")
                {
                    openFileDialog1->Filter =
                                    "Текстовые файлы (*.txt)|*.txt";
                    openFileDialog1->ShowDialog();
                    if (openFileDialog1->FileName == nullptr) return;
                    richTextBox1->LoadFile(openFileDialog1->FileName,
                                   RichTextBoxStreamType::PlainText);
                }
                richTextBox1->Modified = false;
            }
            catch (IO::FileNotFoundException^ Ситуация)
            {
                MessageBox::Show(Ситуация->Message +
                    "\nНет такого файла", "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
            catch (Exception^ Ситуация)
            {
                // Отчет о других ошибках
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
private: System::Void сохранитьВФорматеRTFToolStripMenuItem_Click(
                        System::Object^ sender, System::EventArgs^ e)
        {
            saveFileDialog1->FileName = openFileDialog1->FileName;
            if (saveFileDialog1->ShowDialog() ==
                            Windows::Forms::DialogResult::OK) Запись();
        }
        Void Запись()
        {
            try
            {
                richTextBox1->SaveFile(saveFileDialog1->FileName);
                richTextBox1->Modified = false;
            }
```

продолжение ⇩

Листинг 4.6 (продолжение)

```
            catch (Exception^ Ситуация)
            {  // Отчет обо всех возможных ошибках:
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    private: System::Void выходToolStripMenuItem_Click(System::Object^ sender,
                                                       System::EventArgs^ e)
        {
            this->Close();
        }
    private: System::Void Form1_FormClosing(System::Object^  sender,
            System::Windows::Forms::FormClosingEventArgs^  e)
        { // Обработка момента закрытия формы:
            if (richTextBox1->Modified == false) return;
            // Если текст модифицирован, то выясняем, записывать ли файл?
            auto MBox = MessageBox::Show(
                "Текст был изменен. \nСохранить изменения?",
                "Простой редактор", MessageBoxButtons::YesNoCancel,
                                    MessageBoxIcon::Exclamation);
            // YES – диалог;  NO – выход;  CANCEL - редактирование
            if (MBox==Windows::Forms::DialogResult::No) return;
            if (MBox==Windows::Forms::DialogResult::Cancel) e->Cancel =
            true;
            if (MBox==Windows::Forms::DialogResult::Yes)
            {
                if (saveFileDialog1->ShowDialog() ==
                                    Windows::Forms::DialogResult::OK)
                { Запись(); return; }
                else e->Cancel = true; // Передумал выходить из ПГМ
            } // - DialogResult::Yes
        }
    };
}
```

Структура этой программы аналогична структуре программы простого текстового редактора, рассмотренного выше. В процедуре обработки события загрузки формы задаем начальные значения некоторым переменным (см. текст программы). Здесь же выполняем подписку на обработку одной процедурой ОТКРЫТЬ двух событий — выбор пунктов меню Открыть в формате RTF и Открыть в формате Win1251. Подробно особенности обработки событий создаваемых разными объектами обсуждались в примерах 18, 19 и др. (см. главу 3). Из этих примеров читатели знают, что сведения об объекте, создавшем событие, находятся в объектной переменной sender. Какой пункт меню указал пользователь, можно узнать, конвертировав переменную sender в объект t класса ToolStripMenuItem. В таком случае мы можем прочитать в свойстве Text название пункта меню, которое выбрал пользователь. Таким образом, в строковую переменную Формат попадает или строка «Открыть в формате

RTF», или строка «Открыть в формате Win1251». Метод LoadFile объекта richTextBox1 загружает либо файл в формате RTF, либо файл в обычном текстовом формате. Перехватчик ошибок catch сообщает пользователю либо о том, что такого файла нет, либо, если пользователь использует пункт меню Открыть в формате RTF для открытия текстового файла, он получает сообщение «Недопустимый формат файла».

Сохранение файла (рис. 4.7) выполняется также с использованием стандартного диалога SaveFileDialog. Непосредственное сохранение файла удобнее всего выполнять в отдельной процедуре Запись(), поскольку эту процедуру необходимо вызывать также при выходе из программы, когда в документе имеются несохраненные изменения richTextBox1->Modified = True.

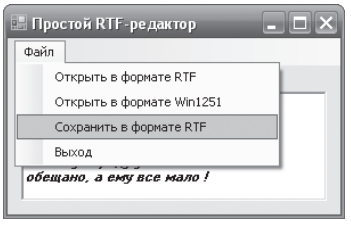

Рис. 4.7. Простой RTF-редактор

В основе процедуры Запись() также лежит блок try...catch: выполнить попытку (try) сохранения файла (SaveFile) и при этом перехватить (catch) возможные недоразумения и сообщить о них пользователю в диалоговом окне MessageBox.

Выход из программы организован абсолютно так же, как и в программе из предыдущего примера. Вдобавок обработаны два события — пункт меню Выход и всевозможные закрытия программы традиционными способами Windows. Предусмотрен диалог с пользователем в случае имеющих место несохраненных данных.

Отмечу, что для закрытия приложения *следует осторожно* пользоваться методом Exit объекта Application (можно сказать, с оглядкой). Этот метод подготавливает приложение к закрытию. Да, метод Application::Exit() *не вызывает события формы* Closing. Но попробуйте проследить за поведением программы после команды Application::Exit с *помощью отладчика* (клавиша F11). Вы убедитесь, что после команды Application::Exit управление перейдет к следующему оператору, затем — к следующему, и так до конца процедуры. Если на пути встретится функция MessageBox, то программа выдаст это диалоговое окно, и все это будет происходить, несмотря на то что уже давно была дана команда Application::Exit. Аналогично ведет себя метод Close элемента Form (если вы работаете с проектом Windows Application), который вызывается таким образом: this->Close(). Да, this->Close вызывает событие формы Closing. Этот метод закрывает форму и *освобождает все ресурсы*. Но для освобождения ресурсов после команды this->Close управление также *пройдет все операторы процедуры. Таким образом, для немедленного выхода из процедуры следует комбинировать названные методы с* return.

Убедиться в работоспособности программы можно, открыв решение RtfРедактор.sln в папке RtfРедактор.

Пример 31. Программа ввода каталога координат (числовых данных) из текстового файла

Следует отметить, что хороший тон программирования предполагает ввод подобного рода данных через удобную для пользователя таблицу с интуитивно понятным интерфейсом. Кстати, для этой цели очень удобно использовать элемент управления DataGridView (просмотр сетки данных) из панели элементов Toolbox. Но сейчас нам важно быстро получить результат, поэтому мы пренебрежем хорошим тоном при решении следующей задачи.

Допустим, что нам необходимо написать программу ввода каталога координат (числовых данных) X, Y, U и V. Предполагается, что пользователь в Блокноте заполняет текстовый файл этими числовыми данными. При этом в каждой строке он пишет координаты одной точки, то есть четыре числа, в качестве разделителя целой и дробной частей пользователь может использовать одновременно и точку, и запятую (в одном числе — точка, в следующем — запятая, потом наоборот и т. д.). Между числами может быть сколько угодно символов пробел и/или знаков табуляции ('\t'). Пользователь может пропускать пустые строки в середине каталога, а также пустые строки в самом конце файла. Программа должна «понять» намерения пользователя и вывести распознанный каталог координат в текстовое поле экранной формы.

На листинге 4.7 приведем пример такого текстового файла.

Листинг 4.7. Содержание текстового файла с каталогом координат

```
167.5 437,5 13.8  120,0
217.5 437.5 62,5  131.2


182.5 413.8 35.0  100
210   410   62.5  101.2
240   415   90    113.8
162,5 387.5 21.2  68.8
191.2 372.5 52.5  61.2
211.2 390   67.5  82.5
240   388.8 96.5  88.8
160   357,5 26.2  40
225   367.5 86.2  65
161.2 341.2 31.2  25
187.5 340   56.2  28.8
212.5 337.5 81.2  32.5
241.2 341.2 108.8 42.5
```

Как мы видим, файл содержит пустые строки в середине и в конце каталога; имеет место и точка, и запятая в качестве разделителя целой и дробной частей числа; между числами разное количество пробелов и невидимых символов табуляции (следствие нажатия клавиши Tab).

Для решения этой задачи запустим систему Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms

Application Visual C++. В панели Toolbox дважды щелкнем на элементе TextBox, а затем, указав в его свойствах свойству Multiline значение True, растянем текстовое поле на всю экранную форму (см. рис. 4.8).

Текст программы приведен в листинге 4.8.

Листинг 4.8. Программа ввода каталога координат

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Чтение текстового файла, содержащего каталог координат.
        // В каждой строке файла должны быть записаны координаты одной точки,
        // (четыре числа). При этом в качестве разделителя целой и дробной
        // частей пользователь может использовать точку и/или запятую.
        // Между числами может быть сколько угодно пробельных символов и/или
        // знаков табуляции ('\t'). Пользователь может пропускать пустые строки
        // в середине каталога и/или в конце файла. Программа должна «понять»
        // введенный текст и вывести распознанный каталог координат в текстовое
        // поле экранной формы
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Объявляем массивы координат вне всех процедур, чтобы
        // они были видны из любой из процедур:
        array<Double> ^X, ^Y, ^U, ^V;
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 int i, n;
                 try
                 {
                     textBox1->Size = Drawing::Size(223, 205);
                     auto ТекущийКаталог =
                     // или String ^ ТекущийКаталог =
                                     IO::Directory::GetCurrentDirectory();
                     // Читаем файл в строковую переменную:
                     String ^ ВесьТекст = IO::File::ReadAllText(
                                       ТекущийКаталог + "\\convert1.txt");
                     // Во всем тексте заменяем точки на запятые, то есть
                     // можно вводить числа с точкой, а можно с запятой:
                     ВесьТекст = ВесьТекст->Replace(".", ",");
                     // Заменяем все знаки табуляции на пробелы:
                     ВесьТекст = ВесьТекст->Replace('\t', ' ');
                     // Делим весь текст на строки с удалением пустых строк:
                     array<Char> ^ Сепаратор1 = { '\r', '\n' };
                     auto Строки = ВесьТекст->Split(Сепаратор1,
                                    StringSplitOptions::RemoveEmptyEntries);
```

продолжение ⇩

Листинг 4.8 (продолжение)

```
// Пустые строки, содержащие только (""), мы удалили, но
// остались строки, заполненные несколькими пробелами.
n = Строки->Length; // - количество элементов массива
// Ищем строки с пробелами (где кроме пробелов в строке
// ничего нет) и удаляем пробелы:
for (i = 0; i <= n - 1; i++) Строки[i] = Строки[i]->Trim();
// Опять объединяем все строки, в них уже нет строк
// с пробелами, вместо них - пустые строки (""):
ВесьТекст = String::Empty;
for (i = 0; i <= n - 1; i++) ВесьТекст += Строки[i] +
"\r\n";
// Метод Split умеет избавляться от пустых строк, он их
// удаляет, если задать параметр RemoveEmptyEntries:
Строки = ВесьТекст->Split(Сепаратор1,
                    StringSplitOptions::RemoveEmptyEntries);
// Задаемся новым сепаратором. В этом сепараторе символ
// табуляции '\t' - необязателен, поскольку мы все эти
// символы заменили на пробелы. Но если мы передумаем делать
// такую замену, то символ табуляции в новом сепараторе
// будет весьма кстати.
n = Строки->Length;
array<Char> ^ Сепаратор2 = { ' ', '\t' };
// Здесь уже можно определиться с размерностью массивов:
X = gcnew array<Double>(n + 1);
Y = gcnew array<Double>(n + 1);
U = gcnew array<Double>(n + 1);
V = gcnew array<Double>(n + 1);
// Признаки успешного преобразования из строки в число:
bool A, B, C, D;
for (i = 0; i <= n - 1; i++)
{ // Преобразование из строки в число:
    array<String^> ^Подстроки = Строки[i]->Split(Сепаратор2,
        StringSplitOptions::RemoveEmptyEntries);
    A = Double::TryParse(Подстроки[0], X[i + 1]);
    B = Double::TryParse(Подстроки[1], Y[i + 1]);
    C = Double::TryParse(Подстроки[2], U[i + 1]);
    D = Double::TryParse(Подстроки[3], V[i + 1]);
    // Если хотя бы одно из преобразований
    // метода TryParse ложно, то:
    if ((A && B && C && D) == false) MessageBox::Show(
        String::Format("В строке {0} - не числовой ввод !",
        i + 1), "Ошибка", MessageBoxButtons::OK,
        MessageBoxIcon::Exclamation);
}
this->Text = String::Format("Количество точек = {0};", n);
textBox1->Multiline = true; textBox1->Clear();
// В цикле добавляем строку координат в текстовое поле:
for (i = 1; i <= n; i++)
```

```
                    textBox1->AppendText(String::Format("X = {0,5:F1};  " +
                        "Y = {1,5:F1};  U = {2,5:F1};  V = {3,5:F1};\r\n",
                                                X[i], Y[i], U[i], V[i]));
                // Использованный формат "{0,5:F1}" означает: взять нулевой
                // выводимый элемент и записать его с выравниванием вправо в
                // следующих пяти позициях в формате с фиксированной точкой
                // и одним десятичным знаком после запятой.
            }
            catch (IO::FileNotFoundException^ Ситуация)
            {
                MessageBox::Show("Нет такого файла\n" + Ситуация->Message,
                    "Ошибка", MessageBoxButtons::OK,
                              MessageBoxIcon::Exclamation);
            }
            catch (Exception ^ Ситуация)
            {
                // Отчет о других ошибках:
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    };
}
```

Как видно из текста программы, процесс распознавания каталога координат происходит при обработке события загрузки формы. Вначале из текущей папки считываем в строковую переменную ВесьТекст содержимое файла convert1.txt и во всем тексте делаем контекстную замену точек на запятые, используя метод Replace. Также заменяем все невидимые символы табуляции (следствие нажатия клавиши Tab) на пробелы. Далее используем очень эффективную функцию Split для копирования всего текста в строковый массив, в качестве разделителя (сепаратора) мы указали здесь символы конца строки. Благодаря параметру RemoveEmptyEntries пустые строки в тексте игнорировались, но остались строки, содержащие пробелы, от которых также было бы неплохо избавиться. Для этого с помощью функции Length определяем количество элементов строкового массива. Заметим, что в других языках Visual Studio C# и Visual Basic эта функция носит название Count. Затем организуем цикл по элементам строкового массива и при этом каждый элемент массива избавим от пробелов в начале и конце строк (функция Trim). После чего опять записываем этот массив в одну строковую переменную ВесьТекст и опять применяем к нему метод Split, чтобы окончательно избавиться от пустых строк в середине и в конце анализируемого текста. Избавившись от избыточных пустых строк, мы можем выяснить количество вводимых данных и назначить, наконец, количество элементов в массивах координат. В следующем цикле преобразуем считанные из файла подстроки в числа с помощью функции TryParse. Эта функция возвращает true при успешном преобразовании. Поскольку мы имеем четыре числа в каждой строке (X, Y, U и V), получим четыре возврата: A, B, C и D. Логическое умножение (конъюнкция) этих булевых переменных ответит на вопрос, было ли хотя бы одно

из четырех преобразований метода TryParse ложно. В этом случае организуем вывод соответствующего сообщения.

Фрагмент работы данной программы можно увидеть на рис. 4.8.

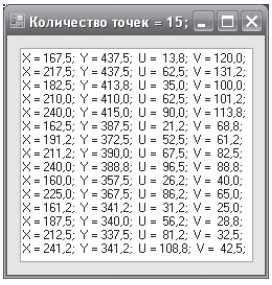

Рис. 4.8. Вывод распознанного каталога координат в текстовое поле

Убедиться в работоспособности программы можно, открыв sln-файл в папке ВводКаталогаКоординат.

Пример 32. Печать текстового документа

Любой текстовый редактор (и не только текстовый) должен иметь возможность печати на принтере. Я сознательно не добавлял такую возможность в текстовые редакторы, приведенные в предыдущих разделах, чтобы не запутать читателя. Понятно, что чем больше функциональности имеет программа, тем сложнее ее программный код, тем труднее текст программы для понимания. А наша задача — выразительно и ярко демонстрировать технологии в максимально простой форме.

Программа, представленная в данном разделе, обладает достаточно скромными возможностями. Она позволяет открыть в стандартном диалоге Windows текстовый файл, просмотреть его в окне программы (в текстовом поле) без возможности изменения текста (ReadOnly) и вывести этот текст на принтер.

Таким образом, чтобы создать данную программу, следует перенести в форму следующие элементы управления: текстовое поле TextBox, меню MenuStrip с пунктами меню: Открыть, Печатать и Выход, а также элементы управления OpenFileDialog и PrintDocument. Текст программы представлен в листинге 4.9.

Листинг 4.9. Печать текстового документа

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа позволяет открыть в стандартном диалоге текстовый файл,
        // просмотреть его в текстовом поле без возможности изменения текста
```

```
    // (ReadOnly) и при желании пользователя вывести этот текст на принтер
    IO::StreamReader ^ Читатель;
private: System::
             Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
         {
             Form1::Text = "Открытие текстового файла и его печать";
             textBox1->Multiline = true; textBox1->Clear();
             textBox1->Size = Drawing::Size(268, 112);
             textBox1->ScrollBars = ScrollBars::Vertical;
             textBox1->ReadOnly = true;
             // До тех пор пока файл не прочитан в текстовое поле,
             // не должен быть виден пункт меню "Печать..."
             печатьToolStripMenuItem->Visible = false;
             openFileDialog1->FileName = nullptr;
         }
private: System::Void открытьToolStripMenuItem_Click(System::Object^ sender,
                                                     System::EventArgs^ e)
         {
             // Щелчок на пункте меню "Открыть":
             openFileDialog1->Filter =
                 "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
             openFileDialog1->ShowDialog();
             if (openFileDialog1->FileName == nullptr) return;
             try
             {  // Создание потока StreamReader для чтения из файла
                 Читатель = gcnew
                     IO::StreamReader(openFileDialog1->FileName,
                     System::Text::Encoding::GetEncoding(1251));
                 // - здесь заказ кодовой страницы Win1251 для русских букв
                 textBox1->Text = Читатель->ReadToEnd();
                 Читатель->Close();
                 печатьToolStripMenuItem->Visible = true;
             }
             catch (IO::FileNotFoundException^ Ситуация)
             {
                 MessageBox::Show(Ситуация->Message +
                                  "\nНет такого файла", "Ошибка",
                     MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
             }
             catch (Exception^ Ситуация)
             {  // Отчет о других ошибках:
                 MessageBox::Show(Ситуация->Message, "Ошибка",
                     MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
             }
         }
private: System::Void печатьToolStripMenuItem_Click(System::Object^ sender,
                                                    System::EventArgs^  e)
         { // Пункт меню "Печать"
             try
```

продолжение ⇘

Листинг 4.9 (продолжение)

```
                {
                    Читатель = gcnew
                        IO::StreamReader(openFileDialog1->FileName,
                              System::Text::Encoding::GetEncoding(1251));
                    // - здесь заказ кодовой страницы Win1251 для русских букв
                    try { printDocument1->Print(); }
                    finally { Читатель->Close(); }
                }
                catch (Exception^ Ситуация)
                { MessageBox::Show(Ситуация->Message); }
            }
    private: System::Void printDocument1_PrintPage(System::Object^  sender,
                System::Drawing::Printing::PrintPageEventArgs^  e)
            { // Событие вывода на печать страницы (PrintPage):
                Single СтрокНаСтранице = 0;
                Single Y = 0;
                Single ЛевыйКрай = (Single)e->MarginBounds.Left;
                Single ВерхнийКрай = (Single)e->MarginBounds.Top;
                String ^Строка = nullptr;
                Drawing::Font ^ Шрифт = gcnew Drawing::Font(
                                              "Times New Roman", 12.0F);
                // Вычисляем количество строк на одной странице
                СтрокНаСтранице = e->MarginBounds.Height /
                                        Шрифт->GetHeight(e->Graphics);
                // Печатаем каждую строку файла
                int i = 0; // - счет строк
                while (i < СтрокНаСтранице)
                {
                    Строка = Читатель->ReadLine();
                    if (Строка == nullptr) break; // выход из цикла
                    Y = ВерхнийКрай + i *
                               Шрифт->GetHeight(e->Graphics);
                    // Печать строки
                    e->Graphics->DrawString(Строка, Шрифт, Brushes::Black,
                                      ЛевыйКрай, Y, gcnew StringFormat());
                    i = i + 1; // или i += 1; - счет строк
                }
                // Печать следующей страницы, если есть еще строки файла
                if (Строка != nullptr) e->HasMorePages = true;
                else e->HasMorePages = false;
            }
    private: System::Void выходToolStripMenuItem_Click(System::Object^  sender,
                                                    System::EventArgs^  e)
            {
                // Выход из программы
                this->Close();
            }
    };
}
```

Здесь при обработке события загрузки формы Form1_Load *запрещаем* пользователю редактировать текстовое поле: ReadOnly = true. Также назначаем свойство печать ToolStripMenuItem->Visible = false (пункт меню Печать), то есть в начале работы программы пункт меню Печать пользователю не виден (поскольку пока распечатывать нечего, необходимо вначале открыть текстовый файл). Остальные присваивания при обработке события Form1_Load очевидны.

При обработке события «щелчок на пункте меню Открыть» вызываем стандартный диалог OpenFileDialog и организуем чтение файла через создание потока StreamReader. Эти процедуры мы уже рассматривали подробно в разделах о текстовых редакторах, поэтому оставим их без комментария. Замечу только, что после чтения файла в текстовое поле назначаем видимость пункту меню Печать: печатьToolStripMenuItem->Visible = true, поскольку уже есть что печатать на принтере (файл открыт).

Представляет интерес обработка события «щелчок на пункте меню Печать». Здесь во вложенных блоках try...finally...catch программа еще раз создает поток StreamReader, а затем запускает процесс печати документа printDocument1->Print. Если ничего более не программировать, только метод printDocument1->Print, то принтер распечатает пустую страницу. Чтобы принтер распечатал текст, необходимо обработать событие PrintPage (см. текст программы), которое создает объект PrintDocument. То есть роль метода Print — это создать событие PrintPage.

Обратите внимание на обработку события PrintDocument1_PrintPage. Пример обработки этого события приведен в MSDN. Вначале перечислены объявления переменных, значения некоторых из них получаем из аргументов события e, например ЛевыйКрай — значение отступа от левого края и т. д. Назначаем шрифт печати — Times New Roman, 12 пунктов.

Далее в цикле while программа читает каждую строку line из файла — Читатель->ReadLine(), а затем распечатывает ее командой (методом) DrawString. Здесь используется графический объект Graphics, который получаем из аргумента события e.

В переменной i происходит счет строк. Если количество строк оказывается большим, чем число строк на странице, то происходит выход из цикла, поскольку страница распечатана. Если есть еще страницы, программа выясняет это, анализируя содержимое переменной Строка. Если ее содержимое отличается от значения nullptr (Строка != nullptr), то аргументной переменной e.HasMorePages назначаем true, что инициирует опять событие PrintPage, и процедура PrintDocument1_PrintPage, начинает свою работу вновь. И так, пока не закончатся все страницы e.HasMorePages = False для печати на принтере.

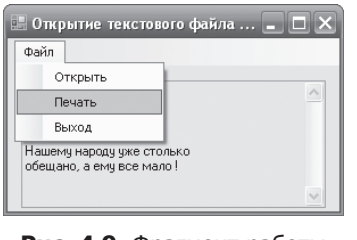

Рис. 4.9. Фрагмент работы программы печати текстового файла

На рис. 4.9 показан интерфейс приложения.

Убедиться в работоспособности программы можно, открыв решение TxtPrint.sln в папке TxtPrint.

Пример 33. Чтение/запись бинарных файлов с использованием потока данных

Обычно программа либо что-то считывает с диска в оперативную память, либо что-то записывает на диск. Писать, читать можно либо в бинарный (двоичный) файл, либо в текстовый (литерный, строковый) файл. Разница между ними состоит в том, что текстовый файл можно прочитать текстовым редактором, например Блокнотом, а бинарный — нет. Название «бинарный» (двоичный) — условное, поскольку, по сути, и текстовый, и бинарный файлы являются двоичными файлами.

Операции с двоичным файлом гораздо более скоростные, и такой файл занимает существенно меньшее пространство на диске. Зато текстовый файл можно читать и редактировать любым текстовым редактором. Обычно программист находит компромисс между этими двумя вариантами.

Приведем пример самого простейшего случая записи на диск бинарного файла с данными об успеваемости одного студента. Понятно, что эта программа может быть маленькой частью большой программы по обработке успеваемости студентов в вузе. Данная программа принимает от пользователя сведения только об одном студенте в текстовые поля формы. При нажатии кнопки Сохранить программа записывает введенные сведения в двоичный файл, а при нажатии кнопки Читать читает эти сведения из двоичного файла в текстовые поля формы.

Итак, в форме имеем три текстовых поля, куда пользователь может записать соответственно номер студента по порядку, фамилию студента и его средний балл успеваемости. Поэтому в форму из панели Toolbox перенесем три текстовых поля TextBox, три метки Label и две командные кнопки: Читать и Сохранить. Таким образом, получим пользовательский интерфейс, показанный на рис. 4.10. Текст программы приведен в листинге 4.10.

Листинг 4.10. Чтение/запись бинарных файлов

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа для чтения/записи бинарных файлов с использованием потока
        данных
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Успеваемость студента";
                label1->Text = "Номер п/п";
                label2->Text = "Фамилия И.О.";
                label3->Text = "Средний балл";
                textBox1->Clear(); textBox2->Clear(); textBox3->Clear();
```

```
            button1->Text = "Читать";
            button2->Text = "Сохранить";
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // ЧТЕНИЕ БИНАРНОГО ФАЙЛА
            // Если такого файла нет
            if (IO::File::Exists("C:\\student.usp") == false) return;
            // Создание потока Читатель
            IO::BinaryReader^ Читатель = gcnew IO::BinaryReader(
                                IO::File::OpenRead("C:\\student.usp"));
            try
            {
                int Номер_пп = Читатель->ReadInt32();
                String ^ ФИО = Читатель->ReadString();
                Single СредБалл = Читатель->ReadSingle();
                textBox1->Text = Convert::ToString(Номер_пп);
                textBox2->Text = Convert::ToString(ФИО);
                textBox3->Text = Convert::ToString(СредБалл);
            }
            finally { Читатель->Close(); }
        }
    private: System::Void button2_Click(System::Object^ sender,
                                        System::EventArgs^ e)
        {
            // ЗАПИСЬ БИНАРНОГО ФАЙЛА
            // Создаем поток Писатель для записи байтов в файл
            auto Писатель = gcnew IO::BinaryWriter(
                IO::File::Open("C:\\student.usp", IO::FileMode::Create));
            try
            {
                int Номер_пп = Convert::ToInt32(textBox1->Text);
                String^ ФИО = Convert::ToString(textBox2->Text);
                Single СредБалл = Convert::ToSingle(textBox3->Text);
                Писатель->Write(Номер_пп);
                Писатель->Write(ФИО);
                Писатель->Write(СредБалл);
            }
            finally { Писатель->Close(); }
        }
    };
}
```

Мы видим, что в процедуре обработки события загрузки формы организована инициализация (присвоение начальных значений) элементам формы: текстовых полей, меток и кнопок.

Запись файла на диск происходит при обработке события button2.Click, то есть «щелчок мышью на кнопке Сохранить» (см. рис. 4.10). Для этого создаем поток

байтов Писатель для открытия файла student.usp. Если такой файл не существует, то он создается (Create), а если файл уже есть, то он перезаписывается. Как видно, для упрощения программы мы не использовали элемент управления OpenFileDialog для открытия файла в диалоге.

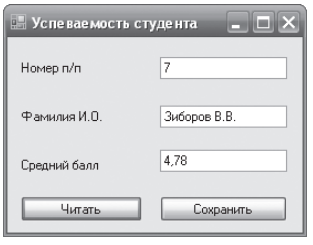

Рис. 4.10. Фрагмент работы программы чтения/записи бинарных файлов

Далее преобразуем записанное в текстовых полях в более естественные типы данных. Номер по порядку Номер_пп — это тип int, преобразование в целый тип может быть реализовано операцией Convert::ToInt32 (можно использовать другие функции преобразования), для переменной СредБалл (средний балл) больше всего подходит тип с плавающей точкой Single, при этом преобразование осуществляется операцией Convert::ToSingle. Преобразование для строковой переменной ФИО является необязательным и приведено для симметрии записей. Операторы Писатель-> Write записывают эти данные в файл. После блока Finally происходит обязательное закрытие (Close) файла.

Чтение файла выполняется при обработке события «щелчок мышью на кнопке Читать». Как уже упоминалось, для максимального упрощения в данной программе не предусмотрено открытие файла через стандартный диалог, поэтому вначале процедуры выясняем, существует ли такой файл. Если файла C:\student.usp нет, то программируем выход (return) из обработчика данного события. Заметьте, чтобы программисту было максимально легко отслеживать ветви оператора if, мы написали: «Если файла нет, то return». При этом длинная ветвь логики «если файл есть» не включена непосредственно в оператор if. Поэтому этот фрагмент программного кода читается (воспринимается) программистом легко.

Далее создается поток байтов Читатель из файла student.usp, открытого для чтения. Чтение из потока в каждую переменную реализовано с помощью функции ReadInt32 — читать из потока Читатель в переменную типа int, аналогично функциям ReadString и ReadSingle. Далее осуществлено конвертирование этих переменных в строковый тип Convert::ToString. Как видно, можно было изначально все текстовые поля записывать в файл без конвертирования, но при дальнейшем развитии этой программы значения полей все равно пришлось бы преобразовывать в соответствующий тип. После блока Finally происходит закрытие (Close) файла. Блок Finally выполняется всегда, даже если перед ним была команда return.

Дальнейшее развитие данной программы может идти по пути добавления в файл сведений о других студентах. В таком случае при чтении файла будет

неопределенность количества студентов. Тогда следует обработать ситуацию достижения конца файла:

```
catch (EndOfStreamException e)
```

а затем закрыть файл.

Убедиться в работоспособности программы можно, открыв решение ReadWriteBin.sln в папке ReadWriteBin.

5 Редактирование графических данных

Пример 34. Простейший вывод отображения графического файла в форму

Поставим задачу вывода в форму какого-либо изображения — растрового графического файла формата BMP, JPEG, PNG или других форматов. Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Двойной щелчок на проекте формы приведет нас на вкладку программного кода. Работать с графикой в форме можно по-разному. Рассмотрим работу с графикой через переопределение метода OnPaint.

Метод OnPaint является членом класса Form. Этот метод можно увидеть, набрав "this->" внутри какой-нибудь процедуры, в раскрывающемся списке *методов и свойств* объекта Form1. Метод OnPaint можно *переопределить*, то есть добавить к уже существующим функциям собственные. Для этого в окне программного кода напишем:

```
protected: virtual void
```

И в появившемся раскрывающемся списке выберем OnPaint. Система сама сгенерирует необходимые строчки процедуры, *подлежащей переопределению*.

Теперь этот программный код следует дополнить командами для вывода в форму изображения из растрового файла poryv.png (листинг 5.1).

Листинг 5.1. Вывод растрового изображения в форму (вариант 1)

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->Text = L"Form1";
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа выводит в форму растровое изображение из графического файла
    protected: virtual void OnPaint( PaintEventArgs^ e ) override
```

```
            {
                Form1::Text = "Рисунок";
                // Размеры формы
                this->Width = 200; this->Height = 200;
                // Создаем объект для работы с изображением
                Image ^ Рисунок = gcnew Bitmap("C:\\poryv.png");
                // Вывод изображения в форму
                e->Graphics->DrawImage(Рисунок, 5, 5);
                // x=5, y=5 - это координаты левого верхнего угла рисунка
                // в системе координат формы: ось x - вниз, ось y - вправо
            }
    };
}
```

Как видно из текста программы, вначале указываем размеры формы при помощи свойств Width и Height, хотя размеры формы *удобно регулировать* на вкладке конструктора формы «визуально». Далее создаем объект Рисунок для работы с изображением с указанием пути к файлу рисунка. Затем обращаемся непосредственно к методу рисования изображения в форме DrawImage, извлекая графический объект Graphics из аргумента e процедуры OnPaint.

Фрагмент работы программы показан на рис. 5.1.

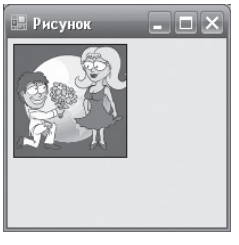

Рис. 5.1. Вывод растрового изображения в форму

Следует отметить, что это не единственный способ работы с графикой. Другой способ — это вызвать тот же метод OnPaint косвенно через событие формы OnPaint. Такой способ работы с графикой представлен в листинге 5.2. Создаем процедуру обработки данного события обычным способом, то есть на вкладке конструктора формы в панели свойств Properties щелкнем на значке молнии и в появившемся списке всех событий для объекта Form1 выберем событие Paint. Объект Graphics получаем из аргумента e события Paint.

Листинг 5.2. Вывод растрового изображения в форму (вариант 2)

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Простейший вывод изображения в форму
    private: System::Void Form1_Paint(System::Object^  sender,
                System::Windows::Forms::PaintEventArgs^  e)
             { // В свойствах формы щелкнем значок молнии и в появившемся списке
               // всех событий для объекта Form1 выберем событие Paint.
                 // Событие Paint - это событие рисования формы:
                 this->Text = "Рисунок";
                 // Создаем объект для работы с изображением:
                 Image^ Рисунок = Image::FromFile("C:\\poryv.png");
```

продолжение ⇩

Листинг 5.2 (продолжение)

```
                // или Image^ Рисунок = gcnew Bitmap("C:\\poryv.png");
                // Вывод изображения в форму:
                e->Graphics->DrawImage(Рисунок, 5, 5);
            }
    };
}
```

Рассмотрим *еще один способ* вывода графики в форму. В этом случае при щелчке на командной кнопке происходит непосредственное создание объекта класса Graphics. Программный код представлен в листинге 5.3.

Листинг 5.3. Вывод растрового изображения в форму (вариант 3)

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Простейший вывод изображения в форму
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                // Событие загрузки формы:
                Form1::Text = "Рисунок";
                button1->Text = "Показать рисунок";
            }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
            {
                // Событие "щелчок на кнопке"
                Image ^ Рисунок = gcnew Bitmap("C:\\poryv.png");
                // Создание графического объекта:
                Graphics ^ Графика = this->CreateGraphics();
                // или Graphics ^ Графика = CreateGraphics();
                Графика->DrawImage(Рисунок, 5, 5);
            }
    };
}
```

Убедиться в работоспособности данных программ можно, открыв соответствующие решения в папках SimpleImage1, SimpleImage2 и SimpleImage3. В заключение замечу, что с рассмотренными в данном разделе методами можно работать *не только* для вывода изображений графических файлов в форму, но и решать многие *другие задачи, связанные с графикой*.

Пример 35. Использование элемента PictureBox для отображения растрового файла с возможностью прокрутки

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в формате BMP, JPEG, GIF, PNG и других используется объект класса PictureBox (графическое поле). Часто рисунок оказывается слишком большим и не помещается целиком в пределах элемента управления PictureBox. Можно воспользоваться свойством элемента SizeMode, указав ему значение StretchImage. В этом случае изображение будет вытягиваться или сужаться, подстраиваясь под размер PictureBox. Очень часто такой подход не устраивает разработчика, поскольку происходит деформация изображения. Решение этой проблемы заключается в организации возможности прокрутки изображения (AutoScroll), но такого свойства у PictureBox нет. Зато такое свойство есть у элемента управления Panel. То есть, разместив PictureBox на элементе Panel с установленным свойством AutoScroll = true и при этом для PictureBox указав SizeMode = AutoSize, мы можем решить задачу прокрутки изображения.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox перетащим на форму элемент управления Panel, а на него поместим элемент PictureBox. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.4.

Листинг 5.4. Вывод изображения на PictureBox с возможностью прокрутки

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа выводит изображение из растрового файла в PictureBox,
        // размещенный на элементе управления Panel, с возможностью прокрутки
        // изображения
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Скроллинг";
                // Назначаем размеры панели:
                panel1->Size = Drawing::Size(200, 151);
                // Назначаем имя файла рисунка:
                pictureBox1->Image = Image::FromFile("C:\\Ris.JPG");
```

продолжение ⇨

Листинг 5.4 (продолжение)

```
            // Или Image ^ Изображение = gcnew Bitmap("C:\\Ris.JPG");
            // а затем pictureBox1->Image = Изображение;
            // Размеры PictureBox в точности соответствуют изображению:
            pictureBox1->SizeMode = PictureBoxSizeMode::AutoSize;
            // Разрешаем прокрутку изображения:
            panel1->AutoScroll = true;
        }
    };
}
```

Мы видим, что весь наш пользовательский программный код написан для обработки события загрузки формы. Напомню: чтобы получить пустой обработчик этого события, достаточно на вкладке Form1.h[Design] дважды щелкнуть в пределах проектируемой формы. Текст программы сопровождают подробные комментарии.

Фрагмент работы программы приведен на рис. 5.2.

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке СкроллингБольшогоРисунка.

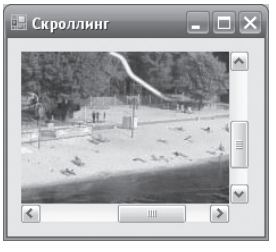

Рис. 5.2. Вывод изображения с возможностью прокрутки

Пример 36. Рисование в форме графических примитивов (фигур)

В векторных чертежах *графическим примитивом* называют элементарные составляющие чертежа: отрезок, дуга, символ, окружность и т. п. Здесь мы имеем дело с растровой графикой, но в данном случае подход тот же — по координатам рисуем те же фигуры. В Visual Studio система координат представлена следующим образом: начало координат — это левый верхний угол формы, ось *Ox* направлена вправо, а *Oy* — вниз.

Наша задача состоит в том, чтобы нарисовать в форме окружность, отрезок, прямоугольник, сектор, текст, эллипс и закрашенный сектор. Выбор того или иного графического примитива можно сделать через элемент управления ListBox (Список). Причем при рисовании очередного графического примитива нужно «стереть» предыдущий рисунок.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Перетащим в форму из окна Toolbox элемент управления ListBox. Далее — двойной щелчок в пределах формы, — и мы попадаем в пустой обработчик события загрузки формы, где *создадим список* графических примитивов, заполняя коллекцию (Items) элементов списка listBox1 (листинг 5.5).

Листинг 5.5. Рисование на форме графических примитивов

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа позволяет рисовать в форме графические примитивы:
        // окружность, отрезок, прямоугольник, сектор, текст, эллипс
        // и закрашенный сектор. Выбор того или иного графического примитива
        // осуществляется с помощью элемента управления ListBox
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Выбери графический примитив";
                listBox1->Items->AddRange(gcnew array<Object^> {"Окружность",
                    "Отрезок", "Прямоугольник", "Сектор",
                    "Текст", "Эллипс", "Закрашенный сектор"});
                Font = gcnew System::Drawing::Font("Times New Roman", 14.F);
             }
    private: System::Void listBox1_SelectedIndexChanged(
                System::Object^  sender, System::EventArgs^  e)
             {
                // Здесь вместо этого события можно было бы обработать
                // событие listBox1_Click.
                // Создание графического объекта
                Graphics ^ Графика = this->CreateGraphics();
                // Создание пера для рисования им фигур
                Pen ^ Перо = gcnew Pen(Color::Red);
                // Создание кисти для "закрашивания" фигур
                Brush ^ Кисть = gcnew SolidBrush(Color::Red);
                // Очистка области рисования путем ее окрашивания
                // в первоначальный цвет формы
                Графика->Clear(SystemColors::Control);
                // или Графика->Clear(Color::FromName("Control"));
                // или Графика->Clear(ColorTranslator::FromHtml("#EFEBDE"));
                switch (listBox1->SelectedIndex) // Выбор фигуры:
                {
                case 0:   // - выбрана окружность:
                    Графика->DrawEllipse(Перо, 50, 50, 150, 150); break;
                case 1:   // - выбран отрезок:
                    Графика->DrawLine(Перо, 50, 50, 200, 200); break;
                case 2:   // - выбран прямоугольник:
                    Графика->DrawRectangle(Перо, 50, 30, 150, 180); break;
                case 3:   // - выбран сектор:
                    Графика->DrawPie(Перо, 40, 50, 200, 200, 180, 225); break;
                case 4:   // - выбран текст:
                    Графика->DrawString("Каждый во что-то верит, но\n" +
```

продолжение ⇩

Листинг 5.5 (продолжение)

```
                        "жизнь преподносит сюрпризы.",
                        Font, Кисть, 10, 100); break;
                case 5:   // - выбран эллипс:
                    Графика->DrawEllipse(Перо, 30, 30, 150, 200); break;
                case 6:   // - выбран закрашенный сектор:
                    Графика->FillPie(Кисть, 20, 50, 150, 150, 0, 45); break;
                }
            }
    };
}
```

В программе, обрабатывая событие изменения выбранного индекса в списке listBox1 (хотя с таким же успехом в этой ситуации можно обрабатывать щелчок на выбранном элементе списка), создаем графический объект Графика, перо — для рисования им фигур, и кисть — для «закрашивания» фигур. Далее очищаем область рисования путем окрашивания формы в первоначальный цвет Control или «#EFEBDE» (как записано в комментарии), используя метод Clear() объекта Графика:

```
Графика->Clear(SystemColors::Control);
```

При очищении области рисования оставляем цвет формы первоначальным — Control. Кстати, этот цвет можно назвать цветом Microsoft: это цвет Windows Explorer, Internet Explorer и т. д.

После очистки формы, используя свойство SelectIndex, которое указывает на номер выбранного пользователем элемента списка (от 0 до 6), рисуем выбранную фигуру. На рис. 5.3 представлен фрагмент работы программы.

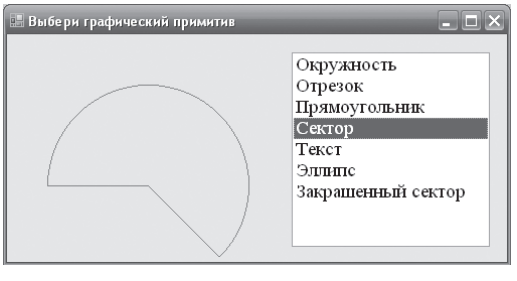

Рис. 5.3. Рисование графического примитива на форме

Убедиться в работоспособности программы можно, открыв решение РисФигур.sln в папке РисФигур.

Пример 37. Выбор цвета с использованием ListBox

В этом разделе нашей задачей будет написание программы, которая *меняет* цвет фона формы BackColor, перебирая константы цвета, предусмотренные в Visual Studio 2010, с помощью элемента управления ListBox.

Для решения данной задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. На вкладке дизайнера формы из панели элементов Toolbox перетащим на форму список ListBox. На вкладке программного кода Form1.cs введем текст, представленный в листинге 5.6.

Листинг 5.6. Выбор цвета с помощью элемента управления ListBox (вариант 1)

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа меняет цвет фона формы BackColor, перебирая константы
        // цвета, предусмотренные в Visual Studio 2010, с помощью элемента
        // управления ListBox
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                // Получаем массив строк имен цветов из перечисления KnownColor
                array<String^>^ ВсеЦвета = Enum::GetNames(KnownColor::typeid);
                // В C#: string[] ВсеЦвета = Enum.GetNames(typeof(KnownColor));
                listBox1->Items->Clear();
                // Добавляем имена всех цветов в список listBox1:
                listBox1->Items->AddRange(ВсеЦвета);
                // Сортировать записи в алфавитном порядке
                listBox1->Sorted = true;
             }
    private: System::Void listBox1_SelectedIndexChanged(
                System::Object^  sender, System::EventArgs^  e)
             {
                // Цвет Transparent является "прозрачным", он не поддерживается
                // для формы:
                if (listBox1->Text == "Transparent") return;
                this->BackColor = Color::FromName(listBox1->Text);
                this->Text = "Цвет: " + listBox1->Text;
             }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы, используя метод Enum::GetNames, получим массив имен цветов в строковом представлении. Теперь этот массив очень легко добавить в список (коллекцию) ListBox методом AddRange. Если вы еще не написали обработку события изменения выбранного индекса, попробуйте уже на данном этапе запустить текущий проект (клавиша F5). Вы увидите форму и заполненные строки элемента управления ListBox цветами из перечисления KnownColor. Обрабатывая событие изменения выбранного индекса в списке ListBox, предпоследней строкой назначаем выбранный пользователем цвет формы (BackColor). Один из цветов перечисления KnownColor — это цвет Control («умалчиваемый» цвет формы), который является базовым цветом во многих про-

граммах Microsoft, в том числе Windows Explorer, Internet Explorer, Visual Studio 2010 и пр. Кроме того, здесь цветов больше, чем в константах цветов (структуре) Color (в структуре Color нет цвета Control). Один из цветов — Transparent — является «прозрачным», и для фона формы он не поддерживается. Поэтому если пользователь выберет этот цвет, то произойдет выход из процедуры (return), и цвет формы не изменится.

На рис. 5.4 приведен пример работы программы. Мы видим, что пользователь выбрал цвет Control, который соответствует цвету формы по умолчанию.

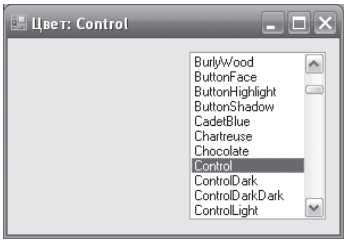

Рис. 5.4. Закраска формы выбранным цветом

Эту программу можно написать более элегантно с использованием цикла for each при заполнении списка формы именами всех цветов (листинг 5.7).

Листинг 5.7. Выбор цвета с помощью элемента управления ListBox (вариант 2)

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа меняет цвет фона формы
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                // Получаем массив строк имен цветов из перечисления KnownColor.
                // Enum::GetNames возвращает массив имен констант
                // в указанном перечислении:
                array<String^>^ ВсеЦвета = Enum::GetNames(KnownColor::typeid);
                // Удаление всех элементов из коллекции:
                listBox1->Items->Clear();
                // Добавляем имена всех цветов в список listBox1:
                for each (String^ s in ВсеЦвета)
                          if (s != "Transparent") listBox1->Items->Add(s);
                // Цвет Transparent является "прозрачным",
                // он не поддерживается для формы
             }
```

```
    private: System::Void listBox1_SelectedIndexChanged(
                 System::Object^  sender, System::EventArgs^  e)
             {
                 // Обработка события изменения выбранного
                 // индекса в списке listBox1:
                 this->BackColor = Color::FromName(listBox1->Text);
                 // Надпись в строке заголовка формы:
                 this->Text = "Цвет: " + listBox1->Text;
             }
    };
}
```

В этом варианте цикл for each (заметим, что эквивалентом этого оператора в Visual C# и Visaul Basic будет оператор foreach) обеспечивает заполнение списка listBox1 именами цветов в строковом представлении, кроме цвета Transparent, поэтому теперь его даже не надо «отсеивать» в процедуре обработки события изменения выбранного индекса.

Мы упомянули 167 констант или 146 цветов из структуры Color. Следует отметить, что в Visual Studio 2010 можно управлять гораздо большим количеством цветов. Система программирования Visual Studio работает с так называемой *RGB-моделью* управления цветом. Согласно этой модели любой цвет может быть представлен как комбинация красного (Red), зеленого (Green) и синего (Blue) цветов. Долю каждого цвета записывают в один байт: 0 означает отсутствие этого цвета, а максимум (255) — максимальное присутствие этого цвета в общей сумме, то есть в результирующем цвете. Например, функция Color::FromArgb(int red, int green, int blue) возвращает цвет, базируясь на этой модели. Информация о цвете элементарной точки (пиксела) может быть записана в три байта, то есть 24 бита. При этом говорят, что глубина цвета равна 24 разрядам. Максимальное число, которое можно записать в 24 бита, равно $2^{24} - 1 = 16\,777\,215$ или приблизительно 17 миллионов. Это означает, что при глубине цвета, равной 24, можно управлять 17 миллионами цветов *(цветовых оттенков)*.

Предлагаю следующую технологию использования любого цвета при разработке программ. Вначале выбираем цвет, который нам хотелось бы использовать (при запуске каких-либо программ или в Интернете на каком-либо сайте). Существуют программы, сообщающие в нескольких принятых кодировках цвет пиксела, на котором находится курсор мыши. Одну такую очень маленькую бесплатную программку Pixie вы можете скачать из Интернета по адресу: http://natty.port5.com или http://www.nattyware.com. Программа сообщает вам цвет пиксела в нескольких форматах, в том числе в формате HTML, например #EFEBDE (этот цвет соответствует цвету Control). Этот цвет можно подать на вход функции ColorTranslator::FromHtml() для перевода в формат, понятный той или иной процедуре (методу) C++.

Убедиться в работоспособности этих программ можно, открыв соответствующие решения в папках ВыборЦвета1 и ВыборЦвета2.

Пример 38. Экранная форма с треугольником прозрачности

Используя графические возможности системы программирования Visual Studio 2010, можно добиться самых невероятных, экзотических эффектов. В данном примере мы рассмотрим возможность программирования экранной формы, в которой будет размещен прозрачный треугольник. Этот треугольник выглядит, как отверстие в форме, через которое видны другие приложения, запущенные в данный момент на вашем компьютере.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Данное приложение назовем ПрозрачныйТреугольник. Из контекстного меню проектируемой формы перейдем на панель свойств Properties объекта Form1. Щелкнув на значке молнии, символизирующем события формы, найдем событие перерисовки элемента управления Form1_Paint и дважды щелкнем по нему. После чего система откроет для нас вкладку программного кода Form1.h с сгенерированным пустым обработчиком события Form1_Paint. Далее введем текст программы, представленный в листинге 5.8.

Листинг 5.8. Программирование экранной формы с треугольником прозрачности

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программирование экранной формы, в которой размещен
        // прозрачный треугольник
    private: System::Void Form1_Paint(System::Object^  sender,
                 System::Windows::Forms::PaintEventArgs^  e)
             {
                 // Событие перерисовки экранной формы:
                 this->ClientSize = System::Drawing::Size(240, 200);
                 // Устанавливаем вершины треугольника:
                 Point p1 = Point(20, 20);
                 Point p2 = Point(225, 66);
                 Point p3 = Point(80, 185);
                 // Инициализируем массив точек:
                 array<Point>^ Точки = { p1, p2, p3 };
                 // Рисуем закрашенный цветом ControlDark многоугольник:
                 e->Graphics->FillPolygon(gcnew SolidBrush(
                                      SystemColors::ControlDark), Точки);
                 // Задаем цвет, который будет выглядеть прозрачным:
                 this->TransparencyKey = SystemColors::ControlDark;
             }
    };
}
```

При обработке события перерисовки экранной формы назначим три вершины треугольника, инициализируя массив точек. Объект Graphics получим из аргумента события e. Воспользуемся методом FillPolygon для рисования закрашенного многоугольника. Цвет закрашивания может быть любым, однако этот же цвет необходимо назначить в качестве прозрачного в свойстве формы TransparencyKey.

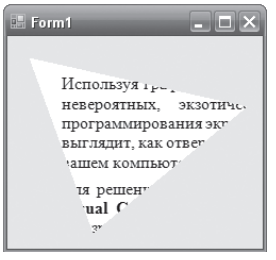

Рис. 5.5. Экранная форма с прозрачным треугольником внутри

Результат работы программы продемонстрирован на рис. 5.5. Здесь форма с треугольником прозрачности показана на фоне текста данной книги, отображаемой в MS Word.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ПрозрачныйТреугольник.

Пример 39. Печать графических примитивов

В данном разделе приведен пример *вывода на печать (на принтер) изображения эллипса*. Понятно, что таким же образом можно распечатывать и другие графические примитивы: прямоугольники, отрезки, дуги и т. д. (см. методы объекта Graphics).

Для написания данной программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Далее из панели элементов Toolbox в форму перенесем элемент управления PrintDocument. Текст программы приведен в листинге 5.9.

Листинг 5.9. Печать графических примитивов

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа выводит на печать (на принтер) изображение эллипса. Понятно,
        // что таким же образом можно распечатывать и другие графические
        // примитивы: прямоугольники, отрезки, дуги и т.д. (см. методы объекта
        // Graphics)
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                printDocument1->Print();
             }
    private: System::Void printDocument1_PrintPage(System::Object^  sender,
                System::Drawing::Printing::PrintPageEventArgs^  e)
             {
                // Выводится на печать эллипс красного цвета внутри
```

продолжение ⇩

Листинг 5.9 (продолжение)

```
                // ограничивающего прямоугольника с вершиной в точке (200, 250),
                // шириной 300 и высотой 200
                Pen ^ Перо = gcnew Pen(Color::Red);
                e->Graphics->DrawEllipse(Перо, Rectangle(200, 250, 300, 200));
                // или e->Graphics->DrawEllipse(Перо, 50, 50, 150, 150);
            }
    };
}
```

Здесь, с целью максимального упрощения программы для генерации события PrintPage при обработке события загрузки формы мы вызываем метод PrintDocument1->Print. В обработчике события PrintPage используем метод DrawEllipse для построения эллипса без заливки. В комментарии приведен вариант построения эллипса другим способом.

Убедиться в работоспособности программы можно, открыв решение ПечатьЭллипса.sln в папке ПечатьЭллипса.

Пример 40. Печать BMP-файла

В данном разделе мы рассмотрим программу, которая выводит на печать графический файл *формата* BMP. На логическом диске C: заранее подготовим графический файл формата BMP и назовем его C:\pic.bmp. Этот файл будет распечатываться на принтере программой, которую мы напишем.

Итак, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем добавим в форму из панели элементов Toolbox командную кнопку Button и объект PrintDocument. Программный код представлен в листинге 5.10.

Листинг 5.10. Печать BMP-файла

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Эта программа выводит на печать файл с расширением bmp
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Печать файла C:\\pic.bmp";
                button1->Text = "Печать";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Пользователь щелкнул на кнопке
```

```
                try
                {
                    printDocument1->Print();
                }
                catch (Exception ^ Ситуация)
                {
                    MessageBox::Show("Ошибка печати на принтере\n",
                                     Ситуация->Message);
                }
            }
    private: System::Void printDocument1_PrintPage(System::Object^  sender,
                System::Drawing::Printing::PrintPageEventArgs^  e)
            {
                // Это событие возникает, когда вызывают метод Print().
                // Рисование содержимого BMP-файла
                e->Graphics->DrawImage(Image::FromFile("C:\\pic.bmp"),
                    e->Graphics->VisibleClipBounds);
                // Следует ли распечатывать следующую страницу?
                e->HasMorePages = false;
            }
    };
}
```

Мы видим, что при нажатии пользователем кнопки вызывается метод printDocument1->Print. Этот метод создает событие PrintPage, которое обрабатывается в обработчике printDocument1_PrintPage. Для вывода на принтер вызывается метод рисования содержимого BMP-файла DrawImage.

Убедиться в работоспособности программы можно, открыв решение ПечатьBMPфайла.sln в папке ПечатьBMPфайла.

Пример 41. Создание JPG-файла «на лету» и вывод его отображения в форму

В некоторых случаях удобнее сначала создать изображение, что-либо на нем нарисовать, например отобразить сегодняшний курс доллара или график какой-нибудь зависимости, актуальный именно на сегодня, затем это изображение записать на диск и вывести его в форму для использования в ходе текущей работы. Таким графиком может быть, скажем, график продаж продукции по месяцам, по неделям или по дням.

Задача, рассматриваемая в данном разделе, состоит в следующем: мы имеем экранную форму и кнопку на ней, при щелчке указателем мыши на кнопке требуется создать изображение и с помощью методов класса Graphics вывести на это изображение текстовую строку, представляющую текущую дату. Далее с целью демонстрации возможностей методов Graphics мы развернем данную строку на некоторый угол относительно горизонта, затем сохраним рисунок в текущий каталог и выведем его отображение в форму.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Выберем имя проекта — Создать_JPG. Из панели Toolbox перетащим на форму элементы управления Button и PictureBox. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.11.

Листинг 5.11. Создание изображения, запись его на диск в формате JPG и вывод отображения файла в форму

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа формирует изображение методами класса Graphics, записывает
        // его на диск в формате JPG-файла и выводит его отображение
        // в экранную форму
   private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 button1->Text = "Показать дату";
             }
    private: System::Void button1_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 pictureBox1->Size = System::Drawing::Size(215, 35);
                 // Создаем точечное изображение размером 215 x 35 точек
                 // с глубиной цвета 24
                 Bitmap ^ Рисунок = gcnew Bitmap(215, 35, System::Drawing::
                                  Imaging:: PixelFormat::Format24bppRgb);
                 // Создаем новый объект класса Graphics из изображения РАСТР
                 Graphics ^ Графика = Graphics::FromImage(Рисунок);
                 // Теперь становятся доступными методы класса Graphics!
                 // Заливка поверхности цветом формы:
                 Графика->Clear(Color::FromName("Control"));
                 // Вывод в строку полной даты:
                 String ^ Дата = String::Format("Сегодня {0:D}", DateTime::Now);
                 // Разворачиваем мир на 356 градусов по часовой стрелке:
                 Графика->RotateTransform(356.0F);
                 // Выводим на изображение текстовую строку Дата,
                 // x=5, y=15 - координаты левого верхнего угла строки
                 Графика->DrawString(Дата, gcnew System::Drawing::
                     Font("Times New Roman", 14,
                     FontStyle::Regular), Brushes::Red, 5, 15);
                 // Сохраняем изображение в файле risunok.jpg:
                 Рисунок->Save("risunok.jpg",
                     System::Drawing::Imaging::ImageFormat::Jpeg);
                 // Задаем стиль границ рисунка:
                 pictureBox1->BorderStyle = BorderStyle::None; // FixedSingle;
                 // Загружаем рисунок из файла:
```

```
                pictureBox1->Image = Image::FromFile("risunok.jpg");
                // Освобождение ресурсов:
                delete Рисунок; delete Графика;
                // Эквивалент C#: Рисунок->Dispose(); и Графика->Dispose();
            }
    };
}
```

Как видно из программного кода, при обработке события «щелчок на кнопке» мы создаем точечное изображение указанного размера, формат Format24bppRgb указывает, что отводится 24 бита на точку: по 8 бит на красный, зеленый и синий каналы. Данное изображение позволяет создать новый объект класса Graphics методом FromImage. Теперь разворачиваем поверхность рисования на 356° методом RotateTransform и выводим на поверхность рисования текстовую строку с текущей датой. Методом Save сохраняем файл изображения в текущей папке в формате JPEG. Далее элементу управления pictureBox1 указываем путь к файлу изображения.

На рис. 5.6 показан фрагмент работы программы.

Убедиться в работоспособности программы из данного раздела можно, открыв соответствующее решение в папке Создать_JPG.

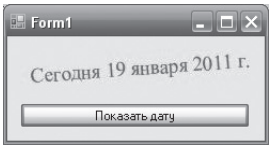

Рис. 5.6. Вывод в форму изображения, сформированного программно

Пример 42. Смена выведенного изображения с помощью обновления формы

При проектировании сценария диалога с пользователем зачастую необходимо предусмотреть для пользователя возможность что-либо изменять в выведенном на экранную форму изображении. Программная реализация такой возможности осуществляется с помощью перерисовки области элемента управления при участии метода Invalidate.

Конкретная постановка задачи состоит в следующем: в экранную форму выводим текстовую строку; щелчок на командной кнопке заставляет текстовую строку разворачиваться на некоторый угол.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Выберем имя проекта — СменаИзображения. Из панели Toolbox перетащим на форму элементы управления Button и Panel. Следует отметить, что элемент Panel предназначен для группировки на нем коллекций элементов управления. Однако элемент Panel часто используют также для вывода на нем графических изображений.

Нам понадобятся три пустых обработчика событий. Для их создания на вкладке дизайнера формы Form.h1[Конструктор] в контекстном меню проектируемой формы выберем команду Свойства. В окне свойств щелкнем на пиктограмме молнии (События) и, выбирая в раскрывающемся меню объекты Form1, panel1 и button1, дваж-

ды щелкнем соответственно на событиях Load, Paint и Click. В результате получим три пустых обработчика событий. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.12.

Листинг 5.12. Программа меняет выведенное в форму изображение, используя метод Invalidate

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа выводит в панель рисования текстовую строку. При щелчке
        // на командной кнопке происходит перерисовка изображения с разворотом
        // текстовой строки
        // ~ ~ ~ ~ ~ ~ ~ ~
        Graphics ^ Графика;
        float Угол;
    private: System::Void Form1_Load(System::Object^  sender,
                System::EventArgs^  e)
             {
                this->Text = "Смена изображения";
                button1->Text = "Развернуть";
                Угол = 0.0F;
             }
    private: System::Void panel1_Paint(System::Object^  sender,
                System::Windows::Forms::PaintEventArgs^  e)
             {
                // Событие Paint происходит при необходимости
                // перерисовки изображения:
                panel1->BackColor = Color::AliceBlue;
                panel1->Cursor = Cursors::Cross;
                // Задаем поверхность для рисования из аргумента события e:
                Графика = e->Graphics;
                // С помощью смещения задаем центр вращения:
                Графика->TranslateTransform(100.0F, 70.0F);
                // Поворот:
                Графика->RotateTransform(Угол);
                // Вывод текстовой строки полужирным и одновременно
                // наклонным шрифтом:
                Графика->DrawString("Весело живём!",
                    gcnew System::Drawing::Font("Comic Sans MS", 14,
                    FontStyle::Bold ▸ FontStyle::Italic), Brushes::Red, -70,
                    -15);
                // Здесь -70, -15 - координаты левого верхнего угла
                // создаваемого текста
             }
```

```
    private: System::Void button1_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 // Разворот на угол в градусах:
                 Угол = Угол + 30.0F;
                 // Обновить панель рисования:
                 panel1->Invalidate();
             }
    };
}
```

Как видно из программного кода, непосредственное рисование изображения происходит при обработке события panel1_Paint. Именно здесь мы выводим текстовую строку, задаем центр вращения и разворачиваем ее на некоторый угол. Значение угла регулируем при обработке события «щелчок на командной кнопке». Можно отследить отладчиком и убедиться, что процесс перерисовки изображения происходит довольно часто. Мы также можем программно инициировать перерисовку, используя метод Invalidate, что мы и запрограммировали при обработке события button1_Click.

Графическая система координат является левой, то есть ось X направлена вправо, а ось Y — вниз. Поскольку углы и в левой, и в правой системах координат измеряются от направления оси X в сторону оси Y, при каждом щелчке указателем мыши на кнопке текстовая строка разворачивается по часовой стрелке.

Фрагмент работы программы представлен на рис. 5.7.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке СменаИзображения.

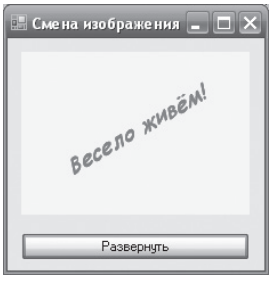

Рис. 5.7. Перерисовка изображения при каждом щелчке мышью

Пример 43. Рисование в форме указателем мыши

В данном примере покажем, как можно рисовать с помощью мыши в форме. То есть наша задача состоит в том, чтобы написать программу, позволяющую при нажатой левой или правой кнопке мыши рисовать в форме. Если пользователь отпустит кнопку мыши, то процесс рисования прекращается. В проектируемой форме следует предусмотреть кнопку Стереть, которая предназначена для очистки формы.

Вначале создадим форму с командной кнопкой так, как мы уже делали прежде. Для этого запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox перетащим на форму элемент управления Button. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.13.

Листинг 5.13. Рисование в форме указателем мыши

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа позволяет при нажатой левой или правой кнопке мыши
        // рисовать в форме
        // ~ ~ ~ ~ ~ ~ ~
        // Булева переменная Рисовать_ли дает разрешение на рисование:
        bool Рисовать_ли;
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Рисую мышью в форме";
                button1->Text = "Стереть";
                Рисовать_ли = false; // в начале - не рисовать
             }
    private: System::Void Form1_MouseDown(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
             {
                // Если нажата кнопка мыши - MouseDown, то рисовать
                Рисовать_ли = true;
             }
    private: System::Void Form1_MouseUp(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
             {
                // Если кнопку мыши отпустили, то НЕ рисовать
                Рисовать_ли = false;
             }
    private: System::Void Form1_MouseMove(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
             {
                // Рисование прямоугольника, если нажата кнопка мыши
                if (Рисовать_ли == true)
                {  // Рисовать прямоугольник в точке (e.X, e.Y)
                    Graphics ^ Графика = CreateGraphics();
                    Графика->FillRectangle(gcnew SolidBrush(Color::Red),
                        e->X, e->Y, 10, 10);
                    // 10x10 пикселов – размер сплошного прямоугольника
                    // e->X, e->Y – координаты указателя мыши
                    delete Графика; // Эквивалент C#: Графика->Dispose();
                }
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Методы очистки формы:
                Graphics ^ Графика = CreateGraphics();
```

```
            Графика->Clear(this->BackColor);
            // Графика->Clear(SystemColors::Control);
            // Графика->Clear(Color::FromName("Control"));
            // Графика->Clear(Color::FromKnownColor(KnownColor::Control));
            // Графика->Clear(ColorTranslator::FromHtml("#EFEBDE"));
            // this->Refresh(); // Этот метод также перерисовывает форму
        }
    };
}
```

Здесь, в начале программы объявлена переменная Рисовать_ли логического типа (bool) со значением false. Эта переменная либо позволяет (Рисовать_ли = true) рисовать в форме при перемещении мыши (событие MouseMove), либо не разрешает делать это (Рисовать_ли = false). Область действия переменной Рисовать_ли — весь класс Form1, то есть изменить или выяснить ее значение можно в любой процедуре этого класса.

Значение переменной Рисовать_ли может изменить либо событие MouseUp (кнопку мыши отпустили, рисовать нельзя, Рисовать_ли = false), либо событие MouseDown (кнопку мыши нажали, рисовать можно, Рисовать_ли = true). При перемещении мыши с нажатой кнопкой программа создает графический объект Graphics пространства имен System::Drawing, используя метод CreateGraphics(), и рисует прямоугольник FillRectangle(), заполненный красным цветом, размером 10 × 10 пикселов. e.X, e.Y — координаты указателя мыши, которые также являются координатами левого верхнего угла прямоугольника.

На рис. 5.8 приведен пример рисования в форме. Чтобы стереть все нарисованное в форме, следует нажать кнопку Стереть. При этом вызывается метод Refresh(), предназначенный для *перерисовывания* формы. В комментарии приведены варианты реализации очистки формы от всего нарисованного на ней пользователем, например, путем создания графического объекта CreateGraphics() и закрашивания формы в ее первоначальный цвет KnownColor::Control.

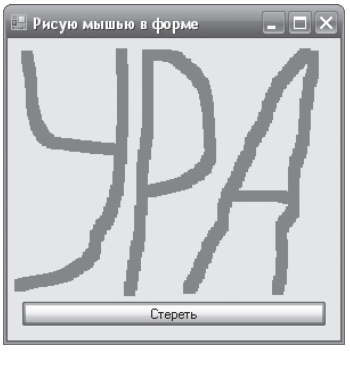

Рис. 5.8. Рисование с помощью указателя мыши в форме

Заметим, что можно было бы очистить область рисования более короткой командой Clear(Color::White), то есть закрасить форму белым цветом (White) либо

выбрать другой цвет из списка 146 цветов после ввода двоеточия (::) за словом Color. Однако ни один из 146 цветов не является первоначальным цветом формы (BackColor). Поэтому задаем этот цвет через другие константы цвета, представленные в перечислении Color::FromKnownColor. Также можно задать цвет как Color::FromName(«Control»). Можно использовать функцию перевода шестнадцатеричного кода цвета ColorTranslator::FromHtml(). Оба эти варианта представлены в комментарии. Цвет #EFEBDE является шестнадцатеричным представлением нужного нам цвета.

Очистить форму от всего нарисованного на ней, можно также свернув ее, а затем восстановив. Рисовать в форме можно как левой, так и правой кнопками мыши.

Убедиться в работоспособности программы можно, открыв решение РисМышью.sln в папке РисМышью.

Пример 44. Управление сплайном Безье

Система Visual Studio 2010 предоставляет возможность построения фундаментального сплайна и сплайна Безье. В принципе, программирование сплайна Безье сводится к подаче на вход соответствующей функции DrawBezier узловых точек, через которые проходит сплайн, а также дополнительных контрольных точек. В сплайне Безье *с помощью контрольных точек управляют* его кривизной, то есть *его формой*. На рис. 5.9 точки P_0 и P_3 являются узловыми, а через контрольные точки P_1 и P_2 проходят касательные к кривой в узловых точках.

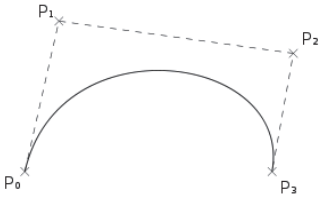

Рис. 5.9. Контрольные точки P_1 и P_2 определяют форму кривой

Перемещая контрольные точки P_1 и P_2 в ту или другую сторону, мы можем управлять формой (кривизной) сплайна. Глядя на рисунок, можно себе представить, как при некотором положении контрольных точек добиться того, что сплайн будет изгибаться в обе стороны, то есть на кривой появится точка перегиба. В русскоязычной документации на сайте www.msdn.com эти точки называют контрольными, а в англоязычной — «control points». Но «control» переводится еще и «управлять». Таким образом, эти точки лучше называть управляющими, поскольку они управляют формой кривой.

Рассмотрим следующую задачу: на экранную форму поместим две узловые точки, а обе управляющие точки соединим в одну. Далее при помощи мыши научимся перемещать эту одну управляющую точку и тем самым изменять форму сплайна.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Назовем этот проект Spline. Нам понадобятся пять пустых обработчиков разных события (листинг 5.14). Их легко получить, если на вкладке конструктора формы Form1.h[Design] через контекстное меню перейти на вкладку свойств, щелкнуть на значке молнии (Events), а затем последовательно выбирать те события, которые указаны в листинге.

Листинг 5.14. Управление формой кривой с помощью одной контрольной точки

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа строит сплайн Безье по двум узловым точкам, а две
        // контрольные (управляющие) точки совмещены в одну. Эта одна
        // управляющая точка отображается в форме в виде красного
        // прямоугольника. Перемещая указателем мыши управляющую точку,
        // мы регулируем форму сплайна (кривой)
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        array<PointF> ^ МассивТочек;
        // Запрещение управлять формой кривой:
        Boolean Управлять;
    private: System::Void Form1_Load(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 Управлять = false;
                 this->Text = "Управление сплайном Безье";
                 МассивТочек = gcnew array<PointF>(4);
                 // Начальная узловая точка:
                 МассивТочек[0] = PointF(50.0f, 50.0f);
                 // Две контрольные (управляющие) точки, мы их совместили в одну:
                 МассивТочек[1] = PointF(125.0f, 125.0f);
                 МассивТочек[2] = PointF(125.0f, 125.0f);
                 // Конечная узловая точка:
                 МассивТочек[3] = PointF(200.0f, 200.0f);
             }
    private: System::Void Form1_Paint(System::Object^  sender,
                 System::Windows::Forms::PaintEventArgs^  e)
             {
                 // Задаем поверхность для рисования из аргумента события e:
                 Graphics ^ Графика = e->Graphics;
                 Pen ^ Перо = gcnew Pen(Color::Blue, 3);
                 // Рисуем начальную и конечную узловые точки диаметром
                 4 пикселя:
                 Графика->DrawEllipse(Перо,
                     МассивТочек[0].X - 2, МассивТочек[0].Y - 2, 4.0f, 4.0f);
```

продолжение ⇩

Листинг 5.14 (продолжение)

```
                Графика->DrawEllipse(Перо,
                    МассивТочек[3].X - 2, МассивТочек[3].Y - 2, 4.0f, 4.0f);
                // Одна управляющая точка в виде прямоугольника красного цвета:
                Перо->Color = Color::Red;
                Графика->DrawRectangle(Перо, МассивТочек[1].X - 2,
                    МассивТочек[1].Y - 2, 4.0f, 4.0f);
                Перо->Color = Color::Blue;
                // Рисуем сплайн Безье:
                Графика->DrawBeziers(Перо, МассивТочек);
                delete Графика;
            }
    private: System::Void Form1_MouseMove(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
            {
                // Событие перемещения указателя мыши в области экранной формы.
                // Если указатель мыши расположен над управляющей точкой
                if (Math::Abs(e->X - МассивТочек[1].X) < 4.0f &&
                    Math::Abs(e->Y - МассивТочек[1].Y) < 4.0f &&
                    // и при этом нажата кнопка мыши
                    Управлять == true)
                { // то меняем координаты управляющей точки
                    МассивТочек[1].X = (float)e->X;
                    МассивТочек[1].Y = (float)e->Y;
                    МассивТочек[2].X = (float)e->X;
                    МассивТочек[2].Y = (float)e->Y;
                    // и обновляем (перерисовываем) форму:
                    this->Invalidate();
                }
            }
    private: System::Void Form1_MouseUp(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
            {
                // Если кнопку мыши отпустили, то запрещаем
                // управлять формой кривой:
                Управлять = false;
            }
    private: System::Void Form1_MouseDown(System::Object^  sender,
                System::Windows::Forms::MouseEventArgs^  e)
            {
                // Если нажата кнопка мыши, то разрешаем
                // управлять формой кривой:
                Управлять = true;
            }
    };
}
```

В программном коде массив из четырех точек (две узловые и две контрольные) объявляем вне процедур класса Form1, чтобы этот массив был «виден» из всех этих

процедур. Начальные значения элементов этого массива задаем при обработке события загрузки формы.

При обработке события перерисовки формы рисуем начальную и конечную узловые точки, одну управляющую точку в виде прямоугольника красного цвета и, наконец, сплайн Безье по текущим значениям массива четырех точек. Булева переменная Управлять, объявленная также вне всех процедур, принимает значение true при нажатой кнопке мыши, а если кнопку мыши отпустили, то этой переменной присваиваем значение false. При обработке события перемещения указателя мыши в пределах экранной формы происходит проверка, если указатель мыши расположен вблизи управляющей точки и при этом нажата кнопка мыши, то координаты управляющей точки назначаем равными текущим координатам положения мыши (их берем из аргументной переменной e) и перерисовываем (то есть обновляем) форму.

Фрагмент работы программы показан на рис. 5.10.

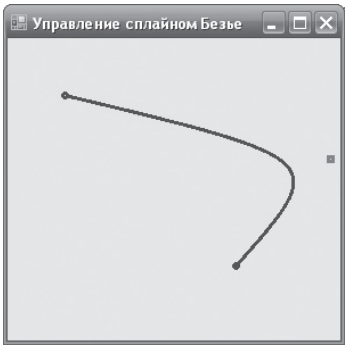

Рис. 5.10. Меняем форму кривой, перемещая мышью красный прямоугольник

Убедиться в работоспособности программы можно, открыв решение Spline.sln в папке Spline.

Пример 45. Построение графика методами класса Graphics

В этом разделе мы, используя в качестве исходных данных, например, объемы продаж каких-либо товаров по месяцам, построим график по точкам. Понятно, что таким же образом можно построить *любой график для других прикладных целей*.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Перенесем из панели элементов Toolbox в проектируемую форму элемент управления PictureBox (графическое поле) и командную кнопку Button. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.15.

Листинг 5.15. Программа для вывода графика в форму

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа рисует график объемов продаж по месяцам. Понятно, что таким
        // же образом можно построить любой график по точкам для других
        // прикладных целей
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Исходные данные для построения графика (то есть исходные точки):
        array<String^>^ Months;
        array<int>^ Sales;
        Graphics ^ Графика;
        // Далее, создаем объект Bitmap, который имеет
        // тот же размер и разрешение, что и PictureBox
        Bitmap ^ Растр;
        int ОтступСлева, ОтступСправа, ОтступСнизу, ОтступСверху;
        int ДлинаВертОси, ДлинаГоризОси, YГоризОси, Xmax, XНачЭпюры;
        // Шаг градуировки по горизонтальной и вертикальной осям:
        double ГоризШаг; int ВертШаг;
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                Months = gcnew array<String^> {"Янв", "Фев", "Март", "Апр",
                "Май",
                                               "Июнь", "Июль", "Авг", "Сент",
                                               "Окт", "Нояб", "Дек"};
                Sales = gcnew array<int> {335, 414, 572, 629, 750, 931,
                    753, 599, 422, 301, 245, 155};
                ОтступСлева = 35;  ОтступСправа = 15;
                ОтступСнизу = 20;  ОтступСверху = 10;
                this->Text = " Построение графика ";
                button1->Text = "Нарисовать график";
                this->ClientSize = System::Drawing::Size(593, 342);
                Растр = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height,
                    pictureBox1->CreateGraphics());
                // pictureBox1->BorderStyle = BorderStyle::FixedSingle;
                YГоризОси = pictureBox1->Height - ОтступСнизу;
                Xmax = pictureBox1->Width - ОтступСправа;
                ДлинаГоризОси = pictureBox1->Width - (ОтступСлева +
ОтступСправа);
                ДлинаВертОси = YГоризОси - ОтступСверху;
                ГоризШаг = (double)(ДлинаГоризОси / Sales->Length);
                ВертШаг = (int)(ДлинаВертОси / 10);
                XНачЭпюры = ОтступСлева + 30;
             } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

```
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
         { // Последовательно вызываем следующие процедуры:
             Графика = Graphics::FromImage(Растр);
             РисуемОси();
             РисуемГоризЛинии();
             РисуемВертЛинии();
             РисованиеЭпюры();
             pictureBox1->Image = Растр;
             // Освобождаем ресурсы, используемые объектом класса Graphics:
             delete Графика; // - эквивалент C#: Графика.Dispose
         } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
private: void РисуемОси()
         {
             //Pen Перо = new Pen(Color.Black, 2);
             Pen ^ Перо = gcnew Pen(Color::Black, 2);
             // Рисование вертикальной оси координат:
             Графика->DrawLine(Перо, ОтступСлева, YГоризОси,
                 ОтступСлева, ОтступСверху);
             // Рисование горизонтальной оси координат:
             Графика->DrawLine(Перо, ОтступСлева, YГоризОси,
                 Xmax, YГоризОси);
             auto Шрифт = gcnew Drawing::Font("Arial", 8);
             for (int i = 1; i <= 10; i++)
             {  // Рисуем "усики" на вертикальной координатной оси:
                 int Y = YГоризОси - i * ВертШаг;
                 Графика->DrawLine(Перо, ОтступСлева - 5, Y, ОтступСлева, Y);
                 // Подписываем значения продаж через каждые 100 единиц:
                 Графика->DrawString((i * 100).ToString(), Шрифт,
                                                 Brushes::Black, 2, Y - 5.F);
             }
             // Подписываем месяцы на горизонтальной оси:
             for (int i = 0; i <= Months->Length - 1; i++)
                 Графика->DrawString(Months[i], Шрифт, Brushes::Black,
                 ОтступСлева + 18.F + (int)(i * ГоризШаг), YГоризОси + 4.F);
         } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
private: void РисуемГоризЛинии()
         {
             Pen ^ ТонкоеПеро = gcnew Pen(Color::LightGray, 1);
             for (int i = 1; i <= 10; i++)
             {  // Рисуем горизонтальные почти "прозрачные" линии:
                 int Y = YГоризОси - ВертШаг * i;
                 Графика->DrawLine(ТонкоеПеро, ОтступСлева + 3, Y, Xmax, Y);
             }
             delete ТонкоеПеро; // - эквивалент C#: ТонкоеПеро.Dispose();
         } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
private: void РисуемВертЛинии()
         {  // Рисуем вертикальные почти "прозрачные" линии
             Pen ^ ТонкоеПеро = gcnew Pen(Color::Bisque, 1);
```

продолжение ⇘

Листинг 5.15 (продолжение)

```
                for (int i = 0; i <= Months->Length - 1; i++)
                {
                    int X = XНачЭпюры + (int)(ГоризШаг * i);
                    Графика->DrawLine(ТонкоеПеро, X, ОтступСверху,
                                                  X, YГоризОси - 4);
                }
                delete ТонкоеПеро;
            } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
    private: void РисованиеЭпюры()
            {
                double ВертМасштаб = (double)ДлинаВертОси / 1000;
                // значения ординат на экране:
                array<int>^ Y = gcnew array<int>(Sales->Length);
                // значения абсцисс на экране:
                array<int>^ X = gcnew array<int>(Sales->Length);
                for (int i = 0; i <= Sales->Length - 1; i++)
                {  // Вычисляем графические координаты точек:
                    Y[i] = YГоризОси - (int)(Sales[i] * ВертМасштаб);
                    // Отнимаем значения продаж, поскольку ось Y экрана
                    // направлена вниз
                    X[i] = XНачЭпюры + (int)(ГоризШаг * i);
                }
                // Рисуем первый кружок:
                Pen ^ Перо = gcnew Pen(Color::Blue, 3);
                Графика->DrawEllipse(Перо, X[0] - 2, Y[0] - 2, 4, 4);
                for (int i = 0; i <= Sales->Length - 2; i++)
                {  // Цикл по линиям между точками:
                    Графика->DrawLine(Перо, X[i], Y[i], X[i + 1], Y[i + 1]);
                    // Отнимаем 2, поскольку диаметр (ширина) точки = 4:
                    Графика->DrawEllipse(Перо, X[i + 1] - 2, Y[i + 1] - 2,
                                                                4, 4);
                }
            } // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
    };
}
```

Как видно из текста программы, вначале объявляем некоторые переменные так, чтобы они были видны из всех процедур класса. Строковый массив Months содержит названия месяцев, которые пользователь нашего программного кода может менять в зависимости от контекста строящегося графика. В любом случае записанные строки в этом массиве будут отображаться по горизонтальной оси графика. Массив целых чисел Sales содержит объемы продаж по каждому месяцу, они соответствуют ординатам графика. Оба массива должны иметь одинаковую размерность, но не обязательно равную двенадцати.

При обработке события «щелчок мыши на кнопке Button» создаем объект класса Graphics, используя элемент управления PictureBox (графическое поле), а затем, вызывая соответствующие процедуры, поэтапно рисуем координатные оси,

сетку из горизонтальных и вертикальных линий и непосредственно эпюру. Чтобы успешно, минимальными усилиями и с возможностью дальнейшего совершенствования программы построить график, следует как можно более понятно назвать некоторые ключевые, часто встречающиеся интервалы и координаты на рисунке. Названия этих интервалов должны быть осмысленными. Скажем, переменная ОтступСлева хранит число пикселов, на которое следует отступить, чтобы построить на графике, например, вертикальную ось продаж. Кроме очевидных названий упомянем переменную YГоризОси, это графическая ордината (ось x направлена слева направо, а ось y — сверху вниз) горизонтальной оси графика, на которой подписываются месяцы. Переменная XMax содержит в себе значение максимальной абсциссы (см. рис. 5.7), правее которого уже никаких построений нет. Переменная XНачЭпюры — это значение абсциссы первой построенной точки графика. Используя такие понятные из контекста названия переменных, да еще и на русском языке, мы значительно облегчаем весь процесс программирования, упрощаем сопровождение и модификацию программы.

Построенный данной программой график показан на рис. 5.11.

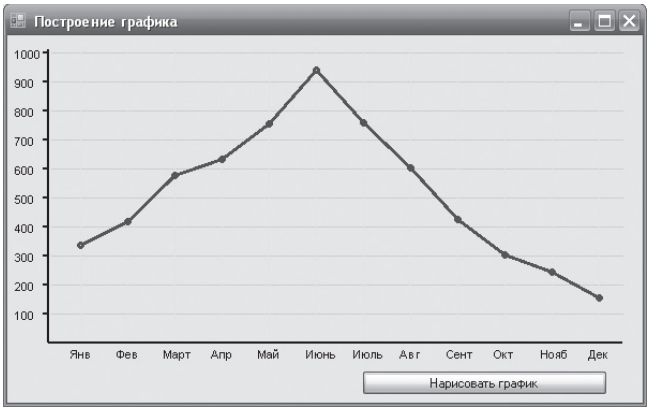

Рис. 5.11. График объемов продаж по месяцам

Убедиться в работоспособности программы можно, открыв решение График.sln в папке График.

Управление буфером обмена с данными в текстовом и графическом форматах

6

Пример 46. Буфер обмена с данными в текстовом формате

Напишем программу для управления буфером обмена с данными в текстовом формате. Эта программа будет позволять *записывать какой-либо текст в буфер обмена (БО), а затем извлекать этот текст из БО. Для этой цели в форме создадим два текстовых поля, а также две командные кнопки под этими полями. Одну кнопку назовем* Записать в БО, а другую — Извлечь из БО.

Чтобы записать какой-либо текст в БО, нужно записать его в верхнее поле, выделить (с помощью клавиш управления курсором при нажатой клавише Shift), а затем нажать кнопку Записать в БО. Нажимая кнопку Записать в БО, мы как бы моделируем комбинацию клавиш Ctrl+C.

Далее записанный в БО текст можно читать в каком-либо текстовом редакторе или вывести в нижнее текстовое поле прямо в нашей форме, для этого служит кнопка Извлечь из БО (рис. 6.1). Текст данной программы приведен в листинге 6.1.

Листинг 6.1. Запись текстовых данных в буфер обмена и их чтение

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Эта программа имеет возможность записи какого-либо текста
        // в буфер обмена, а затем извлечения этого текста из буфера обмена
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
```

```
        {
            this->Text = "Введите текст в верхнее поле";
            textBox1->Clear();  textBox2->Clear();
            textBox1->TabIndex = 0;
            button1->Text = "Записать в БО";
            button2->Text = "Извлечь из БО";
        }
    private: System::Void button1_Click(System::Object^  sender,
            System::EventArgs^  e)
        {
            // Записать выделенный в верхнем поле текст в БО
            if (textBox1->SelectedText != String::Empty)
            {
                Clipboard::SetDataObject(textBox1->SelectedText);
                textBox2->Text = String::Empty;
            }
            else
                textBox2->Text = "В верхнем поле текст не выделен";
        }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Объявление объекта-получателя из БО
            IDataObject ^ Получатель = Clipboard::GetDataObject();
            // Если данные в БО представлены в текстовом формате...
            if (Получатель->GetDataPresent(DataFormats::Text) == true)
                // то записать их в Text тоже в текстовом формате
                textBox2->Text = Получатель->
                                GetData(DataFormats::Text)->ToString();
            else
                textBox2->Text = "Запишите что-либо в буфер обмена";
        }
    };
}
```

Как видно из программного кода, при обработке события «щелчок на верхней кнопке», если текст в верхнем поле выделен, то записываем его (SelectedText) в буфер обмена (Clipboard) командой (методом) SetDataObject, иначе (else) сообщаем в нижнем поле textBox2 о том, что в верхнем поле текст не выделен.

Напомню, что, нажимая кнопку Извлечь из БО, пользователь нашей программы должен увидеть в нижнем поле содержимое буфера обмена. Для этого объявляем объектную переменную Получатель — это объект-получатель из буфера обмена. Данная переменная сознательно названа по-русски для большей выразительности. Далее следует проверка: в текстовом ли формате (DataFormat::Text) данные представлены в буфере обмена. Если формат текстовый, то в текстовое поле TextBox2 записываем содержимое буфера обмена, используя функцию ToString. Эта функция конвертирует строковую часть объекта в строковую переменную.

Пример работы приложения показан на рис. 6.1.

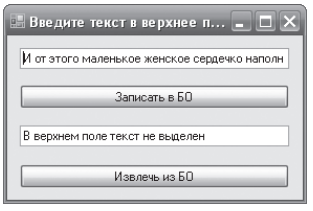

Рис. 6.1. Запись текстовых данных в буфер обмена и их чтение

Убедиться в работоспособности программы можно, открыв решение БуферОбменаTXT.sln в папке БуферОбменаTXT.

Пример 47. Элемент управления PictureBox. Буфер обмена с растровыми данными

Обсудим программу, которая оперирует буфером обмена, когда тот содержит изображение. Здесь мы будем использовать элемент управления PictureBox (графическое поле), который способен отображать в своем поле растровые файлы различных форматов, в том числе BMP, JPEG, PNG, GIF и др. Возможность отображения в PictureBox GIF-файлов позволяет просматривать анимацию, то есть движущееся изображение, в экранной форме вашей программы. Пример отображения такой GIF-анимации вы можете посмотреть, открыв решение PictureBoxGif.sln в папке PictureBoxGif. Однако элемент управления PictureBox не способен отображать Flash-файлы в формате SWF, которые сочетают в себе векторную графику, растровую графику и воспроизведение звука. Замечу, что файлы SWF воспроизводят элемент управления Microsoft WebBrowser, который мы обсудим в *главе 8*.

Итак, наша программа выводит в поле элемента управления PictureBox изображение из растрового файла (например, PNG). При этом изображение записывается в БО. Пользователь может убедиться в этом, например, запустив Paint — стандартный графический редактор ОС Windows. Далее пользователь может поместить в БО любое изображение с помощью какого-нибудь графического редактора, например того же Paint, MS Office Picture Manager, ACDSee или др. Затем, нажав кнопку Извлечь из БО нашей программы, мы получим в форме содержимое БО.

Для создания данной программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели Toolbox в проектируемую форму перетащим элементы управления PictureBox (графическое поле) и кнопку Button. Текст программы приведен в листинге 6.2.

Листинг 6.2. Обмен графических данных через буфер обмена

```
        // ..............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
    }
```

```
#pragma endregion
        // Программа оперирует буфером обмена, когда тот содержит изображение
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 Form1::Text = "Содержимое БО:";
                 button1->Text = "Извлечь из БО";
                 // Размеры графического окна:
                 pictureBox1->Size = Drawing::Size(184, 142);
                 // Записать в PictureBox изображение из файла:
                 try
                 {
                     pictureBox1->Image = Image::FromFile(
                         IO::Directory::GetCurrentDirectory() + "\\poryv.png");
                 }
                 catch (IO::FileNotFoundException^ Ситуация)
                 {   // Обработка исключительной ситуации:
                     MessageBox::Show(Ситуация->Message + "\nНет такого файла",
                         "Ошибка", MessageBoxButtons::OK,
                         MessageBoxIcon::Exclamation);
                     button1->Enabled = false;
                     return;
                 }
                 // Записать в БО изображение из графического окна формы
                 Clipboard::SetDataObject(pictureBox1->Image);
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 // Объявление объекта-получателя из буфера обмена
                 IDataObject ^ Получатель = Clipboard::GetDataObject();
                 Bitmap ^ Растр;
                 // Если данные в БО представлены в формате Bitmap...
                 if (Получатель->GetDataPresent(DataFormats::Bitmap) == true)
                 { // то записать эти данные из БО в переменную
                   // Растр в формате Bitmap:
                     Растр = (Bitmap^)Получатель->GetData(DataFormats::Bitmap);
                     pictureBox1->Image = Растр;
                 }
             }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы в графическое поле pictureBox1 записываем какой-нибудь файл, например poryv.png. Функция GetCurrentDirectory() возвращает полный путь текущей папки. Далее по команде

```
Clipboard::SetDataObject(pictureBox1->Image)
```

происходит запись содержимого графического поля в буфер обмена.

Теперь проверим содержимое БО с помощью какого-либо графического редактора, например Paint. Далее мы можем записать что-либо в буфер обмена, опять

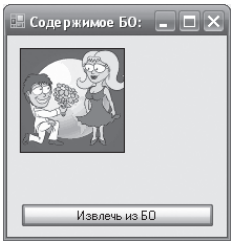

Рис. 6.2. Извлечение изображения из буфера обмена

же используя какой-нибудь графический редактор. Нажатие кнопки Извлечь из БО нашей программы в форме приведет к появлению изображения, находящегося в буфере обмена. Фрагмент работы программы представлен на рис. 6.2.

В программном коде при обработке события «щелчок на кнопке Извлечь из БО» объявляем объектную переменную Получатель. Это объект, с помощью которого мы будем получать изображение, записанное в буфер обмена. Эта переменная сознательно названа по-русски для большей выразительности. Далее проверим, записаны ли данные, представленные в БО, в формате растровой графики Bitmap. Если да, то следует записать данные из БО в переменную Растр в формате Bitmap с помощью объектной переменной Получатель, используя неявное преобразование в переменную типа Bitmap. И наконец, чтобы изображение появилось в элементе управления pictureBox1 в форме, присваиваем свойству Image значение переменной Растр.

Убедиться в работоспособности программы можно, открыв решение БуферОбменаBitmap.sln в папке БуферОбменаBitmap.

Пример 48. Имитация нажатия комбинации клавиш Alt+PrintScreen

Как известно, при нажатии клавиши PrintScreen происходит копирование изображения экрана в буфер обмена, то есть получаем так называемый screen shot — *моментальный снимок экрана* (другое название — screen capture). После извлечения из БО графической копии экрана в любом графическом редакторе (например, Paint) эту графическую копию можно редактировать.

При нажатии комбинации клавиш Alt+PrintScreen в буфер обмена копируется не весь экран, а *только активное окно*. Для имитации нажатия сочетания клавиш Alt+PrintScreen современная система программирования Visual Studio в пространстве имен System::Windows::Forms *имеет класс* SendKeys, который предоставляет методы для отправки приложению сообщений о нажатиях клавиш, в том числе и нашей комбинации клавиш Alt+PrintScreen.

Данная программа сознательно максимально упрощена, это только подчеркнет ее элегантность. Здесь, в форме мы имеем только командную кнопку. При обработке события «щелчок на кнопке» будем имитировать нажатие комбинации клавиш Alt+PrintScreen методом Send класса SendKeys. При этом форму можно как угодно менять — растягивать по ширине и по высоте, но после нажатия кнопки в форме изображение формы запишется в буфер обмена. Следует убедиться в этом: запустить Paint и извлечь графическую копию формы из буфера обмена. Текст программы представлен в листинге 6.3.

Листинг 6.3. Имитация нажатия комбинации клавиш Alt+<PrintScreen>

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программная имитация нажатия клавиш Alt+<PrintScreen>
        // методом Send класса SendKeys
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Имитируем нажатие Alt+<PrintScreen>";
                button1->Text = "методом Send класса SendKeys";
             }
    private: System::Void button1_Click(System::Object^ sender,
                                                    System::EventArgs^ e)
             {
                // Метод SendKeys::Send посылает сообщение активному приложению
                // о нажатии клавиш Alt+<PrintScreen>
                SendKeys::Send("%{PRTSC}");
                // Так можно получить символьное представление клавиши:
                // String ^ QQ = Keys::PrintScreen.ToString();
             }
    };
}
```

В этой программе при обработке события «щелчок на кнопке» мы решаем нашу задачу современными средствами. Метод Send класса SendKeys посылает сообщение активному приложению о нажатии комбинации клавиш Alt+PrintScreen. Кодом клавиши PrintScreen является код {PRTSC}. Чтобы указать сочетание клавиш Alt+PrintScreen, следует в этот код добавить символ процента: %{PRTSC}. Коды других клавиш можно посмотреть по адресу:

http://msdn.microsoft.com/ru-ru/library/system.windows.forms.sendkeys.aspx

Убедиться в работоспособности программы можно, открыв решение AltPrintScreen.sln в папке AltPrintScreen.

Пример 49. Запись содержимого буфера обмена в BMP-файл

Напишем программу, которая читает буфер обмена, и если данные в нем представлены в формате растровой графики, то записывает эти данные в BMP-файл. Для этой цели запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Перенесем из

панели элементов Toolbox в проектируемую форму командную кнопку Button. Текст этой программы приведен в листинге 6.4.

Листинг 6.4. Запись содержимого буфера обмена в BMP-файл

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа читает буфер обмена, и если данные в нем представлены
        // в формате растровой графики, то записывает их в BMP-файл
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Сохраняю копию БО в BMP-файл";
                button1->Text = "Сохранить";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Объявление объекта-получателя из буфера обмена
                auto Получатель = Clipboard::GetDataObject();
                Bitmap ^ Растр;
                // Если данные в буфере обмена представлены в формате Bitmap...
                if (Получатель->GetDataPresent(DataFormats::Bitmap) == true)
                {  // то записать их из БО в переменную Растр в формате Bitmap
                    Растр = (Bitmap^)Получатель->GetData(DataFormats::Bitmap);
                    // Сохранить изображение в файле Clip.bmp
                    Растр->Save("C:\\Clip.BMP");
                    //this->Text = "Сохранено в файле C:\\Clip.BMP";
                    //button1->Text = "Еще записать?";
                    MessageBox::Show
                        ("Изображение из БО записано в файл C:\\Clip.BMP",
                                                              "Успех");
                }
                else
                    // В БО нет данных в формате изображений
                    MessageBox::Show(
                        "В буфере обмена нет данных в формате Bitmap",
                        "Запишите какое-либо изображение в БО");
             }
    };
}
```

В программном коде при обработке события «щелчок на кнопке Сохранить» объявляем объектную переменную Получатель — это объект-получатель из буфера обмена. Далее следует проверка: записаны ли данные, представленные в буфере обмена, в формате растровой графики Bitmap. Если да, то записываем данные

из буфера обмена в переменную Растр в формате Bitmap с помощью объектной переменной Получатель, используя неявное преобразование в переменную типа Bitmap. Сохранить изображение Растр на винчестер мы можем, воспользовавшись методом Save(). Чтобы излишне не запутать читателя и упростить программу, я не стал организовывать запись файла в диалоге. При необходимости вы сделаете это самостоятельно, воспользовавшись элементом управления SaveFileDialog. Фрагмент работы данной программы представлен на рис. 6.3.

Рис. 6.3. Запись изображения из буфера обмена в файл

Убедиться в работоспособности программы можно, открыв решение БуферОбменаSaveBMP.sln в папке БуферОбменаSaveBMP.

Пример 50. Использование таймера Timer

Я несколько раз *встречал задачу записи моментальных снимков экрана (*screen shot) в растровые файлы с некоторым промежутком времени, например пять секунд. Кстати, такая задача была выставлена на тендер на сайте оффшорного программирования (www.rentacoder.com). За ее решение указывалось вознаграждение 100 долларов США. Конечно, сразу появляется очень много желающих эту задачу решить, но попробуйте выиграть тендер.

Следующая задача является частью упомянутой задачи, мы ее сформулируем следующим образом: после запуска программы должна отображаться форма с элементом управления ListBox, а через две секунды в список добавляется запись «Прошло две секунды», далее через каждые две секунды будет происходить добавление в список аналогичной записи. На этой задаче мы освоим технологию работы с таймером.

Для реализации программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем в проектируемую форму из панели элементов Toolbox перетащим элемент управления Timer и список ListBox. Программный код приведен в листинге 6.5.

Листинг 6.5. Использование таймера

```
        // ...............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
    }
```

продолжение ⇨

Листинг 6.5 (продолжение)

```
#pragma endregion
        // Демонстрация использования таймера Timer. После запуска программы
        // показываются форма и элемент управления список элементов ListBox.
        // Через 2 секунды в списке элементов появляется запись «Прошло две
        // секунды», и через каждые последующие 2 секунды в список добавляется
        // аналогичная запись
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 this->Text = "Timer";
                 timer1->Interval = 2000; // - 2 секунды
                 // Старт отчета времени
                 timer1->Enabled = true;  // - время пошло
             }
    private: System::Void timer1_Tick(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 listBox1->Items->Add("Прошло две секунды");
             }
    };
}
```

Экземпляр класса Timer — это невидимый во время работы программы элемент управления, предназначенный для периодического генерирования события Tick. В программном коде сразу после инициализации конструктором компонентов программы задаем интервал времени (Interval), равный 2000 миллисекунд (две секунды). Каждые две секунды *будет возникать событие* Tick, при этом в список элементов listBox1 будет добавляться запись «Прошло две секунды». Этот текст будет выводиться каждые две секунды до тех пор, пока пользователь программы не щелкнет на кнопке формы Закрыть. На рис. 6.4 показан фрагмент работы программы.

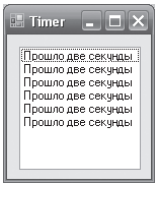

Рис. 6.4. Вывод сообщения

Убедиться в работоспособности программы можно, открыв решение ПростоTimer.sln в папке ПростоTimer.

Пример 51. Запись в файлы текущих состояний экрана каждые пять секунд

Как уже говорилось в предыдущем разделе, работа с таймером — довольно распространенная задача. Напишем очередную программу: после запуска программы через каждые пять секунд снимается текущее состояние экрана и записывается в файлы Pic1.BMP, ..., Pic5.BMP.

Для решения данной задачи мы уже имеем весь инструментарий, он был рассмотрен в предыдущих задачах. А именно, задание интервала времени в пять секунд

с помощью элемента управления Timer, эмуляция нажатия клавиш Alt+PrintScreen для записи текущего состояния экрана в буфер обмена и, наконец, чтение буфера обмена в формате изображения и запись этого изображения *в файл* BMP.

Для решения задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов перетаскиваем в проектируемую форму элементы управления Timer и Button. Текст программы приведен в листинге 6.6.

Листинг 6.6. Запись в файлы текущих состояний экрана с интервалом 5 секунд

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа после запуска каждые пять секунд делает снимок текущего
        // состояния экрана и записывает эти снимки в файлы Pic1.BMP, Pic2.BMP
        // и т. д. Количество таких записей в файл — пять
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        int i; // счет секунд
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 i = 0; // счет секунд
                 this->Text = "Запись каждые 5 секунд в файл";
                 button1->Text = "Пуск";
             }
    private: System::Void timer1_Tick(System::Object^  sender,
                                      System::EventArgs^  e)
             {
                 i = i + 1;
                 this->Text = String::Format("Прошло {0} секунд", i);
                 if (i >= 28) { timer1->Enabled = false; this->Close(); }
                 if (i % 5 != 0) return;
                 // Имитируем нажатие клавиш Alt+<PrintScreen>
                 SendKeys::Send("%{PRTSC}");
                 // Объявление объекта-получателя из буфера обмена
                 auto Получатель = Clipboard::GetDataObject();
                 Bitmap ^ Растр;
                 // Если данные в буфере обмена представлены в формате Bitmap,
                 // то записать
                 if (Получатель->GetDataPresent(DataFormats::Bitmap) == true)
                 {  // эти данные из буфера обмена в переменную Растр в формате
                 Bitmap
                     Растр = (Bitmap^)Получатель->GetData(DataFormats::Bitmap);
                     // Сохранить изображение из переменной Растр
                     // в файл C:\Pic1, C:\Pic2, C:\Pic3, ...
```

продолжение ⇨

Листинг 6.6 (продолжение)

```
                    String ^ ИмяФайла = String::Format("C:\\Pic{0}.BMP", i / 5);
                    Растр->Save(ИмяФайла);
                }
            }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
            {
                this->Text = String::Format("Прошло 0 секунд");
                timer1->Interval = 1000; // равно одной секунде
                timer1->Enabled = true;  // время пошло
            }
    };
}
```

Как видно из программного кода, структура программы включает в себя обработку события «щелчок на кнопке button1_Click» и обработку события Timer1_Tick. В начале программы задаем переменную i, которая считает, сколько раз программа сделает запись в буфер обмена, а из буфера обмена — в файл. При щелчке на кнопке Пуск задаем интервал времени Interval, равный 1000 миллисекунд, то есть одной секунде. Далее даем команду таймеру начать отсчет времени timer1->Enabled = true, и через каждую секунду наступает событие timer1_Tick(), то есть управление переходит этой процедуре.

При обработке события timer1_Tick наращиваем значение переменной i, которая ведет счет секундам после старта таймера. Выражение i % 5 вычисляет целочисленный остаток после деления первого числового выражения на второе. Понятно, что если число i будет кратно пяти, то этот остаток будет равен нулю, и только в этом случае будет происходить имитация нажатия клавиш Alt+PrintScreen и запись содержимого буфера обмена в файл.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке SaveСкриншотКаждые5сек.

7 Ввод и вывод табличных данных. Решение системы уравнений

Пример 52. Формирование таблицы. Функция String::Format

При создании инженерных, экономических и других приложений задача зачастую сводится к вводу данных, расчету (обработке введенных данных), а затем выводу результатов вычислений в виде таблицы. В этом разделе мы рассмотрим типичную задачу: как оптимально сформировать таблицу, а затем вывести ее на экран с возможностью распечатывания на принтере.

Чтобы лучше разобраться в процессе формирования таблицы, абстрагируемся от ввода данных и расчетов и сосредоточимся только на сути. Например, у нас есть список телефонов наших знакомых и нам хотелось бы представить эту информацию *в виде наглядной таблицы. Предположим, что результаты обработки записаны в два массива: массив имен знакомых* Imena и массив телефонов Tel. Наша программа формирует таблицу из этих двух массивов в текстовом поле TextBox. Кроме того, в программе участвует элемент управления MenuStrip для организации раскрывающегося меню, с помощью которого мы можем вывести сформированную таблицу в Блокнот (notepad.exe) с целью последующей корректировки (редактирования) и вывода на печать.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов перетаскиваем в проектируемую форму следующие элементы управления: текстовое поле TextBox со свойством Multiline = true и меню MenuStrip с пунктами меню Файл, Показать таблицу в Блокноте и Выход. Текст программы представлен в листинге 7.1.

Листинг 7.1. Формирование таблицы

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа формирует таблицу из двух строковых массивов в текстовом
        // поле, используя функцию String::Format. Кроме того, в программе
        // участвует элемент управления MenuStrip для организации
        // раскрывающегося меню, с помощью которого пользователь выводит
        // сформированную таблицу в Блокнот с целью последующего редактирования
        // и вывода на печать
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 this->textBox1->Multiline = true;
                 this->textBox1->Size = Drawing::Size(320, 216);
                 this->ClientSize = Drawing::Size(342, 266);
                 this->Text = "Формирование таблицы";
                 array<String^>^ Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                                  "Справка по тел", "Александр Степанович",
                                           "Мама - дом", "Карапузова Таня",
                                           "Погода сегодня", "Театр Браво"};
                 array<String^>^ Tel = {"274-88-17","+38(067)7030356",
                            "22-345-72", "009", "223-67-67 доп 32-67",
                                  "570-38-76", "201-72-23-прямой моб",
                                                "001", "216-40-22"};
                 textBox1->ScrollBars = ScrollBars::Vertical;
                 textBox1->Font = gcnew Drawing::Font("Courier New", 9.0F);
                 textBox1->Text = "ТАБЛИЦА ТЕЛЕФОНОВ\r\n\r\n";
                 for (int i = 0; i <= 8; i++)
                     textBox1->Text += String::Format(
                     "{0, -21} {1, -21}", Imena[i], Tel[i]) + "\r\n";
                 textBox1->Text += "\r\nПРИМЕЧАНИЕ:" +
                     "\r\nдля корректного отображения таблицы" +
                     "\r\nв Блокноте укажите шрифт Courier New";
                 // Запись таблицы в текстовый файл C:\Table.txt.
                 // Создание экземпляра StreamWriter для записи в файл
                 auto Писатель = gcnew IO::
                     StreamWriter("C:\\Table.txt", false,
                         System::Text::Encoding::GetEncoding(1251));
                 // - здесь заказ кодовой страницы Win1251 для русских букв
                 Писатель->Write(textBox1->Text);
                 Писатель->Close();
             }
    private: System::Void показатьТаблицуВБлокнотеToolStripMenuItem_Click(
                                 System::Object^  sender, System::EventArgs^  e)
```

```
        {
            try
            {
                Diagnostics::Process::Start("Notepad", "C:\\Table.txt");
            }
            catch (Exception ^ Ситуация)
            {  // Отчет об ошибках
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    private: System::Void выходToolStripMenuItem_Click(System::Object^  sender,
                                                       System::EventArgs^  e)
        {
            // Выход из программы:
            this->Close();
        }
    };
}
```

Чтобы таблица отображалась корректно в текстовом поле TextBox, мы заказали шрифт Courier New. Особенность этого шрифта заключается в том, что каждый символ (буква, точка, запятая и др.) этого шрифта имеет одну и ту же ширину, как это было на печатающей машинке. Поэтому, пользуясь шрифтом Courier New, удобно строить таблицы. Таким же замечательным свойством обладает, например, шрифт Consolas.

Далее в пошаговом цикле for мы использовали оператор +=, он означает: сцепить текущее содержание текстовой переменной textBox1->Text с текстом, представленным справа. Функция String::Format возвращает строку, сформированную по формату. Формат заключен в кавычки. Ноль в первых фигурных скобках означает: вставить вместо нуля переменную Imena[i], а единица во вторых фигурных *скобках — вставить вместо единицы строку* Tel[i]. Число 21 означает, что длина строки в любом случае будет состоять из 21 символа (недостающими символами будут пробелы), причем знак «минус» заставляет прижимать текст к левому краю. Символы \r\n означают, что следует начать текст с новой строки. Внешний вид таблицы в текстовом поле формы показан на рис. 7.1.

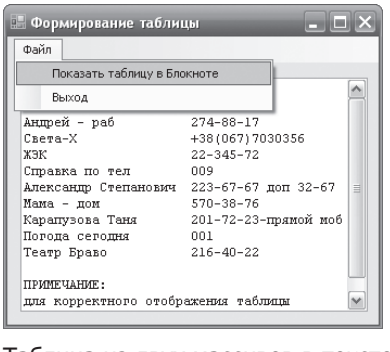

Рис. 7.1. Таблица из двух массивов в текстовом поле

После формирования всех строк textBox1->Text записываем их в текстовый файл C:\Table.txt через StreamWriter с кодовой таблицей Windows 1251. Подробное обсуждение этого фрагмента программы вы можете посмотреть в примере 26 (см. главу 4).

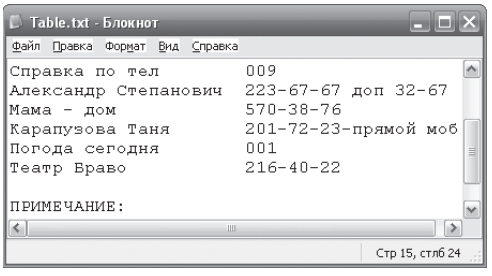

Рис. 7.2. Просмотр созданной таблицы в Блокноте

При выборе пользователем пункта меню Показать таблицу в Блокноте система создает событие, которое обрабатывается в соответствующей процедуре. Здесь вызываем программу операционной системы Блокнот (notepad.exe) для открытия файла C:\Table.txt (рис. 7.2).

Убедиться в работоспособности программы можно, открыв решение ТаблTxt.sln в папке ТаблTxt.

Пример 53. Форматирование Double-переменных в виде таблицы. Вывод таблицы на печать. Поток StringReader

В данном разделе мы рассмотрим программу, которая решает аналогичную задачу, однако в результате вычислений мы будем иметь не строковые переменные String, а два массива переменных с двойной точностью Double. Например, пусть в результате расчетов получены координаты точек на местности *X* и *Y*. Набор этих координат необходимо оформить в виде таблицы. Таблицу следует назвать «Каталог координат». Координаты в таблице должны быть округлены до двух знаков после запятой. Сформированную таблицу следует вывести в текстовое поле TextBox. Далее надо организовать возможность печати таблицы на принтере. Заметьте, что при решении этой задачи мы не пользуемся Блокнотом.

Начнем: как и в предыдущем разделе, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем в проектируемой форме с помощью панели элементов управления Toolbox создадим текстовое поле TextBox, выберем элементы управления MenuStip и PrintDocument. На вкладке Design подготовим пункты меню Печать и Выход, как показано на рис. 7.3. Чтобы растянуть текстовое поле на всю форму, в свойстве Multiline укажем True (разрешим введение множества строк).

Текст программы представлен в листинге 7.2.

Листинг 7.2. Формирование таблицы и вывод ее на печать

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа формирует таблицу на основании двух массивов переменных
        // с двойной точностью. Данную таблицу программа демонстрирует
        // пользователю в текстовом поле TextBox. Есть возможность распечатать
        // таблицу на принтере
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        IO::StringReader ^ Читатель; // - внешняя переменная
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                this->Text = "Формирование таблицы";
                Double X[] = {5342736.17653, 2345.3333, 234683.853749,
                              2438454.825368, 3425.72564, 5243.25,
                              537407.6236, 6354328.9876, 5342.243};
                Double Y[] = {27488.17, 3806703.356, 22345.72,
                              54285.34, 2236767.3267, 57038.76,
                              201722.3, 26434.001, 2164.022};
                // Массив можно объявить также и так:
                // array<Double> ^ Y = {27488.17, 3806703.356, 22345.72,
                //                      54285.34, 2236767.3267, 57038.76,
                //                      201722.3, 26434.001, 2164.022};
                textBox1->Multiline = true;
                textBox1->ScrollBars = ScrollBars::Vertical;
                textBox1->Font = gcnew Drawing::Font("Courier New", 9.0F);
                textBox1->Text = "КАТАЛОГ КООРДИНАТ\r\n";
                textBox1->Text += "---------------------------------\r\n";
                textBox1->Text += "|Пункт|      X    |    Y      |\r\n";
                textBox1->Text += "---------------------------------\r\n";
                for (int i = 0; i <= 8; i++)
                    textBox1->Text += String::Format(
                    "| {0,3:D}|{1,10:F2}|{2,10:F2} |",
                        i, X[i], Y[i]) + "\r\n";
                textBox1->Text += "---------------------------------\r\n";
             }
    private: System::Void печатьToolStripMenuItem_Click(
                System::Object^  sender, System::EventArgs^  e)
             {
                // Пункт меню "Печать"
                try
                {  // Создание потока Читатель для чтения из строки:
                    Читатель = gcnew IO::StringReader(textBox1->Text);
                    try
```

продолжение ⇩

Листинг 7.2 (продолжение)

```
                    {
                        printDocument1->Print();
                    }
                    finally
                    {
                        Читатель->Close();
                    }
                }
                catch (Exception ^ Ситуация)
                {
                    MessageBox::Show(Ситуация->Message);
                }
            }
    private: System::Void выходToolStripMenuItem_Click(
                System::Object^ sender, System::EventArgs^  e)
            {
                // Выход из программы
                this->Close();
            }
    };
}
```

Как видно из текста программы, формирование таблицы также происходит в цикле for с помощью функции String::Format. В фигурных скобках числа 0, 1 и 2 означают, что вместо фигурных скобок следует вставлять переменные i, X[i], Y[i]. Выражение 3:D означает, что переменную i следует размещать в трех символах по формату целых переменных D. Выражение 10:F2 означает, что переменную X(i) следует размещать *в десяти символах по фиксированному формату с двумя знаками после запятой*.

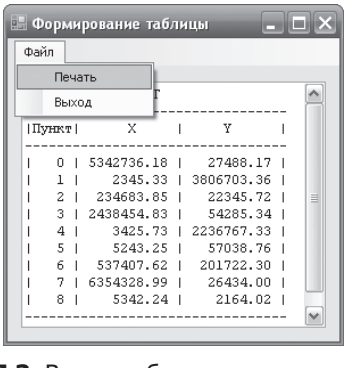

Рис. 7.3. Вывод таблицы в текстовое поле

При обработке события «щелчок на пункте меню Печать» (рис. 7.3) в блоках try... finaly...catch создаем поток Читатель, однако не для чтения из файла, а для чтения из текстовой переменной textBox1->Text. В этом случае мы обращаемся с потоком Читатель так же, как при операциях с файлами, но при этом не обращаемся к внешней

памяти (диску). Поэтому организация многостраничной печати остается абсолютно такой же, как в примере 31 (см. главу 4).

Как видно из приведенной программы, для того чтобы просмотреть, откорректировать и распечатать на принтере таблицу (инженерных или экономических вычислений), совершенно необязательно записывать эту таблицу в текстовый файл и читать его с помощью Блокнота.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТаблTxtPrint.

Пример 54. Вывод таблицы в Internet Explorer

Рассмотрим несколько необычный подход к выводу таблицы для просмотра и распечатывания на принтере. Запишем таблицу в текстовый файл в формате HTML, затем прочитаем ее с помощью обозревателя (браузера) веб-страниц Internet Explorer. HTML (HyperText Markup Language, язык гипертекстовой разметки) содержит специальные инструкции браузеру, с помощью которых создаются веб-страницы. То есть веб-страницы — это документы в формате HTML, содержащие текст и специальные теги (дескрипторы) HTML. По большому счету, теги HTML необходимы для форматирования текста (то есть придания ему нужного вида), который «понимает» браузер. Документы HTML хранятся в виде файлов с расширением htm или html. Теги HTML сообщают браузеру информацию о структуре и особенностях форматирования веб-страницы. Каждый тег содержит определенную инструкцию и заключается в угловые скобки (<>).

Приведем пример простейшей таблицы, записанной на языке HTML (листинг 7.3).

Листинг 7.3. Представление таблицы на языке HTML

```
<title>Пример таблицы</title>
<table border>
<caption>Таблица телефонов</caption>
<tr><td>Андрей – раб<td>274-88-17
<tr><td>Света-Х<td>+38(067)7030356
<tr><td>ЖЭК<td>22-345-72
<tr><td>Справка по тел<td>009
</table>
```

Если строго придерживаться правил языка HTML, то сначала следует написать теги <HTML>, <HEAD>, <TITLE> и т. д., однако современные браузеры понимают и такую разметку, которая приведена в листинге 7.3. В нашем примере даже не указан ни один закрывающий тег для <tr> (тег, задающий строку в таблице) и для <td> (тег, задающий ячейку в таблице). Вам будет полезно набрать приведенный пример в Блокноте, как-нибудь назвать этот файл с расширением htm и открыть его в каком-либо браузере Internet Explorer, Mozilla Firefox или любом другом обозревателе. Также поучительно будет открыть этот файл в редакторе HTML

Microsoft Office SharePoint Designer (или Microsoft Office FrontPage или других) и проследить, как меняется HTML-форматирование при тех или иных изменениях, сделанных в таблице.

Итак, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Наша программа имеет данные, уже знакомые читателю из примера 44. Эти данные находятся в двух массивах: Imena и Tel. На основании этих двух массивов программа формирует таблицу в формате HTML, то есть создает текстовый файл (в нашей программе он называется C:\Tabl_tel.htm), а затем открывает этот файл браузером Internet Explorer. Текст программы приведен в листинге 7.4.

Листинг 7.4. Вывод таблицы в Internet Explorer

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Вывод таблицы в Internet Explorer. Здесь реализован несколько
        // необычный подход к выводу таблицы для ее просмотра и печати на
        // принтере. Программа записывает таблицу в текстовый файл в формате
        // HTML. Теперь у пользователя появляется возможность прочитать эту
        // таблицу с помощью любого обозревателя веб-страниц
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Таблица в формате HTML";
                array<String^> ^ Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                    "Справка по тел", "Александр Степанович", "Мама - дом",
                    "Карапузова Таня", "Погода сегодня", "Театр Браво"};
                array<String^> ^ Tel = {"274-88-17", "+38(067)7030356",
                    "22-345-72", "009", "223-67-67 доп 32-67", "570-38-76",
                    "201-72-23-прямой моб", "001", "216-40-22"};
                String ^ text = "<title>Пример таблицы</title>" +
                                "<table border><caption>" +
                                "Таблица телефонов</caption>\r\n";
                for (int i = 0; i <= 8; i++)
                    text += String::Format("<tr><td>{0}<td>{1}", Imena[i],
                    Tel[i]) + "\r\n";
                text += "</table>";
                // Запись таблицы в текстовый файл C:\Tabl_tel.htm.
                // Создание экземпляра StreamWriter для записи в файл
                auto Писатель = gcnew IO::
                    StreamWriter("C:\\Tabl_tel.htm", false,
                        System::Text::Encoding::GetEncoding(1251));
                // - здесь заказ кодовой страницы Win1251 для русских букв
                Писатель->Write(text); Писатель->Close();
```

```
            try
            {
                Diagnostics::Process::Start("Iexplore",
                                                "C:\\Tabl_tel.htm");
                // Файл HTM можно открывать также с помощью MS_WORD:
                // Diagnostics::Process::Start("WinWord",
                //                                "C:\\Tabl_tel.htm");
            }
            catch (Exception ^ Ситуация)
            {
                // Отчет об ошибках
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    };
}
```

HTML-файл *формируется* с помощью строковой переменной text. Между тегами <title> и </title> указано название страницы. Это текст, который браузер покажет в строке заголовка окна.

Тег <table> указывает на начало таблицы, между тегами <caption> и </caption> расположено название таблицы. Далее в пошаговом цикле for формируется каждая строка таблицы. Используется уже хорошо известный читателю String::Format, то есть вместо фигурной скобки {0} подставляется элемент массива Imena[i], а вместо {1} — Tel[i].

Затем, создавая поток Писатель, сохраняем на диск текстовый файл C:\Tabl_tel.htm. Далее в блоке try...catch открываем этот файл с помощью браузера Internet Explorer.

Результат работы программы приведен на рис. 7.4.

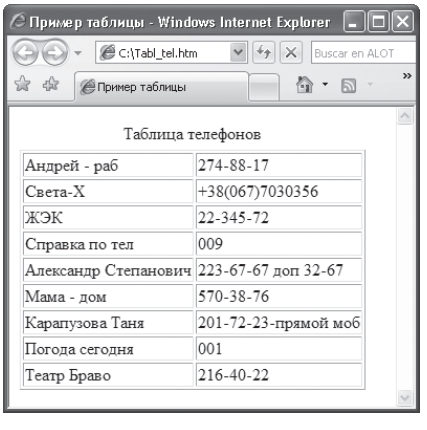

Таблица телефонов

Андрей - раб	274-88-17
Света-Х	+38(067)7030356
ЖЭК	22-345-72
Справка по тел	009
Александр Степанович	223-67-67 доп 32-67
Мама - дом	570-38-76
Карапузова Таня	201-72-23-прямой моб
Погода сегодня	001
Театр Браво	216-40-22

Рис. 7.4. Вывод таблицы в браузер

Очень технологично открыть созданный HTM-файл не веб-браузером, а текстовым редактором MS Word, то есть в программном коде написать:

```
Diagnostics::Process::Start("WinWord", "C:\\Tabl_tel.htm")
```

В этом случае пользователь вашей программы будет иметь возможность редактировать полученную таблицу. Убедиться в работоспособности программы можно, открыв решение Табл_HTM.sln в папке Табл_HTM.

Пример 55. Формирование таблицы с помощью элемента управления DataGridView

Создадим приложение, которое *заполняет два строковых массива и выводит эти массивы на экран в виде таблицы, используя элемент управления* DataGridView (просмотр *сетки данных*). Элемент управления DataGridView предназначен для просмотра таблиц с возможностью их редактирования.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++, при этом получаем стандартную форму. Перенесем в форму элемент управления DataGridView (Сетка данных) *из панели* Toolbox. В данной программе два уже знакомых читателю массива Imena[] и Tel[] выводятся на сетку данных DataGridView. Для максимального упрощения программы формируем таблицу при обработке события загрузки формы. Текст программы приведен в листинге 7.5.

Листинг 7.5. Формирование таблицы с помощью элемента управления DataGridView

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа заполняет два строковых массива и выводит эти массивы
        // на экран в виде таблицы, используя элемент управления DataGridView
        // (Сетка данных). Элемент управления DataGridView предназначен для
        // просмотра таблиц с возможностью их редактирования
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                Form1::Text = "Формирование таблицы";
                array<String^> ^ Imena  = {"Андрей - раб", "Света-Х", "ЖЭК",
                    "Справка по тел", "Ломачинская Светлана", "Мама - дом",
                    "Карапузова Таня", "Погода сегодня", "Театр Браво"};
                array<String^> ^  Tel = {"274-88-17", "+38(067)7030356",
                    "22-345-72", "009", "223-67-67 доп 32-67", "570-38-76",
                    "201-72-23-прямой моб", "001", "216-40-22"};
                // Создание объекта "таблица данных"
                DataTable ^ Таблица = gcnew DataTable();
```

```
            // Заполнение "шапки" таблицы
            Таблица->Columns->Add("Имена");
            Таблица->Columns->Add("Номера телефонов");
            // Заполнение клеток (ячеек) таблицы данных
            for (int i = 0; i <= 8; i++)
             Таблица->Rows->Add( Imena[i], Tel[i] );
            // Для сетки данных указываем источник данных
            dataGridView1->DataSource = Таблица;
        }
    };
}
```

Как видно, в программе *используется объект таблица данных* DataTable. С его помощью сначала заполняем «шапку» таблицы данных, используя метод Columns->Add, а затем — непосредственно ячейки таблицы, используя метод Rows->Add.

Чтобы передать построенную таблицу в элемент управления DataGridView, указываем в качестве источника данных DataSource объект Таблица класса DataTable.

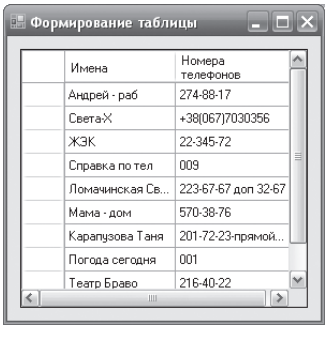

Рис. 7.5. Формирование таблицы в элементе DataGridView

На рис. 7.5 приведен результат работы программы. Заметим, что, щелкая на заголовках колонок, можно расположить записи в колонках в алфавитном порядке для удобного поиска необходимого телефона.

Убедиться в работоспособности программы можно, открыв решение ТаблGrid.sln в папке ТаблGrid.

Пример 56. Отображение данных в форме хэш-таблицы с помощью элемента DataGridView

Структура данных, называемая хэш-таблицей, представляет собой таблицу из двух столбцов, один столбец содержит ключи, а второй — значения. То есть каждая строка в этой таблице образует пару «ключ — значение». Имея ключ в хэш-таблице, можно быстро найти значение. Хэш-таблицу можно назвать таблицей соответствий. Простейшим примером хэш-таблицы является таблица телефонов, которая участвовала в примерах предыдущих разделов, однако там мы программировали ее как

два массива. Если эти два массива поместить в хэш-таблицу, то ключами в данном случае были бы имена, а значением — номер телефона. При этом программирование поиска значения по ключу сводится к тривиальной задаче, операция добавления и удаления пары также упрощается, поскольку хэш-таблица — это объект, который содержит соответствующие методы. В реальной жизни много разнообразных примеров представления данных в виде хэш-таблицы. Например, таблица, где расширения файлов (txt, jpg, mdb, xls) являются ключами, а соответствующими значениями — программы, которые открывают файлы с такими расширениями (Notepad.exe, Pbrush.exe, MSAccess.exe, Excel.exe). Типичнейшим примером являются разнообразные словари или база данных доменных имен, которая сопоставляет доменному имени IP-адрес. По принципу хэш-таблицы организованы объекты ViewState и Session технологии ASP.NET.

Поставим следующую задачу: сопоставим в хэш-таблице государства в качестве ключей, а их столицы — в качестве значений. Далее, используя элемент управления DataGridView, выведем эту хэш-таблицу в форму.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++, в поле Name укажем имя ХэшGridView. Перенесем в форму элемент управления DataGridView (Сетка данных) *из панели* Toolbox. Далее на вкладке Form1.h введем программный код, приведенный на листинге 7.6.

Листинг 7.6. Вывод хэш-таблицы в экранную форму

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // В данной программе используется структура данных, называемая
        // хэш-таблицей. С ее помощью программа ставит в соответствие
        // государствам их столицы. При этом в качестве ключей указываем
        // названия государств, а в качестве значений - их столицы. Далее,
        // используя элемент управления DataGridView, программа выводит
        // эту хэш-таблицу в форму
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 this->Text = "Пример хэш-таблицы";
                 // Создаем новую хэш-таблицу:
                 System::Collections::Hashtable Хэш = gcnew
                             System::Collections::Hashtable();
                 // Заполнение хэш-таблицы.
                 // Можно добавлять записи "ключ - значение" таким образом:
                 Хэш["Украина"] = "Киев";
                 // А можно добавлять так:
                 Хэш.Add("Россия", "Москва");
                 // Здесь государство - это ключ, а столица - это значение
```

```
            Хэш.Add("Белоруссия", "Минск");
            // Создаем обычную таблицу (не хэш):
            DataTable ^ Таблица = gcnew DataTable();
            // Задаем схему таблицы, заказывая две колонки:
            Таблица->Columns->Add("ГОСУДАРСТВА");
            Таблица->Columns->Add("СТОЛИЦЫ");
            // В цикле заполняем обычную таблицу парами
            // из хэш-таблицы по рядам:
            for each (System::Collections::DictionaryEntry ОднаПара in Хэш)
                // Здесь структура DictionaryEntry
                // определяет пару "ключ - значение"
                Таблица->Rows->Add(ОднаПара.Key, ОднаПара.Value);
            // Указываем источник данных для DataGridView:
            dataGridView1->DataSource = Таблица;
        }
    };
}
```

При обработке события загрузки формы создается объект класса Hashtable. Хэш-таблица заполняется тремя парами «код — значение», причем, как показано в программном коде, допустимы обе формы записи: через присваивание и посредством метода Add. Далее создается вспомогательный объект класса DataTable, который следует заполнить данными из хэш-таблицы. Хэш-таблица имеет структуру типа DictionaryEntry, которая позволяет перемещаться по рядам в цикле и таким образом получить все пары из хэш-таблицы. В этом цикле происходит заполнение объекта класса DataTable. Далее, так же как и в предыдущем примере, для dataGridView1 указываем в качестве источника данных заполненный объект DataTable. Пример работы данной программы показан на рис. 7.6.

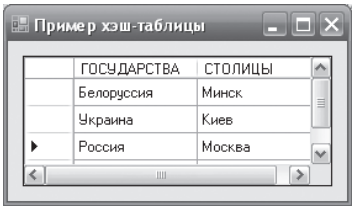

Рис. 7.6. Отображение в форме хэш-таблицы

В заключение отметим, что хэш-таблицу называют *ассоциативным массивом*, но в этом «массиве» роль индекса играет ключ. Для реализации хэш-таблицы можно было бы использовать обычный одномерный массив, в котором элементы с четным индексом являются ключами, а с нечетным — значениями. Однако для реализации трех основных операций с хэш-таблицей: добавления новой пары, операции поиска и операции удаления пары по ключу, потребовалось бы отлаживать довольно-таки много строчек программного кода.

Убедиться в работоспособности данной программы можно, открыв решение ХэшGridView.sln в папке ХэшGridView.

Пример 57. Табличный ввод данных. DataGridView. DataTable. DataSet. Инструмент для создания файла XML

Существует множество задач, предполагающих ввод данных в виде таблиц. Конечно, можно эту таблицу программировать как совокупность текстовых полей TextBox, но в большинстве случаев заранее неизвестно, сколько рядов данных будет вводить пользователь, необходимо предусмотреть скроллинг этой таблицы и т. д. То есть проблем в организации ввода табличных данных достаточно много.

Мы предлагаем для ввода табличных данных использовать элемент управления DataGridView (Сетка данных). Прежде всего, этот элемент управления предназначен для отображения данных, которые удобно представить в виде таблицы, чаще всего источником этих данных является база данных. Однако, кроме отображения, элемент управления DataGridView позволяет также редактировать табличные данные, DataGridView поддерживает выделение, изменение, удаление, разбиение на страницы и сортировку.

Программа, рассматриваемая в данном разделе, предлагает вам заполнить таблицу телефонов знакомых, сотрудников, родственников, любимых и т. д. После щелчка на кнопке Запись данная таблица записывается на диск в файл в формате XML. Для упрощения текста программы предусмотрена запись в один и тот же файл C:\tabl.xml. При последующих запусках данной программы таблица будет считываться из файла, и вы сможете продолжить редактирование таблицы. Поэтому эту программу можно громко назвать «табличным редактором». Щелкая на заголовках колонок, можно расположить записи в колонках в алфавитном порядке для удобного поиска необходимого телефона.

Для написания программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Далее требуется из панели управления Toolbox перенести мышью следующие элементы управления: сетку данных DataGridView и кнопку Button. Текст программы приведен в листинге 7.7.

Листинг 7.7. Заполнение телефонной книги

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа предлагает пользователю заполнить таблицу телефонов его
        // знакомых, сотрудников, родственников, любимых и т. д. После щелчка
        // на кнопке Запись данная таблица записывается на диск в файл в формате
        // XML. Для упрощения текста программы предусмотрена запись в один и тот
        // же файл C:\tabl.xml. При последующих запусках данной программы таблица
        // будет считываться из этого файла, и пользователь может продолжать
```

```
        // редактирование таблицы
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        DataTable ^ Таблица; // Объявление объекта "таблица данных"
        DataSet ^ НаборДанных; // Объявление объекта "набор данных"
    private: System::
                Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
            {
                this->Text = "Почти табличный редактор";
                button1->Text = "Запись";
                Таблица = gcnew DataTable();
                НаборДанных = gcnew DataSet();
                if (IO::File::Exists("C:\\tabl.xml") == false)
                {
                    // Если XML-файла НЕТ:
                    dataGridView1->DataSource = Таблица;
                    // Заполнение "шапки" таблицы
                    Таблица->Columns->Add("Имена");
                    Таблица->Columns->Add("Номера телефонов");
                    // Добавить объект Таблица в DataSet
                    НаборДанных->Tables->Add(Таблица);
                }
                else // Если XML-файл ЕСТЬ:
                {
                    НаборДанных->ReadXml("C:\\tabl.xml");
                    // Содержимое DataSet в виде строки XML для отладки:
                    String ^ СтрокаXML = НаборДанных->GetXml();
                    dataGridView1->DataMember = "Название таблицы";
                    dataGridView1->DataSource = НаборДанных;
                }
            }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
            {
                // Сохранить файл tabl.xml:
                Таблица->TableName = "Название таблицы";
                НаборДанных->WriteXml("C:\\tabl.xml");
            }
    };
}
```

Как видно из текста программы, потребовалось всего лишь несколько строк программного кода для создания такой многофункциональной программы. Это стало возможным благодаря использованию мощной современной технологии ADO.NET. В начале класса объявлены два объекта этой технологии: набор данных DataSet и таблица данных DataTable. Объект класса DataSet является основным компонентом архитектуры ADO.NET. DataSet представляет кэш данных, расположенный в оперативной памяти. DataSet состоит из коллекции объектов класса DataTable. То есть в один объект класса DataSet может входить несколько таблиц, а информацию о них мы можем записывать в файл на диск одним оператором WriteXml, соответственно

читать — ReadXML. Таким образом, в этой программе мы имеем дело преимущественно с тремя объектами: DataSet — кэш данных, DataTable — представляет одну таблицу с данными и DataGridView — элемент управления для отображения данных.

Сразу после инициализации компонентов формы мы обработали две ситуации. Если файла, в который мы сохраняем информацию о таблице, не существует Exists(«C:\\tabl.xml») == false, то назначаем в качестве источника данных DataSource для DataGrid объект класса DataTable и заполняем «шапку» таблицы, то есть указываем названия колонок: «Имена» и «Номера телефонов», а затем добавляем объект DataTable в объект DataSet. Теперь пользователь видит пустую таблицу с двумя колонками и может начать ее заполнение. Если файл существует (ветвь else), то данные в объект DataSet отправляем из XML-файла (ReadXML). Здесь уже в качестве источника данных для сетки данных DataGrid указываем объект DataSet.

При щелчке мышью на кнопке Запись (рис. 7.7) — событие button1.Click — происходит запись XML-файла на диск (WriteXml).

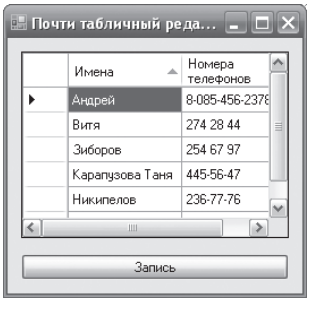

Рис. 7.7. Почти табличный редактор

Здесь используются, так называемые, XML-файлы. Формат этих файлов позволяет *легко и надежно передавать данные с помощью Интернета даже на компьютеры другой платформы* (например, Macintosh). Мы не ставили перед собой цель работать в Интернете, а всего лишь воспользовались данной технологией. Файл формата XML можно просмотреть Блокнотом или с помощью MS Word, поскольку это текстовый файл. Однако следует учесть, что этот файл записан в кодировке UTF-8, поэтому другими текстовыми редакторами, например edit.com или Rpad32.exe (русский Блокнот), его прочитать затруднительно. XML-документ открывается веб-браузером, XML-editor (входит в состав Visual Studio 2010), MS Office SharePoint Designer, MS Front Page и другими программами. При этом применение отступов и различных цветовых решений позволяет более наглядно продемонстрировать структуру данного файла. XML-файл можно открыть табличным редактором MS Excel, и при этом он может отобразиться в виде таблицы. На рис. 7.8 приведен образец представления XML-файла в MS Word.

При использовании нами XML-файлов для программирования простейшего табличного редактора совсем не обязательно вникать в его структуру, тем более при выборе имени файла для сохранения совершенно не обязательно устанавливать расширение файла xml, файл с любым расширением будет читаться методом ReadXml как XML-файл.

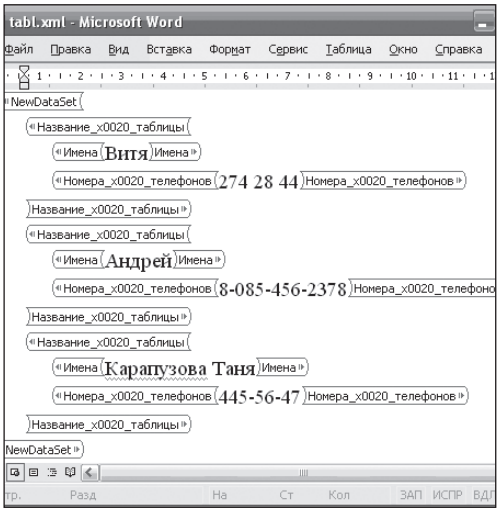

Рис. 7.8. Образец представления XML-файла в MS Word

Программист может получить доступ к полям таблицы. Например, доступ к левой верхней ячейке (полю) таблицы можно получить, используя свойство объекта класса DataTable: Таблица.Rows.Item(0).Item(0). Однако запись этого поля, например, в последовательный файл будет некорректной даже при использовании дополнительной переменной из-за того, что технология ADO.NET предусматривает кэширование данных. Таким образом, чтение и запись данных для подобных таблиц следует организовывать только через методы объекта DataSet.

Замечу, что данная программа может также являться инструментом для создания XML-файлов. Убедиться в работоспособности программы можно, открыв решение ТаблВвод.sln в папке ТаблВвод.

Пример 58. Решение системы линейных уравнений. Ввод коэффициентов через DataGridView

В инженерном деле, в экономических или научных расчетах часто встречается задача по решению системы линейных алгебраических уравнений (СЛАУ). Между тем, решение подобной задачи в последнее время нечасто встречается в книгах по языкам программирования. Программа, рассматриваемая в этом разделе, приглашает пользователя ввести в текстовое поле количество неизвестных (то есть задает размерность системы). Если пользователь благополучно справился с этим заданием, то текстовое поле становится недоступным и появляется элемент управления сетка данных DataGridView, куда пользователь может ввести коэффициенты линейных уравнений и свободные члены. При щелчке на кнопке Решить программа проверяет корректность введенных данных (не должно быть нечисловых симво-

лов, количество строк в матрице коэффициентов должно быть равно количеству неизвестных). Далее решается введенная система уравнений методом Гаусса и выводятся результаты вычислений с помощью MessageBox.

Итак, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Пользовательский интерфейс будет включать в себя (рис. 7.9) метку Label, текстовое поле TextBox, элемент управления DataGridView и кнопку Button. Перенесем названные элементы из панели элементов управления в проектируемую форму. В листинге 7.8 приведен программный код решения задачи.

Листинг 7.8. Решение системы линейных уравнений

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа для решения системы линейных уравнений. Ввод коэффициентов
        // предусмотрен через DataGridView
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Данные переменные объявляем вне всех процедур, чтобы
        // они были видны из любой из процедур:
        int n; // - размерность СЛАУ
        DataTable ^ Таблица;
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 this->Text = "Решение системы уравнений";
                 // Чтобы при старте программы фокус находился в текстовом поле:
                 textBox1->TabIndex = 0;
                 dataGridView1->Visible = false; // сетку данных пока не видно
                 label1->Text = "Введите количество неизвестных:";
                 button1->Text = "Ввести"; // - первоначальная надпись на кнопке
                 Таблица = gcnew DataTable();
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 // Матрица коэффициентов линейных уравнений:
                 array<double,2> ^ A = gcnew array<double,2>(n, n);
                 // Вектор свободных членов:
                 array<double> ^ L = gcnew array<double>(n);
                 int i, j;
                 // Признак ввода числовых данных:
                 bool Число_ли = false;
                 String ^ tmp;   // - временная рабочая переменная
                 if (button1->Text == "Ввести")
```

```
{
    for (; ; )
    {  // Бесконечный цикл, пока пользователь не введет
       // именно число:
        Число_ли = int::TryParse(textBox1->Text,
            Globalization::NumberStyles::Integer,
            Globalization::NumberFormatInfo::CurrentInfo, n);
        if (Число_ли == false) return;
        // Задаем другую надпись на кнопке:
        button1->Text = "Решить";
        // Теперь уже текстовое поле недоступно:
        textBox1->Enabled = false;
        // Теперь уже сетку данных видно:
        dataGridView1->Visible = true;
        dataGridView1->DataSource = Таблица;
        // Создаем "шапку" таблицы
        for (i = 1; i <= n; i++)
        {
            tmp = "X" + Convert::ToString(i);
            Таблица->Columns->Add(gcnew DataColumn(tmp));
        }
        // Колонка правой части системы:
        Таблица->Columns->Add(gcnew DataColumn("L"));
        return;
    }
}
else // button1->Text == "Решить")
{   // Нажали кнопку Решить
    // Таблица->Rows->Count - количество рядов
    if (Таблица->Rows->Count != n)
    {
        MessageBox::Show(
            "Количество строк не равно количеству колонок");
        return;
    }
    // Заполнение матрицы коэффициентов системы A[j, i]
    for (j = 0; j <= n - 1; j++)
    {
        for (i = 0; i <= n - 1; i++)
        {
            A[j, i] = ВернутьЧисло(j, i, Число_ли);
            if (Число_ли == false) return;
        }
        // Правая часть системы B(j, 0)
        L[j] = ВернутьЧисло(j, i, Число_ли);
        if (Число_ли == false) return;
    } // j
}
// Решение системы A*x = L методом Гаусса:
```

продолжение ⇘

Листинг 7.8 (продолжение)

```
            gauss(n, A, L);
            // L - вектор свободных членов системы, сюда
            // же возвращается решение x
            String ^ s = "Неизвестные равны:\n";
            for (j = 1; j <= n; j++)
            {
                tmp = L[j - 1].ToString();
                s = s + "X" + j.ToString() + " = " + tmp + ";\n";
            }
            MessageBox::Show(s);
        }
    private: void gauss(int n, array<double,2> ^ A, array<double> ^ LL)
        {
            // n  - размер матрицы
            // A  - матрица коэффициентов линейных уравнений
            // LL - правая часть, сюда возвращаются значения неизвестных
            int i, j, l = 0;
            Double c1, c2, c3;
            for (i = 0; i <= n - 1; i++) // цикл по элементам строки
            {
                c1 = 0;
                for (j = i; j <= n - 1; j++)
                {
                    c2 = A[j, i];
                    if (Math::Abs(c2) > Math::Abs(c1))
                    {
                        l = j; c1 = c2;
                    }
                }

                for (j = i; j <= n - 1; j++)
                {
                    c3 = A[l, j] / c1;
                    A[l, j] = A[i, j]; A[i, j] = c3;
                } // j

                c3 = LL[l] / c1; LL[l] = LL[i]; LL[i] = c3;

                for (j = 0; j <= n - 1; j++)
                {
                    if (j == i) continue;
                    for (l = i + 1; l <= n - 1; l++)
                    {
                        A[j, l] = A[j, l] - A[i, l] * A[j, i];
                    } // l
                    LL[j] = LL[j] - LL[i] * A[j, i];
                } // j
            } // i
        }
```

```
    private: double ВернутьЧисло(int j, int i, bool & Число_ли)
             {  // j - номер строки, i - номер столбца
                // Передаем аргумент Число_ли по ссылке (bool & Число_ли)
                 double rab; // - рабочая переменная
                 String ^ tmp = Таблица->Rows[j][i]->ToString();
                 Число_ли = Double::TryParse(tmp,
                     System::Globalization::NumberStyles::Number,
                     System::Globalization::NumberFormatInfo::CurrentInfo, rab);
                 if (Число_ли == false)
                 {
                     tmp = String::Format("Номер строки {0}, номер столбца " +
                         "{1},\n в данном поле - не число", j + 1, i + 1);
                     MessageBox::Show(tmp);
                 }
                 return rab;
             }
    };
}
```

Как видно их программного кода, при начальной загрузке программы пользователь не видит (Visible = false) сетку данных, а первоначальная надпись на кнопке — «Ввести». При щелчке на кнопке, если пользователь корректно ввел количество неизвестных, меняем надпись на кнопке (она теперь будет — «Решить»), по количеству неизвестных подготавливаем «шапку» таблицы и размерность сетки данных, куда пользователь будет вводить коэффициенты линейных уравнений и свободные члены.

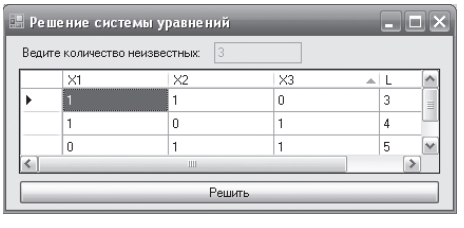

Рис. 7.9. Фрагмент табличного ввода данных

После ввода коэффициентов и щелчка на кнопке Решить происходит проверка количества введенных рядов коэффициентов и проверка на нечисловые символы. После этого вызывается *процедура решения СЛАУ методом Гаусса* gauss, то есть методом *последовательного исключения* неизвестных. В основе процедуры gauss — цикл for по элементам строки матрицы системы. В этот внешний цикл вложены три внутренних цикла по строкам матрицы. После вызова процедуры gauss формируется строковая переменная s для визуализации значений неизвестных. Переменная s выводится посредством диалогового окна MessageBox (рис. 7.10).

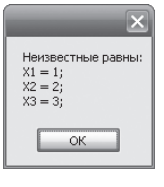

Рис. 7.10. Вывод неизвестных значений в диалоговое окно

Данная программа не предусматривает проверку на вырожденность СЛАУ, однако в этом случае в качестве значений неизвест-

ных пользователь получает либо «бесконечность», либо константу NaN, значение которой является результатом деления на ноль. Программа не предусматривает ограничение сверху на размерность решаемой системы и должна работать при любой разумной размерности. Работа программы должна быть ограничена лишь размером оперативной памяти. Однако автор не тестировал эту программу на решение систем большой размерности. Если количество неизвестных равно одному, то программа также нормально функционирует.

Убедиться в работоспособности программы можно, открыв решение ГауссGrid.sln в папке ГауссGrid.

Пример 59. Организация связанных таблиц

Представляем вашему вниманию пример программирования двух связанных таблиц. Одна таблица содержит сведения о клиентах-заказчиках. В нашем случае это названия организаций и контактная информация (рис. 7.11), хотя в реальной жизни в эту таблицу включают гораздо больше колонок с данными. Вторая таблица содержит в себе данные о заказах, она может содержать в себе номер (идентификацию) заказа, его объем, организацию-заказчика, сроки выполнения, адрес доставки и прочие необходимые сведения. В нашем случае с целью компактного изложения эта таблица будет содержать только номер заказа, его объем и название организации-заказчика (рис. 7.12).

Обе таблицы связываем таким образом, чтобы из одной родительской таблицы можно было получать уточняющие данные из другой дочерней таблицы. Связь осуществляем с помощью одинаковой колонки (столбцу), содержащей название организации. При этом родительская таблица — «Клиенты» (см. рис. 7.11), в заголовке каждой строки будет отображать знак +, раскрывая который щелчком мыши можно получать данные из дочерней таблицы «Заказы» (рис. 7.13).

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Данное решение назовем СвязанныеТаблицы.

Обычно для отображения таблиц пользуются элементом управления DataGridView. Это сравнительно новый элемент управления, он появился в Visual Studio примерно с 2007 года. До 2007 года использовали элемент управления DataGrid. Элемент управления DataGridView поддерживает ряд простых и сложных функций, отсутствовавших в элементе управления DataGrid. Кроме того, архитектура элемента управления DataGridView упрощает его расширение и настройку по сравнению с DataGrid. Почти для всех целей следует использовать элемент управления DataGridView. Единственная функция, доступная в элементе управления DataGrid и недоступная в DataGridView, — иерархическое отображение данных из двух связанных таблиц в едином элементе управления. Для отображения данных из двух связанных таблиц требуется два элемента управления DataGridView. Поэтому для отображения двух связанных таблиц *более технологичным вариантом будет использовать* именно элемент DataGrid.

Итак, в панели ToolBox в узле Данные нам необходим элемент DataGrid. Однако, поскольку он устарел, элемента DataGrid там нет, и нам нужно его добавить в панель элементов. Для этого щелкаем правой кнопкой мыши в пределах панели элементов управления и в появившемся контекстном меню указываем пункт Выбрать элементы. Далее на вкладке Компоненты .NET Framework устанавливаем флажок возле элемента DataGrid пространства имен System.Windows.Forms. После щелчка на кнопке OK указанный элемент появится в панели ToolBox. Теперь мы можем перенести элемент DataGrid в проектируемую форму. Кроме этого, нам понадобится элемент Button (как же без него), его также переносим в нашу форму.

В листинге 7.9 приведен программный код решения задачи.

Листинг 7.9. Программирование двух связанных таблиц

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа создает таблицу с данными о клиентах (названия организаций,
        // контакты) и таблицу с данными о заказах (номер заказа, его объем,
        // организация-заказчик). Между таблицами устанавливается связь
        // посредством одинаковых столбцов (названий организаций). При этом
        // таблица с данными о клиентах будет родительской, а таблица с данными
        // о заказах - дочерней. Каждая строка родительской таблицы отображается
        // со знаком "плюс". При щелчке на знаке "плюс" открывается узел,
        // содержащий ссылку на дочернюю таблицу
        // ~ ~ ~ ~ ~ ~ ~ ~
        Boolean ПоказатьКлиентов;
        System::Data::DataSet ^ НаборДанных;
    private: System::Void Form1_Load(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 button1->Text = "Переключатель таблиц"; button1->TabIndex=0;
                 Form1::Text = "Связанные таблицы"; ПоказатьКлиентов = true;
                 // Р О Д И Т Е Л Ь С К А Я  Т А Б Л И Ц А  "К Л И Е Н Т Ы":
                 System::Data::DataTable ^ Таблица = gcnew DataTable("Клиенты");
                 // Создаем первый столбец в таблице:
                 DataColumn ^ Столбец = gcnew DataColumn("Название организации");
                 Столбец->ReadOnly = true; // - его нельзя модифицировать
                 // Значения в каждой строке столбца должны быть уникальными:
                 Столбец->Unique = true;
                 // Добавляем этот столбец в таблицу:
                 Таблица->Columns->Add(Столбец);
                 // Добавляем второй и третий столбец в таблицу:
                 Таблица->Columns->Add("Контактное лицо");
                 Таблица->Columns->Add("Телефон");
                 // Создаем набор данных:
```

продолжение ⇩

Листинг 7.9 (продолжение)

```
            НаборДанных = gcnew DataSet();
            // Добавляем таблицу "Клиенты" в набор данных:
            НаборДанных->Tables->Add(Таблица);
            // Заполняем строки (ряды) в таблице:
            Таблица->Rows->Add(
                "НИИАСС", "Погребицкий Олег", "095 345 22 37");
            Таблица->Rows->Add(
                "КНУБА", "Борис Григорьевич", "050 456 21 03");
            Таблица->Rows->Add(
                "МИИГАИК", "Стороженко Светлана", "067 456 56 72");
            // Д О Ч Е Р Н Я Я  Т А Б Л И Ц А  "З А К А З Ы":
            Таблица = gcnew DataTable("Заказы");
            // Создаем первый столбец в таблице "Заказы":
            DataColumn ^ Столбец1 = gcnew DataColumn("Номер заказа");
            Столбец1->DataType = System::Type::GetType("System.Int32");
            Столбец1->ReadOnly = true; Столбец1->Unique = true;
            Таблица->Columns->Add(Столбец1);
            // Добавляем второй и третий столбец в таблицу:
            Таблица->Columns->Add("Объем заказа");
            Таблица->Columns->Add("Организация-заказчик");
            НаборДанных->Tables->Add(Таблица);
            // Заполняем строки (ряды) в таблице:
            Таблица->Rows->Add(1, "230000", "НИИАСС");
            Таблица->Rows->Add(2, "178900", "КНУБА");
            Таблица->Rows->Add(3, "300000", "НИИАСС");
            Таблица->Rows->Add(4, "345000", "МИИГАИК");
            Таблица->Rows->Add(5, "308000", "КНУБА");
            // С О З Д А Н И Е  С В Я З Е Й  М Е Ж Д У  Т А Б Л И Ц А М И:
            // Связывем одинаковые столбцы в двух таблицах:
            DataColumn ^ Родитель = НаборДанных->
                Tables["Клиенты"]->Columns["Название организации"];
            DataColumn ^ Дочь = НаборДанных->
                Tables["Заказы"]->Columns["Организация-заказчик"];
            DataRelation ^ Связь1 = gcnew DataRelation(
                "Ссылка на заказы клиента", Родитель, Дочь);
            // В родительской таблице значения в связываемом столбце
            // должны быть уникальными, а в дочерней - нет.
            НаборДанных->Tables["Заказы"]->ParentRelations->Add(Связь1);
            // И С Т О Ч Н И К  Д А Н Н Ы Х  Д Л Я  D A T A G R I D:
            dataGrid1->SetDataBinding(НаборДанных, "Клиенты");
            dataGrid1->CaptionText = "Родительская таблица \"Клиенты\"";
            dataGrid1->CaptionFont = gcnew
                System::Drawing::Font("Consolas", 11);
        }
    private: System::Void button1_Click(System::Object^  sender,
            System::EventArgs^  e)
```

```
            {
                // Показываем либо "таблицу Клиенты", либо таблицу "Заказы":
                ПоказатьКлиентов = !ПоказатьКлиентов;
                if (ПоказатьКлиентов == true)
                {
                    dataGrid1->SetDataBinding(НаборДанных, "Клиенты");
                    dataGrid1->CaptionText = "Родительская таблица \"Клиенты\"";
                }
                else
                {
                    dataGrid1->SetDataBinding(НаборДанных, "Заказы");
                    dataGrid1->CaptionText = "Дочерняя таблица \"Заказы\"";
                }
            }
    };
}
```

При обработке события загрузки формы создаем родительскую таблицу «Клиенты», а также дочернюю таблицу «Заказы». В данной главе мы уже не раз создавали таблицу класса DataTable, поэтому не будем повторяться в комментариях. Укажем только, что создавая первый столбец «Название организации», мы установили уникальность каждого значения в этом столбце, таково требование к родительской таблице. В родительской таблице значения в связываемом столбце не должны повторяться. В то время как для связываемого столбца дочерней таблицы такого ограничения нет.

Создавая связь между таблицами (объект класса DataRelation), указываем одинаковые столбцы в двух таблицах. В набор данных класса DataSet мы добавили две таблицы класса DataTable и объект связи между таблицами класса DataRelation. При подключении набора DataSet к какой-либо базе данных (SQL Server, MS Access, Oracle) можно добавлять записи (строки) в обе таблицы, их модифицировать (изменять), организовывать различные SQL-запросы к данным и более эффективно их использовать.

Обрабатывая событие «щелчок на командной кнопке Button», мы организовали переключение между родительской и дочерней таблицами. Фрагменты работы программы показаны на рис. 7.11—7.13.

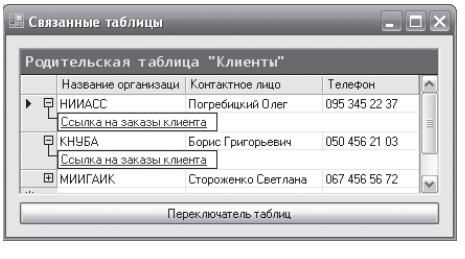

Рис. 7.11. Родительская таблица «Клиенты» содержит ссылки на строки дочерней таблицы

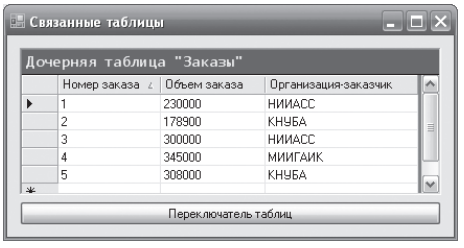

Рис. 7.12. Содержание дочерней таблицы «Заказы»

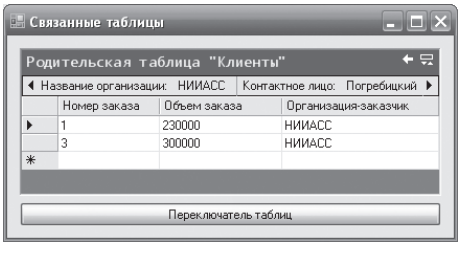

Рис. 7.13. Результат ссылки на заказы организации НИИАСС

Убедиться в работоспособности программы можно, открыв решение СвязанныеТаблицы.sln в папке СвязанныеТаблицы.

Пример 60. Построение графика по табличным данным с использованием элемента Chart

Уважаемый читатель, давайте посмотрим на панель элементов управления Toolbox. Среди множества элементов найдем элемент Chart. Он предназначен для вывода в экранную форму графика (диаграммы). Очень удобно строить этот график по табличным данным, представленным в виде объекта класса DataTable.

В данном примере решим следующую задачу. Даны сведения об объемах продаж за пять месяцев. Требуется наглядно визуализировать эти данные на графике в виде гистограммы.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Данное решение назовем ГрафикChart. Из панели ToolBox перенесем в проектируемую экранную форму следующие элементы: диаграмму Chart и сетку данных DataGridView. Для наглядности мы хотим, чтобы при щелчке мышью на графике он менял свой внешний вид, а именно, чтобы столбики гистограммы отображались в виде цилиндров. Для этого нам понадобится пустой обработчик события — «щелчок на диаграмме Chart». Для получения этого пустого обработчика, используя контекстное меню элемента Chart в конструкторе проектируемой формы, перейдем к свойствам этого элемента. Здесь в окне Properties щелкнем на значке

молнии (События) и в списке возможных событий выберем событие Click. Дважды щелкнув на отображении этого события, мы попадаем на вкладку программного кода с готовым пустым обработчиком этого события. В листинге 7.10 приведен программный код решения задачи.

Листинг 7.10. Вывод в форму графика посредством элемента управления Chart

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа, используя элементы управления Chart и DataGridView, выводит
        // график (диаграмму) зависимости объемов продаж от времени по месяцам.
        // При этом в качестве источника данных указываем объект класса DataTable
        // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Отображать ли столбики гистограммы в виде цилиндров:
        Boolean Цилиндр;
    private: System::Void Form1_Load(System::Object^ sender,
                                                    System::EventArgs^  e)
             {
                 Цилиндр = false;
                 this->Text = "Щелкните на графике";
                 DataTable ^ Таблица = gcnew DataTable();
                 // В этой таблице заказываем две колонки "Месяц" и "Объем
                 продаж":
                 Таблица->Columns->Add("Месяц", String::typeid);
                 // В C#: Таблица.Columns.Add("Месяц", typeof(String));
                 // Значения во второй колонке назначаем типа long:
                 Таблица->Columns->Add("Объем продаж", long::typeid);
                 // В C#: Таблица.Columns.Add("Объем продаж", typeof(long));
                 // Заполняем первую строку (ряд) в таблице:
                 DataRow ^ Ряд = Таблица->NewRow();
                 Ряд["Месяц"] = "Май"; Ряд["Объем продаж"] = 15;
                 Таблица->Rows->Add(Ряд);
                 // Добавляем вторую строку в таблице:
                 Ряд = Таблица->NewRow();
                 Ряд["Месяц"] = "Июнь"; Ряд["Объем продаж"] = 35;
                 Таблица->Rows->Add(Ряд);
                 // Добавляем третью строку:
                 Ряд = Таблица->NewRow();
                 Ряд["Месяц"] = "Июль"; Ряд["Объем продаж"] = 65;
                 Таблица->Rows->Add(Ряд);
                 // Добавляем четвертую строку:
                 Ряд = Таблица->NewRow();
                 Ряд["Месяц"] = "Авг"; Ряд["Объем продаж"] = 85;
                 Таблица->Rows->Add(Ряд);
                 // Добавляем пятую строку:
```

продолжение ⇩

Листинг 7.10 (продолжение)

```
                Ряд = Таблица->NewRow();
                Ряд["Месяц"] = "Сент"; Ряд["Объем продаж"] = 71;
                Таблица->Rows->Add(Ряд);
                // Составленную таблицу указываем в качестве источника данных:
                chart1->DataSource = Таблица;
                // На одном графике можно изобразить несколько зависимостей.
                // Например, первая зависимость - объемы продаж по указанным
                // месяцам в 2009 году, и вторая зависимость - продажи по
                // тем же месяцам в 2010 году.
                // В данном графике мы покажем только одну зависимость, данные
                // для отображения этой зависимости назовем "Series1"
                // По горизонтальной оси откладываем названия месяцев:
                chart1->Series["Series1"]->XValueMember = "Месяц";
                // А по вертикальной оси откладываем объемы продаж:
                chart1->Series["Series1"]->YValueMembers = "Объем продаж";
                // Название графика (диаграммы):
                chart1->Titles->Add("Объемы продаж по месяцам");
                // Задаем тип диаграммы - столбиковая гистограмма:
                chart1->Series["Series1"]->ChartType = System::Windows::Forms::
                    DataVisualization::Charting::SeriesChartType::Column;
                // Тип диаграммы может быть другим, например: Pie, Line и др.
                chart1->Series["Series1"]->Color = Color::Aqua;
                // Легенду на графике не отображаем:
                chart1->Series["Series1"]->IsVisibleInLegend = false;
                // Привязка графика к источнику данных:
                chart1->DataBind();
                // Для сетки данных указываем источник данных
                dataGridView1->DataSource = Таблица;
            }
    private: System::Void chart1_Click(System::Object^  sender,
                                                    System::EventArgs^  e)
            {
                // Обработка события "щелчок на графике"
                Цилиндр = !Цилиндр;
                // Изображение столбиков гистограммы в виде цилиндра:
                if (Цилиндр == true)
                    chart1->Series["Series1"]["DrawingStyle"] = "Cylinder";
                else
                    chart1->Series["Series1"]["DrawingStyle"] = "Default";
            }
    };
}
```

В программном коде при обработке события загрузки формы объявляем объект Таблица класса DataTable. Этот объект представляет одну таблицу данных в оперативной памяти. Чтобы визуализировать эту таблицу на экране, используют элемент — сетка данных DataGridView. Объект класса DataTable используют в качестве исходных данных и для сетки данных DataGridView, и для диаграммы Chart.

В таблице DataTable определяем ее схему, заказывая две колонки «Месяц» и «Объем продаж». А далее заполняем таблицу по ее строкам (рядам), используя метод Add. Заполненную пятью строками таблицу указываем в качестве источника данных для элементов Chart и DataGridView. Далее оформляем внешний вид диаграммы, что подробно представлено в комментариях к программному коду.

Фрагмент работы программы показан на рис. 7.14.

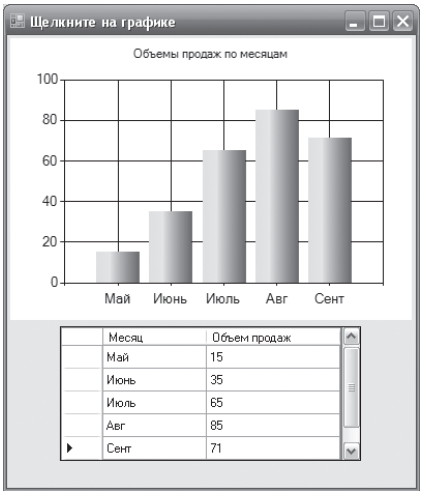

Рис. 7.14. Вывод диаграммы и соответствующей таблицы в экранную форму

Убедиться в работоспособности программы можно, открыв решение ГрафикChart.sln в папке ГрафикChart.

8

Элемент управления WebBrowser

Пример 61. Отображение HTML-таблиц в элементе WebBrowser

Мы знаем, что существуют настольные приложения (или Windows-приложения), которые пользователи запускают непосредственно (например, exe-файлы), и существуют веб-приложения, выполняющиеся внутри браузера. Однако в панели элементов Toolbox имеется элемент управления WebBrowser, который позволяет просматривать веб-приложения внутри экранной формы Windows-приложения.

В данном примере воспользуемся элементом управления WebBrowser для отображения таблицы, записанной на языке HTML с помощью элементарных тегов: тега <tr>, задающего строку в таблице, и тега <td>, задающего ячейку в таблице. Понятно, что сформировав такой HTML-файл, содержащий таблицу, подлежащую выводу на экран, можно вывести этот файл в поле элемента управления WebBrowser с помощью метода Navigate объекта.

Рассмотрим, как, задав на языке HTML какую-нибудь таблицу в строковой переменной, вывести эту таблицу в поле элемента управления WebBrowser, при этом не записывая на жесткий диск компьютера HTML-файл. Для этой цели воспользуемся простейшей таблицей из примера о формировании таблицы.

Приступим к программированию поставленной задачи. Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. В панели элементов Toolbox находим элемент WebBrowser, который перетаскиваем в форму. Текст программы приведен в листинге 8.1.

Листинг 8.1. Отображение таблиц с помощью элемента управления WebBrowser

```
        // ..............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
    }
```

```
#pragma endregion
        // В программе для отображения таблицы используется элемент управления
        // WebBrowser. Таблица записана на языке HTML с помощью элементарных
        // тегов <tr> (строка в таблице) и <td> (ячейка в таблице)
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                // Никакие края (из четырех) не привязаны к экранной форме:
                webBrowser1->Dock = DockStyle::None;
                // или webBrowser1.Navigate("c:\\table.htm");
                String ^ СтрокаHTML = "Какой-либо текст до таблицы" +
                    "<table border> " +
                    "<caption>Таблица телефонов</caption> " +
                    "<tr><td>Андрей – раб<td>274-88-17 " +
                    "<tr><td>Света-Х<td>+38(067)7030356 " +
                    "<tr><td>ЖЭК<td>22-345-72 " +
                    "<tr><td>Справка по тел<td>009 " +
                    "</table> " +
                    "Какой-либо текст после таблицы";
                webBrowser1->Navigate("about:" + СтрокаHTML);
            }
    };
}
```

Как видно из текста программы, здесь реализована обработка события загрузки формы. В процедуре этой обработки в строковой переменной СтрокаHTML задаем HTML-представление простейшей таблицы, состоящей из двух столбцов и четырех строк. Это строковую переменную подаем на вход метода Navigate объекта WebBrowser1, сцепляя ее с ключевым словом about:. В комментарии показано, как можно подать на вход метода Navigate таблицу, если она уже записана в файл table.htm.

В результате работы этой программы получаем отображение таблицы в поле элемента управления WebBrowser, как показано на рис. 8.1.

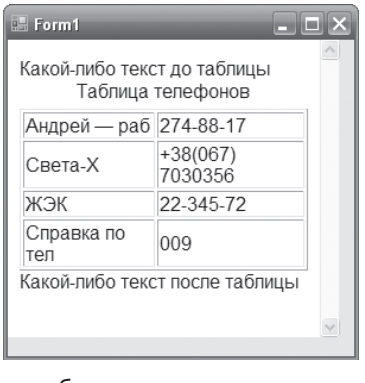

Рис. 8.1. Отображение таблицы в поле элемента управления WebBrowser

Теперь легко можно распечатать данную таблицу на принтере, для этого необходимо попасть в контекстное меню, щелкнув правой кнопкой мыши в пределах таблицы.

Убедиться в работоспособности программы можно, открыв решение ТаблWebHTM.sln в папке ТаблWebHTM.

Пример 62. Отображение Flash-файлов

Ранее мы уже обсуждали элемент управления PictureBox, способный отображать растровые файлы различных форматов. Однако графическое поле PictureBox не умеет отображать Flash-файлы формата SWF. Этот формат очень распространен в Интернете благодаря удачному сочетанию в нем векторной графики, растровой графики, воспроизведения звука и компактности кода, что позволяет быстро загружать SWF-файлы в браузер.

В данном разделе мы создадим программу, которая будет отображать в поле элемента управления WebBrowser файл SWF. Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Перетащим с панели Toolbox элемент управления WebBrowser в форму. Теперь, дважды щелкнув в пределах проектируемой формы, мы попадаем на вкладку программного кода, при этом система автоматически сгенерировала пустой обработчик загрузки формы. Далее мы введем текст из листинга 8.2.

Листинг 8.2. Отображение Flash-файлов

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа использует элемент управления WebBrowser для отображения
        // Flash-файлов
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                // Никакие края (из четырех) не привязаны к экранной форме:
                webBrowser1->Dock = DockStyle::None;
                // webBrowser1.Navigate("www.mail.ru");
                String ^ ИмяФайла = IO::Directory::
                    GetCurrentDirectory() + "\\Shar.swf";
                // Если такого файла нет
                if (IO::File::Exists(ИмяФайла) == false)
                {
                    MessageBox::Show(
                        "Файл " + ИмяФайла + " не найден", "Ошибка");
                    return;
                }
```

```
            webBrowser1->Navigate(ИмяФайла);
        }
    };
}
```

Здесь, в тексте программы вызываем метод Navigate объекта webBrowser1. На вход метода подаем полный путь к Flash-файлу Shar.swf. Полный путь к текущей папке получаем с помощью функции GetCurrentDirectory(). В комментарии показано, как можно вывести веб-страницу из Интернета в поле элемента управления WebBrowser.

В результате работы этой программы получаем отображение файла Shar.swf (рис. 8.2).

Рис. 8.2. Отображение Flash-файла в поле элемента управления WebBrowser

К сожалению, на рисунке не видно, что шар вращается. Убедиться в работоспособности программы можно, открыв решение FlashWeb.sln в папке FlashWeb.

Пример 63. Отображение веб-страницы и ее HTML-кода

Напишем на C++ программу, способную в одном поле отображать веб-страницу, а в другом поле, ниже, показывать HTML-код загружаемой в браузер страницы. Ядром этой программы будет все тот же элемент управления WebBrowser. Таким образом, экранная форма будет содержать на себе элемент управления WebBrowser, два текстовых поля и кнопку (рис. 8.3).

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Используя панель элементов, добавим элемент управления WebBrowser в экранную форму, также добавим два текстовых поля и кнопку. Первое текстовое поле textBox1 предназначено для ввода URL-адреса желаемой веб-страницы (например, www.latino.ho.ua), а второе поле textBox2 растянем, поскольку сюда будет выводиться HTML-код загружаемой в браузер страницы. Для того чтобы иметь возможность растянуть второе поле по вертикали, укажем явно в свойствах этого поля (Properties) Multiline = True. Текст программы приведен в листинге 8.3.

Листинг 8.3. Отображение веб-страницы и ее HTML-кода

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Эта программа использует элемент управления WebBrowser
        // для отображения веб-страницы и ее HTML-кода
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                this->Text = "Веб-страница и ее HTML-код";
                textBox1->Text = String::Empty;
                textBox2->Text = String::Empty;
                textBox2->Multiline = true;
                textBox2->ScrollBars = ScrollBars::Vertical;
                button1->Text = "ПУСК";
                webBrowser1->Dock = DockStyle::None;
            }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
            {
                // Обработка события "щелчок на кнопке ПУСК":
                webBrowser1->Navigate(textBox1->Text);
                // webBrowser1->Navigate("www.latino.ho.ua")
                // webBrowser1->GoBack()     // Назад
                // webBrowser1->GoForward()  // Вперед
                // webBrowser1->GoHome()     // На домашнюю страницу
            }
    private: System::Void webBrowser1_DocumentCompleted(
                System::Object^  sender, System::Windows::
                Forms::WebBrowserDocumentCompletedEventArgs^  e)
            {
                // Обработка события "Веб-документ полностью загружен"
                // Получаем HTML-код из элемента WebBrowser:
                textBox2->Text = webBrowser1->Document->Body->InnerHtml;
            }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы мы задали некоторые уже хорошо известные нам свойства. Свойство ScrollBars, определяющее скроллинг текстового поля, приведено в состояние Vertical, означающее разрешение вертикальной прокрутки.

При обработке события «щелчок на кнопке Пуск» метод Navigate объекта webBrowser1 вызывает в поле элемента управления обозревателя веб-страницу с адресом, указанным пользователем данной программы в текстовом поле textBox1.

В комментарии приведены варианты дальнейшего совершенствования данной программы. Так, метод GoBack возвращает пользователя на предыдущую веб-страницу, метод GoForward — на последующую, а метод GoHome загружает в поле браузера домашнюю веб-страницу. Все эти методы можно было бы развести по разным командным кнопкам, но я не стал этого делать, чтобы не загромождать программу, а наоборот подчеркнуть главное.

Событие DocumentCompleted подразумевает, что документ обозревателя «укомплектован», то есть загрузка страницы в браузер завершена. При обработке этого события функция Body->innerHTML возвращает в свойство Text текстового поля textBox2 HTML-код, соответствующий загруженной в браузер веб-страницы. На рис. 8.3 показан пример работы данной программы.

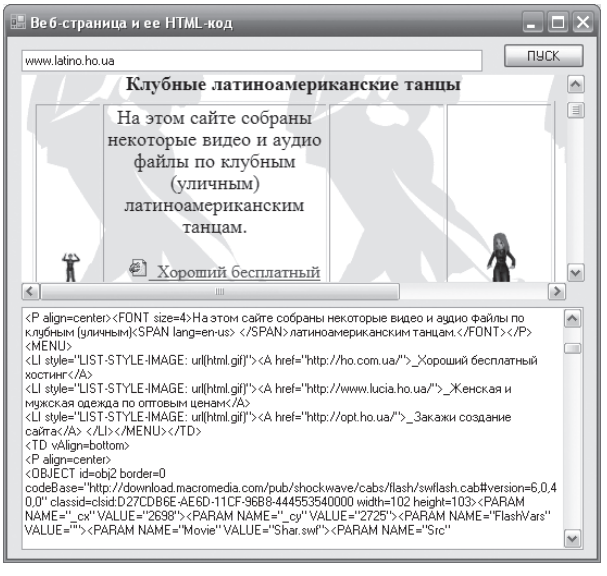

Рис. 8.3. Загрузка веб-страницы и ее HTML-кода

Убедиться в работоспособности программы можно, открыв решение Split.sln в папке Split.

Пример 64. Программное заполнение веб-формы

Наша данная задача состоит в том, чтобы, воспользовавшись элементом управления WebBrowser, отобразить в этом элементе какую-либо веб-страницу, содержащую веб-форму. Затем «программно» заполнить поля формы (например, логин и пароль на почтовом сервере, или строку поиска в поисковой системе, или персональные данные для регистрации в нужных нам ресурсах при оформлении подписки на рассылку и пр.). А при завершении заполнения «программно» щелкнуть на кнопке Submit (она может называться по-разному) для отправки заполненной веб-формы

для обработки на сервер. В итоге в элементе управления WebBrowser будет отображен результат, полученный с сервера после обработки. Подобным средством можно воспользоваться для подбора паролей, однако мы не хотели бы из моральных соображений популяризировать эту возможность. В данном примере мы будем заполнять строку поиска в наиболее используемых поисковых системах, таких как Yandex, Rambler, Google и др.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели элементов Toolbox в экранную форму перенесем элемент управления WebBrowser. Чтобы получить удобный доступ к конкретным дескрипторам (тегам) HTML, составляющим веб-страницу, к текущему проекту добавим ссылку на так называемую «неуправляемую» библиотеку Microsoft.mshtml.dll. Не стоит пугаться слова «неуправляемая», так компания Microsoft называет библиотеки, содержащие «неуправляемый» код, который выполняется не общеязыковой средой, а непосредственно операционной системой. Как видите, имя файла библиотеки содержит не совсем привычные две точки. Этот файл, скорее всего, расположен на вашем компьютере в папке: C:\Program Files\Microsoft Visual Studio 10.0\Visual Studio Tools for Office\PIA\Common.

Для добавления ссылки на эту библиотеку в пункте меню Project выберем команду Add Reference, а на вкладке .NET дважды щелкнем на ссылке Microsoft.mshtml. Теперь в окне Properties в области References (Ссылки) увидим добавленную в наш проект выбранную ссылку.

Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 8.4.

Листинг 8.4. Программное заполнение формы некоторых поисковых систем

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа загружает в элемент WebBrowser начальную страницу поисковой
        // системы http://yahoo.com. Далее, используя указатель на неуправляемый
        // интерфейс DomDocument (свойство объекта класса WebBrowser),
        // приводим его к указателю IHTMLDocument2. В этом случае мы получаем
        // доступ к формам и полям веб-страницы по их именам. Заполняем поле
        // поиска ключевыми словами для нахождения соответствующих веб-страниц,
        // а затем для отправки заполненной формы на сервер "программно" нажимаем
        // кнопку Submit. В итоге получим в элементе WebBrowser результат работы
        // поисковой системы, а именно множество ссылок на страницы, содержащие
        // указанные ключевые слова
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Программное заполнение формы";
```

```
// *** Для сайта "http://google.com":
//String ^ АдресСайта = "http://google.com"
//String ^ ИмяФормы = "f";
//String ^ ИмяПоляФормы = "q";
// Для сайта "http://meta.ua":
//String ^ АдресСайта = "http://meta.ua";
//String ^ ИмяФормы = "sForm";
//String ^ ИмяПоляФормы = "q";
// *** Для сайта "http://yandex.ru":
//String ^ АдресСайта = "http://yandex.ru";
//String ^ ИмяФормы = "form";
//String ^ ИмяПоляФормы = "text";
// *** Для сайта "http://rambler.ru":
//String ^ АдресСайта = "http://rambler.ru";
//String ^ ИмяФормы = "rSearch";
//String ^ ИмяПоляФормы = "query";
// *** Для сайта "http://aport.ru":
//String ^ АдресСайта = "http://aport.ru";
//String ^ ИмяФормы = "aport_search";
//String ^ ИмяПоляФормы = "r";
// *** Для сайта "http://bing.com":
//String ^ АдресСайта = "http://bing.com";
//String ^ ИмяФормы = "sb_form";
// - в HTML-коде нет name формы, но есть id = "sb_form"
//String ^ ИмяПоляФормы = "q";
// *** Для сайта "http://yahoo.com":
String ^ АдресСайта = "http://yahoo.com";
String ^ ИмяФормы = "sf1";  // или "p_13838465-searchform"
String ^ ИмяПоляФормы = "p";  // или "p_13838465-p"
// Загружаем веб-документ в элемент WebBrowser:
webBrowser1->Navigate(АдресСайта);
while (webBrowser1->ReadyState !=
       WebBrowserReadyState::Complete)
{
    Application::DoEvents();
    System::Threading::Thread::Sleep(50);
}
if (webBrowser1->Document == nullptr)
{
    MessageBox::Show(
        "Возможно, вы не подключены к Интернету", "Ошибка");
    return;
}
// Свойство DomDocument приводим к указателю IHTMLDocument2:
mshtml::IHTMLDocument2 ^ Док = (mshtml::
        IHTMLDocument2 ^)webBrowser1->Document->DomDocument;
// В этом случае мы получаем доступ к формам веб-страницы
// по их именам:
mshtml::HTMLFormElement ^ Форма = (mshtml::
```

продолжение ⇘

Листинг 8.4 (продолжение)

```
                    HTMLFormElement ^)Док->forms->item(ИмяФормы, nullptr);
                if (Форма == nullptr)
                {
                    MessageBox::Show(String::Format("Форма с " +
                        "именем \"{0}\" не найдена", ИмяФормы), "Ошибка");
                    return;
                }
                // В форме находим нужное поле по его (полю) имени:
                mshtml::IHTMLInputElement ^ ТекстовоеПоле = (mshtml::
                    IHTMLInputElement ^)Форма->namedItem(ИмяПоляФормы);
                if (ТекстовоеПоле == nullptr)
                {
                    MessageBox::Show(String::Format("Поле формы с " +
                        "именем \"{0}\" не найдено ", ИмяПоляФормы), "Ошибка");
                    return;
                }
                // Заполняем текстовое поле:
                String ^ ТекстЗапроса = "Зиборов Visual";
                ТекстовоеПоле->value = ТекстЗапроса;
                // "Программно" нажимаем кнопку "Submit":
                Форма->submit();
            }
    };
}
```

Как мы видим, весь наш пользовательский программный код написан для обработки события загрузки формы. Здесь мы подготовили для поисковой системы Yahoo! ее веб-адрес, имя формы и имя поля формы, куда пользователь вводит ключевые слова для поиска. Также в комментариях приведены эти параметры и для нескольких других поисковых систем: Google, Meta, Yandex, Rambler, Aport и Bing. Как мы определим эти параметры?

Для этого в браузере, например Internet Explorer, зададим в адресной строке имя поисковой системы Yahoo!, затем в контекстном меню выберем команду Просмотр HTML-кода. В этом случае мы увидим в Блокноте HTML-разметку начальной страницы системы. Используя контекстный поиск программы Блокнот, в разметке поисковой системы Yahoo! найдем тег формы <form>. В этом теге следует найти либо атрибут name, либо атрибут id. Любой из этих атрибутов можно указать в качестве имени формы в нашей программе. Далее между открывающим тегом <form> и закрывающим тегом </form> ищем тег <input>, представляющий поле формы. Этот тег должен иметь тип (type) text, но не hidden (скрытый). В этом теге найдем или атрибут name, или атрибут id. Любой из них можно указать в качестве имени поля формы в нашей программе. Аналогичным образом мы нашли имена форм и полей этих форм и в других веб-страницах поисковых систем, которые представлены в комментариях.

После первичной загрузки веб-документа в элемент управления WebBrowser, используя указатель на неуправляемый интерфейс DomDocument (свойство объекта

класса WebBrowser), приводим его к указателю IHTMLDocument2. В этом случае мы получаем доступ к формам и полям веб-страницы по их именам. Далее заполняем поле поиска ключевыми словами для нахождения ссылок на соответствующие веб-страницы, а затем для отправки заполненной формы на сервер «программно» нажимаем кнопку Submit. В результате получим в элементе WebBrowser результат работы поисковой системы, а именно множество ссылок на страницы, содержащие указанные ключевые слова (рис. 8.4).

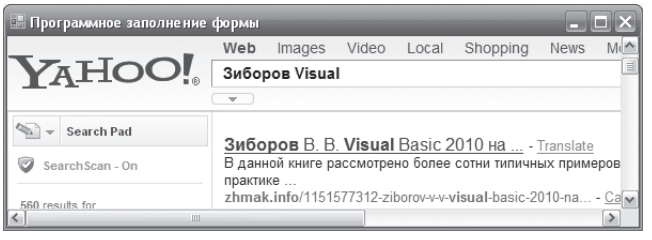

Рис. 8.4. Результат работы поисковой системы

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ЗаполнениеВеб_формы.

Пример 65. Синтаксический разбор веб-страницы без использования элемента WebBrowser

Чтобы иметь доступ к данным какой-либо веб-страницы (к ее гипертекстовой разметке), вовсе не обязательно использовать элемент управления WebBrowser. В примере этого раздела рассмотрим, как можно воспользоваться какими-либо актуальными сведениями, представленными на веб-странице, скопировав их на экранную форму нашей программы. Для более конкретной постановки задач, представим, что нам необходимо по роду наших занятий знать официальный курс доллара США на текущий момент. Где нам взять этот официальный курс? Конечно же, на сайте Центрального банка Российской Федерации www.cbr.ru. Причем курс доллара должен быть постоянно перед нашими глазами, чтобы мы не обращались к этому сайту каждый раз, когда он нам будет нужен, он должен быть на нашем компьютерном Рабочем столе. Исходя из поставленной задачи, воспользуемся классом *WebClient*, с его помощью прочитаем гипертекстовую разметку в строковую переменную, найдем в ней нужные сведения и скопируем их в текстовую метку нашей экранной формы. Заметьте, что в нашем случае участие элемента управления WebBrowser вовсе не обязательно.

Для решения этой задачи запустим Visual Studio 2010, щелкнем пункт меню New Project. В появившемся окне New Project в левой колонке в узле Visual C++ выберем среду CLR, а затем в области шаблоны (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем

имя РазборВебСтраницы и щелкнем на кнопке OK. А с панели элементов Tollbox нам понадобится текстовая метка Label и графическое поле PictureBox. Последнее нам будет нужно для отображения логотипа банка России с их сайта.

Чтобы получить доступ к упомянутому классу WebClient, нам понадобится ссылка на библиотеку System.Net.dll. Для добавления ссылки на эту библиотеку в пункте меню Project выберем команду Add Reference, а на вкладке .NET дважды щелкнем на ссылке System.Net. Теперь в окне Properties в области References (Ссылки) увидим добавленную в наш проект выбранную ссылку.

Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 8.5.

Листинг 8.5. Синтаксический разбор HTML-разметки веб-страницы Центрального банка РФ

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа, используя класс WebClient, читает веб-страницу Центрального
        // банка РФ www.cbr.ru, ищет в ее гипертекстовой разметке курс доллара
        // США и копирует его в текстовую метку Label. Кроме того, элемент
        // управления "графическое поле PictureBox" отображает логотип банка,
        // используя URL-адрес этого изображения
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // В данный проект необходимо добавить ссылку: Project ▸ Properties ▸
        // Add Reference, а затем на вкладке .NET добавить компоненту System.Net
    private: System::Void Form1_Load(System::Object^  sender,
                System::EventArgs^  e)
             {
                this->Text = "Сведения от банка России";
                // Создаем объект для чтения веб-страницы:
                auto КЛИЕНТ = gcnew System::Net::WebClient();
                // Если веб-страница записана в кодировке Win1251, то чтобы
                // русские буквы читались корректно, следует объявить
                // объект Кодировка:
                // auto Кодировка =
                //          System::Text::Encoding::GetEncoding(1251);
                System::IO::Stream ^ ПОТОК;
                String ^ СТРОКА;
                try
                { // Попытка открытия веб-ресурса:
                    ПОТОК = КЛИЕНТ->OpenRead("http://www.cbr.ru/");
                }
                catch (Exception ^ Ситуация)
                {
                    СТРОКА = String::Format(
                        "www.cbr.ru\r\n{0}", Ситуация->Message);
```

```
                label1->Text = СТРОКА;
                return;
            }
            // Чтение HTML-разметки веб-страницы в кодировке
            // Unicode (по умолчанию):
            auto Читатель = gcnew
                System::IO::StreamReader(ПОТОК); //, Кодировка);
            // Копируем HTML-разметку в строковую переменную:
            СТРОКА = Читатель->ReadToEnd();
            // Ищем в HTML-разметке страницы курс доллара США:
            int i = СТРОКА->IndexOf("Доллар США");
            СТРОКА = СТРОКА->Substring(i, 300);
            i = СТРОКА->IndexOf("nowrap");
            СТРОКА = СТРОКА->Substring(i + 7);
            i = СТРОКА->IndexOf(" ");
            СТРОКА = СТРОКА->Remove(i);
            // Вставляем текущую дату:
            СТРОКА = String::Format("Курс доллара США на {0:D}: {1}",
                DateTime::Now, СТРОКА);
            ПОТОК->Close();
            // Копируем в текстовую метку найденный курс доллара:
            label1->Text = СТРОКА;
            // В графическом поле отображаем логотип Центрального банка:
            pictureBox1->ImageLocation =
                "http://www.cbr.ru/images/main_logo.gif";
        }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы создаем объект класса WebClient, с помощью которого читаем веб-страницу. Данная веб-страница записана в кодировке Unicode. Я хотел бы, чтобы вы знали, как определить, в какой кодировке создана та или иная веб-страница. Для этого в любом браузере следует отобразить эту веб-страницу, затем в контекстном меню (нажав правую кнопку мыши) выбрать пункт Кодировка. Если страница была написана в другой кодировке, например очень распространенной Windows1251, то нам пришлось бы создать объект класса Encoding, как показано в комментарии, для корректного отображения русских букв.

Прочитав в строковую переменную HTML-разметку нужного нам веб-ресурса, найдем с помощью строковых операций курс доллара США на текущий момент. Результат работы программы представлен на рис. 8.5.

Убедиться в работоспособности программы можно, открыв решение РазборВебСтраницы.sln в папке РазборВебСтраницы.

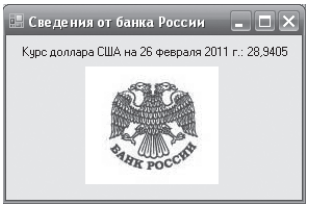

Рис. 8.5. Поиск нужных сведений на веб-ресурсе и их отображение в форме

9 Использование функций MS Word, MS Excel, AutoCAD и MATLAB, а также создание PDF-файла

Пример 66. Проверка правописания в текстовом поле с помощью обращения к MS Word

Пакет приложений Microsoft Office может являться сервером OLE-объектов, и его функции могут использоваться другими приложениями. Продемонстрируем такое использование. Для этого создадим программу, которая позволяет пользователю ввести какие-либо слова, предложения в текстовое поле и после нажатия соответствующей кнопки проверить орфографию введенного текста. Для непосредственной проверки орфографии воспользуемся функцией CheckSpelling объектной библиотеки MS Word.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Перетащим из панели элементов Toolbox в форму текстовое поле. Чтобы растянуть его на всю форму, в свойстве Multiline текстового поля укажем True (разрешим введение множества строк). Также с панели элементов перетащим кнопку Button. Мы должны получить примерно такой дизайн, который представлен на рис. 9.1.

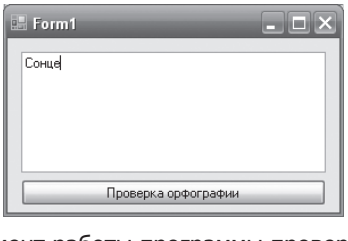

Рис. 9.1. Фрагмент работы программы проверки орфографии

Далее к текущему проекту добавим объектную библиотеку MS Word (библиотеку компонентов). Для этого в пункте меню Project выберем команду Add Reference. Затем, если на вашем компьютере установлен MS Office 2003, то на вкладке COM дважды щелкнем по ссылке на библиотеку Microsoft Word 11.0 Object Libary. Если же установлен MS Office 2007, то дважды щелкнем на ссылке Microsoft Word 12.0 Object Library. Эта объектная библиотека соответствует файлу, расположенному по адресу: C:\Program Files\Microsoft Office\OFFICE11\MSWORD.OLB (или ...\OFFICE12\MSWORD.OLB для MS Office 2007).

Теперь убедимся в том, что данная ссылка благополучно установлена. Для этого в окне Properties в области References (Ссылки) найдем добавленную в наш проект выбранную ссылку Microsoft.Office.Interop.Word. Кроме того, в папке проекта Interop появился файл Interop.Microsoft.Office.Interop.Word.8.4.dll.

Таким образом, мы подключили библиотеку объектов MS Word. Далее введем программный код, представленный в листинге 9.1.

Листинг 9.1. Проверка орфографии (вариант 1)

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа позволяет пользователю ввести какие-либо слова, предложения
        // в текстовое поле и после нажатия соответствующей кнопки проверить
        // орфографию введенного текста. Для непосредственной проверки орфографии
        // воспользуемся функцией CheckSpelling объектной библиотеки MS Word
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                 // В пункте меню Project выберем команду Add Reference.
                 // Затем, если на вашем компьютере установлен MS Office 2007,
                 // на вкладке COM дважды щелкнем по ссылке
                 // на библиотеку Microsoft Word 12.0 Object Libary.
                 textBox1->Clear(); button1->Text = "Проверка орфографии";
                 textBox1->TabIndex = 0; button1->TabIndex = 1;
             }
    private: System::Void button1_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 // Создаем новый экземпляр класса Word::Application:
                 auto Ворд1 =
                     gcnew Microsoft::Office::Interop::Word::Application();
                 // Ворд1->Visible = false;
                 // Переменная с "пустым" значением:
                 Object ^ t = Type::Missing;
                 // Открываем новый документ MS Word:
```

продолжение ⇗

Листинг 9.1 (продолжение)

```
                auto Документ = Ворд1->Documents->Add(t, t, t, t);
                // Вводим в документ MS Word текст из текстового поля:
                Документ->Words->First->InsertBefore(textBox1->Text);
                // Непосредственная проверка орфографии:
                Документ->CheckSpelling(t, t, t, t, t, t, t, t, t, t, t, t);
                // Получаем исправленный текст:
                String ^ ИсправленныйТекст = Документ->Content->default;
                // В Visual Basic и C# к этому свойству следует обратиться
                // так: ИсправленныйТекст = Документ->Content->Text;
                // Возвращаем в текстовое поле исправленный текст:
                textBox1->Text = ИсправленныйТекст->Replace("\r", "");
                Object ^ tt = false;
                    // или tt = Microsoft::Office::Interop::Word::
                    //              WdSaveOptions::wdDoNotSaveChanges;
                Ворд1->Documents->Close(tt, t, t);
                // Закрыть документ Word без сохранения:
                Ворд1->Quit(tt, t, t);
                Ворд1 = nullptr;
            }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы очищается текстовое поле и инициализируется название кнопки Проверка орфографии. При обработке события «щелчок по кнопке Button1_Click» создается новый объект класса Word::Application, и командой Documents->Add открывается новый документ. Здесь мы использовали поле Type::Missing для получения значения параметра метода по умолчанию. Далее весь введенный пользователем текст копируется в этот документ (команда InsertBefore). Затем происходит непосредственная проверка орфографии методом CheckSpelling. Если текст проверен, то правильный вариант написания слова находится теперь в одном из свойств объекта класса Word::Document, а именно в свойстве Content->Text. Если бы мы писали программу на Visual Basic или на C#, то мы бы к этому свойству так и обратились: Content->Text. Однако транслятор Visual C++ требует, чтобы к этому свойству мы обратились, используя ключевое слово default. Теперь исправленный текст возвращаем назад в текстовое поле. Далее документ MS Word закрываем без сохранения изменений wdDoNotSaveChanges, поэтому при работе программы мы этот документ даже не замечаем.

Теперь проверим, как работает написанная нами программа. Запустим ее, нажав на функциональную клавишу F5, и в текстовое поле введем какое-либо слово с ошибкой. После щелчка на кнопке Проверка орфографии получим диалоговое окно, подобное представленному на рис. 9.2.

Выберем правильный вариант написания слова и щелкнем на кнопке Заменить, при этом диалоговое окно закроется, а в нашем текстовом поле на форме окажется исправленное слово.

В листинге 9.2 приведем другой, более краткий вариант решения этой задачи с использованием объекта класса Selection.

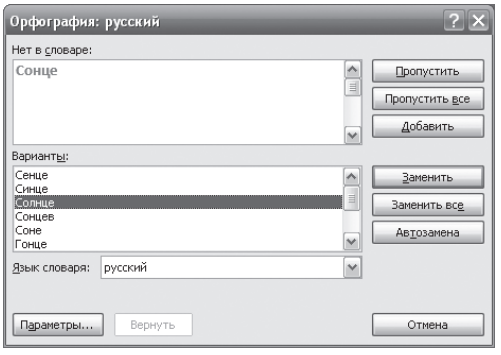

Рис. 9.2. Проверка правописания в текстовом поле

Листинг 9.2. Проверка орфографии (вариант 2)

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа позволяет пользователю проверить
        // орфографию введенного текста
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                textBox1->Clear(); button1->Text = "Проверка орфографии";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                auto Ворд1 =
                    gcnew Microsoft::Office::Interop::Word::Application();
                Ворд1->Visible = false;
                // Перменная с "пустым" значением:
                auto t = Type::Missing;
                // Открываем новый документ MS Word:
                Ворд1->Documents->Add(t, t, t, t);
                // Копируем содержимое текстового окна в документ
                Ворд1->Selection->default = textBox1->Text;
                // Для VB и C# было бы: Ворд1->Selection->Text
                // Непосредственная проверка орфографии:
                Ворд1->ActiveDocument->
                    CheckSpelling(t, t, t, t, t, t, t, t, t, t, t, t);
                // Копируем результат назад в текстовое поле
                textBox1->Text = Ворд1->Selection->default;
```

продолжение ⇗

Листинг 9.2 (продолжение)

```
                Object ^ tt = false;
                Ворд1->Documents->Close(tt, t, t);
                // Закрыть документ Word без сохранения:
                Ворд1->Quit(tt, t, t);
                Ворд1 = nullptr;
            }
    };
}
```

В этом программном коде аналогично создаем новый объект класса Word::Application и, используя метод Add, открываем документ MS Word. Далее воспользуемся объектом Ворд1.Selection, он оперирует с текущей позицией «невидимого» курсора, например, может подать команду ввода текста в документ MS Word (метод TypeText). Мы используем свойство Text этого объекта для копирования содержимого текстового окна в документ MS Word. Соответственно доступ к этому свойству должен быть реализован с помощью предложения: Ворд1->Selection->Text. Так мы бы и поступили, если бы писали программу на VB или C#. Однако транслятор C++ требует, чтобы к этому свойству мы обратились, используя ключевое слово default. Далее происходит непосредственная проверка орфографии аналогично предыдущему варианту решения этой задачи.

Убедиться в работоспособности приведенных программ можно, открыв решения в папках соответственно Орфография1 и Орфография2.

Пример 67. Вывод таблицы средствами MS Word

В данной книге мы уже рассматривали способы формирования таблицы. В этом разделе мы обсудим способ создания таблицы с использованием функции MS Word. Следует отметить, что программировать взаимодействие программы на Visual C++ 2010 с различными офисными приложениями (Word, Excel, Access, PowerPoint и т. д.), а также с AutoCAD и CorelDRAW удобно, поскольку во все эти приложения встроен язык VBA (Visual Basic for Applications), в арсенале которого находятся программные объекты, названия и назначения которых во многом схожи с объектами, используемыми в Visual C++/CLI (так же как и в Visual C#). Кроме того, существует возможность записи макроса с последующим просмотром соответствующей VBA-программы. Например, мы хотим посмотреть, как организована вставка таблицы в редакторе MS Word. Для этого запускаем MS Word, затем в меню Сервис выбираем команду Макрос ▸ Начать запись, далее в диалоговом окне Запись макроса указываем имя макроса и щелкаем на кнопке OK. Теперь в текст MS Word вставляем таблицу, используя пункты меню Таблица ▸ Вставить ▸ Таблица и т. д. После заполнения таблицы нажимаем кнопку Остановить запись. Далее с помощью комбинации клавиш Alt+F11 откроем окно Microsoft Visual Basic, здесь мы увидим текст макроса на языке VBA. Из этого текста мы можем понять основной принцип, имена используемых объектов, функций, свойств и пр.

А теперь рассмотрим конечный результат — программу на Visual C++ 2010, которая, используя функции MS Word, строит таблицу. Итак, запускаем Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Далее к текущему проекту добавим объектную библиотеку MS Word. Для этого в меню Project укажем команду Add Reference и на появившейся вкладке COM дважды щелкнем по ссылке на библиотеку Microsoft Word 11.0 Object Library (или другая версия MS Word, например 12.0 Object Library). На экранную форму перенесем командную кнопку Button, чтобы работа программы выглядела более выразительно. То есть именно после щелчка на кнопке будет формироваться таблица и вызываться MS Word для ее отображения. Далее введем программный код, представленный в листинге 9.3.

Листинг 9.3. Вывод таблицы средствами MS Word

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа вывода таблицы средствами MS Word: запускается
        // программа, пользователь наблюдает, как запускается редактор
        // MS Word и автоматически происходит построение таблицы
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 // В меню Project укажем команду Add Reference и на
                 // появившейся вкладке COM дважды щелкнем по ссылке на
                 // библиотеку Microsoft Word 12.0 Object Library
                 button1->Text = "Пуск"; this->Text = "Построение таблицы";
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             { // Инициализируем два строковых массива:
                 array<String^> ^ Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                                  "Справка по тел", "Александр Степанович",
                                           "Мама - дом", "Карапузова Таня",
                                           "Погода сегодня", "Театр Браво"};
                 array<String^> ^ Tel = {"274-88-17", "+38(067)7030356",
                              "22-345-72", "009", "223-67-67 доп 32-67",
                                    "570-38-76", "201-72-23-прямой моб",
                                                 "001", "216-40-22"};
                 // Создаем новый экземпляр класса Word::_Application:
                 auto Ворд1 =
                       gcnew Microsoft::Office::Interop::Word::Application();
                 Ворд1->Visible = true;
                 // Перменная с "пустым" значением:
                 auto t = Type::Missing;
```

продолжение ⇩

Листинг 9.3 (продолжение)

```
                // Открываем новый документ MS Word:
                auto Документ = Ворд1->Documents->Add(t, t, t, t);
                // Вводим текст в документ MS WORD с текущей позиции:
                Ворд1->Selection->TypeText("ТАБЛИЦА ТЕЛЕФОНОВ");
                // Параметр, указывающий, показывать ли границы ячеек:
                System::Object ^ t1 = Microsoft::Office::Interop::
                    Word::WdDefaultTableBehavior::wdWord9TableBehavior;
                // Параметр, указывающий будет ли приложение Word автоматически
                // изменять размер ячеек в таблице для подгонки содержимого:
                System::Object ^ t2 = Microsoft::Office::Interop::
                    Word::WdAutoFitBehavior::wdAutoFitContent;
                // Создаем таблицу из 9 строк и 2 столбцов:
                Ворд1->ActiveDocument->Tables->Add(Ворд1->Selection->Range,
                                                   9, 2, t1, t2);
                // Заполнять ячейки таблицы можно так:
                for (int i = 1; i <= 9; i++)
                {
                    Ворд1->ActiveDocument->Tables[1]->Cell(i, 1)->
                                        default->InsertAfter(Imena[i - 1]);
                    Ворд1->ActiveDocument->Tables[1]->Cell(i, 2)->
                                          default->InsertAfter(Tel[i - 1]);
                 // Программируя на C#, мы написали бы:
                 // Ворд1.ActiveDocument.Tables[1].Cell(i, 2).
                 //                          Range.InsertAfter(Tel[i - 1]);
                }
                // Назначаем единицы измерения в документе приложения MS Word:
                Object ^ t3 = Microsoft::Office::Interop::Word::WdUnits::wdLine;
                // Параметр, указывающий на девятую строку в документе MS Word:
                Object ^ строка9 = 9;
                // Перевести текущую позицию (Selection) за пределы таблицы,
                // (в девятую строку), чтобы здесь вывести какой-либо текст:
                Ворд1->Selection->MoveDown(t3, строка9, t);
                // И здесь печатаем следующий текст:
                Ворд1->Selection->TypeText("Какой-либо текст после таблицы");
                // Сохранять документ нет смысла, но это решит пользователь:
                // Object ^ ИмяФайла = "C:\\a.doc";
                // Ворд1->ActiveDocument->SaveAs(ИмяФайла, t, t, t, t, t,
                //                          t, t, t, t, t, t, t, t, t, t);
            }
    };
}
```

Заметим, что содержимое текстовой таблицы такое же, как и в примере 44 (см. главу 7). То есть наш сюжет меняется, а действующие персонажи остаются прежними. Данные находятся в двух массивах: Imena[] и Tel[]. Мы создаем экземпляр объекта Word::Application и открываем новый документ Document::Add. Здесь мы использовали поле Type::Missing для получения значения параметра метода по умолчанию. Затем демонстрируем, как можно добавлять какие-либо тексты в но-

вый документ из C++-программы. Например, мы вводим в активный документ текст «ТАБЛИЦА ТЕЛЕФОНОВ», используя объект Selection, определяющий текущую позицию «невидимого» курсора и содержащий методы для ввода в документ MS Word.

Затем мы создаем таблицу, состоящую из девяти строк (рядов) и двух столбцов, причем ширина столбцов будет регулироваться в зависимости от содержимого ячеек (wdAutoFitContent). Затем в цикле заполняем ячейки таблицы и выводим курсор (Selection) за пределы таблицы, чтобы написать какой-либо текст.

После запуска этой программы очень красиво, прямо на наших глазах в редакторе MS Word сформируется таблица (рис. 9.3), которую при желании можно редактировать, сохранять и распечатывать на принтере.

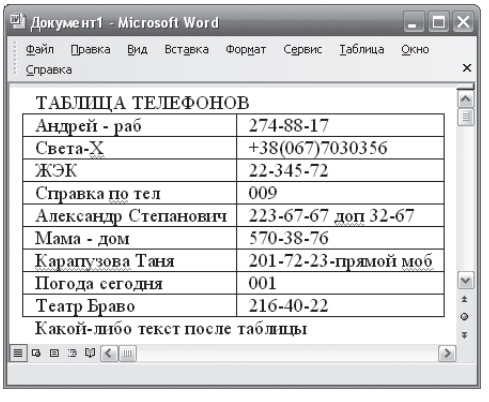

Рис. 9.3. Программно сформированная таблица в редакторе MS Word

Убедиться в работоспособности программы можно, открыв решение ТаблицаWord.sln в папке ТаблицаWord.

Пример 68. Обращение к функциям MS Excel из Visual C++ 2010

Очень заманчиво попробовать обратиться из какой-нибудь нашей C++-программы к функциям Microsoft Excel. Табличный редактор MS Excel содержит очень мощные средства для сложных вычислений и анализа данных, которые могут значительно расширить возможности программ, причем доступ к этим средствам, как мы сейчас убедимся, очень простой и удобный.

В данной программе мы продемонстрируем буквально в трех строчках программного кода обращение к одной простой функции MS Excel, а именно получение значения числа $p = 3{,}14$. Число p представлено в классе Math языка C++. Не будем забывать, что целью данной программы является, прежде всего, демонстрация легкости доступа к функциям MS Excel. Если мы сумеем обратиться к этой

элементарной функции, то нам откроется целый пласт возможностей управления огромной библиотекой объектов MS Excel.

Уже привычно запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Проектируемую экранную форму сделаем совсем маленькой, поскольку число π будем выводить в строку заголовка формы.

Чтобы добавить в текущий проект возможности MS Excel, следует подключить библиотеку объектов MS Excel. Для этого в пункте меню Project выберем команду Add Reference. Затем, если на вашем компьютере установлен MS Office 2003, то на вкладке COM дважды щелкнем по ссылке на библиотеку Microsoft Excel 11.0 Object Library. Если же установлен MS Office 2007, то дважды щелкнем на ссылке Microsoft Excel 12.0 Object Library. То есть процедура добавления новой библиотеки объектов такая же, как и в примерах, посвященных использованию возможностей MS Word, а названия пунктов меню сохранились почти такими же, как в предыдущих версиях Visual Studio. Подключить новую библиотеку объектов в текущий проект можно также через контекстное меню окна Solution Explorer (Обозреватель решений; если в данный момент вы не увидели это окно в своем текущем проекте, можно воспользоваться комбинацией клавиш Ctrl+Alt+L), щелкнув на пункте Add Reference.

Таким образом, мы подключили библиотеку объектов MS Excel. На содержимое этой библиотеки мы можем посмотреть в окне Object Browser (Обозреватель объектов). Чтобы его открыть, удобно воспользоваться комбинацией клавиш Ctrl+Alt+J. В этом окне найдем объект WorksheetFunction, при этом в области Members of 'WorksheetFunction' увидим доступные нам функции MS Excel для объекта WorksheetFunction. Теперь в программном коде обратимся к одной из этих функций, а именно функции Pi().

Для этого перейдем на вкладку Form1.h[Design] и дважды щелкнем в пределах проектируемой формы. При этом произойдет автоматический переход на вкладку программного кода Form1.h и будет создан пустой обработчик события загрузки формы. Здесь напишем программный код, приведенный в листинге 9.4.

Листинг 9.4. Обращение к одной из функций MS Excel

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа обращается к одной простой функции объектной библиотеки
        // MS Excel для получения значения числа Пи = 3,14...
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Чтобы добавить ссылку на объектную библиотеку Excel, в пункте меню
        // Project выберем команду Add Reference. Затем, если на вашем
        // компьютере установлен MS Office 2007, на вкладке COM дважды
        // щелкнем по ссылке на библиотеку Microsoft Excel 12.0 Object Library
    private: System::
                 Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
```

```
        {
            // Создание экземпляра класса Excel::Application:
            auto XL = gcnew Microsoft::Office::Interop::
                                        Excel::Application();
            double PI = XL->WorksheetFunction->Pi();
            // Выводим значение Пи в строку заголовка формы
            this->Text = "PI = " + PI;
            XL->Quit();
        }
    };
}
```

Заметим, что весь наш пользовательский программный код написан для обработки события загрузки формы. Мы создаем объект Excel::Application, с помощью которого получаем доступ для одной из функций MS Excel, возвращающей число $\pi = 3{,}14$.

Результат работы программы показан на рис. 9.4.

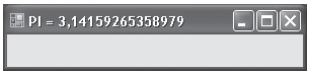

Рис. 9.4. Вывод числа π в заголовок формы

Убедиться в работоспособности программы можно, открыв решение ExcelПи.sln в папке ExcelПи.

Если мы имеем возможность управлять функциями MS Excel в C++-программе, для нас открываются самые широкие перспективы. Например, оцените возможность решать сложнейшие в математическом смысле оптимизационные задачи (то есть задачи нахождения максимума/минимума с набором ограничений), доступные в MS Excel через Сервис ▸ Поиск решения.

Пример 69. Использование финансовой функции MS Excel

Рассмотрим еще один пример обращения к функциям MS Excel из программы на Visual C++ 2010. Допустим, мы взяли кредит на покупку квартиры *100 тыс. долларов под 15 %* годовых, на срок 10 лет. Требуется узнать сумму, которую мы вынуждены будем платить ежемесячно. В русскоязычном MS Excel для подобных расчетов есть функция ПЛТ(), на вход которой следует подать месячную процентную ставку (то есть в нашем случае 0.15/12), срок погашения кредита в месяцах (120 месяцев) и размер кредита ($100 тыс.). Аналогом функции ПЛТ() является функция (метод) Pmt() класса WorksheetFunction, которая имеет такие же аргументы. Таким образом, мы можем написать C#-программу с обращением к функции Pmt() и проверить результат в русскоязычной версии MS Excel. Список всех методов (функций) объекта WorksheetFunction с описанием аргументов можно найти по адресу: http://msdn.microsoft.com/en-us/library/bb225774.aspx.

Для программирования обращений к этим функциям из программы, созданной в Visual Studio 2010, важно найти соответствие русскоязычных функций MS Excel и их аналогов в объекте WorksheetFunction для отладки на тестовых примерах.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. В проектируемую экранную форму из панели Toolbox перенесем три метки, три текстовых поля (для ввода трех вышеперечисленных аргументов функции Pmt()) и кнопку. В текущий проект подключаем библиотеку объектов MS Excel. Для этого в меню Project выберем команду Add Reference, затем на вкладке COM дважды щелкнем на ссылке Microsoft Excel 12.0 Object Library. Теперь можно перейти к программному коду, приведенному в листинге 9.5.

Листинг 9.5. Использование финансовой функции MS Excel

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа использует финансовую функцию Pmt() объектной библиотеки
        // MS Excel для вычисления суммы периодического платежа на основе
        // постоянства сумм платежей и постоянства процентной ставки
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Для подключения библиотеки объектов MS Excel в пункте меню Project
        // выберем команду Add Reference. Затем, если на вашем компьютере
        // установлен MS Office 2007, на вкладке COM дважды щелкнем по
        // ссылке на библиотеку Microsoft Excel 12.0 Object Library
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 this->Text = "Расчет ежемесячных платежей";
                 label1->Text = "Год. ставка в %";
                 label2->Text = "Срок в месяцах";
                 label3->Text = "Размер кредита";
                 textBox1->Clear(); textBox2->Clear(); textBox3->Clear();
                 button1->Text = "Расчет";
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                 try
                 {
                     auto XL = gcnew Microsoft::Office::Interop::
                                              Excel::Application();
                     // Перменная с "пустым" значением:
                     auto t = Type::Missing;
                     // Получаем размер месячного платежа:
```

```
                double pay = XL->WorksheetFunction->Pmt(
                    (Convert::ToDouble(textBox1->Text)) / 1200,
                     Convert::ToDouble(textBox2->Text),
                     Convert::ToDouble(textBox3->Text), t, t);
                // ИЛИ, если использовать функцию Pmt()
                // из Microsoft.VisualBasic:
                // double FV = 0;
                // Microsoft::VisualBasic::DueDate dt =
                //     Microsoft::VisualBasic::DueDate::EndOfPeriod;
                // double pay = Microsoft::VisualBasic::Financial::Pmt(
                //      (Convert::ToDouble(textBox1->Text)) / 1200,
                //       Convert::ToDouble(textBox2->Text),
                //       Convert::ToDouble(textBox3->Text), FV, dt);
                auto Строка = String::Format(
                    "Каждый месяц следует платить {0:$#.##} долларов",
                                                    Math::Abs(pay));
                MessageBox::Show(Строка);
                XL->Quit();
            }
            catch (Exception ^ Ситуация)
            {
                MessageBox::Show(Ситуация->Message, "Ошибка",
                    MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
            }
        }
    };
}
```

Как видно из текста программы, при обработке события загрузки формы очищаются (Clear) текстовые поля, а также подписываются названия этих полей с помощью меток label1—label3 и присваивается название кнопки Button1.

При обработке события «щелчок на кнопке Расчет» создается объект Excel::Application. Объект Excel::Application обеспечивает доступ к функциям MS Excel, в частности к функции Pmt(). На вход функции Pmt() подаем значения текстовых полей, конвертированных из строкового типа в тип Double. При этом первый аргумент функции переводим из годовой процентной ставки в месячную ставку в сотых долях единицы, поэтому делим на 1200. На вход функции также приходится подать еще два параметра, которые являются необязательными. Здесь мы использовали поле Type::Missing для получения значения параметра метода по умолчанию. На выходе функции Pmt() получаем размер месячного платежа, который выводим, используя функцию MessageBox::Show.

Обращение к функциям MS Excel оформляем в блоке try...catch для обработки исключительных ситуаций (Exception). Замечу, что в данной программе мы не предусмотрели диагностику обязательного заполнения всех полей, а также диагностику ввода только числовых данных, чтобы не перегружать текст программы многочисленными очевидными подробностями.

В данной программе мы несколько переменных объявили как auto, это означает, что тип соответствующей переменной выводится из выражения инициализации

(как объявление var в C#), как мы уже отмечали — это новая возможность в Visual C++ 2010.

Интерфейс программы показан на рис. 9.5.

Рис. 9.5. Расчет ежемесячных платежей

Убедиться в работоспособности программы можно, открыв решение ExcelПлт.sln в папке ExcelПлт.

В данном разделе на простом примере мы рассмотрели, как легко подключиться к библиотеке объектов MS Excel и пользоваться ее функциями. Однако функция Pmt() имеется также в среде Visual Studio 2010 в пространстве имен Microsoft::VisualBasic::Financial точно с такими же параметрами. (Более того, эта функция была еще в Visual Basic 6.) Для обращения к этой функции потребовалось бы подключение к Visual Basic, как мы это делали в примере 15 (см. главу 2, это просто). То есть следовало бы в проект добавить ссылку на библиотеку Microsoft.VisualBasic.dll. Для этого в пункте меню Project надо выбрать команду Add Reference, а на вкладке .NET дважды щелкнуть на ссылке Microsoft.VisualBasic. В этом случае можно было бы обращаться к функции Pmt(), *как это представлено в комментарии*. Однако в данном примере показана принципиальная возможность работы с функциями MS Excel из C++-программы.

Пример 70. Решение системы уравнений с помощью функций MS Excel

Используя функции MS Excel, в своей программе, созданной в Visual C++ 2010, можно решать и более серьезные задачи. Например, рассмотрим, как решить *систему линейных алгебраических уравнений*:

$$\begin{aligned} X_1 + X_2 + X_3 &= 6 \\ X_1 + X_2 &= 3 \\ X_2 + X_3 &= 5 \end{aligned}$$

через *обратную матрицу*. Исходную систему уравнений запишем в матричном виде:

$$A \times X = L$$

Здесь A — матрица коэффициентов при неизвестных; X — вектор неизвестных X_1, X_2, X_3; L — вектор свободных членов 6, 3, 5.

Тогда решением системы будет выражение:

$$X = A^1 \times L,$$

где X^1 — обратная матрица.

Для нахождения обратной матрицы в русскоязычной версии MS Excel существует функция МОБР(), а объект WorksheetFunction в библиотеке объектов Microsoft Excel имеет функцию MInverse(). Для умножения обратной матрицы на вектор свободных членов есть соответственно функции МПРОИЗ() и MMult(). Такие функции отсутствуют в Visual Studio 2010, и в данном случае мы получаем реальный положительный эффект от подключения к функциям MS Excel.

Для программной реализации решения поставленной задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. В проектируемую форму из панели Toolbox добавим текстовую метку Label. На ней будем формировать ответ задачи. Кроме того, добавим библиотеку объектов MS Excel. Для этого в пункте меню Project выберем команду Add Reference и на вкладке COM отметим библиотеку Microsoft Excel 12.0 Object Library, а затем щелкнем на кнопке OK.

Программу построим следующим образом (листинг 9.6): при обработке события загрузки формы прямо в тексте программы зададим (инициализируем) прямую матрицу в виде двумерного массива и строку свободных членов в виде одномерного массива. Затем после решения системы выведем ответ на метку label1.

Листинг 9.6. Решение системы линейных уравнений с использованием объектной библиотеки MS Excel

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа решает систему уравнений с помощью функций объектной
        // библиотеки MS Excel
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Для подключения библиотеки объектов MS Excel в пункте меню Project
        // выберем команду Add Reference. Затем, если на вашем компьютере
        // установлен MS Office 2007, на вкладке COM дважды щелкнем по
        // ссылке на библиотеку Microsoft Excel 12.0 Object Library
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 // Матричное уравнение AX = L решаем через
                 // обратную матрицу: X = A(-1)L.
                 // Здесь (-1) - "знак" обратной матрицы.
                 // Решаем систему
                 // X1 + X2 + X3 = 6
```

продолжение ⇗

Листинг 9.6 (продолжение)

```
                // X1 + X2      = 3
                //      X2 + X3 = 5
                // Для этой системы прямая матрица будет иметь вид:
                // array<double,2> ^ A = gcnew array<double,2>(n, n);
                array<double,2> ^ A = { {1, 1, 1},
                                        {1, 1, 0},
                                        {0, 1, 1} };
                // Строка свободных членов:
                // array<double> ^ L = gcnew array<double>(n);
                // Свободные члены:
                array<double> ^ L = { 6, 3, 5 };
                // Создание экземпляра класса Excel::Application:
                auto XL1 = gcnew Microsoft::Office::Interop::
                                            Excel::Application();
                // Вычисление детерминанта матрицы A
                double det_A = XL1->Application->WorksheetFunction->MDeterm(A);
                // Если det_A != 0, то выход из процедуры:
                if (Math::Abs(det_A) < 0.01)
                {
                    label1->Text = "Система не имеет решения, поскольку\n\n" +
                                   "определитель равен нулю";
                    return;
                }
                // Получение обратной матрицы oA:
                Object ^ oA = XL1->Application->
                                     WorksheetFunction->MInverse(A);
                // Функция Transpose преобразует строку свободных
                // членов в вектор:
                Object ^ ВекторL = XL1->Application->
                                    WorksheetFunction->Transpose(L);
                // Умножение обратной матрицы на вектор свободных членов:
                Object ^ X =
                    XL1->Application->WorksheetFunction->MMult(oA, ВекторL);
                // Чтобы ответ приобрел индексированные свойства,
                // преобразуем его в массив:
                array<Object^,2> ^ Xd = (array<Object^,2> ^)X;
                // Получаем двумерный массив, индексы которого
                // начинаются с единицы:
                label1->Text = String::Format(
                    "Неизвестные равны:\n\nX1 = {0};  X2 = {1};  X3 = {2}.",
                                          Xd[1, 1],  Xd[2, 1],  Xd[3, 1]);
            }
    };
}
```

Как видно из текста программы, задавая прямую матрицу, значения коэффициентов присваиваем сразу при объявлении двумерного массива. Аналогично поступаем со строкой (одномерным массивом) свободных членов. Согласно требованию объекта WorksheetFunction возвращаемые обратная матрица и вектор неизвестных

должны быть объявлены как объектные переменные. Вначале вычисляем детерминант (определитель) прямой матрицы, используя функцию MS Excel Mdeterm(). Если прямая матрица *плохо обусловлена*, то есть определитель по абсолютному значению меньше 0.01, то выходим из процедуры и сообщаем пользователю в метке label1, что система не имеет решения.

Если определитель матрицы больше 0.01, то с помощью функции MS Excel MInverse() находим обратную матрицу. Теперь строку (одномерный массив) свободных членов следует преобразовать в вектор, для этой цели MS Excel используем функцию Transpose (то есть транспонировать массив L). Далее обратную матрицу с помощью функции MS Excel MMult() умножаем на вектор свободных членов. В результате функция MMult() возвращает двумерный массив, индексы которого начинаются с единицы (но не с нуля). Следующим оператором форматируем ответ в метке label1.

Результат работы программы приведен на рис. 9.6. Убедиться в работоспособности программы можно, открыв решение ExcelСЛАУ.sln в папке ExcelСЛАУ.

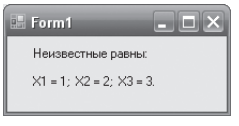

Рис. 9.6. Решение системы линейных алгебраических уравнений

Как видим, довольно сложные задачи можно решать при помощи коротенькой программы благодаря обращению к функциям MS Excel. Причем на компьютере, где будет работать данная программа, вовсе не обязательно должен быть инсталлирован MS Excel. Однако инсталляционный пакет вашей программы должен содержать соответствующую dll-библиотеку.

Пример 71. Построение диаграммы средствами MS Excel

Зачастую какие-либо экономические показатели или технические измерения (геодезические, метрологические, астрономические) необходимо представить в виде графика (диаграммы), например, с целью более наглядной визуализации данных. Иногда сделать это надо очень оперативно. Для этих целей в ячейки рабочего листа MS Excel можно ввести измеренные данные, а затем, чтобы получить график, построенный по этим данным, воспользоваться пунктами меню Вставка ▸ Диаграмма. В данном разделе я покажу, как можно быстро получить график (диаграмму) из программы Visual C++ 2010, используя средства (объекты компонентной библиотеки) MS Excel.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR, поскольку экранная форма нам не требуется. Теперь к текущему проекту добавим библиотеку объектов MS Excel. Для этого в пункте меню Project выберем команду Add Reference, далее на вкладке COM отметим библиотеку Microsoft Excel 12.0 Object Library и щелкнем на кнопке OK. Затем на вкладке программного кода введем текст, приведенный в листинге 9.7.

Листинг 9.7. Построение диаграммы средствами MS Excel

```
// ExcelГрафик.cpp: главный файл проекта.
// Программа строит диаграмму (график), используя объекты
// компонентной библиотеки MS Excel
#include "stdafx.h"
using namespace System;
// Добавляем пространство имен для более коротких выражений:
using namespace Microsoft::Office::Interop::Excel;
// Для подключения библиотеки объектов MS Excel в пункте меню Project
// выберем команду Add Reference. Затем на вкладке COM дважды щелкнем
// по ссылке на библиотеку Microsoft Excel 12.0 Object Library
int main(array<System::String ^> ^args)
{
    // Создаем экземпляр класса Excel::Application:
    _Application ^ XL1 = gcnew Application();
    XL1->Visible = true;
    // Задаем параметр "по умолчанию" для его дальнейшего
    // использования в соответствующих методах:
    Object ^ t = Type::Missing;
    // Создаем новую книгу MS Excel:
    Workbook ^ Книга = XL1->Workbooks->Add(t);
    // Объявляем листы в книге:
    Sheets ^ Листы = Книга->Worksheets;
    // Выбираем первый лист:
    _Worksheet ^ Лист = (_Worksheet ^)Листы->Item[1];
    // Если хотим добавить еще один лист (четвертый) к уже существующим:
    // _Worksheet ^ Лист = safe_cast<_Worksheet^>(Листы->Item[ (Object^)1 ]);
    // Записываем данные по продажам в соответствующих ячейках:
    Лист->Range["A1", t]->Value2 = "Месяцы";
    Лист->Range["A2", t]->Value2 = "Март";
    Лист->Range["A3", t]->Value2 = "Апр";
    Лист->Range["A4", t]->Value2 = "Май";
    Лист->Range["A5", t]->Value2 = "Июнь";
    Лист->Range["A6", t]->Value2 = "Июль";
    Лист->Range["B1", t]->Value2 = "Продажи";
    Лист->Range["B2", t]->Value2 = 138;
    Лист->Range["B3", t]->Value2 = 85;
    Лист->Range["B4", t]->Value2 = 107;
    Лист->Range["B5", t]->Value2 = 56;
    Лист->Range["B6", t]->Value2 = 34;
    // Заказываем построение диаграммы (графика) с умалчиваемыми параметрами:
    _Chart ^ График = (_Chart ^)XL1->Charts->Add(t, t, t, t);
    // Задаем диапазон значений для построения графика:
    График->SetSourceData(Лист->Range["A2", "B6"], XlRowCol::xlColumns);
    // Задаем тип графика "столбиковая диаграмма" (гистограмма):
    График->ChartType = XlChartType::xlColumnClustered;
    // Отключаем легенду графика:
    График->HasLegend = false;
    // График имеет заголовок:
    График->HasTitle = true;
    График->ChartTitle->Caption = "ПРОДАЖИ ЗА ПЯТЬ МЕСЯЦЕВ";
```

```
    // Подпись оси X:
    Axis ^ ГоризонтальнаяОсь = (Axis^)График->Axes(
                    XlAxisType::xlCategory, XlAxisGroup::xlPrimary);
    ГоризонтальнаяОсь->HasTitle = true;
    ГоризонтальнаяОсь->AxisTitle->Text = "Месяцы";
    // Подпись оси Y:
    Axis ^ ВертикальнаяОсь = (Axis^)График->Axes(
                        XlAxisType::xlValue, XlAxisGroup::xlPrimary);
    ВертикальнаяОсь->HasTitle = true;
    ВертикальнаяОсь->AxisTitle->Text = "Уровни продаж";
    // Сохранение графика в растровом файле:
    // XL1->ActiveChart->Export("C:\\ExcelГрафик.jpg", t, t);
    return 0;
}
```

В самом начале программного кода *создаем объект класса* Excel::Application и рабочую книгу. Далее заполняем ячейки первого листа: вначале в ячейки Ai записываем подписи абсцисс гистограммы, а в ячейки Bi — значения ординат.

Саму диаграмму строим при помощи объекта класса _Chart. Напомним читателю, что мы в предыдущих примерах уже строили диаграмму, используя элемент управления экранной формы Chart. Однако, несмотря на похожесть названий, эти классы расположены в разных пространствах имен, поэтому названия их свойств, методов и событий являются разными.

Заказывая построение диаграммы, задаем диапазон значений для ее построения. Затем задаем тип диаграммы — xlColumnClustered, что соответствует гистограмме *(столбиковой диаграмме)*. Далее указываем название гистограммы и подписываем горизонтальную и вертикальную оси. Затем в комментарии приводим возможность сохранять полученную диаграмму на диске в виде jpg-файла с помощью функции Export(). Графическое отображение этого файла можно посмотреть на рис 9.7, а соответствующие исходные данные на листе Excel для построения диаграммы на рис. 9.8.

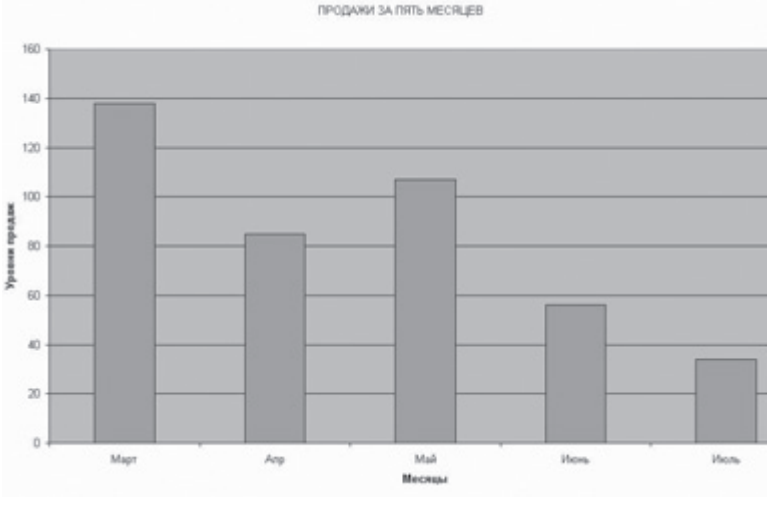

Рис. 9.7. Графическое отображение полученного jpg-файла

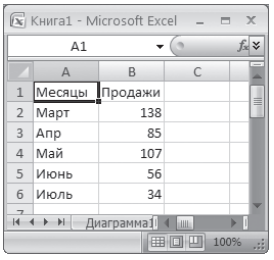

Рис. 9.8. Исходные данные для построения диаграммы на листе MS Excel

Убедиться в работоспособности программы можно, открыв решение ExcelГрафик.sln в папке ExcelГрафик.

Пример 72. Управление функциями AutoCAD из программы на Visual C++ 2010

Если результатом работы вашей программы должен быть какой-либо векторный чертеж (техническая документация, строительный чертеж, географическая карта и пр.), то самый быстрый путь создания такого приложения — это обращение к функциям AutoCAD из вашей C++-программы. AutoCAD (Computer-Aided Design) — это двух- и трехмерная система автоматизированного проектирования и черчения. Эта система, так же как и пакет приложений Microsoft Office, может являться сервером OLE-объектов, и его функции могут использоваться другими приложениями.

Графическими примитивами векторной графики являются отрезки, дуги, окружности, тексты, которые можно выводить под различными углами к горизонту, и, может быть, еще некоторые простейшие геометрические фигуры. Чертеж, подлежащий построению, состоит из совокупности таких элементов. Программа на C++ 2010, обращаясь к соответствующим функциям AutoCAD, формирует такой чертеж и записывает его в dwg-файл. Пользователь может просмотреть этот файл в среде AutoCAD, отредактировать его и вывести на печать.

Приступаем к программированию поставленной задачи. Поскольку в данной задаче нас не интересует пользовательский интерфейс, будем программировать консольное приложение. Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Далее следует подключить библиотеку объектов AutoCAD, для этого в пункте меню Project выбираем команду Add Reference, затем на вкладке COM выберем две ссылки, одну — с названием AutoCAD 2008 Type Library, а вторую — AutoCAD/ObjectDBX Common 17.0 Type Library..

Таким образом, мы подключили две библиотеки объектов AutoCAD версии 2008. Если на вашем компьютере инсталлирована другая версия AutoCAD, то действуйте аналогично. Программа, выводящая в dwg-файл два отрезка, дугу,

а также две горизонтально и вертикально ориентированных строки текста, представлена в листинге 9.8.

Листинг 9.8. Построение отрезков и двух строк текста в AutoCAD

```
// ACADЭлементарныйЧертеж.cpp: главный файл проекта.
// Программа строит средствами объектов библиотеки AutoCAD элементарный
// чертеж из отрезков и некоторого текста. Этот чертеж сохраняется в файле
// формата DWG. Конкретнее: эта программа запускает AutoCAD 2008, рисует
// два отрезка, одну дугу и два текстовых объекта, сохраняет чертеж
// в файле C:\Чертеж.dwg и завершает работу AutoCAD
// ~ ~ ~ ~ ~ ~ ~ ~ ~
// Следует подключить библиотеку объектов AutoCAD. Для этого надо выбрать
// Project ▸ Add Reference – вкладка COM – AutoCAD 2008 Type Library,
// а также AutoCAD/ObjectDBX Common 17.0 Type Library
#include "stdafx.h"
using namespace System;
// Пространство имен для ссылки AutoCAD 2008 Type Library на вкладке COM:
using namespace AutoCAD;
// Для ссылки AutoCAD/ObjectDBX Common 17.0 Type Library:
using namespace AXDBLib;
int main(array<System::String ^> ^args)
{
    // Создаем экземпляр класса AutoCAD::AcadApplication:
    AcadApplication ^ ACAD1 = gcnew AcadApplication();
    // Задаем параметр "по умолчанию" для его дальнейшего
    // использования в соответствующих методах:
    Object ^ t = Type::Missing;
    AcadDocument ^ Документ = ACAD1->Documents->Add(t);
    // Видимость:
    ACAD1->Visible = true;
    // Узловые точки чертежа:
    array<double> ^ T1 = { 10,  10,  0 };
    array<double> ^ T2 = { 200, 200, 0 };
    array<double> ^ T3 = { 200, 10,  0 };
    array<double> ^ T4 = { 15,  200, 0 };
    array<double> ^ T5 = { 100, 100, 0 };
    // Нарисовать отрезок от точки T1 до точки T2:
    Документ->ModelSpace->AddLine(T1, T2);
    // Вертикальный отрезок от точки T2 до точки T3:
    AutoCAD::AcadLine ^ ВертикальныйОтрезок =
                        Документ->ModelSpace->AddLine(T2, T3);
    // Построить дугу с центром в точке T5, радиусом 80,
    // от линии, параллельной оси X длиной 180 градусов:
    AutoCAD::AcadArc ^ Дуга =
                        Документ->ModelSpace->AddArc(T5, 80, 0, Math::PI);
    // или просто  Документ->ModelSpace->AddArc(T5, 80, 0, Math->PI);
    // Горизонтальный текст (с разворотом 0 градусов)
    Документ->ModelSpace->AddText("Горизонтальный", T4, 22);
```

продолжение ⇩

Листинг 9.8 (продолжение)

```
    // Вертикальный текст с разворотом на 90 градусов = ПИ/2:
    AutoCAD::AcadText ^ ВертикТекст =
                        Документ->ModelSpace->AddText("Вертикальный", T1, 22);
    // Задаем разворот текста на 90 гдадусов:
    ВертикТекст->Rotation = Math::PI / 2;
    // Сохраняем чертеж на диске:
    Документ->SaveAs("C:\\Чертеж.dwg", t, t);
    ACAD1->Quit();
    return 0;
}
```

Как видно из текста программы, вначале мы создаем объект класса AcadApplication и принадлежащий ему документ класса AcadDocument. Затем задаем видимость работы AutoCAD Visible = True, при этом AutoCAD только мелькнет на экране. (Заметьте, что для конечной цели, то есть для получения dwg-файла, видимость не обязательна.) Далее задаем пять точек, которые будут участвовать в построении чертежа. Обратите внимание, что каждая точка имеет три координаты, хотя мы собираемся строить плоский чертеж. Третью координату мы будем воспринимать, *как напоминание того, что* AutoCAD способен строить трехмерные чертежи.

Затем рисуем два отрезка AddLine через точки T1, T2 и T2, T3. Далее подаем команду выводить текст *горизонтально*, а — другой текст — *вертикально* с разворотом на 90°, то есть $\pi/2$. Затем, используя метод SaveAs, записываем построенный в документе чертеж в dwg-файл на логический диск C.

В результате работы этой программы получаем чертеж в системе AutoCAD, подобный представленному на рис. 9.9.

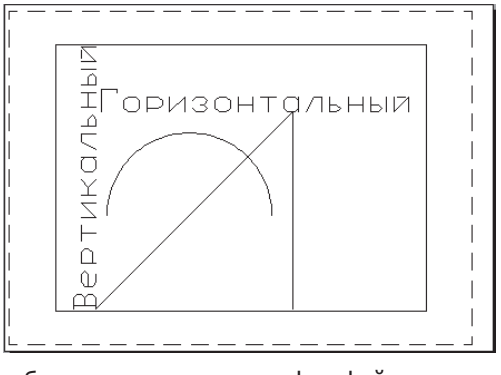

Рис. 9.9. Отображение полученного dwg-файла в системе AutoCAD

Убедиться в работоспособности программы можно, открыв sln-файл в папке ACADЭлементарныйЧертеж.

Пример 73. Вызов MATLAB из вашей программы на Visual C++ 2010

Следующей нашей задачей будет вызов среды MATLAB из C++-программы. Среда MATLAB является стандартным мощным инструментом для работы в различных отраслях математики. При подготовке этого примера я пользовался наиболее распространенной версией MATLAB 6.5. В данном примере продемонстрируем подготовку вводных данных для MATLAB, создание экземпляра объекта типа MATLAB и непосредственный его вызов на выполнение. Результатом наших действий должно стать построение графика функции $y = \sin(x) \times e^x$.

Для программирования этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели элементов Toolbox перенесем командную кнопку Button, чтобы обращение к среде MATLAB происходило при щелчке на этой кнопке и выглядело бы наиболее выразительно. Далее на вкладке программного кода введем текст из листинга 9.9.

Листинг 9.9. Использование возможностей среды MATLAB

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа вызывает простейшую функцию Matlab
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Для успешной работы программы нет необходимости добавлять ссылку на
        // объектную библиотеку через Project ▸ Reference. Однако на компьютере
        // MATLAB должен быть установлен
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                button1->Text = "Вызвать MATLAB";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Получить тип приложения MATLAB:
                Type ^ ТипМатЛаб =
                      Type::GetTypeFromProgID("Matlab.Application");
                // Создать экземпляр объекта типа MATLAB:
                Object ^ МатЛаб = Activator::CreateInstance(ТипМатЛаб);
                // Подготавливаем команды для MATLAB:
                array<Object^> ^ Команды =
                // { "surf(peaks)" };
                { "x = 0:0.1:6.28; y = sin(x).*exp(-x); plot(x,y)" };
```

продолжение ⇩

Листинг 9.9 (продолжение)

```
                // { "s = sin(0.5); c = cos(0.5); y = s*s+c*c; y" };
                // Вызываем MATLAB, подавая ему на вход подготовленные команды:
                Object ^ Результат =
                    ТипМатЛаб->InvokeMember("Execute", System::Reflection::
                    BindingFlags::InvokeMethod, nullptr, МатЛаб, Команды);
                MessageBox::Show(Результат->ToString());
            }
    };
}
```

Как видно из текста программы, при обработке события «щелчок на кнопке» в переменную ТипМатЛаб получаем тип приложения MATLAB. Далее создаем экземпляр объекта этого типа. Затем подготавливаем три команды для MATLAB, разделенные точкой с запятой. Первая команда x = 0:0.1:6.28; задает вектор x (набор чисел) от нуля до 2π (6,28) с шагом 0,1. Вторая команда y = sin(x).*exp(-x); вычисляет второй вектор по значениям первого вектора. Третья команда plot создает график зависимости y от x. Метод Execute выполняет в среде MATLAB подготовленные команды. В результате обращения к MATLAB получим построенный график заданной функции (рис. 9.10).

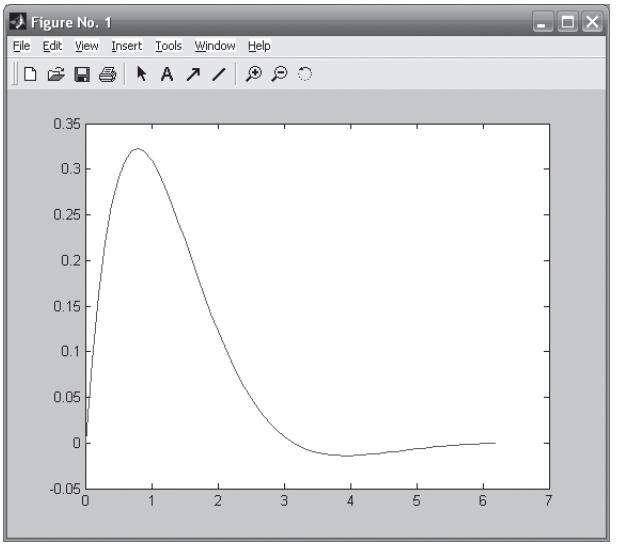

Рис. 9.10. График функции, построенный в среде MATLAB

В комментариях приведены и другие команды, которые можно выполнить, подключаясь к среде MATLAB.

Убедиться в работоспособности программы можно, открыв решение MatlabВызов.sln в папке MatlabВызов.

Пример 74. Решение системы уравнений путем обращения к MATLAB

Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами, которые создатели языка выразили в лозунге «Думай векторно» (от англ. *Think vectorized*). Следует отметить, что среда MATLAB начала свой эволюционный путь с задач матричной алгебры, отсюда и слово MATLAB, которое означает матричная лаборатория (matrix laboratory). Решить систему уравнений, глядя на предыдущий пример, очень просто, нужно всего лишь знать, как строятся команды в MATLAB. Продемонстрируем процесс решения системы линейных уравнений на следующем примере.

$$X_1 + X_2 + X_3 = 6$$
$$X_1 + X_2 = 3$$
$$X_2 + X_3 = 5$$

Данную систему решим через *обратную матрицу*.

Для программирования этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели элементов Toolbox перенесем командную кнопку Button. Далее на вкладке программного кода введем текст из листинга 9.10.

Листинг 9.10. Решение системы линейных уравнений с помощью MATLAB

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа, подготовив команды для решения системы уравнений в среде
        // MATLAB, вызывает его на выполнение этих команд. В результате получаем
        // решение, которое выводим на экран с помощью MessageBox
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Для успешной работы программы нет необходимости добавлять ссылку на
        // объектную библиотеку через Project ▸ Reference. Однако на компьютере
        // MATLAB должен быть установлен
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                button1->Text = "Решить СЛАУ";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Матричное уравнение AX = L решаем через
                // обратную матрицу: X = A(-1)L.
                // Здесь (-1) - "знак" обратной матрицы.
```

продолжение ⇘

Листинг 9.10 (продолжение)

```
                // Решаем систему
                // X1 + X2 + X3 = 6
                // X1 + X2      = 3
                //      X2 + X3 = 5
                // Для решения этой системы в MATLAB следует подать такие
                команды:
                // A = [1 1 1; 1 1 0; 0 1 1]; L = [6; 3; 5];
                // % здесь задание прямой матрицы A и вектора свободных членов L
                // X = inv(A)*L % умножение обратной матрицы на L" };
                // % - это признак комментария в MATLAB
                // Получить тип приложения MATLAB:
                Type ^ ТипМатЛаб = Type::GetTypeFromProgID("Matlab.Application");
                // Создать экземпляр объекта типа MATLAB:
                Object ^ МатЛаб = Activator::CreateInstance(ТипМатЛаб);
                // Подготавливаем команды для MATLAB:
                array<Object^> ^ Команды =
                { "A = [1 1 1; 1 1 0; 0 1 1]; L = [6; 3; 5]; " +
                  "X = inv(A)*L % обратная матрица inv" };
                // Вызываем MATLAB, подавая ему на вход подготовленные команды:
                Object ^ Результат = ТипМатЛаб->InvokeMember("Execute",
                      Reflection::BindingFlags::InvokeMethod,
                      nullptr, МатЛаб, Команды);
                // Таким образом мы могли бы вывести решение на экран:
                // MessageBox::Show(Результат->ToString());
                // Однако этот результат будет внутри строки, а хотелось бы
                // получить ответ в массив double для дальнейшей обработки.
                // Этот массив можно получить методом GetFullMatrix из среды
                // MATLAB, как показано ниже
                Reflection::ParameterModifier p =
                                  Reflection::ParameterModifier(4);
                p[0] = false; p[1] = false; p[2] = true; p[3] = true;
                array<Reflection::ParameterModifier> ^ mods = { p };
                array<double,2> ^ X = gcnew array<double,2>(3, 1);
                array<Object^> ^ Аргументы =
                                { "X", "base", X, gcnew array<double>(0) };
                // Здесь "X" - это название матрицы, которую мы хотим получить.
                // "base" является названием рабочей среды MATLAB, где следует
                // искать матрицу "X".
                Результат = ТипМатЛаб->InvokeMember("GetFullMatrix",
                    Reflection::BindingFlags::InvokeMethod, nullptr, МатЛаб,
                    Аргументы, mods, nullptr, nullptr);
                // Решение системы получаем в матрицу X:
                X = (array<double,2>^)Аргументы[2];
                String ^ Строка = String::Format(
                    "X1 = {0};  X2 = {1};  X3 = {2};",
                              X[0, 0], X[1, 0], X[2, 0]);
                MessageBox::Show(Строка);
            }
    };
}
```

Как видно из программного кода, подход к решению задачи аналогичен подходу предыдущего примера. Мы реализовали обращение к MATLAB, используя метод Execute, решение системы получили в переменную Результат и, как показано в комментарии, можем вывести результат вычислений на экран с помощью MessageBox. Однако для дальнейшей работы полученный вектор неизвестных нам нужен в виде массива Double, а не в виде строки. Конечно, можно выделить из строки решения каждое значение неизвестного с помощью операций со строками, используя функцию Split, которая возвращает строковый массив, содержащий подстроки данного экземпляра.

Однако существует более красивое решение (листинг 9.10). Оно заключается в использовании метода GetFullMatrix. Технологию этого использования мы привели в данной программе. Здесь наиболее важным параметром является объектная переменная Аргументы. Ее первым компонентом является компонент X, содержащий название матрицы, которую мы хотим получить из среды MATLAB, второй компонент — base — является названием рабочей среды (workspace) MATLAB, где следует искать матрицу X. Согласно документации, в среде MATLAB мы имеем две основные рабочие среды: base и global. Третьим компонентом является массив X, куда мы получаем результат решения из среды MATLAB, а четвертым компонентом — массив мнимой части решения, которой в нашей задаче нет, но для общности технологии требуется ее формальное присутствие. На рис. 9.11 приведен результат работы программы.

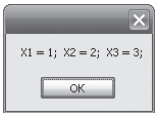

Рис. 9.11. Решение системы трех уравнений с помощью MATLAB

О других способах подключения к MATLAB можно узнать на сайте компании The MathWorks, производителя MATLAB, www.mathworks.com. Убедиться в работоспособности обсуждаемой программы можно, открыв решение MatlabClay.sln в папке MatlabClay.

Пример 75. Создание PDF-файла «на лету» с возможностью вывода кириллических символов

Представьте себе ситуацию, когда требуется вывести отчетный документ (например, результаты технических вычислений или какую-либо бухгалтерскую отчетную документацию), при этом следует сделать так, чтобы пользователь вашей программы не имел возможности редактировать этот документ. Например, вы хотите, чтобы пользователь не удалял ссылку на вас, как на разработчика программы, чтобы он не вносил исправления в результаты вычислений и пр. Если вы пишете какое-либо веб-приложение, и согласно сценарию интерактивного взаимодействия с пользователем в результате будет сформирована, например, квитанция для оплаты услуг, то становится чрезвычайно важным, чтобы пользователь ни случайно, ни сознательно не смог внести никаких изменения в этот финансовый документ. Подобные задачи помогает решать PDF-формат. Формат PDF гарантирует, что при просмотре файла в сети или выводе на печать формат файла останется неизменным

и данные файла не могут быть легко изменены. Формат PDF стал международным стандартом. PDF-файл легко напечатать и использовать для совместной работы, однако трудно изменить.

Примерами таких ситуаций могут служить резюме, юридические документы, бюллетени и любые другие файлы, предназначенные главным образом для чтения и печати. Для просмотра PDF-файла можно использовать официальную бесплатную программу Adobe Reader, а также веб-браузер, например Internet Explorer или Mozilla Firefox (они открывают PDF-документ на отдельной вкладке браузера). Последний факт оказывается очень важным при создании PDF-файла «на лету» (on the fly) во время диалога пользователя с интерактивной веб-страницей. К сожалению, к моменту написания данной книги Microsoft еще не создала стандартных шаблонов для разработки веб-ориентированных приложений на C++ (хотя такие шаблоны для Visual Basic и Visual C# из линейки продуктов Visual Studio давно существуют). Поэтому приведем пример создания PDF-файла в ходе работы стандартного Windows-приложения.

Решить задачу было бы совсем просто, если бы мы с вами жили в англоязычной стране. В этом случае проблем с выводом кириллицы нет, и программа свелась бы к написанию программного кода из пяти строчек. Такую маленькую программку мы сейчас и напишем, чтобы изложить решение, придерживаясь принципа «от простого к сложному». Для этого запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_1. Поскольку Microsoft не разработала инструментов для генерации PDF, мы воспользуемся объектной библиотекой itextsharp.dll, которую бесплатно распространяет американская компания iText Software на сайте www.itextsharp.com. Эту библиотеку вы можете либо скачать на упомянутом сайте (а также ознакомиться с документацией), либо найти ее в архиве с рассмотренными в книге примерами (скачайте его с сайта издательства «Питер» www.piter.com).

Теперь в текущий проект необходимо добавить ссылку на эту библиотеку. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке Browse (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.11.

Листинг 9.11. Создание простейшего PDF-документа, содержащего английский текст

```
// Создать_PDF_1.cpp: главный файл проекта.
// Данная программа "на лету" генерирует PDF-документ
#include "stdafx.h"
// Добавляем ссылку на библиотеку itextsharp.dll:
// Project ▸ Properties ▸ Add References и на вкладке Browse (Обзор)
// находим файл itextsharp.dll.
using namespace System;
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;
int main(array<System::String ^> ^args)
{
    // В текущем каталоге создаем PDF-документ:
    Document ^ Документ = gcnew Document();
```

```
    PdfWriter::GetInstance(Документ, gcnew System::IO::FileStream(
        "Отчет.pdf", System::IO::FileMode::Create));
    Документ->Open();
    Документ->Add(gcnew Paragraph("Hello ! \n Do you like C++ ?"));
    Документ->Close();
    // PDF-документ можно открыть с помощью Adobe Acrobat,
    // если он установлен:
    System::Diagnostics::Process::Start("Отчет.pdf");
    return 0;
}
```

Как видно из текста программы, вначале создаем PDF-документ класса Document из объектной библиотеки itextsharp.dll. Этот документ будет записан в файл Отчет.pdf после закрытия документа(Close). В документ добавлен абзац (Paragraph) с текстом на английском языке. Последней строчкой вызывается созданный PDF-файл с помощью Adobe Acrobat, если он установлен на вашем компьютере, если нет, то помните, что PDF-файлы может открыть интернет-браузер, например Internet Explorer. Результат работы программы мы не приводим, поскольку он очевиден.

Задача решена, однако нет чувства удовлетворения, поскольку мы пока не научились выводить в PDF-документе русские буквы. Для этого *необходимо задать шрифт с кодировкой Windows 1251*. Как это сделать, рассмотрим в следующем примере. Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_2. Далее, как и в предыдущем случае, добавим к нашему проекту ссылку на объектную библиотеку itextsharp.dll. Затем на вкладке программного кода введем текст из листинга 9.12.

Листинг 9.12. Создание PDF-документа с текстом на русском языке

```
// Создать_PDF.cpp: главный файл проекта.
// Данная программа "на лету" генерирует PDF-документ
#include "stdafx.h"
// Добавляем ссылку на библиотеку itextsharp.dll:
// Project ▸ Properties ▸ Add References и на вкладке Browse (Обзор)
// находим файл itextsharp.dll.
using namespace System;
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;
int main(array<System::String ^> ^args)
{
    // В текущем каталоге создаем PDF-документ:
    Document ^ Документ = gcnew Document();
    PdfWriter::GetInstance(Документ, gcnew System::IO::FileStream(
        "Отчет.pdf", System::IO::FileMode::Create));
    Документ->Open();
    // Базовый шрифт создаем, используя один из шрифтов из папки Windows:
    BaseFont ^ БазовыйШрифт = BaseFont::CreateFont(
        //"C:\\WINDOWS\\Fonts\\comic.ttf", "CP1251", BaseFont::EMBEDDED);
```

продолжение ⇗

Листинг 9.12 (продолжение)

```
        //"C:\\WINDOWS\\Fonts\\times.ttf", "CP1251", BaseFont::EMBEDDED);
        "C:\\WINDOWS\\Fonts\\CONSOLA.TTF", "CP1251", BaseFont::EMBEDDED);
    // Задаем шрифт размером 10 пунктов. Можно задать
    // шрифт Font::ITALIC (наклонный) или Font::BOLD (жирный):
    Font ^ Шрифт = gcnew Font(БазовыйШрифт, 10, Font::NORMAL);
    Документ->Add(gcnew Paragraph(
                    "Здравствуйте !\n Вы увлекаетесь C++ ?", Шрифт));
    Документ->Close();
    // PDF-документ можно открыть с помощью интернет-браузера:
    // System.Diagnostics.Process.Start("IExplore.exe", System.IO.
    //             Directory.GetCurrentDirectory() + "\\Отчет.pdf");
    // PDF-документ можно открыть с помощью Adobe Acrobat,
    // если он установлен:
    System::Diagnostics::Process::Start("Отчет.pdf");
    return 0;
}
```

В этом примере мы вначале программы также создаем PDF-документ, но далее создаем базовый шрифт на основе имеющихся на нашем компьютере шрифтов, которые находятся в папке операционной системы C:\Windows\Fonts. Заметим, что для создания документа минимального объема следует использовать векторную графику и, так называемые, «безопасные» шрифты. Всего имеется 14 таких шрифтов, среди них:

- Times (обычный, курсив, полужирный и полужирный курсив);
- Courier (обычный, наклонный, полужирный и полужирный наклонный);
- Helvetica (обычный, наклонный, полужирный и полужирный наклонный);
- Symbol;
- Zapf Dingbats.

Эти шрифты можно использовать без внедрения в документ — их должны правильно отображать все программы. Любые другие шрифты, которые не были внедрены в документ и отсутствуют в системе, будут отображаться одним из имеющихся, что может стать причиной увеличения или уменьшения числа страниц, количества символов в строке, межстрочного интервала, а также возникновению других явлений, связанных с метрикой шрифта.

В результате работы данной программы программа-просмотрщик PDF-документов Adobe Reader откроет созданный файл (рис. 9.12).

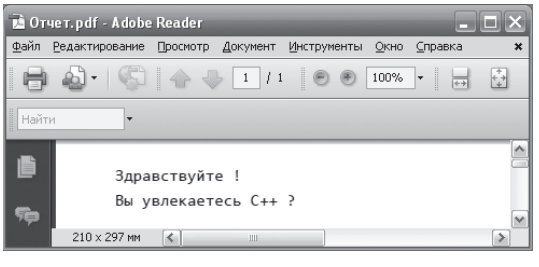

Рис. 9.12. Отображение созданного PDF-файла

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_1 и Создать_PDF_2.

Пример 76. Вывод таблицы в PDF-документ

Довольно часто в PDF-документ требуется вывести какую-либо таблицу. Объектная библиотека itextsharp.dll компании iText Software (см. предыдущий раздел) предоставляет соответствующие классы для решения этой задачи. В данном примере рассмотрим два варианта таблицы. Первый вариант — это вариант «широкой» таблицы с многими колонками на всю страницу формата A4. Пример такой таблицы см. на рис. 9.12. Мы условно назвали ее «широкой», поскольку при взгляде на нее не возникает ощущения пустого места вокруг таблицы. Но если бы такое ощущение возникало, то необходимо было бы перекомпоновать все страницу таким образом, чтобы оптимально заполнялись все пустые места либо другой таблицей, либо каким-нибудь изображением, либо каким-нибудь текстом. А это как раз второй вариант, который мы условно назовем вариантом «узкой» таблицы.

Таким образом, мы формулируем следующую задачу: вывести «широкую» таблицу в PDF-файл. Для ее решения запускаем Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_Табл_1. Теперь в текущий проект необходимо добавить ссылку на объектную библиотеку itextsharp.dll. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке Browse (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.13.

Листинг 9.13. Вывод таблицы в PDF-документ (вариант 1)

```
// Создать_PDF_Табл_1.cpp: главный файл проекта.
// Программа "на лету" создает PDF-файл и записывает в этот
// файл "широкую" таблицу
#include "stdafx.h"
using namespace System;
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;

int main(array<System::String ^> ^args)
{
    // Инициализируем четыре строковых массива:
    array<String^> ^ Номер_п_п = { "N п/п", "1", "2", "3" };
    array<String^> ^ Страны =
                   { "ГОСУДАРСТВА", "Украина", "Россия", "Белоруссия" };
    array<String^> ^ Столицы = { "СТОЛИЦЫ", "Киев", "Москва", "Минск" };
    array<String^> ^ Население =
              { "НАСЕЛЕНИЕ", "2 760 000", "10 380 000", " 1 740 000" };
    // В текущем каталоге создаем PDF-документ:
    Document ^ Документ = gcnew Document();
```

продолжение ⇨

Листинг 9.13 (продолжение)

```
PdfWriter ^ Писатель = PdfWriter::GetInstance(Документ, gcnew System::
      IO::FileStream("ОтчетТабл1.pdf", System::IO::FileMode::Create));
// System::IO::FileMode::Create - если такой файл уже есть, то он
// будет удален, а новый создан
Документ->Open();
// Базовый шрифт создаем из одного из шрифтов из папки Windows:
BaseFont ^ БазовыйШрифт = BaseFont::CreateFont(
    //"C:\\WINDOWS\\Fonts\\comic.ttf", "CP1251", BaseFont::EMBEDDED);
      "C:\\WINDOWS\\Fonts\\times.ttf", "CP1251", BaseFont::EMBEDDED);
    //"C:\\WINDOWS\\Fonts\\CONSOLA.TTF", "CP1251", BaseFont::EMBEDDED);
    // Заказываем шрифт размером 10 пунктов. Можно заказать
    // шрифт Font::ITALIC (наклонный) или Font::BOLD (жирный):
Font ^ Шрифт = gcnew Font(БазовыйШрифт, 12, Font::NORMAL);
// Цвет текста:
Шрифт->Color = BaseColor::BLUE;
// Текст до таблицы:
Документ->Add(gcnew Paragraph("Таблица государств:\n\n", Шрифт));
// Заказываем таблицу с четырьмя колонками:
PdfPTable ^ Табл = gcnew PdfPTable(4);
// Таблицу выровнять в центр:
Табл->HorizontalAlignment = Element::ALIGN_CENTER; // ALIGN_LEFT;
PdfPCell ^ Ячейка = gcnew PdfPCell(gcnew Phrase("Ячейка", Шрифт));
// Пропорции размеров колонок в таблице:
array<float> ^ ШиринаКолонок = { 10.0f, 30.0f, 30.0f, 30.0f };
Табл->SetTotalWidth(ШиринаКолонок);
// Степень заливки ячейки:
Ячейка->GrayFill = 0.92f;
// Высота ячейки:
Ячейка->FixedHeight = 20.0f;
// Цвет границ ячеек:
Ячейка->BorderColor = BaseColor::BLUE;
// Выравнивание содержимого ячеек:
Ячейка->HorizontalAlignment = Element::ALIGN_LEFT; // ALIGN_CENTER;
Ячейка->VerticalAlignment = Element::ALIGN_MIDDLE;
// В цикле создаем каждую ячейку и добавляем ее в таблицу:
for (int i = 0; i <= 3; i++)
{
    Ячейка->Phrase = gcnew Phrase(Номер_п_п[i], Шрифт);
    Табл->AddCell(Ячейка);
    Ячейка->Phrase = gcnew Phrase(Страны[i], Шрифт);
    Табл->AddCell(Ячейка);
    Ячейка->Phrase = gcnew Phrase(Столицы[i], Шрифт);
    Табл->AddCell(Ячейка);
    Ячейка->Phrase = gcnew Phrase(Население[i], Шрифт);
    Табл->AddCell(Ячейка);
}
Документ->Add(Табл);
Документ->Add(gcnew Paragraph("Текст после таблицы", Шрифт));
```

```
    Документ->Close(); Писатель->Close();
    // PDF-документ можно открыть с помощью интернет-браузера:
    System::Diagnostics::Process::Start("IExplore.exe", System::IO::
             Directory::GetCurrentDirectory() + "\\ОтчетТабл1.pdf");
    // PDF-документ можно открыть с помощью Adobe Acrobat,
    // если он установлен:
    // System::Diagnostics::Process::Start("ОтчетТабл.pdf");
    return 0;
}
```

Как видно из программного кода, мы используем те же операторы создания PDF-документа и задания нужного шрифта, что и в примерах из предыдущего раздела. Далее, программируя непосредственно таблицу, мы создаем объект класса PdfPTable. Этот объект содержит ряд свойств и методов для построения таблицы. Составной частью любой таблицы, ее «кирпичиком», является ячейка. Ячейку таблицы задаем, используя класс PdfPCell. В цикле добавляем создаваемые ячейки в искомую таблицу. Последним оператором открываем созданный PDF-документ с помощью браузера Internet Explorer. Внешний вид полученной таблицы показан на рис. 9.13.

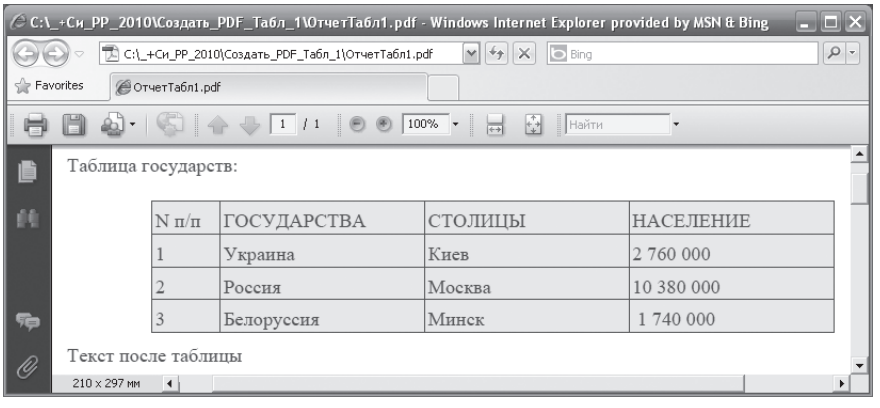

Рис. 9.13. Отображение браузером Internet Explorer созданного PDF-файла

К недостаткам приведенного подхода отнесем невозможность изменения ширины всей таблицы, поскольку таблица всегда будет занимать в ширину всю страницу. Теперь решим эту же задачу по выводу таблицы в PDF-документ иным способом. В этом случае выводить будем, так называемую, «узкую» таблицу с малым числом колонок.

Для решения этой задачи, так же как и в предыдущем случае, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_Табл_2. Необходимым условием работы программы является добавление в текущий проект ссылки на объектную библиотеку itextsharp.dll. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке Browse (Обзор)

найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.14.

Листинг 9.14. Вывод таблицы в PDF-документ (вариант 2)

```
// Создать_PDF_Табл_2.cpp: главный файл проекта.
// Программа "на лету" создает PDF-файл и записывает в этот
// файл "узкую" таблицу с малым числом колонок
#include "stdafx.h"
using namespace System;
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;
int main(array<System::String ^> ^args)
{
    // Инициализируем два строковых массива:
        array<String^> ^ Страны = { "ГОСУДАРСТВА", "Украина", "Россия",
                                                          "Белоруссия" };
        array<String^> ^ Столицы = { "СТОЛИЦЫ", "Киев", "Москва", "Минск" };
    // В текущем каталоге создаем PDF-документ:
    Document ^ Документ = gcnew Document();
    PdfWriter ^ Писатель = PdfWriter::GetInstance(Документ, gcnew System::
        IO::FileStream("ОтчетТабл.pdf", System::IO::FileMode::Create));
    // System::IO::FileMode::Create - если такой файл уже есть, то он
    // будет удален, а новый создан
    Документ->Open();
    // Базовый шрифт создаем из одного из шрифтов из папки Windows:
    BaseFont ^ БазовыйШрифт = BaseFont::CreateFont(
    //"C:\\WINDOWS\\Fonts\\comic.ttf", "CP1251", BaseFont::EMBEDDED);
      "C:\\WINDOWS\\Fonts\\times.ttf", "CP1251", BaseFont::EMBEDDED);
    //"C:\\WINDOWS\\Fonts\\CONSOLA.TTF", "CP1251", BaseFont::EMBEDDED);
    // Заказываем шрифт размером 10 пунктов. Можно заказать
    // шрифт Font.ITALIC (наклонный) или Font.BOLD (жирный):
    Font ^ Шрифт = gcnew Font(БазовыйШрифт, 10, Font::NORMAL, BaseColor::BLUE);
    // или цвет текста отдельно: Шрифт->Color = BaseColor::RED;
    PdfPTable ^ Таблица = gcnew PdfPTable(2);
    PdfPCell ^ Ячейка = gcnew PdfPCell();
    Ячейка->HorizontalAlignment = Element::ALIGN_LEFT;//.ALIGN_CENTER;
    // Две ячейки объединить в одну:
    Ячейка->Colspan = 2;
    // Границы ячейки не показывать:
    Ячейка->Border = 0;
    // Высота ячейки:
    Ячейка->FixedHeight = 16.0f;
    Ячейка->Phrase = gcnew Phrase("Какой-либо текст до таблицы", Шрифт);
    Таблица->AddCell(Ячейка);
    Ячейка->BackgroundColor = BaseColor::LIGHT_GRAY;
    // Не объединять ячейки:
    Ячейка->Colspan = 1;
    // Границы ячеек показывать:
```

```
    Ячейка->Border = 15;
    for (int i = 0; i <= 3; i++)
    {
        Ячейка->Phrase = gcnew Phrase(Страны[i], Шрифт);
        Таблица->AddCell(Ячейка);
        Ячейка->Phrase = gcnew Phrase(Столицы[i], Шрифт);
        Таблица->AddCell(Ячейка);
    }
    Ячейка->Colspan = 2; // две ячейки объединить в одну
    Ячейка->Border = 0;  // не показывать границ ячейки
    Ячейка->Phrase = gcnew Phrase("Некоторый текст после таблицы", Шрифт);
    Ячейка->BackgroundColor = BaseColor::WHITE;
    Таблица->AddCell(Ячейка);
    // Регулируем ширину таблицы:
    Таблица->TotalWidth = Документ->PageSize->Width - 400;
    // Третий и четвертый параметры - это координаты
    // левого верхнего угла таблицы:
    Таблица->WriteSelectedRows(0, -1, 40, Документ->PageSize->Height - 30,
        Писатель->DirectContent);
    Документ->Close(); Писатель->Close();
    // PDF-документ можно открыть с помощью интернет-браузера:
    System::Diagnostics::Process::Start("IExplore.exe", System::IO::
        Directory::GetCurrentDirectory() + "\\ОтчетТабл.pdf");
    return 0;
}
```

В программном коде появилась возможность регулировать ширину таблицы с помощью метода WriteSelectedRows, который выводит таблицу по координатам левого верхнего угла. Такой подход позволяет компоновать страницу так, как это делается при создании HTML-страницы. Очень часто HTML-страницу разбивают на ячейки невидимой таблицы, какие-то ячейки объединяют, при этом размещение данных в такой невидимой таблице становится оптимальным. Результат работы данной программы приведен на рис. 9.14.

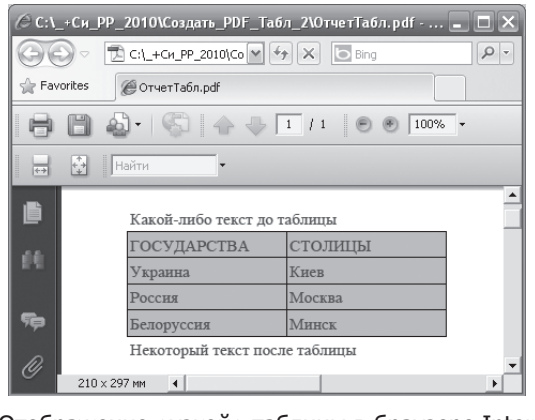

Рис. 9.14. Отображение «узкой» таблицы в браузере Internet Explorer

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_Табл_1 и Создать_PDF_Табл_2.

Пример 77. Вывод графических данных в PDF-документ

При рассмотрении возможности вывода графических данных в PDF-документ сначала покажем, как можно вывести файл растровой графики в PDF-документ. Для этого запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_граф_1. Далее добавим в текущий проект ссылку на объектную библиотеку itextsharp.dll. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке Browse (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.15.

Листинг 9.15. Вывод отображений графических файлов в PDF-документ

```
// Создать_PDF_граф_1.cpp: главный файл проекта.
// Программа выводит в PDF-документ таблицу, состоящую из двух колонок
// и трех строк. Ячейки в первой строке объединены, и в нее мы вывели некоторый
// текст. В две ячейки второй строки мы вывели два изображения. Ячейки
// в третьей строке мы также объединили в одну ячейку и вывели здесь текст.
// Границы первой строки и последней мы не показываем
#include "stdafx.h"
using namespace System;
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;
int main(array<System::String ^> ^args)
{
    // В текущем каталоге создаем PDF-документ:
    Document ^ Документ = gcnew Document();
    PdfWriter ^ Писатель = PdfWriter::GetInstance(Документ, gcnew System::
             IO::FileStream("ТаблГраф.pdf", System::IO::FileMode::Create));
    // System::IO::FileMode::Create - если такой файл уже есть, то он
    // будет удален, а новый создан
    Документ->Open();
    // Базовый шрифт создаем из одного из шрифтов из папки Windows:
    BaseFont ^ БазовыйШрифт = BaseFont::CreateFont(
        "C:\\WINDOWS\\Fonts\\comic.ttf", "CP1251", BaseFont::EMBEDDED);
    // Заказываем шрифт размером 10 пунктов:: Можно заказать
    // шрифт Font::ITALIC (наклонный) или Font::BOLD (жирный):
    Font ^ Шрифт = gcnew Font(БазовыйШрифт, 10, Font::NORMAL, BaseColor::BLUE);
    // или цвет текста отдельно: Шрифт::Color = BaseColor::RED;
    PdfPTable ^ Таблица = gcnew PdfPTable(2);
    PdfPCell ^ Ячейка = gcnew PdfPCell();
    Ячейка->HorizontalAlignment = Element::ALIGN_CENTER;
```

```
    // Две ячейки объединить в одну:
    Ячейка->Colspan = 2;
    // Границы ячейки не показывать:
    Ячейка->Border = 0;
    // Высота ячейки:
    Ячейка->FixedHeight = 16.0F;
    Ячейка->Phrase = gcnew Phrase(
                  "Как же нам найти гармонию между Ж и М ?", Шрифт);
    Таблица->AddCell(Ячейка);
    // Не объединять ячейки:
    Ячейка->Colspan = 1;
    // Границы ячеек показывать:
    Ячейка->Border = 15;
    // Увеличиваем высоту ячеек для изображений:
    Ячейка->FixedHeight = 83.0F;
    // Вставляем изображения в ячейки:
    Ячейка->Image = Image::GetInstance("C:\\g.jpg");
    Таблица->AddCell(Ячейка);
    Ячейка->Image = Image::GetInstance("C:\\m.jpg");
    Таблица->AddCell(Ячейка);
    Ячейка->Colspan = 2; // две ячейки объединить в одну
    Ячейка->Border = 0;  // не показывать границ ячейки
    Ячейка->Phrase = gcnew Phrase(
           "Может, примем друг друга такими, как мы есть ?", Шрифт);
    Ячейка->BackgroundColor = BaseColor::WHITE;
    Таблица->AddCell(Ячейка);
    // Регулируем ширину таблицы:
    Таблица->TotalWidth = Документ->PageSize->Width - 380;
    // Третий и четвертый параметры - это координаты
    // левого верхнего угла таблицы:
    Таблица->WriteSelectedRows(0, -1, 40,
             Документ->PageSize->Height - 30, Писатель->DirectContent);
    Документ->Close(); Писатель->Close();
    // PDF-документ можно открыть с помощью интернет-браузера:
    System::Diagnostics::Process::Start("IExplore.exe", System::IO::
                 Directory::GetCurrentDirectory() + "\\ТаблГраф.pdf");
    return 0;
}
```

Эта программа очень похожа на последнюю программу из предыдущего раздела, только здесь мы во второй строке таблицы выводим в двух ячейках соответственно отображения двух графических файлов. Результат работы программы показан на рис. 9.15.

Довольно часто изображение, подлежащее выводу в PDF-файл, требуется сформировать в процессе работы программы. Следовательно, вам не удастся заранее подготовить графические файлы на все случаи жизни. Поэтому в примере, представленном ниже, мы покажем, как можно в ходе работы программы сформировать изображение методами класса Graphics, а затем вывести это изображение в PDF-документ. Причем мы сделаем это в консольном приложении среды CLR, которое

не содержит объектной библиотеки, поддерживающей класс Graphics, поскольку в консольном приложении нет экранной формы и выводить графику некуда (разве что в растровый файл). Таким образом, изображение мы будем создавать «виртуальное», без «подложки», но выводить его будем в PDF-файл.

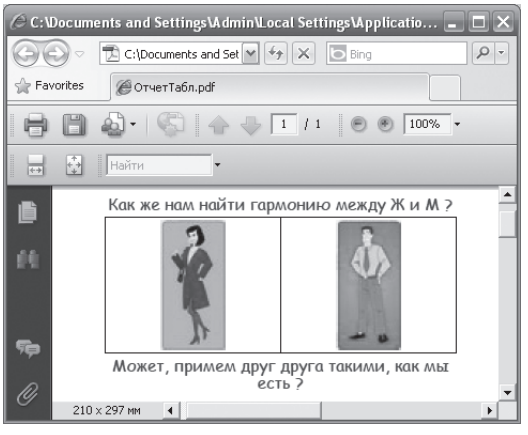

Рис. 9.15. Вывод отображений двух графических файлов

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Выберем имя проекта — Создать_PDF_граф_2. Далее, чтобы получить доступ к графическим возможностям, добавим ссылку на объектную библиотеку System.Drawing.dll. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке .NET дважды щелкнем на компоненте System.Drawing. Также добавим в текущий проект ссылку на объектную библиотеку itextsharp.dll для работы с PDF-документом. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference, затем на вкладке Browse (Обзор) найдем названную библиотеку. Далее на вкладке программного кода введем текст из листинга 9.16.

Листинг 9.16. Вывод изображения, сформированного в консольном приложении, в PDF-файл

```
// Создать_PDF_граф_2.cpp: главный файл проекта.
// Программа формирует изображение методами класса Graphics. Рисуем
// текстовую строку с использованием линейного градиента. Чтобы
// подчеркнуть, что мы создаем именно рисунок, а не просто текст,
// текстовую строку разворачиваем на некоторый угол к горизонту.
// Далее выводим сформированное изображение из оперативной памяти
// в PDF-файл
#include "stdafx.h"
// Здесь добавлены ссылки на две объектные библиотеки:
// System.Drawing.dll и itextsharp.dll
using namespace System;
```

```
// Следует добавить эти две директивы:
using namespace iTextSharp::text;
using namespace iTextSharp::text::pdf;
int main(array<System::String ^> ^args)
{
    // Создаем точечное изображение размером 215 x 35 точек
    // с глубиной цвета 24
    auto Рисунок = gcnew System::Drawing::
         Bitmap(415, 35, System::Drawing::
         Imaging::PixelFormat::Format24bppRgb);
    // Создаем новый объект класса Graphics из изображения Рисунок:
    auto Графика = System::Drawing::Graphics::FromImage(Рисунок);
    auto Точка1 =  System::Drawing::PointF(20.0f, 20.0f);
    auto Точка2 =  System::Drawing::PointF(360.0f, 20.0f);
    auto Градиент = gcnew
        System::Drawing::Drawing2D::LinearGradientBrush(Точка1, Точка2,
        System::Drawing::Color::Yellow, System::Drawing::Color::Red);
    auto Шрифт = gcnew System::Drawing::Font("Times gcnew Roman", 14.0F);
    Графика->Clear(System::Drawing::Color::White);
    // Разворачиваем мир на 356 градусов по часовой стрелке:
    Графика->RotateTransform(356.0F);
    // 20.0f, 20.0f - это координаты левого нижнего угла строки:
    Графика->DrawString("Записываем графику в PDF-документ",
                                          Шрифт, Градиент, 20.0f, 20.0f);
    // Освобождение ресурсов:
    delete Графика; // Эквиваент C#: Графика.Dispose();
    // Создаем PDF-документ:
    Document ^ Документ = gcnew Document();
    PdfWriter ^ Писатель = PdfWriter::GetInstance(Документ, gcnew System::
             IO::FileStream("ТаблГраф.pdf", System::IO::FileMode::Create));
    Документ->Open();
    auto Изо = iTextSharp::text::Image::
                                    GetInstance(Рисунок, BaseColor::WHITE);
    Документ->Add(Изо);
    Документ->Close();
    delete Рисунок; // Эквивалент C#: Рисунок.Dispose();
    System::Diagnostics::Process::Start("ТаблГраф.pdf");
    return 0;
}
```

Как видно из программного кода, мы создаем экземпляр класса Graphics на основе изображения класса Image с заданным размером и форматом. Далее создаем горизонтальный линейный градиент между двумя точками, от желтого цвета до красного. Градиент обеспечивает постепенный переход от одного цвета к другому (рис. 9.16). Сформированный рисунок передаем методом GetInstance с преобразованием в PDF-изображение. Затем обычным способом выводим изображение в PDF-документ (см. рис. 9.16).

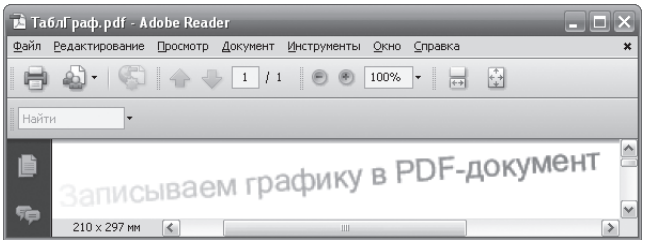

Рис. 9.16. Вывод изображения, сформированного методами класса Graphics

Убедиться в работоспособности обсуждаемых программ можно, открыв соответствующие решения в папках Создать_PDF_граф_1 и Создать_PDF_граф_2.

10 Обработка баз данных с использованием технологии ADO.NET

Пример 78. Создание базы данных SQL Server

В этом разделе мы рассмотрим, как можно создать базу данных SQL Server в среде Visual Studio 2010. В этой простейшей базе данных будет всего одна таблица, содержащая сведения о телефонах наших знакомых, то есть в этой таблице будем иметь всего три колонки: Имя, Фамилия и Номер телефона.

Для этого запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++, зададим имя — БД_SQL_Server. Далее создадим новую базу данных SQL Server. Для этого в пункте меню Project выберем команду Add Existing Item (то есть добавить существующий элемент) и в появившемся одноименном окне увидим файлы текущего проекта. Среди этих файлов имеем файл с расширением sdf, в нашем случае он называется по имени проекта — БД_SQL_Server.sdf. То есть среда Visual Studio 2010 для проектов C++ сразу создает соответствующий файл базы данных sdf. Добавим этот файл в наш проект, для этого дважды щелкнем на изображении файла БД_SQL_Server.sdf. Теперь убедимся, что этот файл появился в окне обозревателя решений, и теперь уже здесь дважды щелкнем на его изображении. При этом откроется окно Server Explorer (Обозреватель баз данных). В этом окне щелчком мыши развернем узел БД_SQL_Server.sdf и выберем Tables (Таблицы). Щелкнем правой кнопкой мыши на пункте Tables, а затем выберем пункт Create Table (Создать таблицу). Откроется окно New Table. Назовем новую таблицу (поле Name) БД телефонов. Заполним структуру таблицы, как показано на рис. 10.1.

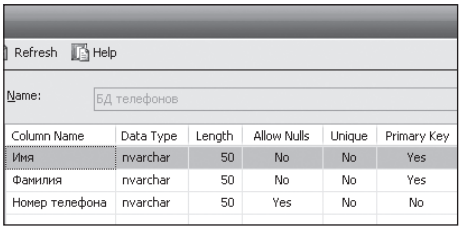

Рис. 10.1. Формирование структуры новой таблицы

Нажмем кнопку OK, чтобы создать таблицу и закрыть окно New Table.

Чтобы исключить *повторяющиеся записи (то есть строки в таблице), следует назначить первичные ключи (или ключевые столбцы). Ключевым столбцом* назначают столбец в таблице, который всегда содержит уникальные (неповторяющиеся в данной таблице) значения. Однако в нашей таблице могут присутствовать люди с одинаковыми именами или одинаковыми фамилиями, то есть в нашей таблице в качестве первичных ключей следует использовать одновременно два столбца: столбец Имя и столбец Фамилия. Представим себе, что наша таблица уже содержит около ста записей, и при попытке ввести вторую строку, содержащую то же самое значение, появляется сообщение об ошибке. Это очень технологично и удобно!

Чтобы добавить первичные ключи в таблицу, в Server Explorer развернем узел Tables. Далее щелкнем правой кнопкой мыши на нашей только что созданной таблице и выберем пункт Edit Table Schema, затем для полей Имя и Фамилия укажем для параметра Allow Nulls (разрешить нулевые значения) значение No, то есть сделаем эти поля (ячейки таблицы) обязательными для заполнения пользователем. Далее для параметра Unique (Являются ли эти поля уникальными?) ответим No, поскольку и имена, и фамилии повторяются. И наконец, назначим колонки Имя и Фамилия *первичными ключами* (Primary Key ▸ Yes). Нажмем кнопку OK для сохранения этих настроек и для закрытия окна Edit Table ▸ БД телефонов.

Теперь добавим данные в таблицу. Для этого в окне Server Explorer щелкнем правой кнопкой мыши на пункте БД телефонов и выберем команду Show Table Data (показать таблицу данных). Откроется окно данных таблицы, как показано на рис. 10.2, но в нашем случае оно пока пустое.

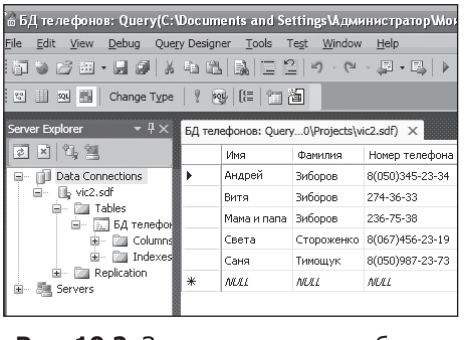

Рис. 10.2. Заполнение ячеек таблицы

Далее заполним данную таблицу, для нашей демонстрационной цели введем пять строчек даных. После ввода в меню File выберем команду Save All для сохранения проекта и базы данных. Теперь убедимся, что в папке проекта имеется файл БД_SQL_Server.sdf. Его можно открыть вне проекта с помощью Microsoft Visual Studio 2010 для редактирования базы данных.

Пример базы данных можно найти в папке БД_SQL_Server.

Пример 79. Отображение таблицы базы данных SQL Server на консоли

Имея базу данных, например базу данных SQL Server, в виде файла БД_SQL_Server.sdf, созданного в предыдущем разделе, покажем, как легко можно вывести таблицу из этой базы, например, на консоль.

Для этой цели для упрощения строки подключения скопируем файл, созданный в предыдущем разделе, в корневой каталог диска C: и запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Зададим имя проекта — БД_SQL_Server_Консоль. Далее для манипулирования данными в базе SQL Server добавим в текущий проект объектную библиотеку System.Data.SqlServerCe.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Data.SqlServerCe. Ниже приведем программный код обсуждаемого консольного приложения.

Листинг 10.1. Чтение всех записей из таблицы БД SQL Server и вывод их на консоль

```
// БД_SQL_Server_Консоль.cpp: главный файл проекта.
// Программа читает все записи из таблицы БД SQL Server (файл *.sdf)
// и выводит их на консоль с помощью объектов Command и DataReader
#include "stdafx.h"
using namespace System;
// Добавим в наш проект объектную библиотеку System.Data.SqlServerCe.dll,
// для этого выберем пункты меню: Project ▸ Properties ▸ Add Reference и на
// вкладке .NET дважды щелкнем по ссылке System.Data.SqlServerCe, а в тексте
// программы добавим директиву:
using namespace System::Data::SqlServerCe;

int main(array<System::String ^> ^args)
{
    // Создаем объект класса Connection:
    SqlCeConnection ^ Подключение = gcnew SqlCeConnection();
    // Передаем ему строку подключения:
    Подключение->ConnectionString = "Data Source=\"C:\\БД_SQL_Server.sdf\"";
    Подключение->Open();
    // Создаем объект класса Command: передавая ему SQL-команду
    SqlCeCommand ^ Команда = gcnew SqlCeCommand();
```

продолжение ⇩

Листинг 10.1 (продолжение)

```
    Команда->Connection = Подключение;
    // Передаем объекту Command SQL-команду:
    Команда->CommandText = "Select * From [БД телефонов]";
    // ДРУГИЕ КОМАНДЫ:
    // Выбрать все записи и сортировать их по колонке "Имя":
    // Команда.CommandText = "Select * From [БД телефонов] order by Имя";
    // Аналогично по колонке "Номер телефона":
    // Команда.CommandText =
    //    "Select * From [БД телефонов] ORDER BY [Номер телефона]";
    // Выполняем SQL-команду и закрываем подключение:
    SqlCeDataReader ^ Читатель = Команда->
        ExecuteReader(System::Data::CommandBehavior::CloseConnection);
    // Задаем строку заголовка консоли:
    Console::Title = "Таблица БД:";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    // Выводим имена полей (колонок) таблицы:
    Console::WriteLine("{0,-11} {1,-11} {2,-15}\n", Читатель->GetName(0),
        Читатель->GetName(1), Читатель->GetName(2));
    while (Читатель->Read() == true)
        // Цикл, пока не будут прочитаны все записи.
        // Читатель.FieldCount - количество полей в строке.
        // Здесь три поля: 0, 1 и 2.
        // Минус прижимает строку влево:
        Console::WriteLine("{0,-11} {1,-11} {2,-15}", Читатель->GetValue(0),
        Читатель->GetValue(1), Читатель->GetValue(2));
    Читатель->Close(); Подключение->Close();
    // Приостановить выполнение программы до нажатия какой-нибудь клавиши:
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, вначале мы создаем объект Подключение класса Connection и передаем ему строку подключения. В строке подключения полный доступ к sdf-файлу заключен в двойные кавычки (\"). Это сделано для того, чтобы корректно читались длинные имена папок и файлов, содержащие пробелы.

Далее создаем объект класса Command и передаем ему простейшую SQL-команду:

```
Select * From [БД телефонов]
```

то есть выбрать все записи из таблицы [БД телефонов]. Название таблицы в SQL-запросе заключено в квадратные скобки из-за пробела в имени таблицы. Заметьте, что в комментарии указаны возможные варианты SQL-запроса: сортировать записи *по колонке* Имя (ORDER BY Имя) и по колонке Номер телефона (ORDER BY [Номер телефона]).

Затем, используя объект класса DataReader, выполняем SQL-команду. Далее в цикле построчно читаем таблицу базы данных. При работе DataReader в памяти

хранится только одна строка (запись) данных. Объект класса DataReader имеет булеву функцию Read, которая возвращает true, если существует следующая строка данных, и false, если такие строки (записи) уже исчерпались. Причем с помощью DataReader невозможно заранее узнать количество записей в таблице.

Результат работы программы показан на рис. 10.3.

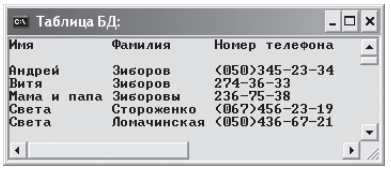

Рис. 10.3. Чтение таблицы базы данных SQL Server на консоль

Убедиться в работоспособности данной программы можно, открыв соответствующий sln-файл решения в папке БД_SQL_Server_Консоль.

Создание базы данных в среде MS Access

Вначале создадим базу данных (БД) vic.mdb средствами Access пакета MS Office. Поясню сразу, что реальный положительный эффект при решении какой-либо задачи информатизации с использованием баз данных можно ощутить, когда количество записей (то есть количество строк в таблице) превышает 100 тысяч. В этом случае очень важным (решающим) фактором оказывается скорость выборки. Однако для примера работы мы будем оперировать совсем маленькой БД.

Чтобы вы смогли повторить мои действия, создадим базу данных телефонов ваших контактов. Структура (поля) таблицы будет следующей: Номер п/п, ФИО и Номер телефона. Для этого запускаем на компьютере офисное приложение Microsoft Access, далее в меню Создать выбираем команду Новая база данных (или нажимаем комбинацию клавиш Ctrl+N), задаем папку для размещения БД и имя файла — vic.mdb. Затем в появившемся окне vic: база данных выбираем команду Создание таблицы в режиме конструктора. Далее задаем три поля (то есть три колонки в будущей таблице): имя первого поля — Номер п/п, тип данных — Счетчик; следующее имя поля — ФИО, тип данных — Текстовый; третье имя поля — Номер телефона, тип данных — Текстовый.

При сохранении структуры таблицы появится запрос на имя таблицы, укажем: БД телефонов. В БД может быть несколько таблиц, а данную таблицу мы назвали именно так. Заметьте, что при работе в обычных приложениях если вы решили отказаться от внесенных изменений в документе, то его просто закрывают без сохранения. Однако при работе с БД все изменения сохраняются на диске без нашего ведома. Запись на диск происходит напрямую, минуя операционную систему.

Далее с помощью двойного щелчка в пределах созданной таблицы приступаем к ее заполнению. В нашем примере в таблице всего семь записей (рис. 10.4).

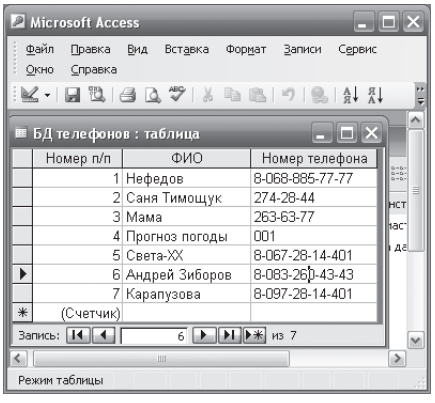

Рис. 10.4. Заполнение таблицы базы данных в среде MS Access

Теперь закроем БД Access и откроем созданную нами таблицу БД vic.mdb в среде Visual Studio 2010.

Пример 80. Редактирование таблицы базы данных MS Access в среде Visual Studio без написания программного кода

Запускаем Visual Studio 2010, однако заказывать новый проект мы не будем. Сейчас наша цель — открыть созданную нами базу данных в среде Visual Studio. Для этого выбираем пункт меню View ▸ Server Explorer (также можно воспользоваться комбинацией клавиш Ctrl+Alt+S). Затем, щелкнув правой кнопкой мыши на значке Data Connections, выбираем пункт Add Connection и указываем в качестве источника данных (Data source) Microsoft Access Database File (OLE DB), нажав кнопку Change. Далее с помощью кнопки Browse задаем путь и имя БД, например C:\vic.mdb. Теперь проверяем подключение — кнопка Test Connection. Успешное подключение выдаст сообщение, представленное на рис. 10.5.

Проверка подключения выполнена, щелкаем на кнопке OK. Теперь в окне Server Explorer указателем мыши раскрываем узлы, символизирующие базу данных, таблицы, поля в таблице.

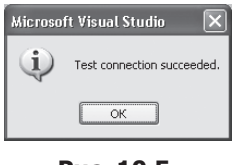

Рис. 10.5. Тестирование подключения

Далее щелкнем правой кнопкой мыши на узле БД телефонов и выбираем команду Retrieve Data. В результате в правом окне получим содержимое этой таблицы, как показано на рис. 10.6. Здесь данную таблицу мы можем редактировать, то есть изменять содержимое любой записи (Update), добавлять новые записи (Insert), то есть новые строки в таблицу, удалять записи (Delete). Кроме того, щелкая правой кнопкой мыши в пределах таблицы и выбирая в контекстном меню

пункты Pane ▸ SQL, можно осуществлять SQL-запросы к базе данных, в том числе и наиболее часто используемый запрос Select.

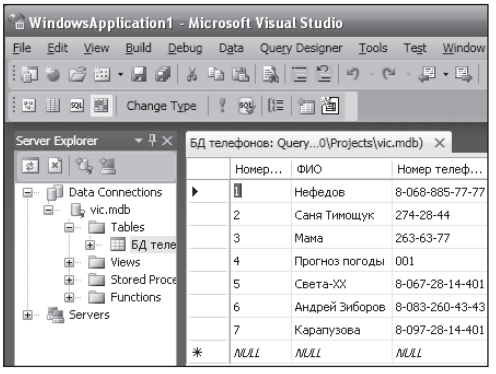

Рис. 10.6. Редактирование таблицы базы данных в среде Visual Studio

Пример 81. Чтение всех записей из таблицы БД MS Access на консоль с помощью объектов классов Command и DataReader

Напишем программу, которая при минимальном количестве строк программного кода выводит на экран все записи (то есть все строки) таблицы базы данных. При этом воспользуемся наиболее современной технологией ADO.NET. Нам понадобятся четыре объекта. Объект Connection обеспечивает соединение с базой данных. Объект Command обеспечивает привязку SQL-выражения к соединению с базой данных. А с помощью объектов DataSet и DataReaders можно просмотреть результаты запроса.

Мы рассмотрим четыре основных действия над базой данных: Select (выборка записей из таблицы БД), Insert (вставка записей), Update (модификация записей в таблице БД), Delete (удаление некоторых записей из таблицы).

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Зададим имя данного проекта БДDataReader. Нам нужно вывести на экран самым простым способом таблицу. Если мы будем выводить ее при помощи функции MessageBox::Show, то ровных колонок в окне MessageBox::Show мы не получим, поскольку буквы, используемые в этой функции, имеют разную ширину. Обычно в таком случае для вывода таблиц используют шрифт Courier New или Consolas, но объект MessageBox не содержит возможностей смены шрифта. Поэтому мы пойдем по самому короткому пути и выведем таблицу из базы данных на консоль, то есть на черный экран DOS. В этом случае у нас реализуется моноширинный шрифт, в котором все символы имеют одинаковую ширину. Например, буква «Ш» и символ «.» (точка) имеют

одинаковую ширину, следовательно, колонки в построенных таблицах будут ровными.

Теперь на вкладке программного кода напишем текст из листинга 10.2.

Листинг 10.2. Чтение всех записей из таблицы БД MS Access и вывод их на консоль

```
// БДDataReader1.cpp: главный файл проекта.
// Программа читает все записи из таблицы БД MS Access и выводит их
// на консоль с помощью объектов Command и DataReader
#include "stdafx.h"
using namespace System;
// Добавляем эту директиву для краткости выражений:
using namespace System::Data::OleDb;
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем объект класса Connection
    auto Подключение = gcnew OleDbConnection();
    // Передаем ему строку подключения:
    Подключение->ConnectionString = "Data Source=\"C:\\vic.mdb\";User " +
        "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";";
    Подключение->Open();
    // Создаем объект класса Command:
    auto Команда = gcnew OleDbCommand();
    Команда->Connection = Подключение;
    // Передаем ему SQL-команду:
    Команда->CommandText = "Select * From [БД телефонов]";
    // Выбрать все записи и сортировать их по колонке "ФИО":
    // Команда->CommandText = "Select * From [БД телефонов] order by ФИО";
    // Аналогично по колонке "Номер п/п":
    // Команда->CommandText =
    //                "Select * From [БД телефонов] ORDER BY 'Номер п/п'";
    // Выполняем SQL-команду:
    OleDbDataReader ^ Читатель = Команда->
        ExecuteReader(System::Data::CommandBehavior::CloseConnection);
    Console::WriteLine("Таблица БД:\n");
    while (Читатель->Read() == true)
        // Цикл, пока не будут прочитаны все записи.
        // Читатель->FieldCount - количество полей в строке.
        // Здесь три поля: 0, 1 и 2.
        // Минус прижимает строку влево:
        Console::WriteLine("{0,-3} {1,-15} {2,-15}", Читатель->GetValue(0),
        Читатель->GetValue(1), Читатель->GetValue(2));
    Читатель->Close(); Подключение->Close();
    // Приостановить выполнение программы до нажатия какой-нибудь клавиши:
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, вначале мы создаем объект Подключение класса Connection и передаем ему строку подключения. В строке подключения полный доступ к mdb-файлу заключен в двойные кавычки. Это сделано для того, чтобы корректно читались длинные имена папок и файлов, содержащие пробелы.

Далее создаем объект класса Command и передаем ему простейшую SQL-команду:

```
Select * From [БД телефонов]
```

то есть выбрать *все записи* из таблицы [БД телефонов]. Название таблицы в SQL-запросе заключено в квадратные скобки из-за пробела в имени таблицы. Заметьте, что в комментарии указаны возможные варианты SQL-запроса: *сортировать* записи *по колонке* ФИО (ORDER BY ФИО) и по колонке Номер п/п (ORDER BY 'Номер п/п').

Затем, используя объект класса DataReader, выполняем SQL-команду. Далее в цикле построчно читаем таблицу базы данных. При работе DataReader в памяти хранится только одна строка (запись) данных. Объект класса DataReader имеет Булеву функцию Read, которая возвращает true, если существует следующая строка данных, и false, если такие строки (записи) уже исчерпались. Причем с помощью DataReader невозможно заранее узнать количество записей в таблице.

Результат работы программы показан на рис. 10.7.

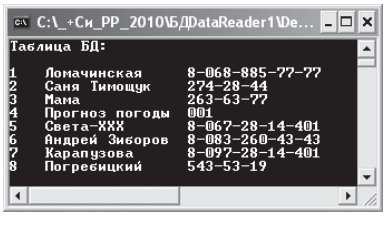

Рис. 10.7. Отображение таблицы базы данных на консоли

Таким образом, мы получили простейшую программу для просмотра таблицы базы данных. С ее помощью можно *только просматривать данные*, но нельзя их редактировать.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке БДDataReader.

Пример 82. Создание базы данных MS Access в программном коде

Создадим программу, которая во время своей работы создает базу данных Access, то есть файл new_BD.mdb. Эта база данных будет пустой, то есть она не будет содержать ни одной таблицы. Наполнять базу данных таблицами можно впоследствии как из программного кода Visual C++ 2010, так и используя MS Access. Заметим, что в этом примере технология ADO.NET *не использована*.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Для добавления в наш проект DLL-библиотеки ADOX выполним следующее: в пункте меню Project выберем команду Properties ▸ Add Reference, затем на вкладке COM дважды щелкнем по ссылке Microsoft ADO Ext. 2.8 for DDL and Security, добавив тем самым эту библиотеку в текущий проект. Убедиться в том, что теперь существует ссылка на эту библиотеку, можно в окне Properties. Здесь, щелкнув на узле References, увидим ветвь ADOX. Теперь мы можем ссылаться на это имя в программном коде. Далее вводим программный код, приведенный в листинге 10.3.

Листинг 10.3. Создание БД во время работы программы

```
// БДСоздание.cpp: главный файл проекта.
// Программа создает базу данных MS Access, то есть файл new_BD.mdb.
// Эта база данных будет пустой, то есть не будет содержать ни одной таблицы.
// Наполнять базу данных таблицами можно будет впоследствии
// как из программного кода C++ 2010, так и используя MS Access.
// В этом примере технология ADO.NET не использована
#include "stdafx.h"
// Добавим в наш проект библиотеку ADOX: Project ▸ Add Reference, и на
// вкладке COM выбираем Microsoft ADO Ext. 2.8 for DDL and Security
using namespace System;
// Для вызова MessageBox выберем следующие пункты меню:
// Project ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке
// System.Windows.Forms.dll, а в тексте программы добавим директиву:
using namespace System::Windows::Forms;
int main(array<System::String ^> ^args)
{
    ADOX::Catalog ^ Каталог = gcnew ADOX::Catalog();
    try
    {
        Каталог->Create("Provider=Microsoft.Jet." +
            "OLEDB.4.0;Data Source=C:\\new_BD.mdb");
        MessageBox::Show("База данных C:\\new_BD.mdb успешно создана");
    }
    catch (System::Runtime::InteropServices::COMException ^ Ситуация)
    { MessageBox::Show(Ситуация->Message); }
    finally
    { Каталог = nullptr; }
    return 0;
}
```

Чтобы был доступен объект MessageBox для вывода сообщений, добавим в проект еще одну dll-библиотеку. Для этого, как и в предыдущем случае, укажем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке System.Windows.Forms.dll, а в тексте программы добавим директиву:

```
Using namespace System::Windows::Forms;
```

Ключевое слово using используется для импортирования пространства имен, которое содержит класс MessageBox.

Программа работает следующим образом: создаем экземпляр класса ADOX::catalog, одна из его функций Create способна создавать базу данных, если на ее вход подать *строку подключения*. Заметим, что в строку подключения входит также и полный путь к создаваемой БД. Функция Create заключена в блоки try...catch, которые обрабатывают *исключительные ситуации*. После запуска этого приложения получим сообщение о создании базы данных (рис. 10.8).

Если запустить наше приложение еще раз, то мы получим сообщение о том, что такая база данных уже существует (рис. 10.9), поскольку БД new_BD.mdb только что создана.

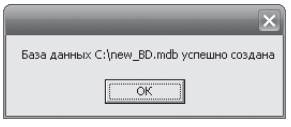

Рис. 10.8. Сообщение о создании базы данных

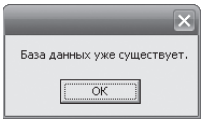

Рис. 10.9. База данных существует

Данное сообщение генерировалось обработчиком исключительной ситуации.

Программный код этой программы можно посмотреть, открыв решение БДСоздание.sln в папке БДСоздание.

Пример 83. Запись структуры таблицы в пустую базу данных MS Access. Программная реализация подключения к БД

Теперь здесь и далее мы будем использовать *только* самую современную технологию ADO.NET. Создадим программу, которая записывает структуру таблицы, то есть «шапку» таблицы, в существующую БД. В этой БД может не быть ни одной таблицы, то есть БД может быть пустой. Либо в БД могут уже быть таблицы, но название новой таблицы должно быть уникальным.

Создадим базу данных new_BD.mdb в корневом каталоге логического диска C:, используя MS Access или программным путем, как это было показано в предыдущем разделе. Никакие таблицы в базе данных создавать не будем, то есть наша БД будет пустой. Теперь запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Затем напишем программный код, представленный в листинге 10.4.

Листинг 10.4. Создание таблицы в БД MS Access

```
// БдСоздТаблицы.cpp: главный файл проекта.
// Программа записывает структуру таблицы в пустую базу данных MS Access.
// Программная реализация подключения к БД. В этой БД может еще не быть
// ни одной таблицы, то есть БД может быть пустой. Либо в БД могут уже быть
```

продолжение ⇘

Листинг 10.4 (продолжение)

```
// таблицы, но название новой таблицы должно быть уникальным
#include "stdafx.h"
using namespace System;
// Для вызова MessageBox выберем следующие пункты меню:
// Project ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке
// System.Windows.Forms.dll, а в тексте программы добавим директиву:
using namespace System::Windows::Forms;
// Добавляем эту директиву для более краткого обращения к классам
// обработки данных:
using namespace System::Data::OleDb;
int main(array<System::String ^> ^args)
{ // ЗАПИСЬ СТРУКТУРЫ ТАБЛИЦЫ В ПУСТУЮ БД:
    // Создание экземпляра объекта Connection с указанием строки
    // подключения:
    auto Подключение = gcnew OleDbConnection(
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\new_BD.mdb");
    // Открытие подключения:
    Подключение->Open();
    // Создание экземпляра объекта класса Command
    // с заданием SQL-запроса:
    auto Команда = gcnew OleDbCommand("CREATE TABLE [" +
        "БД телефонов] ([Номер п/п] counter, [ФИО] ch" +
        "ar(20), [Номер телефона] char(20))", Подключение);
    try // Выполнение команды SQL:
    {
        Команда->ExecuteNonQuery();
        MessageBox::Show(
            "Структура таблицы 'БД телефонов' записана в пустую БД");
    }
    catch (Exception ^ Ситуация)
    { MessageBox::Show(Ситуация->Message); }
    Подключение->Close();
    return 0;
}
```

Для работы функции MessageBox::Show следует в текущий проект добавить ссылку на DLL-библиотеку. Для этого в пункте меню Project ▸ Properties выберем команду Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Windows.Forms.dll.

Как видно из текста программы, вначале мы создаем экземпляр класса Connection с указанием строки подключения, это позволит нам управлять этой строкой *программно*. Далее создаем экземпляр класса Command с заданием SQL-запроса. В этом запросе создаем (CREATE) новую таблицу с именем БД телефонов с тремя полями: Номер п/п типа счетчик (counter), ФИО и Номер телефона. Имя таблицы и имена полей заключены в квадратные скобки, поскольку они содержат пробелы.

Чтобы выполнить эту SQL-команду, вызываем метод ExecuteNonQuery, который заключим в блоки try...catch для обработки исключительных ситуаций. Если SQL-запрос благополучно выполнился, то получаем сообщение: «Структура таблицы 'БД телефонов' записана в пустую БД». А если, например, таблица с таким именем

уже имеется в базе данных, то управление передается блоку catch (перехват исключительной ситуации), и мы получаем сообщение о том, что такая таблица базы данных уже существует (рис. 10.10).

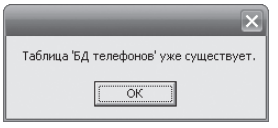

Рис. 10.10. Сообщение о существовании таблицы

Таким образом, в данной программе сначала организовано подключение Connection к БД через строку подключения и открытие подключения Open. Затем задание SQL-запроса в объекте Command и выполнение запроса функцией ExecuteNonQuery. Если связывание данных организовать программно, то мы получим большую гибкость для тех случаев, когда, например, на стадии разработки неизвестно заранее, где (на каком диске, в какой папке) будет находиться БД.

Убедиться в работоспособности программы можно, открыв решение БдСоздТаблицы.sln в папке БдСоздТаблицы.

Пример 84. Добавление записей в таблицу базы данных MS Access

Совсем маленькую программу из предыдущего раздела можно использовать для выполнения *любого запроса*, обращенного к базе данных. Например, модифицируем всего лишь одну строчку программного кода программы из предыдущего примера для добавления новой записи в таблицу БД. Для этого при создании экземпляра объекта Command зададим SQL-запрос на вставку (INSERT) новой записи в таблицу БД.

Заметим, что в SQL-запросе мы сознательно обратились к таблице по имени [бд телефонов], то есть со строчной буквы, хотя следовало бы с прописной. Дело в том, что в именах таблиц следует *точно* указывать регистр символа, поскольку их поиск ведется с учетом регистра (case-sensitive search). Однако это не обязательно при наличии только одной таблицы с таким именем, поскольку при этом используется поиск без учета регистра (case-insensitive search).

Свойству Connection объекта класса Command следует дать ссылку на объект класса Connection:

```
Команда->Connection = Подключение;
```

Причем для добавления записи в таблицу БД такая ссылка обязательна в отличие от предыдущего примера, где мы создавали новую таблицу в существующей БД.

Программный код будет выглядеть так, как представлено в листинге 10.5.

Листинг 10.5. Добавление записей в таблицу базы данных MS Access

```
// БдДобавлЗаписи.cpp: главный файл проекта.
// Программа добавляет запись в таблицу базы данных MS Access. Для этого
// при создании экземпляра объекта Command задаем SQL-запрос
// на вставку (Insert) новой записи в таблицу базы данных
#include "stdafx.h"
using namespace System;
```

продолжение ⇩

Листинг 10.5 (продолжение)

```
// Для вызова MessageBox выберем следующие пункты меню:
// Project ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке
// System.Windows.Forms.dll, а в тексте программы добавим директиву:
using namespace System::Windows::Forms;
// Добавляем эту директиву для более краткого обращения к классам
// обработки данных:
using namespace System::Data::OleDb;
// ДОБАВЛЕНИЕ ЗАПИСИ В ТАБЛИЦУ БД:
int main(array<System::String ^> ^args)
{
        // Создание экземпляра объекта Connection
        // с указанием строки подключения:
        auto Подключение = gcnew OleDbConnection(
           "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\new_BD.mdb");
        // Открытие подключения:
        Подключение->Open();
        // Создание экземпляра объекта Command с заданием SQL-запроса:
        auto Команда = gcnew OleDbCommand(
                "INSERT INTO [бд телефонов] (" +
                "Фио, [номер телефона]) VALUES ('Света-Х', '521-61-41')");
        // Для добавления записи в таблицу БД эта команда обязательна:
        Команда->Connection = Подключение;
        // Выполнение команды SQL:
        Команда->ExecuteNonQuery();
        MessageBox::Show("В таблицу 'БД телефонов' добавлена запись");
        Подключение->Close();
    return 0;
}
```

Зачастую, отлаживая программный код на Visual Studio C++, при работе с БД появляется необходимость проверки работы программы, например требуется узнать, создалась ли таблица в БД, добавилась ли запись в таблице БД, правильно ли сформирован SQL-запрос. Не обязательно запускать MS Access, чтобы выполнить SQL-запрос или проверить правильность его синтаксиса. Это можно сделать в среде Visual Studio. Для этого в пункте меню View выбираем команду Other Windows ▸ Server Explorer (комбинация клавиш Ctrl+Alt+S), далее в списке подключений указываем полный путь к нужной БД. Затем, щелкая правой кнопкой мыши на значке нужной таблицы, в контекстном меню выбираем пункт Retrieve Data. При этом в панели инструментов (Toolbar) появляется значок SQL, после щелчка по этому значку (или нажатия комбинации клавиш Ctrl+3) получим окно SQL-запроса. В этом окне мы можем задавать SQL-запрос, а затем, например, щелкая правой кнопкой мыши, либо проверять его синтаксис, либо выполнять.

Убедиться в работоспособности программы можно, открыв решение БдДобавлЗаписи.sln в папке БдДобавлЗаписи.

Пример 85. Чтение всех записей из таблицы базы данных с помощью объектов классов Command, DataReader и элемента управления DataGridView

Покажем, как легко и «малой кровью» можно вывести таблицу базы данных на элемент управления DataGridView (сетка данных, то есть таблица данных) с использованием объектов классов Command и DataReader из предыдущей программы.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox добавим в проектируемую форму элемент управления DataGridView и растянем его на всю форму. На вкладке Form1.h напишем программный код, представленный в листинге 10.6.

Листинг 10.6. Чтение всех записей из таблицы БД

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа читает все записи из таблицы базы данных с помощью объектов
        // Command, DataReader на элемент управления DataGridView (сетка данных)
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 // Создаем объект Connection и передаем ему строку подключения:
                 auto Подключение = gcnew OleDb::OleDbConnection(
                     "Data Source=\"C:\\vic.mdb\";User " +
                     "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
                 Подключение->Open();
                 // Создаем объект Command, передавая ему SQL-команду
                 auto Команда = gcnew OleDb::
                     OleDbCommand("Select * From [БД телефонов]", Подключение);
                 // Выполняем SQL-команду
                 auto Читатель = Команда-> ExecuteReader();
                 // (CommandBehavior.CloseConnection)
                 auto Таблица = gcnew DataTable();
                 // Заполнение "шапки" таблицы
                 Таблица->Columns->Add(Читатель->GetName(0));
                 Таблица->Columns->Add(Читатель->GetName(1));
                 Таблица->Columns->Add(Читатель->GetName(2));
                 while (Читатель->Read() == true)
```

продолжение ⇩

Листинг 10.6 (продолжение)

```
                    // Заполнение клеток (ячеек) таблицы
                    Таблица->Rows->Add(Читатель->GetValue(0),
                    Читатель->GetValue(1), Читатель->GetValue(2));
                // Здесь три поля: 0, 1 и 2
                Читатель->Close(); Подключение->Close();
                dataGridView1->DataSource = Таблица;
            }
    };
}
```

Как мы можем видеть, эта программа очень похожа на предыдущую. После выполнения SQL-команды создаем объект DataTable, который в конце программного кода задаем как источник (DataSource) для сетки данных dataGridView1. Заполняем «шапку» таблицы, то есть названия колонок, методом Add.

Далее, как и в предыдущей программе, в цикле While заполняем ячейки таблицы. Фрагмент работы программы показан на рис. 10.11.

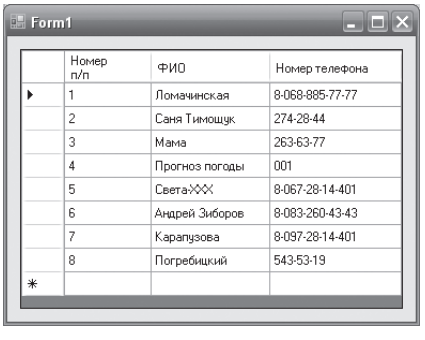

Рис. 10.11. Отображение таблицы базы данных на элементе DataGridView

В этой таблице мы можем сортировать записи по любой из колонок, щелкая мышью на названиях соответствующих колонок. Можем редактировать (изменять) содержимое ячеек, но в базу данных эти изменения не попадут (сохранения не произойдет).

Одно из ключевых преимуществ использования объекта DataReader — это его быстродействие и использование небольшого количества оперативной памяти. Однако применение циклического считывания данных сводит эти преимущества на нет.

Убедиться в работоспособности программы можно, открыв решение БдReaderGridView.sln в папке БдReaderGridView.

Пример 86. Чтение данных из БД в сетку данных DataGridView с использованием объектов классов Command, Adapter и DataSet

Рассмотрим пример чтения таблицы с помощью объекта Adapter из базы данных посредством выбора нужных данных и передачи их объекту DataSet. Очень удобно прочитать таблицу, записанную в DataSet, используя элемент управления DataGridView (сетка данных, то есть таблица данных), указав в качестве источника данных для сетки DataGridView объект класса DataSet.

Поскольку нам нужен элемент управления DataGridView, мы создаем новый проект с экранной формой. Для этого, как обычно, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox добавляем в форму элемент управления DataGridView и растягиваем его на всю форму, как показано на рис. 10.12.

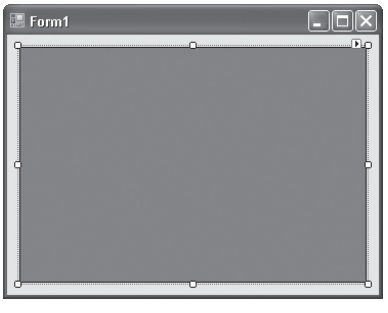

Рис. 10.12. Проектирование экранной формы

Далее пишем программный код, представленный в листинге 10.7.

Листинг 10.7. Чтение данных из БД в сетку данных DataGridView

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа читает из БД таблицу в сетку данных DataGridView
        // с использованием объектов класса Command, Adapter и DataSet
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                this->Text = "Чтение таблицы из БД:";
                auto Подключение = gcnew OleDb::OleDbConnection(
                    "Data Source=\"C:\\vic.mdb\";User " +
```

продолжение ⇨

Листинг 10.7 (продолжение)

```
                    "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
                Подключение->Open();
                auto Команда = gcnew OleDb::OleDbCommand(
                    "Select * From [БД телефонов]", Подключение);
                // Выбираем из таблицы только те записи, поле ФИО которых
                // начинается на букву "М":
                // auto Команда =
                //   gcnew OleDb::OleDbCommand("SELECT * FRO" +
                //     "M [БД телефонов] WHERE (фио LIKE 'м%')", Подключение);
                // Создаем объект класса Adapter и выполняем SQL-запрос
                auto Адаптер = gcnew OleDb::OleDbDataAdapter(Команда);
                // Создаем объект класса DataSet
                auto НаборДанных = gcnew DataSet();
                // Заполняем DataSet результатом SQL-запроса
                Адаптер->Fill(НаборДанных, "БД телефонов");
                // Содержимое DataSet в виде строки XML для отладки:
                auto СтрокаXML = НаборДанных->GetXml();
                // Указываем источник данных для сетки данных:
                dataGridView1->DataSource = НаборДанных;
                // Указываем имя таблицы в наборе данных:
                dataGridView1->DataMember = "БД телефонов";
                Подключение->Close();
            }
    };
}
```

Как видно из текста программы, вначале мы создали объект класса Connection, передавая строку подключения. Затем, создавая объект класса Command, задаем SQL-команду выбора всех записей из таблицы БД телефонов. Здесь мы можем задать любую SQL-команду. В комментарии приведен пример такой команды, которая содержит SELECT и LIKE, в которой предлагается выбрать из таблицы БД телефонов только записи, в которых поле ФИО начинается на «м». Оператор LIKE используется для поиска по шаблону (pattern matching) вместе с символами универсальной подстановки (метасимволами) «звездочка» (*) и «знак вопроса» (?). Строка шаблона заключена в апострофы. Заметим также, что большинство баз данных использует символ % вместо значка * в LIKE-выражениях.

Далее при создании объекта класса Adapter выполняем SQL-команду и при выполнении метода Fill заполняем объект класса DataSet таблицей, полученной в результате SQL-запроса. Затем указываем в качестве источника данных для сетки данных dataGridView1 объект класса DataSet. Этого оказывается достаточным для вывода на экран результатов SQL-запроса (рис. 10.13).

Так же как и при использовании объекта класса DataReader в предыдущем примере, в полученной таблице мы можем сортировать записи по любой из колонок. Можем редактировать (изменять) содержимое ячеек, но в базу данных эти изменения не попадут (сохранения не произойдет).

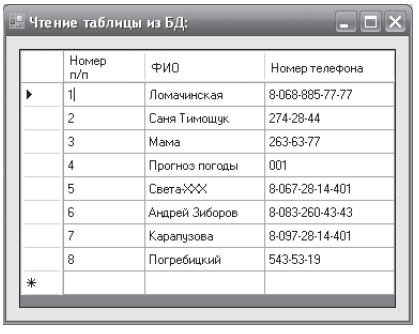

Рис. 10.13. Вывод результата SQL-запроса

Заметим, что здесь с помощью визуального проектирования выполнено только перетаскивание в форму сетки данных DataGridView, остальное сделано программно, что обеспечивает большую гибкость программы.

Убедиться в работоспособности программы можно, открыв решение БдАдаптерGridView.sln в папке БдАдаптерGridView.

Пример 87. Обновление записей в таблице базы данных MS Access

Одним из основных четырех действий над данными в БД (Select, Insert, Update и Delete) является модификация (Update, обновление) данных. Автор поставил задачу написать маленькую программу для обновления записей в таблице базы данных, но с большим удобством (гибкостью) управления программным кодом.

Рассматриваемая в данном примере программа имеет форму, сетку данных DataGridView, в которую *из базы данных считывается таблица при нажатии кнопки* Читать из БД. Пользователь имеет возможность *редактировать* данные в этой таблице, после чего, при нажатии кнопки Сохранить в БД, данные в базе данных будут *модифицированы*, то есть заменены новыми.

Для написания этой программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Из панели Toolbox добавляем в форму элемент управления DataGridView и две командные кнопки. Программный код представлен в листинге 10.8.

Листинг 10.8. Обновление записей в таблице базы данных MS Access

```
        // ..............
        // Программный код, расположенный выше, создан средой Visual Studio
        // автоматически, поэтому автором не приводится
        this->ResumeLayout(false);
    }
```

продолжение ⇨

Листинг 10.8 (продолжение)

```
#pragma endregion
        // Программа обновляет записи (Update) в таблице базы данных MS Access
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Объявляем эти переменные вне всех процедур, чтобы
        // они были видны из любой из процедур:
        DataSet ^ НаборДанных;
        OleDb::OleDbDataAdapter ^ Адаптер;
        OleDb::OleDbConnection ^ Подключение;
        OleDb::OleDbCommand ^ Команда;
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
             {
                НаборДанных = gcnew DataSet();
                Подключение = gcnew OleDb::
                    OleDbConnection( // Строка подключения:
                    "Data Source=\"C:\\vic.mdb\";User " +
                    "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
                Команда = gcnew OleDb::OleDbCommand();
                button1->Text = "Читать из БД"; button1->TabIndex = 0;
                button2->Text = "Сохранить в БД";
             }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             {
                // Читать из БД:
                if (Подключение->State ==
                                ConnectionState::Closed) Подключение->Open();
                Адаптер = gcnew OleDb::OleDbDataAdapter(
                                "Select * From [БД телефонов]", Подключение);
                // Заполняем DataSet результатом SQL-запроса
                Адаптер->Fill(НаборДанных, "БД телефонов");
                // Содержимое DataSet в виде строки XML для отладки:
                String ^ СтрокаXML = НаборДанных->GetXml();
                // Указываем источник данных для сетки данных:
                dataGridView1->DataSource = НаборДанных;
                // Указываем имя таблицы в наборе данных:
                dataGridView1->DataMember = "БД телефонов";
                Подключение->Close();
             }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
             { // Сохранить в базе данных
                Команда->CommandText = "UPDATE [БД телефонов] SET [Но" +
                    "мер телефона] = ?, ФИО = ?  WHERE ([Номер п/п] = ?)";
                // Имя, тип и длина параметра
                Команда->Parameters->Add("Номер телефона",
                           OleDb::OleDbType::VarWChar, 50, "Номер телефона");
                Команда->Parameters->Add(
                               "ФИО", OleDb::OleDbType::VarWChar, 50, "ФИО");
```

```
                Команда->Parameters->Add
                   (gcnew OleDb::OleDbParameter("Original_Номер_п_п",
                    OleDb::OleDbType::Integer,
                    0, System::Data::ParameterDirection::
                    Input, false, (Byte)0, (Byte)0, "Номер п/п",
                    System::Data::DataRowVersion::Original, nullptr));
                Адаптер->UpdateCommand = Команда;
                Команда->Connection = Подключение;
                try
                {  // Update возвращает количество измененных строк
                    int kol = Адаптер->Update(НаборДанных, «БД телефонов»);
                    MessageBox::Show("Обновлено " + kol + " записей");
                }
                catch (Exception ^ Ситуация)
                { MessageBox::Show(Ситуация->Message, "Недоразумение"); }
            }
    };
}
```

Как видно из кода, мы имеем три процедуры обработки событий: загрузки формы, «щелчок на кнопке Читать из БД» и «щелчок на кнопке Сохранить в БД». Чтобы объекты классов DataSet, DataAdapter, Connection и Command были видны в этих трех процедурах, объявляем эти объекты внешними внутри класса Form1.

При программировании чтения из базы данных вначале с помощью SQL-запроса мы выбрали все записи из таблицы (Select * From [БД телефонов]) и с помощью объекта класса Adapter поместили в набор данных DataSet. А затем указали объект класса DataSet в качестве источника (DataSource) для сетки данных dataGridView1. Фрагмент работы программы после чтения из базы данных представлен на рис. 10.14.

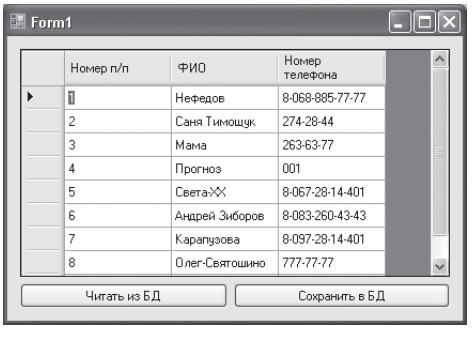

Рис. 10.14. Фрагмент работы программы обновления данных

Для нас будет представлять интерес программирование модификации записей базы данных. Эта возможность реализуется при обработке события «щелчок мышью на кнопке Сохранить в БД». Здесь свойству CommandText присвоено значение текста SQL-запроса. В качестве заменителей параметров используются вопросительные знаки. В данном SQL-запросе имеют место три вопросительных знака. Им соответствуют три параметра, которые должны указываться строго в порядке

следования вопросительных знаков. Эти параметры задаем с использованием метода Parameters->Add. Здесь указываем имя поля (например, «Номер телефона»), тип, длину параметра и значение по умолчанию. Заметим, что третий параметр («Номер п/п») задается как новый, поскольку он не должен подлежать редактированию со стороны пользователя, а будет устанавливаться автоматически, независимо от действий пользователя.

Далее в блоке try...catch вызываем непосредственно метод Update, который возвращает количество (kol) *обновленных* записей. В случае неудачного обновления обрабатывается исключительная ситуация Exception: объект Exception обеспечивает соответствующее сообщение об ошибке.

Убедиться в работоспособности программы можно, открыв решение БдUpdate.sln в папке БдUpdate.

Пример 88. Удаление записей из таблицы базы данных с использованием SQL-запроса и объекта класса Command

Можно также удалять записи (строки из таблицы БД), формируя в программном коде соответствующий SQL-запрос, передаваемый в объект класса Command. Именно объект Command обеспечивает привязку SQL-выражения к соединению с базой данных. Напишем самый простой пример такой программы.

В данном случае экранная форма нам не нужна, поэтому выберем, как и в некоторых предыдущих примерах, шаблон консольного приложения. Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Console Application CLR. Чтобы иметь доступ к функции MessageBox::Show, добавим к проекту ссылку на динамическую библиотеку Forms.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке на библиотеку System.Windows.Forms.dll. Отметим, что при этом в окне Referenses среди ссылок References появится соответствующая этой библиотеке ссылка.

Далее напишем программный код из листинга 10.9.

Листинг 10.9. Удаление записей из таблицы БД

```
// БдУдаленЗаписи.cpp: главный файл проекта.
// Программа удаляет запись из таблицы БД с использованием SQL-запроса
// и объекта класса Command
#include "stdafx.h"
using namespace System;
// Для вызова MessageBox добавим в наш проект пункты меню:
// Project ▸ Add Reference и на вкладке .NET дважды щелкнем по ссылке
// System.Windows.Forms.dll, а в тексте программы добавим директиву:
using namespace System::Windows::Forms;
int main(array<System::String ^> ^args)
```

```
{
        // Создаем объект Connection и передаем ему строку подключения
       auto Подключение = gcnew Data::OleDb::
              OleDbConnection( // Строка подключения:
                 "Data Source=\"C:\\vic.mdb\";User " +
                 "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
        Подключение->Open();
        // Создаем объект класса Command, передавая ему SQL-команду
        auto Команда = gcnew Data::OleDb::OleDbCommand(
                    "Delete * From [БД телефонов] Where " +
                    "ФИО Like 'Vi%'", Подключение);
        // Выполнение команды SQL
        int i = Команда->ExecuteNonQuery();
        // i - количество удаленных записей
        if (i > 0) MessageBox::Show(
            "Записи, содержащие в поле ФИО фрагмент 'Vi*', удалены");
        if (i == 0) MessageBox::Show(
            "Запись, содержащая в поле ФИО фрагмент 'Vi*', не найдена");
        Подключение->Close();
    return 0;
}
```

Здесь при создании объекта класса Command задан SQL-запрос на удаление (Delete) всех записей, содержащий в поле ФИО фрагмент текста Vi*, причем строчные и прописные буквы являются равнозначными, то есть будут удалены записи, содержащие Vi*, vi*, VI* и прочие комбинации. Таким образом, поиск записей ведется без учета регистра (case-insensitive search).

Замечу, что здесь для выполнения команды SQL использован метод ExecuteNonQuery. Он возвращает в переменную i количество удаленных записей. Если i = 0, значит, записей с таким контекстом не найдено, и ни одна запись не удалена.

Убедиться в работоспособности программы можно, открыв решение БдУдаленЗаписи.sln в папке БдУдаленЗаписи.

Использование технологии LINQ

11

Технология LINQ (Language Integrated Query) предназначена для обработки (для организации запросов и преобразований) практически любого источника данных, начиная от массивов, файлов, строк, коллекций объектов .NET Framework, баз данных SQL Server, наборов данных ADO.NET (DataSet) и XML-документов. LINQ упрощает ситуацию, предлагая стандартные шаблоны для работы с данными в различных видах источников и различных форматов. Стандартные шаблоны включают в себя основные операции запросов LINQ: фильтрация, упорядочение, группировка, соединение, выбор (проецирование), статистическая обработка. По форме синтаксис языка LINQ очень похож на язык запросов SQL.

Следует отметить, что ту технологию LINQ-запросов со стандартными операторами запросов (from, where, select и др.), которую имеют языки Visual Basic и C#, язык MS Visual C++ для среды .NET *не поддерживает*. Технология LINQ-запросов успешно работает в Visual Basic и C#, начиная с версии Visual Studio 2008. Вероятно, у Microsoft до внедрения такой же технологии и в C++/CLI 2010 еще «не дошли руки». Во время компиляции выражения LINQ-запроса преобразуются в вызовы методов. Это как раз то, что понятно среде CLR .NET — вызовы методов. Поэтому мы можем на языке MS Visual C++ для среды .NET обращаться непосредственно к этим методам технологии LINQ. Таким образом, в данной главе мы продемонстрируем использование синтаксиса LINQ-методов вместо синтаксиса LINQ-запросов.

Пример 89. Манипулирование массивом данных методами класса Linq::Enumerable

Продемонстрируем возможность выборки из массива необходимых элементов и совершения с ними некоторых преобразований с помощью методов класса Linq::Enumerable. Конкретная задача состоит в следующем: мы имеем строковый массив имен людей, из него извлекаем имена длиной шесть символов, записывая их в список (коллекцию). При этом избавляемся от дублирования элементов в списке, сортируем их в алфавитном порядке и переводим все символы в верхний регистр.

Для решения этой задачи запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR, укажем имя Name — LinqМассив. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.1 приведен программный код обсуждаемого консольного приложения.

Листинг 11.1. Извлечение данных из строкового массива

```
// LinqМассив.cpp: главный файл проекта.
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим для краткости выражений:
using namespace System::Linq;
bool Предикат(String ^ S)
{
    // Если число букв равно шести, то возвращаем true:
    bool A = false;
    if (S->Length == 6) A = true;
    return A;
}
String ^ Предикат2(String ^ S)
{
    // Переводим строку в верхний регистр:
    S = S->ToUpper();
    return S;
}
int main(array<System::String ^> ^args)
{
    // Программа в строковом массиве имен выбирает имена, состоящие из
    // шести букв. В списке выбранных имен сортируем их в алфавитном порядке,
    // все строки переводим в верхний регистр и избавляемся от дублирования
    // имен. При этом вместо синтаксиса LINQ-запросов (как в VB и C#)
    // используем синтаксис LINQ-методов
    Console::Title = "Фильтрация массива методами класса Linq::Enumerable";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    auto СтрокаИмен =
        "Витя Лариса Лариса Лена Андрей Женя \n" +
        "Александр Лариса Виктор Света Оксана Наташа";
    // Из строки имен получаем массив имен, задавая в качестве
    // разделителя подстрок символ пробела:
    auto Имена = СтрокаИмен->Split(' ');
    Console::WriteLine("ЗАДАЧА 1. В списке имен:\n\n" + СтрокаИмен);
    Console::WriteLine(
        "\nвыбираем имена с количеством букв равным\n" +
```

продолжение ⇩

Листинг 11.1 (продолжение)

```
        "шести, сортируем полученный список,       \n" +
        "переводим в верхний регистр и избавляемся \n" +
        "от дублирования имен:                     \n");
    // Из массива имен получаем список имен, длина которых - шесть букв:
    auto Запрос = Enumerable::Where<String^>(Имена, gcnew
                                Func<String ^,bool>(Предикат));
    // Сортируем полученный список в алфавитном порядке:
    Запрос = Enumerable::OrderBy(Запрос, gcnew
                                Func<String ^,String ^>(Предикат2));
    // Переводим в верхний регистр:
    Запрос = Enumerable::Select(Запрос, gcnew
                                Func<String ^,String ^>(Предикат2));
    // Избавляемся от дублирования имен:
    Запрос = Enumerable::Distinct(Запрос);
    // Выводим на консоль результаты:
    for each (String ^ x in Запрос)
                Console::WriteLine("{0} ", x);
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, после присвоения массиву Имена начальных значений, используя LINQ-функцию Where, создаем запрос, который предусматривает выбор из массива Имена строк длиной (Length) ровно шесть символов. Последнее условие задается в отдельной процедуре Предикат. Запись выбранных имен осуществляется в список Запрос. Для сортировки списка в алфавитном порядке имен используем LINQ-функцию OrderBy, а для перевода в верхний регистр — Select. Далее для удаления повторяющихся имен в списке используем функцию Distinct. Цикл for each выводит на консоль результат манипуляций с массивом и списком.

На рис. 11.1 приведен фрагмент работы программы.

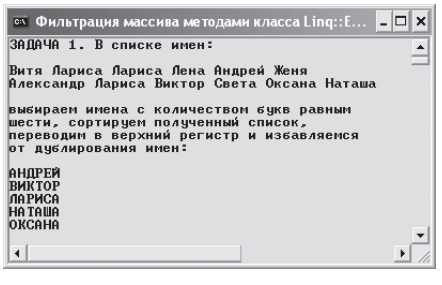

Рис. 11.1. Манипулирование массивом данных с помощью технологии LINQ

Убедиться в работоспособности программы можно, открыв решение LinqМассив.sln в папке LinqМассив.

Пример 90. Запрос к коллекции (списку) данных методами LINQ

В некоторых случаях хранение данных в коллекции (скажем, в списке типа List) может оказаться более эффективным, чем в массиве. Например, если число элементов в массиве изменяется часто или нельзя заранее определить максимальное количество необходимых элементов, то при использовании коллекции можно добиться большей производительности. Но если размер массива не изменяется или изменяется довольно редко, то использование массива приведет к большей эффективности. Как всегда, производительность в большей степени зависит от конкретного приложения. Как советуют в документации MSDN, зачастую стоит потратить время на испытание и массива, и коллекции, чтобы выбрать наиболее практичный и эффективный вариант.

В этом разделе решим две типичные задачи. Первая состоит в следующем: мы имеем список сотрудников предприятия, в котором есть следующие поля: имя сотрудника, его возраст и отметка, курит ли он. Из этого списка следует выбрать только некурящих сотрудников для повышения им зарплаты. Кроме того, из исходного списка выберем также сотрудников, чей возраст превышает 33 года («возраст Христа»). Таким образом, мы должны получить два новых списка и вывести на консоль фамилии из обоих списков.

Для решения этой задачи запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR, укажем имя Name — LinqСписок1. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.2 приведем программный код данного приложения.

Листинг 11.2. Извлечение данных из списка (вариант 1)

```
// LinqСписок1.cpp: главный файл проекта.
// Программа из списка сотрудников некоторого предприятия выбирает
// в отдельный список только некурящих сотрудников
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим эти пространства имен для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
// Обьявляем структуру (или класс):
value struct Сотрудник
// или value class Сотрудник
{
public: String ^ Имя;
public: int Возраст;
```

продолжение ⇨

Листинг 11.2 (продолжение)

```
public: bool КуритЛи;
};
bool Предикат(Сотрудник S)
{
    // Если сотрудник не курит, то заносим его в список некурящих:
    bool A = false;
    if (S.КуритЛи == false) A = true;
    return A;
    // Можно было бы записать короче:
    // return !S.КуритЛи;
    // но более запутанно
}
bool Предикат2(Сотрудник S)
{
    // Если сотруднику больше 33 лет, то заносим его в список "взрослых":
    bool A = false;
    if (S.Возраст > 33) A = true;
    return A;
    // Можно было бы записать короче:
    // return S.Возраст > 33;
    // но более запутанно
}
int main(array<System::String ^> ^args)
{
    Console::Title = "Фильтрация списка методами класса Linq::Enumerable";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    // Создаем список сотрудников:
    List<Сотрудник> ^ Сотрудники = gcnew List<Сотрудник>();
    // Инициализация списка сотрудников (их для упрощения всего четыре):
    Сотрудник Сотрудник1 = {"Зиборов Виктор", 45, false};
    Сотрудники->Add(Сотрудник1);
              Сотрудник Сотрудник2 = {"Еременко Татьяна", 22, true};
              Сотрудники->Add(Сотрудник2);
    Сотрудник Сотрудник3 = {"Стороженко Светлана", 32, false};
    Сотрудники->Add(Сотрудник3);
              Сотрудник Сотрудник4 = {"Тимощук Александр", 43, true};
              Сотрудники->Add(Сотрудник4);
    // Из списка сотрудников получаем новый список некурящих сотрудников:
    auto Некурящие = Enumerable::Where<Сотрудник>(Сотрудники, gcnew
                                        Func<Сотрудник ,bool>(Предикат));
    // Выводим полученный список на консоль:
    Console::WriteLine("Некурящие сотрудники:\n");
    for each (Сотрудник x in Некурящие)
        Console::WriteLine("{0} ", x.Имя);
    // Из списка сотрудников получаем новый список "взрослых" сотрудников,
    // возраст которых превышает 33 года:
```

```
    auto Взрослые = Enumerable::Where<Сотрудник>(Сотрудники, gcnew
                                        Func<Сотрудник ,bool>(Предикат2));
    // Выводим полученный список на консоль:
    Console::WriteLine("\n\"Взрослые\" сотрудники:\n");
    for each (Сотрудник x in Взрослые)
        Console::WriteLine("{0} ", x.Имя);
    Console::ReadKey();
    return 0;
}
```

Можно заметить, что значительная часть программного кода выполняет инициализацию списка сотрудников предприятия. Для упрощения кода мы рассматриваем список из четырех сотрудников. Этот фрагмент программы мы представили в «шахматном» порядке для большей структурированности программного кода. При создании списка объявлена структура Сотрудник, которая содержит три поля: Имя, Возраст и булеву переменную КуритЛи. Как и для случая манипуляций с массивом, для фильтрации исходного списка данных используем метод Where класса Enumerable. При этом условие выбора задаем в отдельной процедуре Предикат. В результате получаем новый список «Некурящие». Полученный список выводим на консоль, используя цикл for each (рис. 11.2). Аналогично получаем второй список «Взрослые», здесь условие выбора для этого списка задаем в другой процедуре Предикат2.

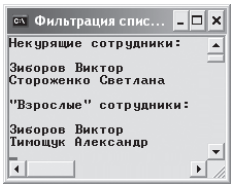

Рис. 11.2. Выборка данных из списка с помощью технологии LINQ

Вторая задача немного сложнее: требуется создать список студентов факультета, содержащий фамилию студента и массив полученных им текущих оценок, то есть массив оценок должен быть «вложен» в список студентов. Из списка студентов необходимо выбрать тех, кто имеет в перечне своих оценок хотя бы одну двойку, а также упорядочить полученный список в алфавитном порядке по фамилиям.

Как и в предыдущем случае, запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqСписок2. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core.

В листинге 11.3 приведен программный код данного приложения.

Листинг 11.3. Извлечение данных из списка (вариант 2)

```
// LinqСписок.cpp: главный файл проекта.
// Имеем список студентов с их фамилиями и текущими оценками. Программа
// фильтрует этот список для получения нового списка студентов, у которых
// среди текущих оценок имеется хотя бы одна двойка
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
```

продолжение ⇘

Листинг 11.3 (продолжение)

```
// Добавим для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
// Объявляем класс:
value class Студент
    // или структуру: value struct Студент
{
public: String ^ Фамилия;
public: array<int> ^ Оценки;
};
bool Предикат(Студент S)
{
    bool A = false;
    // Если хотя бы одна оценка - двойка, то выход из цикла,
    // а студента объявляем двоечником: A = true:
    for each(int i in S.Оценки)
        if (i <= 2) { A = true; break; }
        return A;
}
String ^ Предикат2(Студент S)
{
    // Сортировка в алфавитном порядке по фамилии:
    return S.Фамилия;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем массив объектов типа Студент:
    array<Студент> ^ Массив =
    { "Зиборов",     gcnew array<int>{5, 4, 4, 5},
      "Стороженко",  gcnew array<int>{3, 3, 2, 4},
      "Ломачинская", gcnew array<int>{3, 4, 4, 5},
      "Погребицкий", gcnew array<int>{2, 4, 3, 2},
      "Тимощук",     gcnew array<int>{2, 3, 4, 3},
    };
    // Создаем список студентов из массива:
    List<Студент> ^ Студенты = Enumerable::ToList<Студент>(Массив);
    // Запрос на студентов-двоечников:
    auto Двоечники = Enumerable::Where<Студент>(Студенты, gcnew
                                    Func<Студент ,bool>(Предикат));
    // Сортируем полученный список студентов в алфавитном порядке:
    Двоечники = Enumerable::OrderBy(Двоечники, gcnew
                                Func<Студент, String ^>(Предикат2));
    // Вывод результата запроса на консоль:
    for each (Студент ^ x in Двоечники)
        Console::WriteLine("{0} ", x->Фамилия);
    Console::ReadKey();
    return 0;
}
```

Как видно, в начале программы объявляем класс (можно структуру) Студент, который имеет поля: фамилию студента и массив оценок. Далее заполняем вспомогательный массив, из которого создаем список студентов. Затем строим запрос методами технологии LINQ сначала на выявление из списка студентов, имеющих неудовлетворительные оценки (метод Where), а затем сортируем их по фамилии в алфавитном порядке (метод OrderBy). Результат работы программы приводим на рис. 11.3.

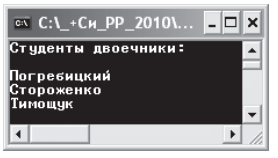

Рис. 11.3. Выборка двоечников из списка студентов

Убедиться в работоспособности этих двух программ можно, открыв соответствующие решения в папках LinqСписок1 и LinqСписок2.

Пример 91. Группировка данных методом GroupBy

Приведем примеры группировки данных в соответствии с заданной функцией селектора ключа методом GroupBy. В первом примере будем группировать данные в списке типа List. Каждая запись в этом списке представляет собой два поля: название месяца в году и количество дней в этом месяце. В этой задаче требуется получить два производных списка, в первый список будут входить месяцы, количество дней в которых равно 31, а во второй — остальные месяцы.

Как и в предыдущем случае, для решения этой задачи запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqМесяцы. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.4 приведен программный код данного приложения.

Листинг 11.4. Группировка элементов списка данных методом GroupBy

```
// Linq_Месяцы.cpp: главный файл проекта.
// Программа создает список месяцев в году с указанием их названия
// и количеством дней в месяце. Затем создает запрос на группировку этих
// данных, то есть следует создать одну группу месяцев, в которых
// содержится 31 день, и другую группу, в которую входят прочие месяцы.
// Результат этого запроса программа выводит на консоль
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
value struct Месяц
```

продолжение ⇩

Листинг 11.4 (продолжение)

```
{
public: String ^ Название;
public: int Дней;
};
bool Предикат(Месяц t)
{
    bool A = false;
    if (t.Дней == 31) A = true;
    return A;
}
Месяц Предикат2(Месяц t)
{
    return t;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем список месяцев - их 12:
    auto Месяцы = gcnew List<Месяц>(12);
    auto Месяц1 = Месяц();
    // Инициализация списка:
    Месяц1.Название = "Январь";   Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Февраль";  Месяц1.Дней = 28; Месяцы->Add(Месяц1);
    Месяц1.Название = "Март";     Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Апрель";   Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Май";      Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Июнь";     Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Июль";     Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Август";   Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Сентябрь"; Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Октябрь";  Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Ноябрь";   Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Декабрь";  Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    // Запрос на группировку данных, записанных в список Месяцы. Этот список
    // делим на две группы: одна группа включает в себя месяцы, содержащие
    // 31 день, вторая - прочие месяцы:
    auto Запрос = Enumerable::GroupBy<Месяц, bool, Месяц>(Месяцы,
                          gcnew Func<Месяц, bool>(Предикат),
                          gcnew Func<Месяц, Месяц>(Предикат2));
    // Выводим результаты запроса на консоль:
    Console::WriteLine("Две группы месяцев: \n");
    // Цикл по группам:
    for each (IGrouping<bool,Месяц> ^ Группа in Запрос)
    {
        if(Группа->Key == true)
            Console::WriteLine("Месяцы, содержащие 31 день: \n");
        else Console::WriteLine("\nПрочие месяцы: \n");
```

```
        // Цикл по месяцам в группе:
        for each (Месяц М in Группа)
            Console::WriteLine("{0} - {1}", М.Название, М.Дней);
    }
    // Ждем от пользователя нажатия какой-либо клавиши:
    Console::ReadKey();
    return 0;
}
```

В программном коде мы вначале создаем список типа List. В угловых скобках указываем, что каждая запись в списке будет представлять собой структуру Месяц, состоящую из двух полей строкового и целого типов. После инициализации списка организуем запрос на получение двух производных групп методом GroupBy. Результат запроса попадает в переменную Запрос. Для вывода обеих групп на консоль используем два вложенных цикла for each. Внешний цикл выполняется по группам, а второй выводит непосредственно название каждого месяца и соответственное количество дней на консоль. Результат работы программы представлен на рис. 11.4.

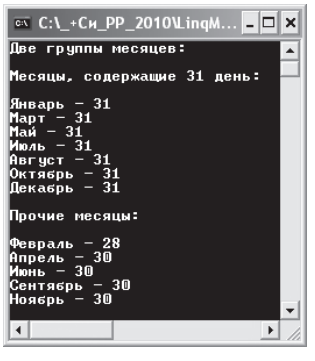

Рис. 11.4. Группировка списка по количеству дней в месяцах

Вторая задача, которую мы решим в данном разделе, манипулирует со словарем данных типа Dictionary. В словаре данных будем записывать информацию о некоторых товарах. В реальной ситуации данные о каждом товаре займут большое количество полей, то есть колонок в соответствующих таблицах. В нашем примере каждый товар будет иметь только название и цену. Чтобы ориентироваться в большом множестве товаров, это множество разбивают на группы. В нашей задаче мы наши товары, записанные в словарь данных, разделим на две ценовые группы, соответственно дороже и дешевле 90 рублей. Это деление выполним, используя LINQ-технологию.

Как и в предыдущем случае, запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqЦеныНаПродукты. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET

дважды щелкнем на ссылке System.Core. В листинге 11.5 приведен программный код данного приложения.

Листинг 11.5. Группировка элементов словаря данных Dictionary методом GroupBy

```
// LinqЦеныНаПродукты.cpp: главный файл проекта.
// Программа создает словарь данных Dictionary продуктов питания. В словаре -
// всего два поля: наименование товара и его цена. Методом GroupBy из данных
// словаря создаем две ценовых группы: товары дороже и дешевле 90 рублей.
// Обе группы товаров выводим на консоль
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
//
// Добавим эти пространства имен для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
bool Предикат(KeyValuePair<String^, float> t)
{
    bool A = false;
    if (t.Value > 90) A = true;
    return A;
}
KeyValuePair<String^, float> Предикат2(KeyValuePair<String^, float> t)
{
    return t;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем и заполняем словарь данных:
    auto Продукты = gcnew Dictionary<String^, float>();
    Продукты->Add("Творог",  115.50F);
    Продукты->Add("Хлеб",    18.75F );
    Продукты->Add("Печенье", 93.75F );
    Продукты->Add("Чай",     76.25F );
    Продукты->Add("Мясо",    150.00F);
    Продукты->Add("Гречка",  62.50F );
    // Запрос на группировку данных, записанных в словарь Продукты:
    auto Запрос = Enumerable::GroupBy<
        KeyValuePair<String^, float>,
        bool, KeyValuePair<String^, float>>(Продукты,
        gcnew Func<KeyValuePair<String^, float>, bool>(Предикат),
        gcnew Func<KeyValuePair<String^, float>,
                   KeyValuePair<String^, float>>(Предикат2));
    // Выводим результаты запроса на консоль:
    Console::WriteLine("Две ценовых группы: \n");
    for each (IGrouping<bool,KeyValuePair<String^, float>> ^
                                                    ЦеноваяГруппа in Запрос)
```

```
    {
        if(ЦеноваяГруппа->Key == true)
            Console::WriteLine("Товары дороже 90 руб: \n");
        else Console::WriteLine("\nТовары дешевле 90 руб: \n");
        for each (KeyValuePair<String^, float> Прод in ЦеноваяГруппа)
            Console::WriteLine("{0} - {1}",Прод.Key, Прод.Value);
    }
    Console::ReadKey();
    return 0;
}
```

В начале программы формируем словарь продуктов питания. Далее организуем запрос на группировку данных на две группы методом GroupBy. Запрос получился несколько громоздким из-за использования системной структуры KeyValuePair. Однако только так мы можем использовать метод GroupBy при обращении к словарю данных. Вывод результата запроса организуем с помощью двух вложенных циклов for each. Внешний цикл выполняется по ценовым группам, а внутренний — по товарам.

Фрагмент работы программы показан на рис. 11.5.

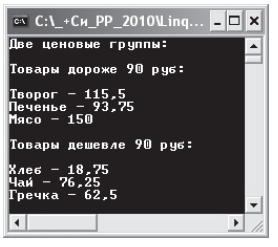

Рис. 11.5. Группировка словаря данных методом GroupBy

Убедиться в работоспособности программ, рассмотренных в данном разделе, можно, открыв соответствующие решения в папках LinqМесяцы и LinqЦеныНаПродукты.

Пример 92. Создание XML-документа методами классов пространства имен System::Xml::Linq

Итак, кроме пространства имен System::Xml, содержащего классы для обработки XML-документов, в Visual Studio 2010 существует пространство имен System::Xml::Linq, содержащее классы, которые позволяют легко и эффективно изменять документы XML, а также организовывать поиск данных. В данном примере представим сведения о наших повседневных телефонных контактах в виде XML-документа, используя класс XDocument пространства System::Xml::Linq. Эти данные будет иметь интуитивно понятную структуру: имя контакта, домашний и мобильный телефоны. Создав такой XML-документ и получив соответствующий XML-файл, его будет удобно просматривать в MS Excel в виде таблицы, содержащей три столбца: имя контакта, домашний и мобильный телефоны. Попутно обсудим структуру XML-документа.

Итак, запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выбирем шаблон Console Application CLR, укажем имя Name — LinqСоздатьXML-документ. Далее, чтобы иметь доступ к пространству имен System::Xml::Linq, добавляем в текущий проект объектную библиотеку System.XML.

Linq.dll. Для этого выбираем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкаем на ссылке System.XML.Linq. В листинге 11.6 приведен программный код обсуждаемого консольного приложения.

Листинг 11.6. Создание XML-документа, представляющего телефонную книгу

```
// LinqСоздатьXML-документ.cpp: главный файл проекта.
// Программа создает типичный XML-документ. С ее помощью можно разобраться
// в структуре XML-документа. В комментариях приведена терминология содержимого
// XML-документа: корневой элемент, вложенные элементы, имя элемента и его
// значение, а также атрибуты элемента, их имена и значения. XML-документ
// представляет телефонную книгу, содержащую имя контакта, номер домашнего
// телефона, а также мобильного. Программа после создания XML-документа
// отображает его на консоли, а также записывает его в файл. Если этот файл
// открыть с помощью MS Excel, то мы получим таблицу из трех столбцов
#include "stdafx.h"
using namespace System;
// Следует добавить эту директиву:
using namespace System::Xml::Linq;
int main(array<System::String ^> ^args)
{
    Console::Title = "Корневой элемент XML-документа";
    // Создаем новый XML-документ:
    XDocument ^ XMLдокумент = gcnew XDocument(
        // Комментарий в XML-документе:
        gcnew XComment(
        "Телефонная_книга - это корневой элемент XML-документа:"),
        gcnew XElement("Телефонная_книга", // - имя корневого элемента

        gcnew XComment(
        "Элемент СТРОКА содержит атрибут Контакт и два вложенных элемента"),
        gcnew XElement("СТРОКА", // - имя (Name) элемента
        gcnew XAttribute("Контакт", "Олег"),
        gcnew XElement("Домашний_телефон", "236-23-67"), // - имя элемента
                                                          // и его значение
        gcnew XElement("Мобильный_телефон", "+7(495)625-31-43")),

        gcnew XComment("Атрибут Контакт имеет значение 'Прогноз погоды':"),
        gcnew XElement("СТРОКА",
        gcnew XAttribute("Контакт", "Прогноз погоды"), // - атрибут элемента
                                                       // СТРОКА
        gcnew XElement("Домашний_телефон", "001"),
        gcnew XElement("Мобильный_телефон", "")), // - имя элемента
                                                  // и его зачение (Value)
        gcnew XComment(
        "Поскольку каждый элемент Контакт имеет атрибут и два вложенных=>"),
        gcnew XElement("СТРОКА",
        gcnew XAttribute("Контакт", "Борис Григорьевич"),
        // Здесь имя атрибута - "Контакт"
        gcnew XElement("Домашний_телефон", "402-12-45"),
```

```
        gcnew XElement("Мобильный_телефон", "+7(495)536-79-94")),

        gcnew XComment(
        "=> элемента, в MS Excel отобразится таблица с тремя колонками"),
        gcnew XElement("СТРОКА",
        gcnew XAttribute("Контакт", "Света"),  // - значение атрибута - Света
        gcnew XElement("Домашний_телефон", ""),
        gcnew XElement("Мобильный_телефон", "+7(495)615-24-41")))
        );
    // Сохранить XML-документ:
    XMLдокумент->Save("C:\\Зиборов.XML");
    Console::WriteLine(XMLдокумент);
    Console::ReadKey();
    return 0;
}
```

Чтобы понять текст программы, рассмотрим структуру полученного XML-файла, а для этого откроем этот файл с помощью Internet Explorer (рис. 11.6).

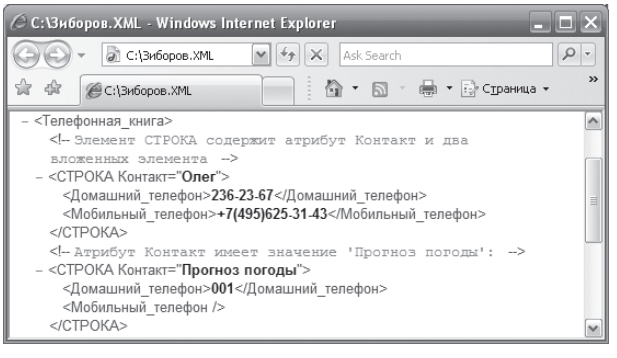

Рис. 11.6. XML-файл, открытый в Internet Explorer

Здесь весь XML-документ вложен в так называемый корневой элемент между начальным тегом <Телефонная_книга> и конечным тегом </Телефонная_книга>. Четыре элемента СТРОКА вложены в корневой элемент. В соответствующей таблице MS Excel элементы СТРОКА будут представлять строку в таблице. В свою очередь, элемент СТРОКА содержит в себе атрибут Контакт и два вложенных в него элемента, имена (Name) которых — Домашний_телефон и Мобильный_телефон. Именно поэтому в MS Excel отобразится таблица с тремя колонками (один атрибут и два элемента): «Контакт», «Домашний_телефон» и «Мобильный_телефон».

Элемент может иметь один или несколько атрибутов (а может и не иметь, как, скажем, элемент Домашний_телефон), например, первый элемент СТРОКА имеет атрибут с именем (Name) Контакт и со значением атрибута (Value) — 001.

После запуска данной программы будет на консоль выведено содержимое XML-документа без XML-объявления (скриншот консоли мы не приводим), а также будет создан XML-файл. Открыв этот файл с помощью MS Excel, получим таблицу телефонных контактов (рис. 11.7).

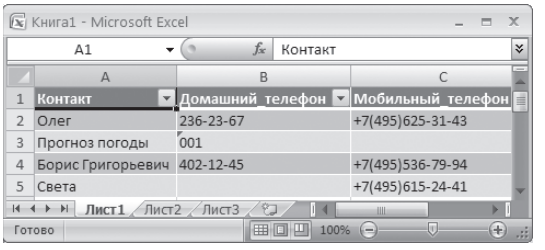

Рис. 11.7. XML-файл, открытый в MS Excel

Убедиться в работоспособности программы можно, открыв решение LinqСоздать-XML-документ.sln папки LinqСоздатьXML-документ.

Пример 93. Извлечение значения элемента из XML-документа

Предположим, что, решая задачу, мы получили строку XML-данных, например, от удаленной веб-службы, обеспечивающей нас прогнозом погоды. В этой текстовой XML-строке содержатся метеорологические показатели для указанного нами района на текущую дату. В данной задаче мы извлекаем из этих XML-данных только значение температуры.

Несколько слов о структуре XML-документа (рис. 11.8).

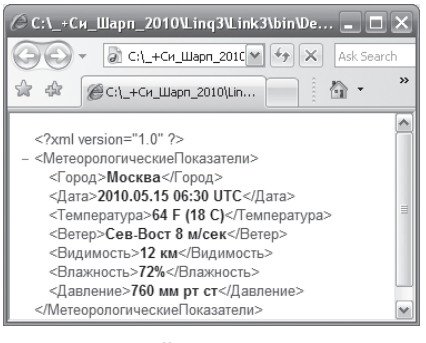

Рис. 11.8. Содержимое XML-файла с метеорологическими показателями

Как видно, XML-документ начинается с XML-объявления (XML declaration), в котором содержится информация о версии (version information parameter). Остальной XML-документ состоит из вложенных друг в друга элементов. Элемент — это блок разметки между начальным тегом, например <Город>, и конечным тегом </Город>. Самый внешний элемент, в данном случае — это тег <МетеорологическиеПоказатели>, его называют *корневым элементом* (root element). Как видите, этот корневой элемент содержит в себе все показатели, и, таким образом, глубина вложенности в этой иерархии равна двум. Следует отметить, что глубина вложенности такого XML-дерева может быть практически любой.

Итак, задача поставлена, сущность XML-данных понятна, приступаем к решению задачи. Для этой цели после запуска Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++, укажем имя решения Name — ПоискXmlЭлемента1. Чтобы иметь доступ к пространству имен System::Xml::Linq, добавим в текущий проект объектную библиотеку System.XML.Linq.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.XML.Linq.

Далее, в конструкторе формы из панели элементов Toolbox перетащим текстовое поле TextBox для вывода в него строки с данными XML и значения температуры из соответствующего элемента XML-дерева. Поскольку мы предполагаем вывод в текстовое поле не одной, а нескольких строчек, в свойствах объекта textBox1 укажем true напротив свойства Multiline. Затем на вкладке программного кода введем текст, представленный в листинге 11.7.

Листинг 11.7. Извлечение значения элемента из XML-данных (1 вариант)

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Дана строка XML, содержащая прогнозные метеорологические показатели
        // для Москвы на заданную дату. Программа извлекает из корневого элемента
        // XML-документа значение температуры элемента "Температура"
        // ~ ~ ~ ~ ~ ~ ~ ~
        // Следует добавить ссылку на библиотеку System.XML.Linq.dll. Для
        // этого: Project ▸ Properties ▸ Add Reference и на вкладке .NET
        // дважды щелкнем на ссылке System.XML.Linq
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 this->Text = "Поиск значения элемента в XML-документе";
                 textBox1->Multiline = true;
                 textBox1->Font = gcnew
                                  System::Drawing::Font("Consolas", 9.0F);
                 String ^ СтрокаXML =
                     "<?xml version=\"1.0\"?>                        \r\n" +
                     "   <МетеорологическиеПоказатели>               \r\n" +
                     "      <Город>Москва</Город>                    \r\n" +
                     "      <Дата>2011.05.15 06:30 UTC</Дата>        \r\n" +
                     "      <Температура> 64 F (18 C)</Температура> \r\n" +
                     "      <Ветер>Сев-Вост 8 м/с</Ветер>          \r\n" +
                     "      <Видимость>12 км</Видимость>             \r\n" +
                     "      <Влажность> 72%</Влажность>              \r\n" +
                     "      <Давление>760 мм рт ст</Давление>        \r\n" +
                     "   </МетеорологическиеПоказатели>";
                 // Загрузка корневого элемента из строки, содержащей XML:
```

продолжение ⇩

Листинг 11.7 (продолжение)

```
            auto КорневойЭлемент =
                        System::Xml::Linq::XElement::Parse(СтрокаXML);
            // Или корневой элемент XML-документа получаем через файл:
            // Записываем строку, содержащую XML, в файл:
            // System::IO::File::WriteAllText(
            //                 "C:\\ПоказателиПогоды.xml", СтрокаXML);
            // Загружаем корневой элемент XML:
            // КорневойЭлемент =
            //    System::Xml::Linq::XElement::Load(
            //                             "C:\\ПоказателиПогоды.xml");
            // Из корневого элемента извлекаем вложенный в него элемент
            // "Температура" и получаем соответствующее значение (Value)
            // этого элемента:
            String ^ Температура =
                       КорневойЭлемент->Element("Температура")->Value;
            textBox1->Text =
                       "Строка XML:\r\n\r\n" + СтрокаXML + "\r\n\r\n";
            textBox1->Text += "Значение температуры = " + Температура;
        }
    };
}
```

В начале текста программы задаем текстовую строку, содержащую XML-данные. Далее, используя метод Parse класса XElement пространства имен Linq, получаем корневой элемент XML-документа. В комментарии показано, как можно получить корневой элемент через запись/чтение XML-файла. Затем с помощью метода Element извлекаем значение (Value) элемента Температура, которое выводим в текстовое поле.

Фрагмент работы программы приведен на рис. 11.9.

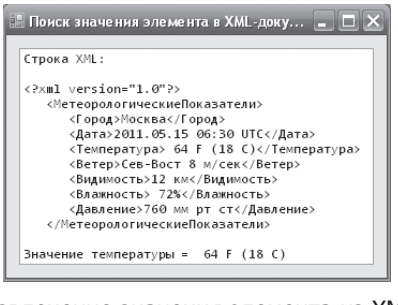

Рис. 11.9. Извлечение значения элемента из XML-документа

Убедиться в работоспособности программы можно, открыв решение ПоискXmlЭлемента1.sln в папке ПоискXmlЭлемента1.

Теперь решим похожую задачу по извлечению значения элемента, но пусть XML-данные будут представлены в другой форме, а именно каждый метеорологический показатель вложим в один и тот же элемент <Показатель> </Показатель>;

в этом случае глубина вложенности элементов будет уже равна трем (листинг 11.8) (ситуация из реальной жизни). Естественно спросить: что мы будет с этого иметь? Дело в том, что если соответствующий XML-файл открыть с помощью табличного редактора MS Excel, то мы сможем увидеть эти XML-данные в виде наглядной таблицы, даже не ссылаясь на таблицу стилей — файл XSLT (не путать с XLS-файлом) (рис. 11.10).

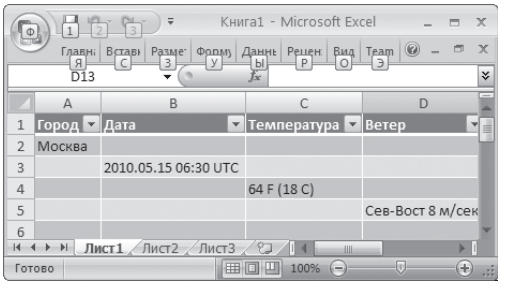

Рис. 11.10. Представление XML-данных в виде таблицы в MS Excel

Если бы мы программировали на Visual Basic 2010 или C# 2010, то для получения значения температуры мы бы воспользовались типовым LINQ-запросом. Однако, как мы уже отмечали, язык MS Visual C++ для среды .NET не поддерживает технологию LINQ-запросов. Между тем, мы можем с успехом использовать методы классов пространства имен System::Xml::Linq. Рассмотрим, как решить задачу по извлечению значения элемента методом Where технологии LINQ.

Для этого запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR. Выберем имя решения Name — ПоискXmlЭлемента2. Чтобы иметь доступ к пространству имен System::Xml::Linq, добавим в текущий проект объектную библиотеку System.XML.Linq.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.XML.Linq.

Теперь на вкладке программного кода введем текст, представленный в листинге 11.8.

Листинг 11.8. Извлечение значения элемента из XML-данных (2 вариант)

```
// ПоискXmlЭлемента2.cpp: главный файл проекта.
// Дана строка XML, которая содержит прогнозные метеорологические
// показатели для Москвы на заданную дату. При этом каждый
// метеорологический показатель вложен в один и тот же элемент
// <Показатель> </Показатель>. Это обеспечивает удобный просмотр
// соответствующего XML-файла в MS Excel в виде таблицы. Программа
// находит в корневом элементе данного XML-документа элемент
// "Температура" и извлекает из него значение температуры
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Xml.Linq
using namespace System;
```

продолжение ⇩

Листинг 11.8 (продолжение)

```
// Добавим эти пространства имен для краткости выражений:
using namespace System::Xml::Linq;
using namespace System::Collections::Generic;
bool Предикат(XElement ^ t)
{
    bool A = false;
    if(t->Element("Температура") != nullptr) A = true;
        return A;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    Console::Title = "Поиск значения элемента в XML-документе";
    // Инициализация XML-строки:
    String ^ СтрокаXML =
        "<?xml version=\"1.0\"?>"                      +
        "<МетеорологическиеПоказатели>"                +
        "  <Показатель>"                               +
        "     <Город>Москва</Город>"                   +
        "  </Показатель>"                              +
        "  <Показатель>"                               +
        "     <Дата>2010.05.15 06:30 UTC</Дата>"       +
        "  </Показатель>"                              +
        "  <Показатель>"                               +
        "     <Температура> 64 F (18 C)</Температура>" +
        "  </Показатель>"                              +
        "  <Показатель>"                               +
        "     <Ветер>Сев-Вост 8 м/с</Ветер>"          +
        "  </Показатель>"                                +
        "  <Показатель>"                                 +
        "     <Видимость>12 км</Видимость>"              +
        "  </Показатель>"                                +
        "  <Показатель>"                                 +
        "     <Влажность> 72%</Влажность>"               +
        "  </Показатель>"                                +
        "  <Показатель>"                                 +
        "     <Давление>760 мм рт ст</Давление>"         +
        "  </Показатель>"                                +
        "</МетеорологическиеПоказатели>";
    auto КорневойЭлемент = XElement::Parse(СтрокаXML);
    // Получаем коллекцию дочерних элементов с именем «Показатель»
    auto СписокЭлементов = КорневойЭлемент->Elements(«Показатель»);
    // В коллекции СписокЭлементов ищем элемент с именем "Температура":
    auto ЭлемТемпер = Linq::Enumerable::Where<XElement^>(СписокЭлементов,
                                gcnew Func<XElement^, bool>(Предикат));
    // В списке ЭлемТемпер - ровно одна запись:
    for each (XElement ^ Элемент in ЭлемТемпер)
    Console::WriteLine("Значение температуры = " + Элемент->Value);
```

```
    // Если не пользоваться методом Where, то значение
    // температуры можно получить таким образом:
    // String ^ Темп = String::Empty;
    // for each (XElement ^ Элемент in СписокЭлементов)
    // {
    //    if (Элемент->Element("Температура") != nullptr) Темп =
    //        Элемент->Element("Температура")->Value;
    // }
    // Console::WriteLine("Значение температуры = " + Темп);
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, вначале мы загружаем строку с XML-данными в XML-элемент методом Parse. Затем получаем коллекцию дочерних элементов с именем «Показатель», используя метод Elements класса XElement. Теперь делаем запрос методом Where на выборку из этой коллекции элементов с именем «Температура». Таких элементов в производном списке — ровно один. Далее в комментарии приведено, как можно найти в данном XML-документе значение температуры без использования метода Where.

Фрагмент работы программы показан на рис. 11.11.

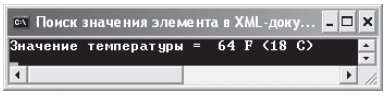

Рис. 11.11. Извлечение значения температуры из XML-документа

Убедиться в работоспособности программ, рассмотренных в данном разделе можно, открыв соответствующие решения в папках ПоискXmlЭлемента1 и ПоискXmlЭлемента2.

Пример 94. Поиск строк (записей) в XML-данных

Мы имеем XML-данные, в которых содержится традиционная для нашей книги таблица с именами и телефонами, причем имена в этой телефонной табличке повторяются, например, строка с именем «Витя» содержит мобильный телефон, а потом по мере знакомства с этим Витей у нас появляется и его домашний телефон. Наша задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML, см. листинг 11.9) найти все строчки с именем «Витя». Эта маленькая и несерьезная, на первый взгляд, задача аналогична, например, следующей «серьезной» задаче: в некоторой громадной базе данных, которую мы получили на каком-то этапе обработки в виде XML, нам требуется «отфильтровать» записи на предмет содержания в каком-либо поле определенной строки.

Прежде чем решать данную задачу, давайте посмотрим на отображение обсуждаемых XML-данных в табличном редакторе MS Excel (рис. 11.12).

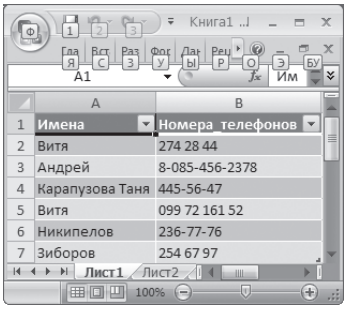

Рис. 11.12. Отображение XML-данных в MS Excel

Мы видим, что в редакторе MS Excel наши XML-данные представлены весьма наглядно. И становится понятным, что мы хотим получить, а именно получить все номера телефонов, расположенные напротив имени «Витя».

Для решения этой задачи запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR. Укажем имя решения Name — ПоискСтрокВXml. Чтобы иметь доступ к пространству имен System::Xml::Linq, добавим в текущий проект объектную библиотеку System.XML.Linq.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.XML.Linq. Затем на вкладке программного кода введем текст, представленный в листинге 11.9.

Листинг 11.9. Извлечение значений элементов из XML-данных

```
// ПоискСтрокВXml.cpp: главный файл проекта.
// Имеем XML-данные, в которых содержится таблица с именами
// и телефонами, причем имена в этой телефонной табличке повторяются.
// Задача состоит в том, чтобы в данной таблице телефонов
// (представленной в виде XML) найти все строчки с именем "Витя"
// с помощью методов классов пространства имен System::Xml::Linq
#include "stdafx.h"
// Добавим в текущий проект объектную библиотеку System.XML.Linq.dll
using namespace System;
// Добавим эту директиву для краткости выражений:
using namespace System::Xml::Linq;
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    Console::Title = "Поиск в XML-данных методами Linq";
    // Инициализация XML-строки:
    String ^ СтрокаXML =
        "<?xml version=\"1.0\"?>                                    " +
        "   <ТаблицаТелефонов>                                      " +
        "      <Строка>                                             " +
        "        <Имена>Витя</Имена>                                " +
        "        <Номера_телефонов>274 28 44</Номера_телефонов>     " +
        "      </Строка>                                            " +
```

```
        "      <Строка>                                                  " +
        "        <Имена>Андрей</Имена>                                   " +
        "        <Номера_телефонов>8-085-456-2378</Номера_телефонов>"  +
        "      </Строка>                                                 " +
        "      <Строка>                                                  " +
        "        <Имена>Карапузова Таня</Имена>                          " +
        "        <Номера_телефонов>445-56-47</Номера_телефонов>     " +
        "     </Строка>                                                  " +
        "     <Строка>                                                  " +
        "       <Имена>Витя</Имена>                                     " +
        "       <Номера_телефонов>099 72 161 52</Номера_телефонов>  " +
        "     </Строка>                                                 " +
        "     <Строка>                                                  " +
        "       <Имена>Никипелов</Имена>                                " +
        "       <Номера_телефонов>236-77-76</Номера_телефонов>      " +
        "     </Строка>                                                 " +
        "     <Строка>                                                  " +
        "       <Имена>Зиборов</Имена>                                  " +
        "       <Номера_телефонов>254 67 97</Номера_телефонов>      " +
        "     </Строка>                                                 " +
        "   </ТаблицаТелефонов>";
    auto КорневойЭлемент = XElement::Parse(СтрокаXML);
    // Запись строки, содержащей XML, в файл:
    // IO::File::WriteAllText(«C:\\ТаблицаТелефонов.xml», СтрокаXML);
    // КорневойЭлемент = XElement::Load(«C:\\ТаблицаТелефонов.xml»);
    // Получаем коллекцию дочерних элементов с именем «Строка»
    auto ВсеЗаписи = КорневойЭлемент->Elements(«Строка»);
    Console::WriteLine("Строки, содержащие имя \"Витя\":");
    for each (XElement ^ x in ВсеЗаписи)
        if (x->Element("Имена")->Value == "Витя")
            Console::WriteLine(x->Element("Номера_телефонов")->Value);
    // Таких записей в этой коллекции - две
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, в начале программы мы инициализируем (то есть присваиваем начальные значения) XML-строку. Далее извлекаем корневой элемент из XML-документа, он, по сути, отличается от XML-документа отсутствием XML-объявления (в этом можно убедиться в отладчике программы). В комментарии указано, как можно получить корневой элемент в том случае, если он представлен в виде XML-файла во внешней памяти. Затем получаем коллекцию дочерних элементов с именем «Строка». А затем в цикле for each ищем элементы со значением атрибута «Витя». Фрагмент работы программы показан на рис. 11.13.

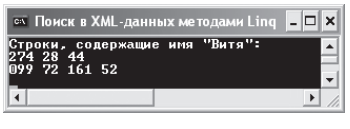

Рис. 11.13. В XML-документе найдены строки методами класса Linq

Убедиться в работоспособности программы можно, открыв решение ПоискСтрокВXml.sln из папки ПоискСтрокВXml.

Пример 95. Получение производных XML-данных от XML-источника

В этом примере у нас имеется источник данных в виде XML-файла, содержащего сведения о некоторых книгах, издаваемых в издательстве «Питер». Поскольку мы хотим представить очень маленький и выразительный пример, то в этом файле содержатся сведения всего о пяти книгах. Фрагмент содержимого этого файла-источника в Internet Explorer см. на рис. 11.14, а его отображение в виде таблицы в MS Excel см. на рис. 11.15. Из этого XML-файла требуется выбрать книги только одного автора — И. Квинта и записать их в другой, производный XML-файл.

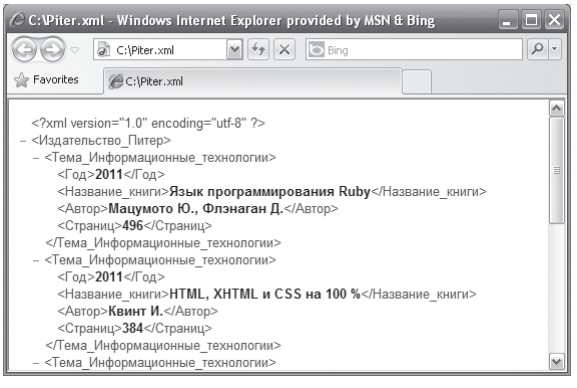

Рис. 11.14. Фрагмент XML-файла — источника данных

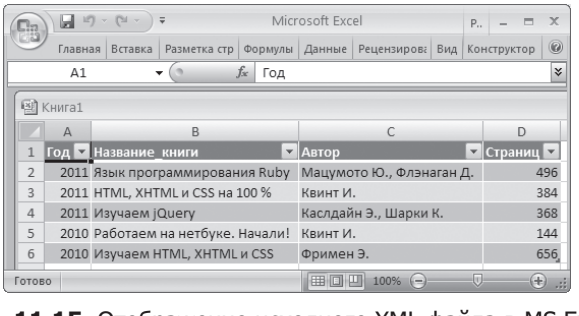

Рис. 11.15. Отображение исходного XML-файла в MS Excel

Для решения этой задачи запустим Visual Studio 2010, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR, укажем имя Name — LinqPiter. Далее, чтобы иметь доступ к пространству имен System::Xml::Linq, добавим в текущий

проект объектную библиотеку System.XML.Linq.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.XML.Linq. Ниже в листинге 11.10 приведен программный код данного консольного приложения.

Листинг 11.10. Выборка данных из XML-файла и их запись в другой XML-файл

```
// LinqPiter.cpp: главный файл проекта.
// Программа читает XML-файл (источник), содержащий сведения о книгах по
// программированию, и выбирает книги автора И. Квинта. Выбранные книги
// программа записывает в другой (производный) XML-файл
#include "stdafx.h"
// Добавим ссылку на System.XML.Linq, для этого выберем пункты
// меню Project | Properties | Add Reference и на вкладке .NET дважды
// щелкнем на ссылке System.Xml.Linq
using namespace System;
// Добавим эти директивы для краткости выражений:
using namespace System::Xml;
using namespace System::Xml::Linq;
int main(array<System::String ^> ^args)
{
    XmlWriterSettings ^ Установки = gcnew XmlWriterSettings();
    // Создавать отступы для элементов:
    Установки->Indent = true;
    // Создание производного XML-файла:
    XmlWriter ^ XmlПисатель =
        XmlWriter::Create("C:\\ТолькоКвинт.xml", Установки);
    //Добавляем начальный (корневой) элемент:
    XmlПисатель->WriteStartElement("Книги_И_Квинта");
    // Комментарий:
    XmlПисатель->WriteComment("Эти данные мы выбрали из Piter.xml");
    // Загружаем XML-документ - источник:
    XDocument ^ XmlДокумент = XDocument::Load("C:\\Piter.xml");
    // Извлекаем корневой элемент:
    XElement ^ КорневЭлемент =
                    XmlДокумент->Element("Издательство_Питер");
    // Получаем список элементов "Книга":
    auto СписокКниг = КорневЭлемент->
                    Elements("Тема_Информационные_технологии");
    // В этом списке анализируем дочерний элемент "Автор":
    for each (XElement ^ Элемент in СписокКниг)
        if (Элемент->Element("Автор")->Value == "Квинт И.")
            // Добавлям в производный XML-файл только книги И. Квинта:
            Элемент->WriteTo(XmlПисатель);
    // Добавляем конечный тег корневого элемента производного XML-документа:
    XmlПисатель->WriteEndElement();
    XmlПисатель->Flush();
    XmlПисатель->Close();
    return 0;
}
```

В программном коде создаем производный XML-файл, куда будем записывать выбранные сведения из файла-источника C:\Piter.xml. Из исходного XML-документа (источника) извлекаем корневой элемент, а из него получаем список элементов «Книга». В этом списке с помощью цикла for each организуем поиск дочернего элемента «Автор» по содержимому элемента «Квинт И.». В результате работы программы получим производный XML-файл с выбранными данными (рис. 11.16 и 11.17).

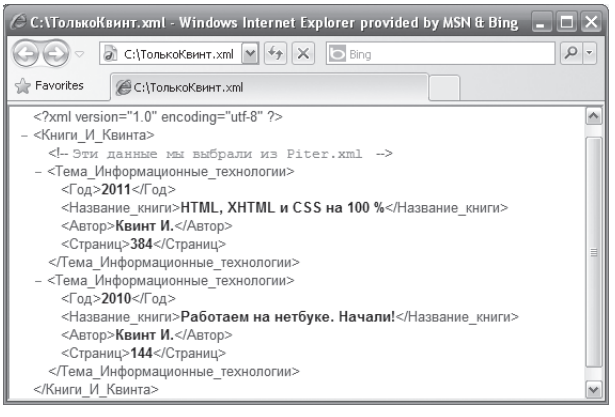

Рис. 11.16. Содержимое производного XML-файла

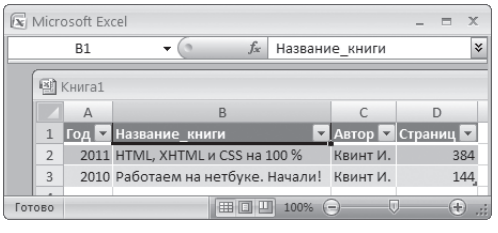

Рис. 11.17. Отображение выбранных XML-данных в MS Excel

Убедиться в работоспособности программы можно, открыв решение LinqPiter.sln из папки LinqPiter.

Пример 96. Организация поиска в наборе данных DataSet

Весьма полезной оказывается организация поиска в наборах данных DataSet, используемых, например, при работе с базами данных. Объект класса DataSet представляет расположенный в памяти кэш (cache) данных (кэш — это промежуточная память с быстрым доступом, содержащая информацию, которая может быть запрошена с наибольшей вероятностью). Реляционные базы данных работают чаще

всего с совокупностью таблиц. Каждая из этих таблиц задается как объект класса DataTable, один такой объект представляет ровно одну таблицу данных. Как правило, набор данных DataSet содержит в себе несколько объектов (таблиц) DataTable. Запросы LINQ к таблицам данных, кэшированным в объекте DataSet, упрощают и ускоряют процесс отбора.

Данная задача состоит в том, чтобы создать программу, которая обеспечивает ввод простейшей таблицы, содержащей два поля — название города и численность его населения. Программа способна фильтровать данные в таблице: мы отберем города, население которых превышает миллион жителей.

Для решения этой задачи запустим Visual Studio 2010 и выберем проект шаблона Windows Forms Application, укажем имя Name — ПоискВDataSet. Далее в конструкторе формы из панели элементов Toolbox перетащим элемент управления для отображения и редактирования табличных данных DataGridView, две командные кнопки Button и текстовое поле TextBox. Одна кнопка предназначена для команды сохранения данных, другая — для поиска данных в таблице, а текстовое поле — для вывода в него найденных строк из таблицы. В свойствах текстового поля разрешим ввод множества строк, для этого свойство Multiline переведем в состояние true.

Для реализации поиска в наборе DataSet в текущий проект следует добавить объектную библиотеку System.Data.DataSetExtensions.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Data.DataSetExtensions. Теперь на вкладке программного кода введем текст, представленный в листинге 11.11.

Листинг 11.11. Извлечение полей из набора данных DataSet

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // В данной программе экранная форма содержит элемент управления для
        // отображения и редактирования табличных данных DataGridView, две
        // командные кнопки и текстовое поле. При старте программы, если есть
        // соответствующий файл XML, то программа отображает в элемент
        // DataGridView таблицу городов - название города и численность
        // населения. При щелчке на кнопке "Сохранить" все изменения в таблице
        // записываются в XML-файл. При щелчке на второй кнопке "Найти"
        // выполняется поиск городов-миллионеров в наборе данных DataSet искомой
        // таблицы. Результат запроса выводится в текстовое поле
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Следует добавить ссылку System.Data.DataSetExtensions на вкладке .NET
        DataTable ^ Таблица;   // Объявление объекта таблица данных
        DataSet ^ НаборДанных; // Объявление объекта набор данных
    private: System::Void Form1_Load(System::Object^  sender,
                 System::EventArgs^  e)
```

продолжение ⇩

Листинг 11.11 (продолжение)

```
        {
            Таблица = gcnew DataTable();
            НаборДанных = gcnew DataSet();
            Form1::Text = "Поиск в наборе данных DataSet";
            button1->Text = "Сохранить"; button2->Text = "Найти";
            textBox1->Multiline = true; button2->TabIndex = 0;
            if (System::IO::File::Exists("C:\\Города.xml") == false)
            {
                // Если XML-файла НЕТ:
                // Заполнение "шапки" таблицы
                Таблица->Columns->Add("Город");
                Таблица->Columns->Add("Население");
                // Добавить объект Таблица в DataSet
                НаборДанных->Tables->Add(Таблица);
                dataGridView1->DataSource = Таблица;
            }
            else // Если XML-файл ЕСТЬ:
            {
                НаборДанных->ReadXml("C:\\Города.xml");
                // Содержимое DataSet в виде строки XML для отладки:
                // String ^ СтрокаXML = НаборДанных->GetXml();
                Таблица = НаборДанных->Tables["Города"];
                dataGridView1->DataMember = "Города";
                dataGridView1->DataSource = НаборДанных;
            }
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Щелчок мышью на кнопке "Сохранить" -
            // сохранить файл Города.xml:
            Таблица->TableName = "Города";
            НаборДанных->WriteXml("C:\\Города.xml");
        }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Щелчок мышью на кнопке "Поиск" - запрос городов-миллионеров:
            textBox1->Clear(); // - очистка текстового поля
            // Извлекаем коллекцию строк (рядов) из таблицы DataTable:
            EnumerableRowCollection<DataRow^> ^ ВсеГорода =
                                DataTableExtensions::AsEnumerable(Таблица);
            // Создаем словарь городов:
            Generic::Dictionary<String^, String^> СловарьГородов =
                            gcnew Generic::Dictionary<String^, String^>();
            // Цикл по всем рядам таблицы:
            for each (DataRow ^ Ряд in ВсеГорода)
            {
                // В каждом ряду находим название города и его население:
```

```
                String ^ Город = DataRowExtensions::
                                 Field<String^>(Ряд, "Город");
                String ^ Население = DataRowExtensions::
                                 Field<String^>(Ряд, "Население");
                // Города с населением более 1 млн добавляем в словарь
городов:
                if (Convert::ToInt32(Население) >= 1000000)
                    СловарьГородов.Add(Город, Население);
            }
            textBox1->Text = "Города-миллионеры:\r\n";
            // Вывод результата запроса в текстовое поле textBox1:
            for each (String ^ Город in СловарьГородов.Keys)
                textBox1->Text +=
                     Город + " - " + СловарьГородов[Город] + "\r\n";
        }
    };
}
```

В начале программы объявляем объекты классов DataSet и DataTable так, чтобы они были видимыми из всех процедур класса Form1. Затем при обработке события загрузки формы проверяем, существует ли файл Города.xml, куда мы записываем искомую таблицу. Если файл не существует, то есть пользователь первый раз запустил нашу программу, то мы создаем таблицу, состоящую из двух полей (колонок): «Город» и «Население», добавляем (Add) эту таблицу в набор данных, а также указываем таблицу в качестве источника данных (DataSource) для сетки данных dataGridView1. Если же файл Города.xml уже создан, то мы считываем его в набор данных DataSet и из него заполняем таблицу данных, а также этот набор данных указываем в качестве источника для сетки данных.

При обработке события «щелчок мышью на кнопке Запись» программируем сохранение редактируемой таблицы в файле Города.xml. В процедуре обработки события «щелчок на кнопке Найти» организуем поиск в заполненной пользователем таблице DataTable, являющейся представителем DataSet. Условием поиска является отбор таких полей таблицы, где население больше миллиона жителей. Результат запроса выводим в текстовое поле, используя цикл for each.

Фрагмент работы программы показан на рис. 11.18.

Убедиться в работоспособности программы можно, открыв решение ПоискВDataSet.sln из папки ПоискВDataSet.

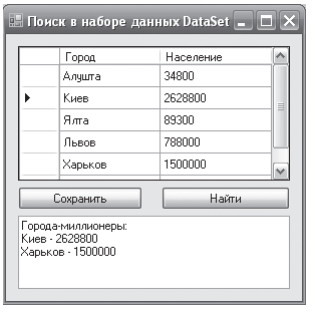

Рис. 11.18. Поиск городов-миллионеров в наборе данных

12

Другие задачи, решаемые с помощью Windows Application

Пример 97. Проверка вводимых данных с помощью регулярных выражений

Данные, вводимые пользователем, должны быть проверены программой на достоверность. В этом примере мы обсудим синтаксический разбор введенной пользователем текстовой строки на соответствие ее фамилии на русском языке, а также разбор строки на соответствие ее положительному рациональному числу.

Начнем с первой задачи: имеем на форме текстовое поле, метку и кнопку. В метке записано приглашение пользователю ввести фамилию на русском языке. После ввода программа должна сравнить эту строку с некоторым образцом (шаблоном, pattern) и сделать заключение, соответствует ли введенное пользователем шаблону русской фамилии.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов управления Toolbox в форму с помощью мыши перетащим текстовое поле TextBox, метку Label и командную кнопку Button. В листинге 12.1 приведен текст программы.

Листинг 12.1. Проверка вводимой фамилии с помощью регулярных выражений

```
            // .............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Проверка данных, вводимых пользователем, на достоверность.
```

```
        // Программа осуществляет синтаксический разбор введенной пользователем
        // текстовой строки на соответствие ее фамилии на русском языке
    private: System::
               Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
             {
                 label1->Text = "Введите фамилию на русском языке:";
                 button1->Text = "Проверка";
             }
    private: System::Void button1_Click(System::Object^ sender,
                                                    System::EventArgs^ e)
             {
                 textBox1->Text = textBox1->Text->Trim();
                 if (System::Text::RegularExpressions::Regex::Match(
                     textBox1->Text,
                             "^[А-ИК-ЩЭ-Я][а-яА-Я]*$")->Success != true)
                     MessageBox::Show ("Неверный ввод фамилии", "Ошибка");
             }
    };
}
```

При обработке события «щелчок мышью на кнопке» текстовое поле textBox1->Text обрабатывается методом Trim, который удаляет все пробельные символы в начале и в конце строки. Ключевым моментом программы является *проверка соответствия* введенной пользователем текстовой строки и шаблона с помощью функции Regex::Match (от англ. *match* — соответствовать):

```
Regex::Match(textBox1->Text, "^[А-ИК-ЩЭ-Я][а-яА-Я]*$")
```

Match представляет результаты из *отдельного совпадения* регулярного выражения. Как видно, мы начали регулярное выражение с символа ^ и закончили символом $. Символы ^ и $ соответствуют *началу* и *концу строки* соответственно. Это заставляет регулярное выражение оценивать всю строку, а не возвращать соответствие, если успешно совпадает подстрока.

Далее в первых квадратных скобках указан диапазон допустимых букв для установления соответствия первой букве фамилии. Первая буква должна быть прописной, то есть быть буквой верхнего регистра, в диапазоне алфавита (и таблицы символов) от А до И, от К до Щ и от Э до Я, то есть недопустим ввод букв Й, Ъ, Ы, Ь в качестве первой буквы фамилии. Далее в следующих квадратных скобках указан диапазон букв либо нижнего, либо верхнего регистров, причем символ * означает, что второй диапазон символов может встретиться в строке ноль или более раз. Фрагмент работы программы представлен на рис. 12.1.

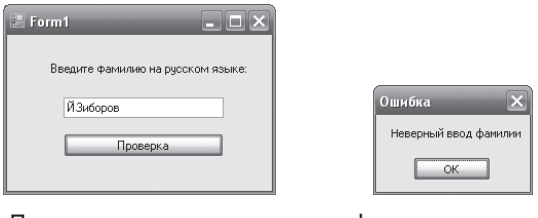

Рис. 12.1. Проверка корректности ввода фамилии на русском языке

Вторая задача, которую мы рассмотрим в данном примере, — это проверка правильности ввода *положительного рационального числа*. Следует допустить возможность ввода любых вариантов, например «2010», «2.9», «5.», «.777», то есть допустим ввод цифровых символов и точки (или запятой).

Пользовательский интерфейс для решения данной задачи такой же, как и для предыдущей, поэтому сразу приведу программный код решения (листинг 12.2).

Листинг 12.2. Проверка вводимого числа с помощью регулярных выражений

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Проверка данных, вводимых пользователем, на достоверность. Программа
        // осуществляет синтаксический разбор введенной пользователем текстовой
        // строки на соответствие ее положительному рациональному числу
    private: System::
                Void Form1_Load(System::Object^  sender, System::EventArgs^  e)
            {
                label1->Text = "Введите положительное рациональное число:";
                button1->Text = "Проверка";
            }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
            {
                textBox1->Text = textBox1->Text->Trim();
                if (System::Text::RegularExpressions::Regex::Match(
                    textBox1->Text,
                    "^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
                    )->Success == false)
                    MessageBox::Show("Некорректный ввод", "Ошибка");
            }
    };
}
```

Данная программа построена аналогично предыдущей, в комментарии нуждается шаблон

```
"^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
```

Здесь между символами |, означающими логическое ИЛИ, между круглыми скобками представлены *три группы выражений*. *Первая группа* допускает ввод цифровых символов от 0 до 9 до десятичной точки. Знак «плюс» (+) означает, что цифровой символ может встретиться в строке один или более раз. Символ * означает, что цифровой символ может встретиться в строке ноль или более раз. Таким образом, первая группа допускает ввод, например, рационального числа 6. (с точкой на конце). Аналогично работает *вторая группа* выражений, она допус-

кает ввод, например, числа .777. *Третья группа* проверяет соответствие с любыми целыми числами.

Убедиться в работоспособности программ, рассмотренных в данном примере, можно, открыв решение ПроверкаФамилии.sln в папке ПроверкаФамилии и ПроверкаЧисла.sln в папке ПроверкаЧисла.

Пример 98. Управление прозрачностью формы

Создадим программу, которая демонстрирует стандартное Windows-окно, то есть стандартную форму. Щелчок мышью в пределах этой формы *начинает постепенный процесс* исчезновения формы, форма становится все более прозрачной, а затем исчезает вовсе, далее она постепенно проявляется снова и т. д. Еще один щелчок в пределах формы останавливает этот процесс, а следующий щелчок процесс возобновляет и т. д.

Для написания этой программы запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Теперь добавим в стандартную форму из панели элементов управления Toolbox объект Timer (Таймер). Это невидимый во время работы программы элемент управления предназначен для периодического генерирования события Tick, которое происходит, когда таймер работает и прошел заданный интервал времени. Этот интервал времени timer1->Interval по умолчанию равен 100 миллисекунд, его в нашей программе мы изменять не будем.

В данной программе мы будем обрабатывать три события. Пустой обработчик события загрузки формы можно получить, дважды щелкнув в пределах проектируемой формы. Обработчик события истечения заданного временного интервала получим, дважды щелкнув на изображении объекта Timer на вкладке дизайнера формы Form1.h[Design]. И пустой обработчик события «щелчок в пределах экранной формы» можно получить в окне свойств формы, щелкнув на значке молнии и выбрав событие Click. Управлять прозрачностью формы можно с помощью свойства формы Opacity, задавая уровень непрозрачности от нуля до единицы. Текст программы приведен в листинге 12.3.

Листинг 12.3. Управление прозрачностью формы

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа демонстрирует стандартную форму. Щелчок мышью в пределах
        // этой формы начинает постепенный процесс исчезновения формы: форма
        // становится все более прозрачной, а затем исчезает вовсе. Далее она
        // постепенно проявляется снова и т. д. Еще один щелчок в пределах
        // формы останавливает этот процесс, а следующий щелчок процесс
```

продолжение ⇘

Листинг 12.3 (продолжение)

```
        // возобновляет и т. д.
        double s; // - шаг изменения прозрачности
    private: System::
                 Void Form1_Load(System::Object^  sender, System::EventArgs^ e)
             {
                 s = 0.1;
                 Form1::Text = "Щелкните на форме";
                 // Timer1->Interval() = 400;
             }
    private: System::Void timer1_Tick(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 if (this->Opacity <= 0 || this->Opacity >= 1) s = -s;
                 this->Opacity += s;
             }
    private: System::Void Form1_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 timer1->Enabled = !timer1->Enabled;
             }
    };
}
```

Как видно из программного кода, при обработке события «щелчок в пределах формы» запускается таймер, а еще один щелчок мыши его останавливает. Каждые 100 миллисекунд возникает событие timer1_Click. Обрабатывая его, мы меняем значение непрозрачности Opacity от нуля до единицы с шагом 0.1, который задаем через внешнюю переменную s.

Этот пример на языке Visual Basic приведен на сайте

http://subscribe.ru/archive/comp.soft.prog.visualbnet/200512/16181816.html,

автор переписал его на C++. Текст этой программы можно посмотреть, открыв решение Opacity.sln в папке Opacity.

Пример 99. Время по Гринвичу в полупрозрачной форме

Время, дату, день недели очень легко выяснить, посмотрев в правый нижний угол рабочего стола Windows. Однако в том случае, если вы, например, читаете новости, которые поступают в реальном времени, и время публикаций указывают по Гринвичу (GMT — Greenwich Meridian Time), это могут быть, например, новости валютного рынка или фондового рынка Форекс или иной экономический календарь, то в этом случае важно знать, насколько актуальна, свежа новость. Конечно, можно держать в голове, что московское время отличается от гринвичского на 4 часа, а киевское — на 3 часа, а затем мучительно соображать, необходимо прибавить эти 3 часа к киевскому времени или, наоборот, отнять.

Кроме того, следует помнить, что гринвичское время не переводится весной и осенью на час. Поэтому, чтобы выяснить правильное время по Гринвичу, можно на стандартном рабочем столе Windows справа внизу двойным щелчком на отображении текущего местного времени на вкладке Часовой пояс выбрать Время по Гринвичу и сбросить флажок Автоматический переход на летнее время и обратно. При этом очень удобно для сравнения времени иметь перед глазами сайт www.central-european-time.com, где показано текущее местное время, среднеевропейское (или центрально-европейское время, Central European Time (CET)) и время по Гринвичу.

Предлагаю написать маленькую программу, которая в полупрозрачной экранной форме отображает текущее время по Гринвичу. То есть мы напишем программу, которая будет демонстрировать текущее время по Гринвичу, и при этом, в силу своей полупрозрачности, форма не будет закрывать другие приложения.

Для решения этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Теперь из панели элементов управления Toolbox перетащим в проектируемую экранную форму метку Label и объект Timer. Этот объект позволит обрабатывать событие timer1_Tick через заданные интервалы времени. Далее на вкладке визуального проекта программы Form1.h [Design] растянем мышью форму и расположим метку примерно так, как показано на рис. 12.2.

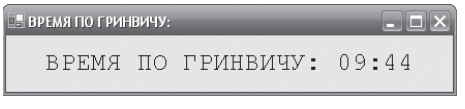

Рис. 12.2. Фрагмент работы программы определения времени по Гринвичу

Код данной программы представлен в листинге 12.4.

Листинг 12.4. Время по Гринвичу в полупрозрачной форме

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа в полупрозрачной экранной форме отображает текущее время
        // по Гринвичу. Таким образом, программа демонстрирует текущее время
        // по Гринвичу и при этом не закрывает собой другие приложения
        // ~ ~ ~ ~ ~ ~ ~ ~ ~
        // Булева переменная t каждую секунду меняет свое значение
        // на противоположное:
        bool t;
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
```

продолжение ⇩

Листинг 12.4 (продолжение)

```
            {
                t = false;
                this->Text = "ВРЕМЯ ПО ГРИНВИЧУ:";
                Form1::Opacity = 0.75;   // Уровень непрозрачности формы
                label1->Font = gcnew System::Drawing::Font("Courier New",
                18.0F);
                label1->Text = String::Empty;
                timer1->Interval = 1000; // 1000 миллисекунд = 1 секунда
                timer1->Start();
            }
    private: System::Void timer1_Tick(System::Object^  sender,
                                      System::EventArgs^  e)
            {
                // Обработка события, когда прошел заданный интервал
                // времени: 1000 миллисекунд = 1 секунда
                label1->Text = "ВРЕМЯ ПО ГРИНВИЧУ: ";
                String ^ Время;
                t = !t;   // То же, что и t = true ^ t;
                if (t == true) Время = String::Format("{0:t}", DateTime::UtcNow);
                else Время = String::Format("{0} {1:00}", DateTime::UtcNow.Hour,
                    DateTime::UtcNow.Minute);
                label1->Text = label1->Text + Время;
            }
    private: System::Void label1_MouseEnter(System::Object^  sender,
                                            System::EventArgs^  e)
            {
                // Указатель мыши входит в область метки
                this->Opacity = 1;
            }
    private: System::Void Form1_MouseLeave(System::Object^  sender,
                                           System::EventArgs^  e)
            {
                // Указатель мыши выходит за пределы формы
                this->Opacity = 0.75;
            }
    };
}
```

При обработке события загрузки формы задаем текст «ВРЕМЯ ПО ГРИНВИЧУ:» в заголовке формы (свойство this->Text), указываем уровень непрозрачности формы Form1::Opacity = 0.75 (или this->Opacity = 0.75). Если Opacity = 0, то изображение формы совсем пропадает, программа будет благополучно работать в оперативной памяти, и мы будем догадываться о ее работе только потому, что она будет напоминать о своем существовании в свернутом виде на панели задач. При Opacity = 1 будем иметь обычное непрозрачное изображение формы.

Далее включаем таймер timer1->Start() и задаем интервал работы таймера timer1->Interval = 1000 (1000 миллисекунд, то есть 1 секунда). Это означает, что через

каждую секунду мы имеем возможность обрабатывать событие «прошла одна секунда» timer1_Tick.

Справиться с задачей демонстрации времени было бы легко, если бы нам надо было просто отображать текущее время. В этом случае мы бы написали:

```
Время = String::Format("{0:t}", DateTime::Now)
```

Однако по условию задачи нам нужно показывать время по Гринвичу. Система Visual Studio 2010 имеет свойство UtcNow пространства имен System::DateTime. Это свойство возвращает так называемое *универсальное координированное время* (Universal Coordinated Time, UTC), которое с точностью до долей секунды совпадает с временем по Гринвичу. Поэтому выражение

```
Время = String::Format("{0:t}", DateTime::UtcNow);
```

копирует в строку время значение текущего гринвичского времени.

Далее, чтобы визуализировать ход времени, между значением часа и значением минут зададим двоеточие (:), которое появляется в течение одной секунды и исчезает тоже на одну секунду. Для этого у нас есть внешняя булева переменная t, которая периодически (каждую секунду) меняет свое значение с true на false, потом с false на true и т. д., то есть на противоположное: t = !t. Таким образом, при обработке события Tick (события, когда прошел заданный интервал времени Interval, равный 1000 миллисекунд = 1 секунда) объекта timer1 в метку label1 копируется новое значение времени то с двоеточием, то без него.

Как видно, в тексте данной программы использованы переменные с русскими именами, что *добавило выразительности* программному коду. Мы реализовали возможность изменения прозрачности формы при наведении курсора мыши в область метки при обработке события label1_MouseEnter вхождения курсора мыши в границы метки. Из эстетических соображений (что, конечно же, может быть подвергнуто сомнению) мы запрограммировали возврат опять к прозрачной форме при выходе курсора мыши за пределы *формы*, а не *метки*, то есть при наступлении события Form1_MouseLeave.

Текст этой программы можно посмотреть, открыв решение Гринвич.sln в папке Гринвич.

Пример 100. Ссылка на процесс, работающий в фоновом режиме, в форме значка в области уведомлений

Для новичка название этого раздела может показаться чрезвычайно «заумным». На самом деле ничего сложного в этом нет, как и во многом другом в области компьютерных технологий, если разобраться в сути слов и попробовать практически испытать работу элементов этих технологий. В данном разделе речь пойдет о программах, работающих на компьютере *в фоновом режиме* при использовании, например, антивирусных программ или элемента управления громкостью. Такие

программы внешне напоминают о своем существовании лишь присутствием *значка в области уведомлений* панели задач в правом нижнем углу экрана. Кстати, программу «Время по Гринвичу в полупрозрачной форме», рассмотренную в предыдущем разделе, уместно было бы запрограммировать именно со значком в области уведомлений. Однако, чтобы замысел одного примера не смешивался с замыслом другого, а программный код при этом выглядел *компактно* и *выразительно*, эти программы приведены в разных разделах.

Программа, рассматриваемая в данном разделе, также способна работать в фоновом режиме. Внешне программа имеет форму с меткой, где записаны возможности программы, командную кнопку, при щелчке на которой форма исчезает, но зато появляется значок в области уведомлений. При этом в оперативной памяти программа существует и продолжает работать в фоновом режиме. При щелчке правой кнопкой мыши на этом значке появляется *контекстное меню* из двух пунктов. Один пункт меню сообщает пользователю время работы данного компьютера с тех пор, как стартовала операционная система, а другой пункт меню обеспечивает выход из данной программы. При двойном щелчке на значке появляется опять исходная форма, а значок исчезает. Таким образом, проектируемая программа способна переводить свою работу в фоновый режим и обратно.

Для программирования данной задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. В конструкторе формы из панели элементов перенесем в форму метку, командную кнопку, элемент управления NotifyIcon и контекстное меню ContextMenuStrip. Щелкая на значке ContextMenuStrip, спроектируем два пункта контекстного меню: Время старта ОС и Выход. Теперь перейдем на вкладку программного кода, где напишем текст программы, представленный в листинге 12.5.

Листинг 12.5. Перевод работы программы в фоновый режим и обратно

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Эта программа сообщает пользователю время, прошедшее с момента
        // старта операционной системы на данном компьютере. Доступ к этой
        // информации реализован через контекстное меню значка в области
        // уведомлений панели задач
    private: System::Void Form1_Load(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 this->Text = "Создаю значок в области уведомлений";
                 // Задаем размер метки
                 label1->Size = System::Drawing::Size(292, 90);
                 // Разрешаем продолжение текста в метке с новой строки
                 label1->AutoSize = false;
```

```
            label1->Text =
                "При щелчке на командной кнопке данная программа " +
                "размещает значок в область уведомлений. Щелкните " +
                "правой кнопкой мыши на этом значке для доступа к " +
                "контекстному меню с пунктами \"Время\"" +
                "работы ОС\" и \"Выход\"" +
                ". Двойной щелчок на значке возвращает " +
                "на экран данную форму.";
            button1->Text = "Разместить значок в область уведомлений";
            //            SystemIcons - это стандартные значки Windows
            notifyIcon1->Icon = SystemIcons::Shield;   // значок щита
            //            SystemIcons::Information - значок сведений
            notifyIcon1->Text = "Время работы ОС";
            notifyIcon1->Visible = false;
            // "Привязываем" контекстное меню к значку notifyIcon1
            notifyIcon1->ContextMenuStrip = contextMenuStrip1;
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Скрыть экранную форму и сделать видимым значок
            // в области уведомлений
            this->Hide(); notifyIcon1->Visible = true;
        }
    private: System::Void notifyIcon1_MouseDoubleClick(System::Object^ sender,
            System::Windows::Forms::MouseEventArgs^  e)
        {
            // Чтобы получить пустой обработчик этого события, можно,
            // например, в конструкторе формы дважды щелкнуть на значке
            // notifyIcon1
            notifyIcon1->Visible = false; this->Show();
        }
    private: System::Void времяСтартаToolStripMenuItem_Click(
            System::Object^  sender, System::EventArgs^  e)
        {
            double Время_работы_ОС_в_минутах =
                System::Environment::TickCount / 1000 / 60;
            // Формат "{0:F0}» округляет Double до целого значения:
            String ^ ss = String::Format("ОС стартовала {0:F0} минут назад",
                Время_работы_ОС_в_минутах);
            MessageBox::Show(ss);
        }
    private: System::Void выходToolStripMenuItem_Click(System::Object^  sender,
                                                       System::EventArgs^  e)
        {
            delete notifyIcon1; // Стиль C#: notifyIcon1.Dispose();
            Application::Exit();
        }
    };
}
```

Как видно из программного кода, при обработке события загрузки формы выполняем очевидные инициализации свойств элементов управления, в том числе для компонента notifyIcon1. Для него среди стандартных значков Windows выбираем *значок щита* (notifyIcon1->Icon = SystemIcons::Shield), а сам значок делаем невидимым (notifyIcon1->Visible = false). При обработке события «щелчок на командной кнопке» скрываем экранную форму и делаем видимым значок в области уведомлений. Обработка события «двойной щелчок» на этом значке обеспечивает *противоположное действие*, а именно показывает форму, но скрывает значок. Две последние процедуры обрабатывают выбор пунктов меню Время старта ОС и Выход. На рис. 12.3 и 12.4 приведены фрагменты работы программы в обычном и фоновом режимах.

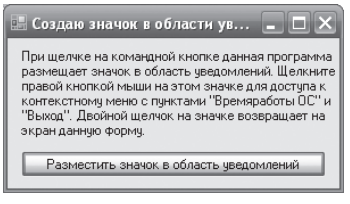

Рис. 12.3. Внешний вид формы при работе программы в обычном режиме

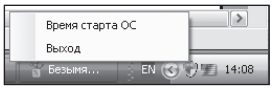

Рис. 12.4. Контекстное меню при работе программы в фоновом режиме

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке Значок_в_области_уведомлений.

Пример 101. Нестандартная форма. Перемещение формы мышью

Обычно экранную форму, любое Windows-окно мы перемещаем при нажатой левой кнопке мыши, «зацепив» за заголовок, то есть за синюю полосу вверху окна. В заголовке расположены кнопки Свернуть, Свернуть в окно и Закрыть. Для демонстрации работы с событиями мыши, с аргументами процедуры обработки этих событий MouseEventArgs напишем программу, с помощью которой появляется возможность перемещать форму, «зацепив» ее указателем мыши за любую часть формы, а не только за заголовок. Подобный пример, только на языке Visual Basic, приведен в книге «Visual Basic .NET. Библия пользователя»[1].

Итак, запускаем Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++, и, таким об-

[1] *Ивьен Б., Берес Дж.* Visual Basic .NET. Библия пользователя. М.: Вильямс, 2002. С. 591.

разом, мы получим стандартную экранную форму. Далее вводим текст программы, приведенный в листинге 12.6.

Листинг 12.6. Перемещение формы мышью

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Нестандартная форма. Программа позволяет перемещать форму мышью,
        // "зацепив" ее не только за заголовок, а в любом месте формы
        bool Перемещение;
        // Перемещаем форму только тогда, когда Перемещение = true
        int MouseDownX;
        int MouseDownY;
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 Перемещение = false;
             }
    private: System::Void Form1_MouseDown(System::Object^  sender,
                 System::Windows::Forms::MouseEventArgs^  e)
             {
                 // Здесь обрабатываем событие "щелчок любой кнопкой мыши".
                 // Во время щелчка левой кнопкой мыши запоминаем текущее
                 // положение мыши
                 if (e->Button == System::Windows::Forms::
                                      MouseButtons::Left)
                 {
                     Перемещение = true;
                     MouseDownX = e->X;
                     MouseDownY = e->Y;
                 }
             }
    private: System::Void Form1_MouseUp(System::Object^  sender,
                 System::Windows::Forms::MouseEventArgs^  e)
             {
                 // Здесь обрабатываем событие, когда
                 // пользователь отпустил кнопку мыши
                 if (e->Button == System::Windows::Forms::
                           MouseButtons::Left) Перемещение = false;
             }
    private: System::Void Form1_MouseMove(System::Object^  sender,
                 System::Windows::Forms::MouseEventArgs^  e)
             {
                 // Здесь обрабатываем событие, когда указатель
                 // мыши перемещается в пределах формы.
```

продолжение ⇩

Листинг 12.6 (продолжение)

```
                // Перемещаем форму только тогда, когда Перемещение = true
                if (Перемещение == true)
                {
                    auto Точка = System::Drawing::Point();
                    Точка.X = this->Location.X + (e->X - MouseDownX);
                    Точка.Y = this->Location.Y + (e->Y - MouseDownY);
                    this->Location = Точка;
                }
            }
    };
}
```

Вначале программы объявлены три внешние переменные, видимые в пределах класса Form1. Одна из переменных — Булева переменная Moving — это как бы флажок, который сигнализирует, следует ли перемещать экранную форму. Если пользователь нажал левую кнопку мыши (событие MouseDown, e->Button == Left), то Moving = true, то есть следует перемещать форму. Если пользователь отпустил кнопку мыши (событие MouseUp), то Moving = false, то есть форму перемещать не следует. Остальные две переменные целого типа int предназначены для запоминания текущего положения (e->X и e->Y) мыши. Перемещаем форму только тогда, когда перемещается указатель мыши (событие MouseMove), и при этом флажок Moving = true.

Убедиться в работоспособности программы можно, открыв решение ПеремещениеФормы.sln в папке ПеремещениеФормы.

Пример 102. Воспроизведение звуков операционной системы

Вы знаете, читатель, что операционная система Windows имеет в своем арсенале множество звуков, которые она использует, «обыгрывая» то или иное событие. В данном примере покажем, как мы, разработчики программного обеспечения, можем использовать эти звуки в своих собственных программах. Для демонстрации некоторых возможностей напишем программу, содержащую несколько командных кнопок на экранной форме, при нажатии на каждую из которых будем извлекать тот или иной звук.

Запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Программу назовем Звуки_ОС. Из панели элементов перенесем в форму шесть командных кнопок. Получив соответственно шесть пустых обработчиков событий введем текст программы, приведенный в листинге 12.7.

Листинг 12.7. Воспроизведение звуков

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа воспроизводит некоторые звуки операционной системы Windows
        System::Media::SoundPlayer ^ Плеер;
    private: System::Void Form1_Load(System::Object^  sender,
                System::EventArgs^  e)
             {
                this->Text = "Звуковые сигналы ОС";
                Плеер = gcnew System::Media::SoundPlayer();
                // Задаем названия командных кнопок:
                button1->Text = "Звук торжества \"та-да\"";
                button2->Text = "Звук завершения процесса";
                button3->Text = "Звук ошибки";
                button4->Text = "Частота звука = 1000 гц";
                button5->Text = "Вход в Windows XP";
                button6->Text = "Выход из Windows XP";
             }
    private: System::Void button1_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                //Воспроизведение звукового WAV-файла
                Плеер->SoundLocation = "c:\\windows\\media\\tada.wav";
                Плеер->Play();
             }
    private: System::Void button2_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Класс SystemSounds получает звуки, связанные с набором
                // типов звуковых событий операционной системы Windows:
                System::Media::SystemSounds::Asterisk->Play();
             }
    private: System::Void button3_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                System::Media::SystemSounds::Beep->Play();
             }
    private: System::Void button4_Click(System::Object^  sender,
                System::EventArgs^  e)
             {
                // Звуковой сигнал частотой 1000 Гц и длительностью 0::5 секунд:
                Console::Beep(1000, 500);
             }
```

продолжение ⇘

Листинг 12.7 (продолжение)

```
    private: System::Void button5_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 Плеер->SoundLocation =
                     "c:\\windows\\media\\Вход в Windows XP.wav";
                 Плеер->Play();
             }
    private: System::Void button6_Click(System::Object^  sender,
                 System::EventArgs^  e)
             {
                 Плеер->SoundLocation =
                     "c:\\windows\\media\\Выход из Windows XP.wav";
                 Плеер->Play();
             }
    };
}
```

Мы видим, что программный код довольно простой (или «прозрачный», как говорят программисты). Следует отметить, что многие звуки находятся в системной папке C:\Windows\Media, и вы можете найти здесь что-либо подходящее для ваших целей.

Внешний вид пользовательского интерфейса программы показан на рис. 12.5.

Убедиться в работоспособности программы можно, открыв решение Звуки_ОС.sln в папке Звуки_ОС.

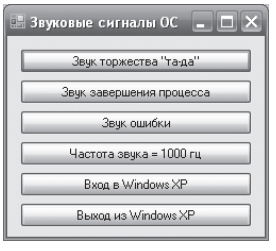

Рис. 12.5. Выбор некоторых звуков операционной системы Windows

Пример 103. Проигрыватель Windows Media Player 11

Используя возможность подключения к проекту соответствующей библиотеки, можно написать программу для проигрывания различных файлов мультимедиа AVI, MPEG, MP3, MP4, FLV, WMV и других форматов. Причем ядром вашей программы будет, например, один из самых современных плееров Windows Media Player версии 11. А вот пользовательский интерфейс, то есть способ управления плеером: кнопки на форме или в виде выпадающего меню, цвет оболочки, значения параметров по умолчанию, размер экрана и пр., вы можете создать свой собственный.

Итак, запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Назовем этот новый проект Player. Теперь нам нужно добавить на проектируемую форму элемент управления Windows Media Player. Однако в панели Toolbox нет такого элемента. Чтобы добавить на панель Toolbox элемент управления Windows Media Player, следует подключить библиотеку Windows Media Player. Для этого, щелкнув правой

кнопкой мыши в пределах панели элементов, в контекстном меню выберем пункт Choose Items (Выбрать элементы). Далее на вкладке COM Components установим флажок Windows Media Player и щелкнем на кнопке OK. При этом на панели элементов появится значок Windows Media Player, его-то мы и перетащим на форму. Растяните изображения формы и плеера так, как вам того хочется. После этого в папках проекта Debug и Interop должны появиться два файла: AxInterop.WMPLib.1.0.dll и Interop.WMPLib.1.0.dll. В этих двух файлах находятся динамически вызываемые объекты для работы элемента управления Windows Media Player.

Далее для программирования меню перенесем с панели Toolbox в форму значок MenuStrip. Слева вверху формы появится проект первого пункта меню Type Here. Здесь мы проектируем пункты меню будущей программы. Меню Файл имеет пункты Открыть и Выход. А меню Сервис *содержит пункты, показанные на рис.* 12.6.

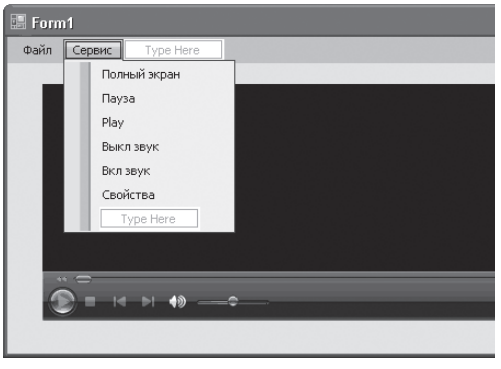

Рис. 12.6. Проектирование пунктов меню

Программный код приведен в листинге 12.8.

Листинг 12.8. Проигрыватель Windows Media Player 11

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // Программа реализует функции проигрывателя Windows Media Player 11
        OpenFileDialog ^ openFileDialog1;
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 openFileDialog1 = gcnew OpenFileDialog();
                 // ВЕРСИЯ ПЛЕЕРА
                 this->Text = "Windows Media Player, версия = " +
                     axWindowsMediaPlayer1->versionInfo;
             }
```

продолжение ⇩

Листинг 12.8 (продолжение)

```
private: System::Void открытьToolStripMenuItem_Click(System::Object^  sender,
             System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Открыть.
             // Пользователь выбирает файл:
             openFileDialog1->ShowDialog();
             // Передача плееру имени файла
             axWindowsMediaPlayer1->URL = openFileDialog1->FileName;
             // axWindowsMediaPlayer1->URL = "C:\\WINDOWS\\Media\\tada.wav";
             // Команда на проигрывание файла
             axWindowsMediaPlayer1->Ctlcontrols->play();
             // ИЛИ ТАК: передача имени файла и сразу PLAY
             // axWindowsMediaPlayer1->openPlayer(openFileDialog1->FileName);
         }
private: System::Void выходToolStripMenuItem_Click(System::Object^ sender,
             System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Выход
             Application::Exit();
         }
private: System::Void полныйЭкранToolStripMenuItem_Click(
             System::Object^  sender, System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Полный экран.
             // Если плеер пребывает в состоянии PLAY, то можно
             // перейти в режим полного экрана:
             if (axWindowsMediaPlayer1->playState ==
                 WMPLib::WMPPlayState::wmppsPlaying)
                 axWindowsMediaPlayer1->fullScreen = true;
         }
private: System::Void паузаToolStripMenuItem_Click(
             System::Object^  sender, System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Пауза
             axWindowsMediaPlayer1->Ctlcontrols->pause();
         }
private: System::Void playToolStripMenuItem_Click(
             System::Object^  sender, System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Play
             axWindowsMediaPlayer1->Ctlcontrols->play();
         }
private: System::Void выклЗвукToolStripMenuItem_Click(
             System::Object^  sender, System::EventArgs^  e)
         {
             // ПУНКТ МЕНЮ Выкл звук
             axWindowsMediaPlayer1->settings->mute = true;
         }
```

```
    private: System::Void вклЗвукToolStripMenuItem_Click(
                 System::Object^  sender, System::EventArgs^  e)
             {
                 // ПУНКТ МЕНЮ Вкл звук
                 axWindowsMediaPlayer1->settings->mute = false;
             }
    private: System::Void свойстваToolStripMenuItem_Click(
                 System::Object^  sender, System::EventArgs^  e)
             {
                 // ПУНКТ МЕНЮ Свойства
                 axWindowsMediaPlayer1->ShowPropertyPages();
             }
    };
}
```

Как видно из текста программы, перед загрузкой формы объявлено создание *экземпляра класса* OpenFileDialog. Этот объект можно было бы перенести в форму из панели элементов управления, как мы это делали в примере 25 (см. главу 4), а можно его создать программно, как в данном программном коде.

При обработке события загрузки формы задаем текст строки заголовка формы «Windows Media Player, версия = ». При этом номер версии получаем из свойства versionInfo.

При обработке события «щелчок мышью по пункту меню Открыть» функция ShowDialog обеспечивает выбор нужного файла мультимедиа. Передача плееру имени выбранного файла происходит через свойство плеера URL. Далее следует команда на проигрывание файла, хотя можно было бы предусмотреть эту команду в отдельном пункте меню. В комментарии приведена возможность проигрывания файла мультимедиа при передаче имени этого файла одной функцией openPlayer.

Далее по тексту программы реализованы очевидные функции в соответствующих обработчиках событий: переход в режим полного экрана (это возможно, только если плеер пребывает в состоянии проигрывания файла wmppsPlaying), Пауза, Play, ВклЗвук, ВыклЗвук, вызов меню Свойства, выход из программы.

Фрагмент работы программы представлен на рис. 12.7.

Рис. 12.7. Проигрыватель в режиме воспроизведения

Убедиться в работоспособности программы можно, открыв решение Player.sln в папке Player.

Пример 104. Воспроизведение только звуковых файлов

Представим ситуацию, когда нам нужно воспроизводить только звуковые файлы, такие как mp3, wma, mid и пр. Такой ситуацией может быть ваше желание воспроизводить музыку в фоновом режиме в ходе работы вашей программы. Следует учесть, что такой сервис не должен быть навязчивым, и поэтому необходимо предусмотреть возможность выключения такого музыкального сопровождения.

Для программирования этой задачи запустим Visual Studio 2010 и в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Назовем этот проект PlayerТолькоЗвук. Теперь из панели элементов управления Toolbox перетащим в проектируемую экранную форму три командые кнопки Button. Первая кнопка нам нужна для выбора звукового файла, который будет подлежать воспроизведению, а остальные две — для регулировки громкости.

Далее наступает очень важный момент — следует добавить ссылку на нужную объектную библиотеку. Для этого выбираем пункты меню Project ▸ Properties (Свойства) ▸ Add Reference (Добавить ссылку) и на вкладке Browse (Обзор) находим файл C:\Windows\system32\wmp.dll. Заметьте, что в этой программе мы не добавляли Windows Media Player в графический интерфейс формы. Несмотря на то что в данной программе, как и в предыдущей, мы добавили ссылку на разные библиотеки, на самом деле мы используем один и тот же Widows Media Plaer, причем одной и той же версии. Теперь перейдем на вкладку программного кода и введем текст, представленный в листинге 12.9.

Листинг 12.9. Воспроизведение звуковых файлов

```
            // ..............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
        }
#pragma endregion
        // Программа предназначена для воспроизведения только звуковых файлов
        WMPLib::WindowsMediaPlayer ^ Плеер;
        OpenFileDialog ^ ОткрытьФайл;
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
             {
                 // Добавляем ссылку на объектную библиотеку. Для этого выбираем
                 // пункты меню Project ▸ Properties ▸ Add Reference и на
                 // вкладке Browse (Обзор) найдем файл C:\Windows\system32\wmp.dll
```

```
            Плеер = gcnew WMPLib::WindowsMediaPlayer();
            ОткрытьФайл = gcnew OpenFileDialog();
            // ВЕРСИЯ ПЛЕЕРА
            this->Text = "Windows Media Player " + Плеер->versionInfo;
            button1->Text = "Файл";
            button2->Text = "Увеличить громкость";
            button3->Text = "Уменьшить громкость";
            // Задаем путь к файлу:
            // Плеер->URL = "C:\\VenSalvrme.wma";
            Плеер->URL = "C:\\WINDOWS\\Media\\town.mid";
            // Плеер->URL = "C:\\juliana-best_of_best.mp3";
            Плеер->settings->volume = 10;
            // Команда на проигрывание файла
            Плеер->controls->play();
        }
    private: System::Void button1_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Кнопка Файл:
            // Пользователь выбирает файл:
            ОткрытьФайл->ShowDialog();
            // Передача плееру имени файла
            Плеер->URL = ОткрытьФайл->FileName;
            // Команда на проигрывание файла
            Плеер->controls->play();
        }
    private: System::Void button2_Click(System::Object^  sender,
                                        System::EventArgs^  e)
        {
            // Увеличиваем громкость с каждым щелчком:
            Плеер->settings->volume = Плеер->settings->volume + 10;
        }
    private: System::Void button3_Click(System::Object^  sender,
            System::EventArgs^  e)
        {
            // Уменьшаем громкость с каждым щелчком:
            Плеер->settings->volume = Плеер->settings->volume - 10;
        }
    };
}
```

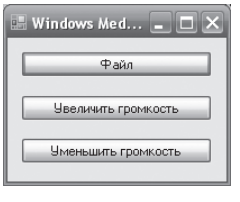

Рис. 12.8. Проигрыватель в режиме воспроизведения

В программном коде при обработке события загрузки формы создаем новый объект класса WindowsMediaPlayer. Задаем путь к звуковому файлу в свойстве URL. Причем если нам известен адрес этого файла в сети Интернет, мы можем указать его в этом свойстве. При обработке события «щелчок мышью на первой кнопке» организуем диалог на открытие файла, и после выбора файла пользователем подаем команду на проигрывание выбранного файла.

Внешний вид графического интерфейса данной программы представлен на рис. 12.8

Убедиться в работоспособности программы можно, открыв решение PlayerТолькоЗвук.sln в папке PlayerТолькоЗвук.

Пример 105. Программирование контекстной справки. Стандартные кнопки в форме

Напишем программу, демонстрирующую организацию помощи пользователю вашей программы. В данной программе предусмотрена экранная форма, которая в заголовке имеет кнопку Справка (*в виде вопросительного знака*) и кнопку Закрыть. Здесь реализована контекстная помощь — после щелчка мыши на кнопке Справка можно получить контекстную всплывающую подсказку *по тому или иному элементу управления, находящемуся в форме*.

Для этого обычным путем создадим новый проект из шаблона Windows Forms Application Visual C++, получим стандартную форму. Используя панель Toolbox, добавим в форму три текстовых поля, три метки и две кнопки, как показано на рис. 12.9.

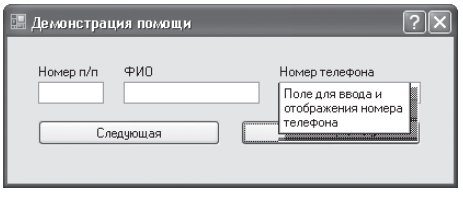

Рис. 12.9. Фрагмент работы программы с контекстной справкой

Таким образом, мы получили пользовательский интерфейс редактирования таблицы телефонов, знакомый нам из предыдущих разделов.

На вкладке программного кода напишем следующий текст (листинг 12.10).

Листинг 12.10. Программирование контекстной справки

```
            // ...............
            // Программный код, расположенный выше, создан средой Visual Studio
            // автоматически, поэтому автором не приводится
            this->ResumeLayout(false);
            this->PerformLayout();
        }
#pragma endregion
        // В программе предусмотрена экранная форма, которая в заголовке имеет
        // только кнопку Справка (в виде вопросительного знака) и кнопку
        // Закрыть. Здесь реализована контекстная помощь, когда после щелчка
        // мыши на кнопке Справка можно получить контекстную всплывающую
        // подсказку по тому или иному элементу управления, находящемуся в форме
    private: System::Void Form1_Load(System::Object^  sender,
                                     System::EventArgs^  e)
```

```
        {
            MaximizeBox = false; // - отмена кнопки Развернуть
            MinimizeBox = false; // - отмена кнопки Свернуть
            // Чтобы стиль формы содержал кнопку помощи,
            // то есть вопросительный знак
            HelpButton = true;
            this->Text = "Демонстрация помощи";
            button1->Text = "Следующая"; button2->Text = "Предыдущая";
            textBox1->Clear(); textBox2->Clear(); textBox3->Clear();
            label1->Text = "Номер п/п"; label2->Text = "ФИО";
            label3->Text = "Номер телефона";
            HelpProvider ^ helpProvider1 = gcnew HelpProvider();
            helpProvider1->SetHelpString(textBox1,
                "Здесь отображаются номера записи по порядку");
            helpProvider1->SetHelpString(textBox2,
                "Поле для редактирования имени абонента");
            helpProvider1->SetHelpString(textBox3,
                "Поле для ввода и отображения номера телефона");
            helpProvider1->SetHelpString(button1,
                "Кнопка для перехода на следующую запись");
            helpProvider1->SetHelpString(button2,
                "Кнопка для перехода на предыдущую запись");
            // Назначаем, какой help-файл будет вызываться
            // при нажатии клавиши <F1>
            helpProvider1->HelpNamespace = "mspaint.chm";
        }
    };
}
```

При обработке события загрузки формы мы устанавливаем все параметры формы. Так, чтобы строка заголовка формы содержала только кнопку Справка (вопросительный знак), следует запретить в форме кнопки Развернуть и Свернуть:

```
MaximizeBox = false;
MinimizeBox = false;
```

а вместо них задать кнопку Справка: HelpButton = true. Далее объявляем объект helpProvider1, который мы могли бы добавить в форму, используя Toolbox. Затем для каждого из текстовых полей и кнопок задаем текст контекстной подсказки. Эта подсказка будет работать так же, как и подсказка ToolTip (см. пример 8 в главе 1). Свойству HelpNamespace назначаем help-файл, который будет вызываться при нажатии функциональной клавиши F1. Мы указали файл mspaint.chm, который является справочным для графического редактора MS Paint, встроенного в ОС Windows, то есть он имеется на любом компьютере под управлением Windows. Формат CHM был разработан Microsoft для гипертекстовых справочных систем. У CHM-файлов, как правило, есть Содержание — отдельная панель со списком статей для упрощения навигации. В наличии содержания, пожалуй, и заключается главное отличие CHM-файлов от использовавшихся ранее HLP-файлов справки Windows.

Убедиться в работоспособности программы можно, открыв решение Help.sln в папке Help.

Создание инсталляционного пакета для распространения программы

Допустим, вы отладили свою первую (или очередную) программу и теперь вам надо *создать дистрибутив* (инсталляционный пакет) для распространения программы (дистрибуции). Инсталляционный пакет будет включать в себя набор файлов (setup.exe, *.msi, *.dll, *.tlb и т. д.). Пользователь вашей программы должен запустить setup.exe, и в режиме диалога будет происходить распаковка msi-архива, регистрация в системном реестре и пр. Для создания таких инсталляционных пакетов существует множество программ-инсталляторов, например InstallShild, InnoSetup и др. Однако можно воспользоваться системой Visual Studio 2010. Покажем процесс создания инсталляционного пакета на примере программы управления функциями AutoCAD (см. пример 60 в главе 9).

Запустим систему Visual Studio 2010, закажем новый проект, в окне New Project раскроем узел Other Project Types и узел Setup and Deployment (Установка и распространение), а затем выберем пункт Visual Studio Installer и шаблон Setup Project. В строке Location укажем папку, в которой будет располагаться наш инсталляционный пакет, например папку C:\New, и нажмем кнопку OK. Далее в пункте меню Project выберем команду Add ▸ File. Здесь добавим exe-файл и прочие файлы, которые мы хотим поместить в msi-архив инсталляционного пакета, в нашем случае это файл ACADm.exe, также добавим файл Interop.AutoCAD.dll. При этом в окне Solution Explorer (Обозревателе решений) мы увидим зависимые от exe-файла ссылки на tlb-, dll- и msm-файлы. В нашем случае в окне Solution Explorer появится файл ACAD.TLB. Можно также добавить необходимые на ваш взгляд компоненты, например какой-либо ocx-файл или тестовый файл с какими-либо пояснениями. После создания дистрибутива все эти файлы окажутся внутри msi-архива.

Теперь дадим команду на непосредственное создание инсталляционного пакета Build ▸ Build Setup1. После завершения данного процесса в папке C:\New\Debug появятся файлы Setup.exe и Setup1.msi. Эти файлы скопируем на какой-либо съемный носитель, например на компакт-диск или флэш-накопитель. Теперь инсталляционный пакет готов для распространения.

На компьютере вашего пользователя запустим setup.exe, далее в диалоге укажем папку, например C:\Program Files\Acadm\, и в конце диалога получим сообщение от системы о завершении инсталляции. Теперь ваша программа находится в указанной папке, в ней — exe-файл и все связанные с ним файлы dll- и tlb-файлы. Программа зарегистрирована в системном реестре, и вы можете найти ее среди установленных программ в Панели управления в разделе Установка и удаление программ.

Желаю вам, уважаемые читатели, не только получить удовольствие от процесса программирования на MS Visual C++, но и при этом заработать достойные деньги.

Извините, все.

Приложение. Описание архива с файлами примеров

Таблица П1. Содержимое архива с файлами примеров

Название папки	Описание программы	Номер примера
First	Первая простейшая программа с текстовой меткой и кнопкой, щелчок на которой вызывает появление диалогового окна, в котором написано: «Всем привет!»	1
Hover	Простейшая программа с экранной формой, меткой, командной кнопкой и диалоговым окном, отслеживание события MouseHover	2
ВыборДаты	Программа выбора нужной даты	3
Корень	Программа вводит через текстовое поле число, при щелчке на командной кнопке извлекает из него квадратный корень и выводит результат на метку label1. В случае ввода не-числа сообщает пользователю об этом выводом предупреждения красного цвета также на метку label1	4
Паспорт	Программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, «находящийся за спиной пользователя», увидит только звездочки	5
Флажок1	Программа управляет стилем шрифта текста, введенного на метку Label посредством флажка CheckBox	6
Флажок2	Совершенствование предыдущей программы. Побитовый оператор ^ — исключающее ИЛИ	7
Вкладки	Программа, позволяющая выбрать текст из двух вариантов, задать цвет и размер шрифта для этого текста на трех вкладках TabControl с использованием переключателей RadioButton	8

Название папки	Описание программы	Номер примера
Visible	Программа пишет в метку Label некий текст, а пользователь с помощью командной кнопки делает этот текст либо видимым, либо невидимым. Здесь использовано свойство Visible. При зависании мыши над кнопкой появляется подсказка «Нажми меня» (свойство ToolTip)	9
Комби	Программа, реализующая функции калькулятора. Здесь для отображения вариантов выбора арифметических действий используется комбинированный список ComboBox	10
Unico	Программа демонстрирует возможность вывода в текстовую метку, а также в диалоговое окно MessageBox греческих букв. Программа приглашает пользователя ввести радиус *R*, чтобы вычислить длину окружности	11
Сумма	Программа организует ввод двух чисел, их сложение и вывод суммы на консоль	12
ТаблКорней	Консольное приложение задает цвета и заголовок консоли, а затем выводит таблицу извлечения квадратного корня от нуля до десяти	13
ConsoleMessageBox	Консольное приложение выводит в окно MessageBox текущую дату и время в различных форматах, используя String::Format	14
ССылкаНаVisualBasic	В данном консольном приложении Visual C++ используем функции Visual Basic. Приложение приглашает пользователя ввести два числа, анализирует, числа ли ввел пользователь, и выводит результат суммирования на экран. При этом используем функции Visual Basic: InputBox, IsNumeric (для контроля, число ли ввел пользователь) и MsgBox	15
Месяцы	Программа создает словарь данных типа Dictionary и записывает в этот словарь названия месяцев и количество дней в каждом месяце. Ключом словаря является название месяца, а значением — количество дней. Используя цикл for each, программа выводит на консоль только те месяцы, количество дней в которых равно 30	16

Название папки	Описание программы	Номер примера
Мониторинг	Программа отображает координаты курсора мыши относительно экрана и элемента управления. Программа содержит форму, список элементов ListBox и два текстовых поля. Программа заполняет список ListBox данными о местоположении и изменении положения курсора мыши. Кроме того, в текстовых полях отображаются координаты положения курсора мыши относительно экрана и элемента управления ListBox	17
NewButton	Программа создает командную кнопку в форме «программным» способом, то есть с помощью написания программного кода, не используя при этом панель элементов управления Toolbox. Программа задает свойства кнопки: ее видимость, размеры, положение, надпись на кнопке и подключает событие «щелчок на кнопке»	18
ДваСобытияОднаПроц	В форме имеем две командные кнопки, и при нажатии указателем мыши на любую из них получаем номер нажатой кнопки. При этом в программе предусмотрена только одна процедура обработки событий	19
Калькулятор	Программа Калькулятор с кнопками цифр. Управление калькулятором возможно только мышью. Данный калькулятор выполняет только арифметические операции	20
ССылкиLinkLabel	Программа обеспечивает ссылку для посещения почтового сервера www.mail.ru, ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот с помощью элемента управления LinkLabel	21
СобытияКлавиатуры	Программа, информирующая пользователя о тех клавишах и о комбинациях клавиш, которые тот нажал	22
ТолькоЦифры	Программа анализирует каждый символ, вводимый пользователем в текстовое поле формы. Если символ не является цифрой или Backspace, то текстовое поле получает запрет на ввод такого символа	23

Название папки	Описание программы	Номер примера
ТолькоЦифры+ТчкОрЗпт	Программа разрешает ввод в текстовое поле только цифровых символов, а также разделитель целой и дробной частей числа (то есть точки или запятой)	24
Пгм-ноВызвать-КликКнопки	На экранной форме имеем две кнопки. Щелчок на первой кнопке вызывает появление окна с сообщением о произошедшем событии нажатия первой кнопки. Щелкая на второй кнопке, мы имитируем нажатие первой кнопки путем программного вызова события нажатия первой кнопки	25
TxtUnicode	Программа для чтения/записи текстового файла в кодировке Unicode	26
TXT_1251	Программа для чтения/записи текстового файла в кодировке Windows 1251	27
ТекстовыйРедактор	Простой текстовый редактор	28
Тестирование	Программа тестирует студента по какому-либо предмету обучения	29
RtfРедактор	Программа простейшего RTF-редактора	30
ВводКаталогаКоординат	Чтение текстового файла, содержащего каталог координат. В каждой строке файла должны быть записаны координаты одной точки (четыре числа). При этом в качестве разделителя целой и дробной частей пользователь может использовать точку и/или запятую. Между числами может быть сколько угодно пробельных символов и/или знаков табуляции ('\t'). Пользователь может пропускать пустые строки в середине каталога и/или в конце файла. Программа должна «понять» введенный текст и вывести распознанный каталог координат в текстовое поле экранной формы	31
TxtPrint	Программа позволяет открыть в стандартном диалоге текстовый файл, просмотреть его в текстовом поле без возможности изменения текста (ReadOnly) и при желании пользователя вывести этот текст на принтер	32
ReadWriteBin	Программа для чтения/записи бинарных файлов с использованием потока данных	33

Название папки	Описание программы	Номер примера
SimpleImage1, SimpleImage2 и SimpleImage3	Три варианта программ, выводящих в форму растровое изображение из графического файла	34
СкроллингБольшого-Рисунка	Программа выводит изображение из растрового файла в PictureBox, который размещен на элементе управления Panel, с возможностью прокрутки изображения	35
РисФигур	Программа позволяет рисовать в форме графические примитивы: окружность, отрезок, прямоугольник, сектор, текст, эллипс и закрашенный сектор. Выбор того или иного графического примитива осуществляется с помощью элемента управления ListBox	36
ВыборЦвета1 и ВыборЦвета2	Два варианта программы, меняющей цвет фона формы BackColor, перебирая константы цвета, предусмотренные в Visual Studio 2010, с помощью элемента управления ListBox	37
ПрозрачныйТреугольник	Программирование экранной формы, в которой размещен прозрачный треугольник	38
ПечатьЭллипса	Программа выводит на печать (на принтер) изображение эллипса. Понятно, что таким же образом можно распечатывать и другие графические примитивы: прямоугольники, отрезки, дуги и т. д. (см. методы объекта Graphics)	39
ПечатьBMPфайла	Эта программа выводит на печать файл с расширением bmp	40
Создать_JPG	Программа формирует изображение методами класса Graphics, записывает его на диск в формате JPG-файла и выводит его отображение в экранную форму	41
СменаИзображения	Программа выводит в панель рисования текстовую строку. При щелчке на командной кнопке происходит перерисовка изображения с разворотом текстовой строки	42
РисМышью	Программа позволяет при нажатой левой или правой кнопке мыши рисовать в форме	43

Название папки	Описание программы	Номер примера
Spline	Программа строит сплайн Безье по двум узловым точкам, а две контрольные (управляющие) точки совмещены в одну. Эта одна управляющая точка отображается в форме в виде красного прямоугольника. Перемещая указателем мыши управляющую точку, мы регулируем форму сплайна (кривой)	44
График	Программа рисует график объемов продаж по месяцам. Понятно, что таким же образом можно построить любой график по точкам для других прикладных целей	45
БуферОбменаTXT	Эта программа имеет возможность записи какого-либо текста в буфер обмена, а затем извлечения этого текста из буфера обмена	46
БуферОбменаBitmap	Программа оперирует буфером обмена, когда тот содержит изображение	47
AltPrintScreen	Программная имитация нажатия клавиш Alt+PrintScreen методом Send класса SendKeys	48
БуферОбменаSaveBMP	Программа читает буфер обмена, и если данные в нем представлены в формате растровой графики, то записывает их в BMP-файл	49
ПростоTimer	Демонстрация использования таймера Timer. После запуска программы показываются форма и элемент управления «список элементов ListBox». Через две секунды в списке элементов появляется запись «Прошло две секунды», и через каждые последующие две секунды в список добавляется аналогичная запись	50
SaveСкриншот Каждые5сек	Программа после запуска каждые пять секунд делает снимок текущего состояния экрана и записывает эти снимки в файлы Pic1.BMP, Pic2.BMP и т. д. Количество таких записей в файл — пять	51
ТаблTxt	Программа формирует таблицу из двух строковых массивов в текстовом поле, используя функцию String::Format. Кроме того, в программе участвует элемент управления MenuStrip для организации раскрывающегося меню, с помощью которого пользователь выводит сформированную таблицу в Блокнот с целью последующего редактирования и вывода на печать	52

Название папки	Описание программы	Номер примера
ТаблTxtPrint	Программа формирует таблицу на основании двух массивов переменных с двойной точностью. Данную таблицу программа демонстрирует пользователю в текстовом поле TextBox. Есть возможность распечатать таблицу на принтере	53
Табл_HTM	Вывод таблицы в Internet Explorer. Здесь реализован несколько необычный подход к выводу таблицы для ее просмотра и печати на принтере. Программа записывает таблицу в текстовый файл в формате HTML. Теперь у пользователя появляется возможность прочитать эту таблицу с помощью обозревателя веб-страниц Internet Explorer или другого браузера	54
ТаблGrid	Программа заполняет два строковых массива и выводит эти массивы на экран в виде таблицы, используя элемент управления DataGridView (Сетка данных). Элемент управления DataGridView предназначен для просмотра таблиц с возможностью их редактирования	55
ХэшGridView	В данной программе используется структура данных, называемая хэш-таблицей. С ее помощью программа ставит в соответствие государствам их столицы. При этом в качестве ключей указываем названия государств, а в качестве значений — их столицы. Далее, используя элемент управления DataGridView, программа выводит эту хэш-таблицу в форму	56
ТаблВвод	Программа предлагает пользователю заполнить таблицу телефонов его знакомых, сотрудников, родственников, любимых и т. д. После щелчка на кнопке Запись данная таблица записывается на диск в файл в формате XML. Для упрощения текста программы предусмотрена запись в один и тот же файл C:\tabl.xml. При последующих запусках данной программы таблица будет считываться из этого файла, и пользователь может продолжать редактирование таблицы	57
ГауссGrid	Программа для решения системы линейных уравнений. Ввод коэффициентов предусмотрен через DataGridView	58

Название папки	Описание программы	Номер примера
СвязанныеТаблицы	Программа создает таблицу с данными о клиентах (названия организаций, контакты) и таблицу с данными о заказах (номер заказа, его объем, организация-заказчик). Между таблицами устанавливается связь посредством одинаковых столбцов (названий организаций). При этом таблица с данными о клиентах будет родительской, а таблица с данными о заказах — дочерней. Каждая строка родительской таблицы отображается со знаком «плюс». При щелчке на знаке «плюс» открывается узел, содержащий ссылку на дочернюю таблицу	59
ГрафикChart	Программа, используя элементы управления Chart и DataGridView, выводит график (диаграмму) зависимости объемов продаж от времени по месяцам. При этом в качестве источника данных указываем объект класса DataTable	60
ТаблWebHTM	В программе для отображения таблицы используется элемент управления WebBrowser. Таблица записана на языке HTML с помощью элементарных тегов <tr> (строка в таблице) и <td> (ячейка в таблице)	61
FlashWeb	Программа использует элемент управления WebBrowser для отображения Flash-файлов	62
Split	Эта программа использует элемент управления WebBrowser для отображения веб-страницы и ее HTML-кода	63
ЗаполнениеВеб_формы	Программа загружает в элемент WebBrowser начальную страницу поисковой системы http://yahoo.com. Далее, используя указатель на неуправляемый интерфейс DomDocument (свойство объекта класса WebBrowser), приводим его к указателю IHTMLDocument2. В этом случае мы получаем доступ к формам и полям веб-страницы по их именам. Заполняем поле поиска ключевыми словами для нахождения соответствующих веб-страниц, а затем для отправки заполненной формы на сервер «программно» нажимаем кнопку Submit. В итоге получим в элементе WebBrowser результат работы поисковой системы, а именно множество ссылок на страницы, содержащие указанные ключевые слова	64

Название папки	Описание программы	Номер примера
РазборВебСтраницы	Программа, используя класс WebClient, читает веб-страницу Центрального банка РФ www.cbr.ru, ищет в ее гипертекстовой разметке курс доллара США и копирует его в текстовую метку Label. Кроме того, элемент управления «графическое поле PictureBox» отображает логотип банка, используя URL-адрес этого изображения	65
Орфография1 и Орфография2	Два варианта программы, позволяющей пользователю ввести какие-либо слова или предложения в текстовое поле и после нажатия соответствующей кнопки проверить орфографию введенного текста. Для непосредственной проверки орфографии воспользуемся функцией CheckSpelling объектной библиотеки MS Word	66
ТаблицаWord	Программа вывода таблицы средствами MS Word: запускается программа, пользователь наблюдает, как запускается редактор MS Word и автоматически происходит построение таблицы	67
ExcelПи	Программа обращается к одной простой функции объектной библиотеки MS Excel для получения значения числа π = 3,14...	68
ExcelПлт	Программа использует финансовую функцию Pmt() объектной библиотеки MS Excel для вычисления суммы периодического платежа на основе постоянства сумм платежей и постоянства процентной ставки	69
ExcelСЛАУ	Программа решает систему уравнений с помощью функций объектной библиотеки MS Excel	70
ExcelГрафик	Программа строит диаграмму (график), используя объекты компонентной библиотеки MS Excel	71
ACADЭлементарный Чертеж	Программа строит средствами объектов библиотеки AutoCAD элементарный чертеж из отрезков и некоторого текста. Этот чертеж сохраняется в файле формата DWG. Конкретнее: эта программа запускает AutoCAD 2008, рисует два отрезка, одну дугу и два текстовых объекта, сохраняет чертеж в файле C:\Чертеж.dwg и завершает работу AutoCAD	72

Название папки	Описание программы	Номер примера
MatlabВызов	Программа вызывает простейшую функцию Matlab	73
MatlabСлау	Программа, подготовив команды для решения системы уравнений в среде MATLAB, вызывает его на выполнение этих команд. В результате получаем решение, которое выводим на экран с помощью MessageBox	74
Создать_PDF_1 и Создать_PDF_2	Первая программа «на лету» генерирует PDF-документ с английским текстом, а вторая — с русским	75
Создать_PDF_Табл_1 и Создать_PDF_Табл_2	Два варианта программ, создающих «на лету» PDF-файл и записывающих в этот файл таблицу данных	76
Создать_PDF_граф_1 и Создать_PDF_граф_2	Два варианта программ, выводящих в PDF-документ изображение	77
БД_SQL_Server	Создание базы данных SQL Server в среде Visual Studio 2010. В этой простейшей базе данных будет всего одна таблица, содержащая сведения о телефонах ваших знакомых, то есть в этой таблице будем иметь всего три колонки: Имя, Фамилия и Номер телефона	78
БД_SQL_Server_Консоль	Программа читает все записи из таблицы БД SQL Server (файл *.sdf) и выводит их на консоль с помощью объектов Command и DataReader	79
БДDataReader	Программа читает все записи из таблицы БД MS Access и выводит их на консоль с помощью объектов Command и DataReader	81
БДСоздание	Программа создает базу данных MS Access, то есть файл new_BD.mdb. Эта база данных будет пустой, то есть не будет содержать ни одной таблицы. Наполнять базу данных таблицами можно будет впоследствии как из программного кода C++ 2010, так и используя MS Access. В этом примере технология ADO.NET не использована	82

Название папки	Описание программы	Номер примера
БдСоздТаблицы	Программа записывает структуру таблицы в пустую базу данных MS Access. Программная реализация подключения к БД. В этой БД может не быть ни одной таблицы, то есть БД может быть пустой. Либо в БД могут быть таблицы, но название новой таблицы должно быть уникальным	83
БдДобавлЗаписи	Программа добавляет запись в таблицу базы данных MS Access. Для этого при создании экземпляра объекта Command задаем SQL-запрос на вставку (Insert) новой записи в таблицу базы данных	84
БдReaderGridView	Программа читает все записи из таблицы базы данных с помощью объектов Command, DataReader на элемент управления DataGridView (сетка данных)	85
БдАдаптерGridView	Программа читает из БД таблицу в сетку данных DataGridView с использованием объектов класса Command, Adapter и DataSet	86
БдUpdate	Программа обновляет записи (Update) в таблице базы данных MS Access	87
БдУдаленЗаписи	Программа удаляет запись из таблицы БД с использованием SQL-запроса и объекта класса Command	88
LinqМассив	Программа в строковом массиве имен выбирает имена, состоящие из шести букв. В списке выбранных имен сортируем их в алфавитном порядке, все строки переводим в верхний регистр и избавляемся от дублирования имен. При этом вместо синтаксиса LINQ-запросов (как в VB и C#) используем синтаксис LINQ-методов	89
LinqСписок1 и LinqСписок2	Два варианта программ, которые, фильтруя данные в некотором списке, формируют, таким образом, новый список	90
LinqМесяцы и LinqЦеныНаПродукты	Программы выполняют группировку некоторых данных. То есть в списке данных данные делят на две группы по заданному критерию	91

Название папки	Описание программы	Номер примера
LinqСоздатьXML-документ	Программа создает типичный XML-документ. С ее помощью можно разобраться в структуре XML-документа. В комментариях приведена терминология содержимого XML-документа: корневой элемент, вложенные элементы, имя элемента и его значение, а также атрибуты элемента, их имена и значения. XML-документ представляет телефонную книгу, содержащую имя контакта, номер домашнего и мобильного телефона. Программа после создания XML-документа отображает его на консоли, а также записывает его в файл. Если этот файл открыть с помощью MS Excel, то мы получим таблицу из трех столбцов	92
ПоискXmlЭлемента1 и ПоискXmlЭлемента2	Дана строка XML, содержащая прогнозные метеорологические показатели для Москвы на заданную дату. Варианты программ извлекают из корневого элемента XML-документа значение температуры элемента «Температура»	93
ПоискСтрокВXml	Имеем XML-данные, в которых содержится таблица с именами и телефонами, причем имена в этой телефонной табличке повторяются. Задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML) найти все строчки с именем «Витя» с помощью методов классов пространства имен System::Xml::Linq	94
LinqPiter	Программа читает XML-файл (источник), содержащий сведения о книгах по программированию, и выбирает книги автора Зиборова. Выбранные книги программа записывает в другой (производный) XML-файл	95
ПоискВDataSet	В данной программе экранная форма содержит элемент управления для отображения и редактирования табличных данных DataGridView, две командные кнопки и текстовое поле. При старте программы, если есть соответствующий файл XML, программа отображает в элемент DataGridView таблицу городов: название города и численность населения. При щелчке на кнопке «Сохранить» все изменения в таблице записываются в XML-файл. При щелчке на второй кнопке «Найти» выполняется поиск городов-миллионеров в наборе данных DataSet искомой таблицы. Результат запроса выводится в текстовое поле	96

Название папки	Описание программы	Номер примера
ПроверкаФамилии и ПроверкаЧисла	Проверка данных, вводимых пользователем, на достоверность	97
Opacity	Программа демонстрирует стандартную форму. Щелчок мышью в пределах этой формы начинает постепенный процесс исчезновения формы: форма становится все более прозрачной, а затем исчезает вовсе. Далее она постепенно проявляется снова и т. д. Еще один щелчок в пределах формы останавливает этот процесс, а следующий щелчок процесс возобновляет и т. д.	98
Гринвич	Программа в полупрозрачной экранной форме отображает текущее время по Гринвичу. Таким образом, программа демонстрирует текущее время по Гринвичу и при этом не закрывает собой другие приложения	99
Значок_в_области_уведомлений	Эта программа сообщает пользователю время, прошедшее с момента старта операционной системы на данном компьютере. Доступ к этой информации реализован через контекстное меню значка в области уведомлений панели задач	100
ПеремещениеФормы	Нестандартная форма. Программа позволяет перемещать форму мышью, «зацепив» ее не только за заголовок, а за любое место формы	101
Звуки_ОС	Программа воспроизводит некоторые звуки операционной системы Windows	102
Player	Программа реализует функции проигрывателя Windows Media Player 11	103
PlayerТолькоЗвук	Программа предназначена для воспроизведения только звуковых файлов	104
Help	В программе предусмотрена экранная форма, которая в заголовке имеет только кнопку «Справка» (в виде вопросительного знака) и кнопку «Закрыть». Здесь реализована контекстная помощь — после щелчка мыши на кнопке «Справка» можно получить контекстную всплывающую подсказку по тому или иному элементу управления, находящемуся в форме	105

Виктор Владимирович Зиборов

MS Visual C++ 2010 в среде .NET. Библиотека программиста

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Кривцов*
Ведущий редактор	*Ю. Сергиенко*
Художественный редактор	*Л. Адуевская*
Корректор	*В. Листова*
Верстка	*Е. Егорова*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 29.07.11. Формат 70x100/16. Усл. п. л. 25,800. Тираж 1000. Заказ 0000.
Отпечатано по технологии CtP в ОАО «Печатный двор» им. А. М. Горького.
197110, Санкт-Петербург, Чкаловский пр., 15.

ВАМ НРАВЯТСЯ НАШИ КНИГИ? ЗАРАБАТЫВАЙТЕ ВМЕСТЕ С НАМИ!

У Вас есть свой сайт?
Вы ведете блог?
Регулярно общаетесь на форумах? Интересуетесь литературой, любите рекомендовать хорошие книги и хотели бы стать нашим партнером?
ЭТО ВПОЛНЕ РЕАЛЬНО!

СТАНЬТЕ УЧАСТНИКОМ ПАРТНЕРСКОЙ ПРОГРАММЫ ИЗДАТЕЛЬСТВА «ПИТЕР»!

- *Зарегистрируйтесь на нашем сайте в качестве партнера по адресу* ***www.piter.com/ePartners***
- *Получите свой персональный уникальный номер партнера*
- *Выбирайте книги на сайте* ***www.piter.com,*** *размещайте информацию о них на своем сайте, в блоге или на форуме и добавляйте в текст ссылки на эти книги (на сайт* ***www.piter.com****)*

ВНИМАНИЕ! *В каждую ссылку необходимо добавить свой персональный уникальный номер партнера.*

С этого момента получайте 10% от стоимости каждой покупки, которую совершит клиент, придя в интернет-магазин «Питер» по ссылке с Вашим партнерским номером. А если покупатель приобрел не только эту книгу, но и другие издания, Вы получаете дополнительно по 5% от стоимости каждой книги.

Деньги с виртуального счета Вы можете потратить на покупку книг в интернет-магазине издательства «Питер», а также, если сумма будет больше 500 рублей, перевести их на кошелек в системе Яндекс.Деньги или Web.Money.

Пример партнерской ссылки:
http://www.piter.com/book.phtml?978538800282 – обычная ссылка
http://www.piter.com/book.phtml?978538800282**&refer=0000** – партнерская ссылка, где 0000 – это ваш уникальный партнерский номер

Подробно о Партнерской программе ИД «Питер» читайте на сайте WWW.PITER.COM

ПРЕДСТАВИТЕЛЬСТВА ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
предлагают эксклюзивный ассортимент компьютерной, медицинской, психологической, экономической и популярной литературы

РОССИЯ

Санкт-Петербург м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-73, 703-73-72; e-mail: sales@piter.com

Москва м. «Электрозаводская», Семеновская наб., д. 2/1, корп. 1, 6-й этаж
тел./факс: (495) 234-38-15, 974-34-50; e-mail: sales@msk.piter.com

Воронеж Ленинский пр., д. 169; тел./факс: (4732) 39-61-70
e-mail: piterctr@comch.ru

Екатеринбург ул. Бебеля, д. 11а; тел./факс: (343) 378-98-41, 378-98-42
e-mail: office@ekat.piter.com

Нижний Новгород ул. Совхозная, д. 13; тел.: (8312) 41-27-31
e-mail: office@nnov.piter.com

Новосибирск ул. Станционная, д. 36; тел.: (383) 363-01-14
факс: (383) 350-19-79; e-mail: sib@nsk.piter.com

Ростов-на-Дону ул. Ульяновская, д. 26; тел.: (863) 269-91-22, 269-91-30
e-mail: piter-ug@rostov.piter.com

Самара ул. Молодогвардейская, д. 33а; офис 223; тел.: (846) 277-89-79
e-mail: pitvolga@samtel.ru

УКРАИНА

Харьков ул. Суздальские ряды, д. 12, офис 10; тел.: (1038057) 751-10-02
758-41-45; факс: (1038057) 712-27-05; e-mail: piter@kharkov.piter.com

Киев Московский пр., д. 6, корп. 1, офис 33; тел.: (1038044) 490-35-69
факс: (1038044) 490-35-68; e-mail: office@kiev.piter.com

БЕЛАРУСЬ

Минск ул. Притыцкого, д. 34, офис 2; тел./факс: (1037517) 201-48-79, 201-48-81
e-mail: gv@minsk.piter.com

Ищем зарубежных партнеров или посредников, имеющих выход на зарубежный рынок.
Телефон для связи: **(812) 703-73-73**. **E-mail:** fuganov@piter.com

---

Издательский дом «Питер» приглашает к сотрудничеству авторов. Обращайтесь по телефонам: **Санкт-Петербург** – **(812) 703-73-72**, **Москва** – **(495) 974-34-50**

Заказ книг для вузов и библиотек по тел.: (812) 703-73-73.
Специальное предложение – e-mail: kozin@piter.com

Заказ книг по почте: на сайте **www.piter.com**; по тел.: (812) 703-73-74
по ICQ 413763617

УВАЖАЕМЫЕ ГОСПОДА!
КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
ВЫ МОЖЕТЕ ПРИОБРЕСТИ
ОПТОМ И В РОЗНИЦУ У НАШИХ
РЕГИОНАЛЬНЫХ ПАРТНЕРОВ.

ДАЛЬНИЙ ВОСТОК

Владивосток
«Приморский торговый дом книги»
тел./факс: (4232) 23-82-12
e-mail: bookbase@mail.primorye.ru

Хабаровск, «Деловая книга», ул. Путевая, д. 1а
тел.: (4212) 36-06-65, 33-95-31
e-mail: dkniga@mail.kht.ru

Хабаровск, «Книжный мир»
тел.: (4212) 32-85-51, факс: (4212) 32-82-50
e-mail: postmaster@worldbooks.kht.ru

Хабаровск, «Мирс»
тел.: (4212) 39-49-60
e-mail: zakaz@booksmirs.ru

ЕВРОПЕЙСКИЕ РЕГИОНЫ РОССИИ

Архангельск, «Дом книги», пл. Ленина, д. 3
тел.: (8182) 65-41-34, 65-38-79
e-mail: marketing@avfkniga.ru

Воронеж, «Амиталь», пл. Ленина, д. 4
тел.: (4732) 26-77-77
http://www.amital.ru

Калининград, «Вестер»,
сеть магазинов «Книги и книжечки»
тел./факс: (4012) 21-56-28, 6 5-65-68
e-mail: nshibkova@vester.ru
http://www.vester.ru

Самара, «Чакона», ТЦ «Фрегат»
Московское шоссе, д.15
тел.: (846) 331-22-33
e-mail: chaconne@chaccone.ru

Саратов, «Читающий Саратов»
пр. Революции, д. 58
тел.: (4732) 51-28-93, 47-00-81
e-mail: manager@kmsvrn.ru

СЕВЕРНЫЙ КАВКАЗ

Ессентуки, «Россы», ул. Октябрьская, 424
тел./факс: (87934) 6-93-09
e-mail: rossy@kmw.ru

СИБИРЬ

Иркутск, «ПродаЛитЪ»
тел.: (3952) 20-09-17, 24-17-77
e-mail: prodalit@irk.ru
http://www.prodalit.irk.ru

Иркутск, «Светлана»
тел./факс: (3952) 25-25-90
e-mail: kkcbooks@bk.ru
http://www.kkcbooks.ru

Красноярск, «Книжный мир»
пр. Мира, д. 86
тел./факс: (3912) 27-39-71
e-mail: book-world@public.krasnet.ru

Новосибирск, «Топ-книга»
тел.: (383) 336-10-26
факс: (383) 336-10-27
e-mail: office@top-kniga.ru
http://www.top-kniga.ru

ТАТАРСТАН

Казань, «Таис»,
сеть магазинов «Дом книги»
тел.: (843) 272-34-55
e-mail: tais@bancorp.ru

УРАЛ

Екатеринбург, ООО «Дом книги»
ул. Антона Валека, д. 12
тел./факс: (343) 358-18-98, 358-14-84
e-mail: domknigi@k66.ru

Екатеринбург, ТЦ «Люмна»
ул. Студенческая, д. 1в
тел./факс: (343) 228-10-70
e-mail: igm@lumna.ru
http://www.lumna.ru

Челябинск, ООО «ИнтерСервис ЛТД»
ул. Артиллерийская, д. 124
тел.: (351) 247-74-03, 247-74-09,
247-74-16
e-mail: zakup@intser.ru
http://www.fkniga.ru, www.intser.ru

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **postbook@piter.com**
- по телефону: **(812) 703-73-74**
- по почте: **197198, Санкт-Петербург, а/я 127, ООО «Питер Мейл»**
- по ICQ: **413763617**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа Вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате все виды электронных денег: от традиционных Яндекс.Деньги и Web-money до USD E-Gold, MoneyMail, INOCard, RBK Money (RuPay), USD Bets, Mobile Wallet и др.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения Вами заказа.

Все посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку Ваших покупок максимально быстро. Дату отправления Вашей покупки и предполагаемую дату доставки Вам сообщат по e-mail.

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, факс, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM