Никита Культин

Microsoft© Visual C++
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.068+800.92VisualC++
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Microsoft Visual C++ в задачах и примерах. — СПб.: БХВ-Петербург, 2010. — 272 с.: ил. + CD-ROM

ISBN 978-5-9775-0458-4

Книга представляет собой сборник программ и задач для самостоятельного решения. Примеры различной степени сложности — от простейших до приложений работы с графикой и базами данных Microsoft Access и Microsoft SQL Server Compact Edition — демонстрируют назначение базовых компонентов, раскрывают тонкости разработки приложений Windows Forms в Mirosoft Visual C++. Справочник, входящий в книгу, содержит описание базовых компонентов, событий, исключений и наиболее часто используемых функций.

На прилагаемом компакт-диске находятся проекты, представленные в книге.

Для начинающих программистов

УДК 681.3.068+800.92VisualC++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.09.09.
Формат $60\times90^{1}/_{16}$. Печать офсетная. Усл. печ. л. 17.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

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

ISBN 978-5-9775-0458-4

Оглавление

Предисловие

В последнее время в общем объеме вновь создаваемого программного обеспечения различного назначения увеличивается доля .NET-приложений, программ, ориентированных на платформу .NET. Это объясняется, прежде всего, новыми возможностями, которые предоставляет платформа прикладным программам, а также тем, что технология .NET поддерживается новейшими операционными системами.

Microsoft .NET — это технология, основанная на идее универсального программного кода, который может быть выполнен любым компьютером, вне зависимости от используемой операционной системы. Универсальность программного кода обеспечивается за счет предварительной (выполняемой на этапе разработки) компиляции исходной программы в *универсальный промежуточный код* (CIL-код, Common Intermediate Language), который во время загрузки (запуска) программы транслируется в *выполняемый*. Преобразование промежуточного кода в выполняемый производит JIT-компилятор (от Jast In Time — в тот же момент, "на лету"), являющийся элементом виртуальной выполняющей системы (Virtual Execution System, VES). Выполнение .NET-приложений в операционной системе Windows обеспечивает Microsoft .NET Framework.

Чтобы понять, что такое .NET, какие возможности она предоставляет программисту, необходимо опробовать ее в деле. Для этого нужно изучить среду и технологию разработки, понять назначение и возможности компонентов, их свойства и методы. И здесь хорошим подспорьем могут стать примеры — программы, разработанные другими программистами.

Среда разработки Microsoft Visual C++ является инструментом разработки приложений различного типа для Windows. В ней интегрированы удобный дизайнер форм, специализированный редактор кода, отладчик и другие полезные инструменты.

Книга, которую вы держите в руках, посвящена практике программирования в Microsoft Visual C++, разработке Windows Forms-приложений. В ней собраны разнообразные примеры, которые демонстрируют назначение базовых компонентов, технологии работы с файлами, графикой и базами данных.

Состоит книга из двух частей. Первая часть содержит примеры программ и задачи для самостоятельного решения. Примеры представлены в виде краткого описания, диалоговых окон и хорошо документированных текстов программ.

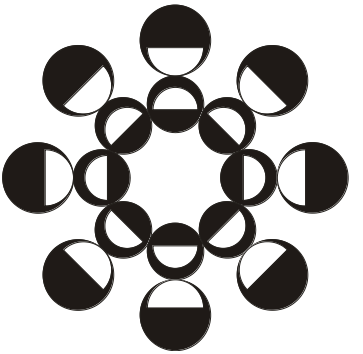
Вторая часть книги — это краткий справочник. В нем можно найти описание базовых компонентов и наиболее часто используемых функций.

Научиться программировать можно, только программируя, решая конкретные задачи. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, старайтесь понять, как работают программы. Не бойтесь экспериментировать — совершенствуйте программы, вносите в них изменения. Чем больше вы сделаете самостоятельно, тем большему научитесь!

Описание компакт-диска

Прилагаемый компакт-диск содержит проекты, приведенные в книге. Каждый проект находится в отдельном каталоге. Для того чтобы увидеть, как работает приложение, загрузите проект в Microsoft Visual C++, откомпилируйте его и затем запустите. Следует обратить внимание, что некоторые программы (например, работа с базами данных) требуют, чтобы на компьютере был установлен соответствующий сервер баз данных.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера, в папку проектов Microsoft Visual Sudio.



ЧАСТЬ I

Примеры и задачи

Базовые компоненты

В этом разделе приведены примеры, демонстрирующие назначение и технологию работы с базовыми компонентами.

Общие замечания

- Процесс создания программы состоит из двух шагов: сначала создается форма, затем — функции обработки *событий*.
- Форма создается путем помещения в нее необходимых компонентов и последующей их настройки.
- В форме практически любого приложения есть компоненты, обеспечивающие взаимодействие программы с пользователем. Такие компоненты называют базовыми.
- К базовым компонентам можно отнести:
  - `Label` — поле отображения информации;
  - `TextBox` — поле ввода-редактирования текста (данных);
  - `Button` — командная кнопка;
  - `CheckBox` — флажок;
  - `RadioButton` — радио-кнопка;
  - `ListBox` — список выбора;
  - `ComboBox` — поле редактирования со списком выбора.
- Вид компонента и его поведение определяют значения *свойств* (характеристик) компонента (описание свойств базовых компонентов можно найти в справочнике, во второй части книги).
- Основную работу в программе выполняют функции обработки событий (описание основных событий можно найти в справочнике, во второй части книги).

- Исходную информацию программа может получить из поля редактирования (компонент `TextBox`), списка (компонент `ListBox`), комбинированного списка (компонент `ComboBox`).
- Для ввода значений логического типа можно использовать компоненты `CheckBox` и `RadoiButton`.
- Результат работы программы можно вывести в поле отображения текста (компонент `Label`), в поле редактирования или в окно сообщения (метод `MessageBox::Show( )`).
- Для преобразования строки в целое число нужно использовать функцию `Convert.ToInt32()`, в дробное число — `Convert.ToDouble()`.
- Для преобразования численного значения в строку нужно использовать метод `ToString()`. В качестве параметра метода можно указать формат отображения: "`C`" — денежный с разделителями групп разрядов и обозначением валюты (`currency`); "`N`" — числовой с разделителями групп разрядов (`numeric`); "`F`" — числовой без разделителей групп разрядов (`fixed`).

Мили-километры

Программа **Мили-километры**, ее форма приведена на рис. 1.1, демонстрирует использование компонентов `TextBox` и `Label` для ввода исходных данных и отображения результата. Программа спроектирована таким образом, что в поле редактирования пользователь может ввести только правильные данные — дробное число. Значения свойств формы приведены в табл. 1.1.

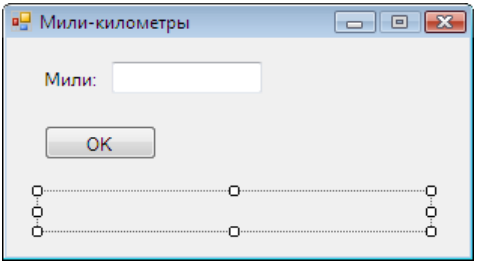


Рис. 1.1. Форма программы **Мили-километры**

Таблица 1.1. *Значения свойств формы*

Свойство	Значение	Комментарий
Text	Мили-километры	Текст заголовка
StartPosition	CenterScreen	Начальное положение окна — в центре экрана
FormBorderStyle	FixedSingle	Тонкая граница окна. Пользователь не сможет изменить размер окна путем перемещения его границы
MaximizeBox	False	Кнопка **Развернуть окно** недоступна. Пользователь не сможет развернуть окно программы на весь экран
Font	Tahoma; 9pt	Шрифт, наследуемый компонентами формы

```
// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double mile; // расстояние в милях
    double km;   // расстояние в километрах

    // Если в поле редактирования нет данных,
    // то при попытке преобразовать пустую
    // строку в число возникает исключение.
    try
    {
        mile = Convert::ToDouble(textBox1->Text);

        km = mile * 1.609344;

        label2->Text = mile.ToString("n") + " miles — " +
                       km.ToString("n") + " км";
```

```
    }

    catch (System::FormatException^ ex )
    {
        // обработка исключения:
        // — сообщение
        MessageBox::Show(
            "Надо ввести исходные данные", "Мили-километры",
            MessageBoxButtons::OK,
            MessageBoxIcon::Exclamation);

        // — установить курсор в поле редактирования
        textBox1->Focus();
    }
}

// нажатие клавиши в поле textBox
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    // Правильными символами считаются цифры,
    // запятая, <Enter> и <Backspace>.
    // Будем считать правильным символом
    // также точку, но заменим ее запятой.
    // Остальные символы запрещены.
    // Чтобы запрещенный символ не отображался
    // в поле редактирования, присвоим
    // значение true свойству Handled параметра e

    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        // цифра
        return;
    }

    if (e->KeyChar == '.')
```

```
    {
      // точку заменим запятой
      e->KeyChar = ',';
    }

    if (e->KeyChar == ',')
    {
        if (textBox1->Text->IndexOf(',') != -1)
        {
            // запятая уже есть в поле редактирования
            e->Handled = true;
        }
        return;
    }

    if (  Char::IsControl(e->KeyChar) )
    {
        // <Enter>, <Backspace>, <Esc>
        if ( e->KeyChar == (char) Keys::Enter)
            // нажата клавиша <Enter>
            // установить "фокус" на кнопку OK
            button1->Focus();
            return;
    }

    // остальные символы запрещены
    e->Handled = true;
}
```

Фунты-килограммы

Программа **Фунты-килограммы**, ее форма приведена на рис. 1.2, показывает, как можно управлять доступностью командной кнопки в зависимости от наличия данных в поле редактирования.

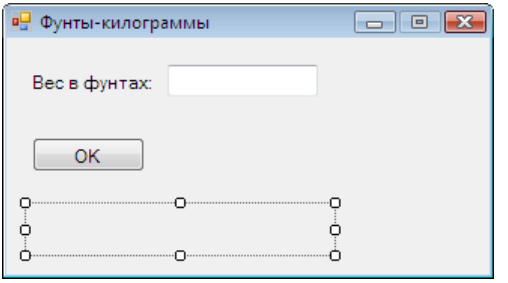


Рис. 1.2. Форма программы **Фунты-килограммы**

```
// пользователь изменил данные в поле редактирования
private: System::Void textBox1_TextChanged
           (System::Object^  sender, System::EventArgs^  e)
{
    label2->Text = ""; // очистить поле отображения
                       // результата расчета

    if (textBox1->Text->Length == 0)
        // в поле редактирования нет данных
        // сделать кнопку OK недоступной
        button1->Enabled = false;
    else
        // сделать кнопку OK доступной
        button1->Enabled = true;
}

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{
    double funt; // вес в фунтах
    double kg;   // вес в килограммах

    funt = Convert::ToDouble(textBox1->Text);
```

```
    // 1 фунт = 409,5 грамма
    kg = funt * 0.4095;

    label2->Text = funt.ToString("N") + " ф. = " +
                   kg.ToString("N") + " кг";
}

// нажатие клавиши в поле textBox1
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
        // цифры — правильные символы
        return;

    if (e->KeyChar == '.')
        // точку заменим на запятую
        e->KeyChar = ',';

    if (e->KeyChar == ',')
    {
        // в поле редактирования не может
        // быть больше одной запятой и запятая
        // не может быть первым символом
        if ( (textBox1->Text->IndexOf(',') != -1) ||
                    ( textBox1->Text->Length == 0))
        {
            e->Handled = true;
        }
        return;
    }

    if (  Char::IsControl(e->KeyChar) )
    {
        // <Enter>, <Backspace>, <Esc>
```

```
        if ( e->KeyChar == (char) Keys::Enter)
            // нажата клавиша <Enter>
            // установить курсор на кнопку OK
            button1->Focus();
            return;
    }

    // остальные символы запрещены
    e->Handled = true;
}
```

Конвертор

Программа **Конвертор**, ее форма приведена на рис. 1.3, демонстрирует обработку одной функцией событий от нескольких однотипных компонентов. Функции обработки событий `KeyPress` и `TextChanged` для компонента `textBox1` (поле **Курс**) создаются обычным образом, затем они указываются в качестве функций обработки соответствующих событий для компонента `textBox2` (поле **Цена**).

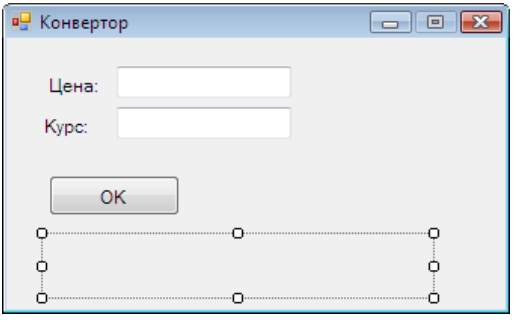


Рис. 1.3. Форма программы **Конвертор**

```
// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{
```

```
    double usd; // цена в долларах
    double k;   // курс
    double rub; // цена в рублях

    usd =  System::Convert::ToDouble(textBox1->Text);
    k =  System::Convert::ToDouble(textBox2->Text);

    rub = usd * k;

    label3->Text = usd.ToString("n") + "$ = " +
                   rub.ToString("c");
}

// Эта функция обрабатывает нажатие клавиши в полях
// редактирования textBox1 (Курс) и textBox2 (Цена).
// Сначала надо обычным образом создать функцию
// обработки события KeyPress для компонента
// textBox1, затем — указать ее в качестве
// обработчика этого же события для компонента textBox2
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
         return;

    if (e->KeyChar == '.') e->KeyChar = ',';

    if (e->KeyChar == ',')
    {

        if (sender->Equals(textBox1)) {
            if ((textBox1->Text->IndexOf(',') != -1) ||
                (textBox1->Text->Length == 0))
            {
                e->Handled = true;
            }
```

```
        }
        else {
            if ((textBox2->Text->IndexOf(',') != -1) ||
                (textBox2->Text->Length == 0))
            {
                e->Handled = true;
            }
        }
        return;
    }

    if (Char::IsControl(e->KeyChar))
    {
        if (e->KeyChar == (char)Keys::Enter)
        {
            if (sender->Equals(textBox1))
               // клавиша <Enter> нажата в поле Курс
               // переместить курсор в поле Цена
               textBox2->Focus();
            else
               // клавиша <Enter> нажата в поле Цена
               // установить фокус на кнопку OK
               button1->Focus();
        }
        return;
     }

     // остальные символы запрещены
     e->Handled = true;
}

// Функция обрабатывает событие TextChanged (изменился
// текст в поле редактирования) обоих компонентов TextBox.
// Сначала надо обычным образом создать функцию
// обработки события TextChanged для компонента
// textBox1, затем — указать ее в качестве
```

```
// обработчика события TextChanged для компонета textBox2
private: System::Void textBox1_TextChanged(System::Object^
sender, System::EventArgs^  e)
{
    label3->Text = ""; // очистить поле отображения
                       // результата расчета

    if ((textBox1->Text->Length == 0) || (textBox2->Text-
>Length == 0))
        // в поле редактирования нет данных
        // сделать кнопку OK недоступной
        button1->Enabled = false;
    else
        // сделать кнопку OK доступной
        button1->Enabled = true;
}
```

Фотоателье

Программа **Фото**, ее форма приведена на рис. 1.4, позволяет рассчитать стоимость печати фотографий. Демонстрирует использование компонента `RadioButton`.

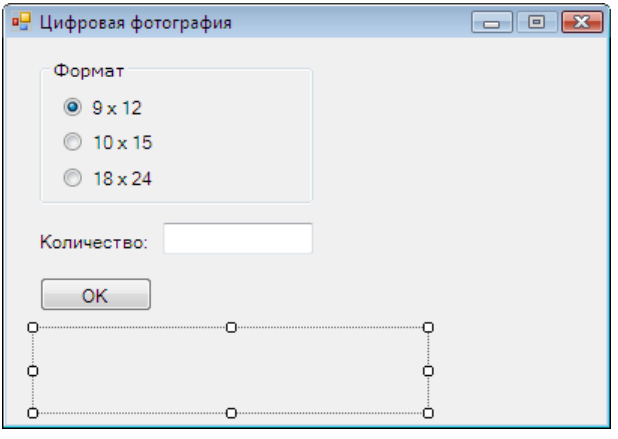


Рис. 1.4. Форма программы **Фото**

```
// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double cena = 0 ; // цена
    int n;            // кол-во фотографий
    double sum;       // сумма

    if (radioButton1->Checked)
        cena = 8.50;
    if (radioButton2->Checked)
        cena = 10;
    if (radioButton3->Checked)
        cena = 15.5;

    n = Convert::ToInt32(textBox1->Text);
    sum = n * cena;

    label2->Text = "Цена: " + cena.ToString("c") +
                "\nКоличество: " + n.ToString() + "шт.\n" +
                "Сумма заказа: " + sum.ToString("C");
}

// изменилось содержимое поля редактирования
private: System::Void textBox1_TextChanged(System::Object^
sender, System::EventArgs^  e)
{
    if (textBox1->Text->Length == 0)
        button1->Enabled = false;
    else
        button1->Enabled = true;

    label2->Text = "";
}

// щелчок на радиокнопке
```

```
// Функция обрабатывает событие Click компонентов
// radioButton1, radioButton2 и radioButton3
private: System::Void radioButton1_Click(System::Object^
sender, System::EventArgs^  e)
{
    label2->Text = "";

    // установить курсор в поле Количество
    textBox1->Focus();

}

// Чтобы в поле Количество можно было
// ввести только целое число
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
       return;

    if (Char::IsControl(e->KeyChar))
    {
        if (e->KeyChar == (char)Keys::Enter)
        {
            // нажата клавиша <Enter>
            button1->Focus();
        }
        return;
    }

    // остальные символы запрещены
    e->Handled = true;
}
```

Комплектация

Программа **Комплектация**, ее окно приведено на рис. 1.5, позволяет посчитать стоимость автомобиля в зависимости от выбранной комплектации. Демонстрирует использование компонента `CheckBox`. Для отображения картинки используется компонент `PictureBox`. Загрузка картинки выполняется в начале работы программы. Делает это конструктор формы.

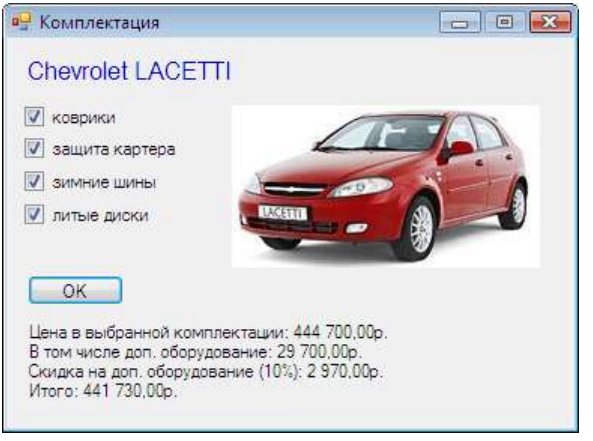


Рис. 1.5. Окно программы **Комплектация**

```
// конструктор
Form1(void)
  {
    InitializeComponent();

    this->pictureBox1->Image =
              Image::FromFile(
              Application::StartupPath+"\\Lacetti_r.jpg");

     /* получить имя папки Изображения можно так:
System::Environment::GetFolderPath(System::Environment::
SpecialFolder::MyPictures)
```

```
    */
        }

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double cena;      // цена в базовой комплектации
    double dop;       // сумма за доп. оборудование
    double discount; // скидка
    double total;     // общая сумма

    cena = 415000;
    dop = 0;

    if (checkBox1->Checked)
    {
        // коврики
        dop += 1200;
    }

    if (checkBox2->Checked)
    {
        // защита картера
        dop += 4500;
    }

    if (checkBox3->Checked)
    {
        // зимние шины
        dop += 12000;
    }

    if (checkBox4->Checked)
    {
```

```
        // литые диски
        dop += 12000;
    }

    total = cena + dop;

    System::String ^st;
    st = "Цена в выбранной комплектации: " +
                  total.ToString("C");
    if ( dop != 0)
    {
        st += "\nВ том числе доп. оборудование: " +
                dop.ToString("C");
    }

    if ((checkBox1->Checked) && (checkBox2->Checked) &&
        (checkBox3->Checked) && (checkBox4->Checked))
    {
        // скидка предоставляется, если
        // выбраны все опции
        discount = dop * 0.1;
        total = total — discount;
        st += "\nСкидка на доп. оборудование (10%): " +
                       discount.ToString("C") +
                      "\nИтого: " + total.ToString("C");
     }

    label2->Text = st;
}


// щелчок на компоненте CheckBox.
// Функция обрабатывает событие Click на
// компонентах checkBox1–checkBox4
```

```
private: System::Void checkBox1_Click(System::Object^  sender,
                                      System::EventArgs^  e)
{
    label2->Text = "";
}
```

Жалюзи

Программа **Жалюзи**, ее форма приведена на рис. 1.6, демонстрирует использование компонента `ComboBox`. Компонент используется для выбора материала (пластик, алюминий, соломка, текстиль). Следует обратить внимание на то, что настройку компонента `ComboBox` выполняет конструктор формы.

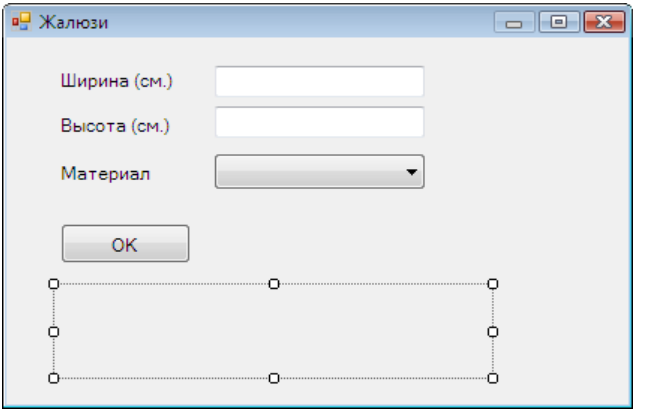


Рис. 1.6. Форма программы **Жалюзи**

```
// конструктор
Form1(void)
  {
    InitializeComponent();

    // настройка компонентов
    comboBox1->DropDownStyle = ComboBoxStyle::DropDownList;
```

```
    // сформировать список
    comboBox1->Items->Add("пластик");
    comboBox1->Items->Add("алюминий");
    comboBox1->Items->Add("бамбук");
    comboBox1->Items->Add("соломка");
    comboBox1->Items->Add("текстиль");

    // Элементы списка нумеруются с нуля.
    // По умолчанию выбран первый элемент
     comboBox1->SelectedIndex = 0;
  }

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double w;
    double h;
    double cena = 0; // цена за 1 кв.м.
    double sum;

    w = Convert::ToDouble(textBox1->Text);
    h = Convert::ToDouble(textBox2->Text);

    switch (comboBox1->SelectedIndex)
    {
        case 0: cena = 100;  break; // пластик
        case 1: cena = 250;  break; // алюминий
        case 2: cena = 170;  break; // бамбук
        case 3: cena = 170;  break; // соломка
        case 4: cena = 120;  break; // текстиль
    }

    sum = (w * h) / 10000 * cena;
    label4->Text =
                "Размер: " + w + " x " + h + " см.\n" +
                "Цена (р./м.кв.): " + cena.ToString("c") +
```

```
                "\nСумма: " + sum.ToString("c");
}

// Нажатие клавиши в поле редактирования.
// Функция обрабатывает событие KeyPress
// компонентов textBox1 и textBox2
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
         return;

    if (Char::IsControl(e->KeyChar))
    {
        if (e->KeyChar == (char)Keys::Enter)
        {
            if (sender->Equals(textBox1))
                // клавиша <Enter> нажата в поле Ширина
                // переместить курсор в поле Высота
                textBox2->Focus();
            else
                // клавиша <Enter> нажата в поле Высота
                // переместить фокус на comboBox1
                button1->Focus();
         }
         return;
    }

    // остальные символы запрещены
    e->Handled = true;
}

// в списке Материал пользователь
// выбрал другой элемент
private: System::Void comboBox1_SelectedIndexChanged
              (System::Object^  sender, System::EventArgs^
```

```
e)
{
    label4->Text = "";
}

// пользователь изменил размер (содержимое
// textBox1 или textBox2)
private: System::Void textBox1_TextChanged(System::Object^
sender, System::EventArgs^  e)
{
    label4->Text = "";

    if ((textBox1->Text->Length == 0) ||
        (textBox2->Text->Length == 0))

        button1->Enabled = false;
     else
        button1->Enabled = true;
}
```

Калькулятор

Программа **Калькулятор** демонстрирует создание компонентов в коде. Кнопки калькулятора, объединенные в массив `btn` компоненты `Button`, создает и настраивает конструктор формы. Он же назначает кнопкам функции обработки события `Click`. Форма и окно программы приведены на рис. 1.7.

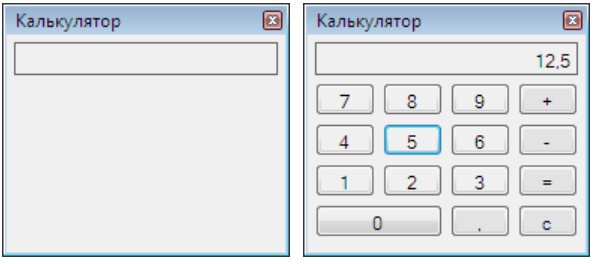


Рис. 1.7. Форма и окно программы **Калькулятор**

```
// переменные класса
private:

  static int bw = 40, bh = 22; // размер кнопки

  // расстояние между кнопками по X и Y
  static int dx = 5, dy = 5;

  // кнопки
  array<System::Windows::Forms::Button^>  ^btn;

  // текст на кнопках
  static array<String^> ^btnText = {"7","8","9","+",
                    "4","5","6","-",
                    "1","2","3","=",
                    "0",",","c"};

  // при нажатии кнопку будем идентифицировать
  // по значению свойства Tag
  static array<int^> ^btnTag =
                {7,8,9,-3,
                 4,5,6,-4,
                 1,2,3,-2,
                 0,-1,-5};

  double ac; // аккумулятор
  int op;    // код операции

  bool fd;
  // fd == true — ждем первую цифру числа,
  // например, после нажатия кнопки "+"
  // fd == false — ждем следующую цифру

// конструктор формы
public:
  Form1(void)
```

```
{
    InitializeComponent();

    // установить размер формы (клиентской области)
    int cw = 4*bw + 5*dx;
    int ch = 5*bh + 7*dy;
    this->ClientSize = System::Drawing::Size(cw,ch);

    // задать размер и положение индикатора
    label1->SetBounds(dx, dy, 4*bw + 3*dx, bh);
    label1->Text = "0";

    int x, y;// координаты кнопки

    y = label1->Bottom + dy ;

    // кнопки калькулятора объединены в массив
    btn = gcnew array<System::Windows::Forms::Button^>(15);

    int k =0;     // номер настраиваемой кнопки

    for ( int i=0; i < 4; i++)
    {
        x = dx;
        for (int j = 0; j < 4; j++)
        {
            if ( !((i == 3 ) && (j == 0)))
            {
              // создать кнопку
              btn[k] = gcnew
                        System::Windows::Forms::Button;

              // настроить кнопку
              btn[k]->SetBounds(x, y, bw, bh);
              btn[k]->Tag = btnTag[k];
```

```
    btn[k]->Text = btnText[k];

    if ( Convert::ToInt32(btn[k]->Tag) > 0) {
        // кнопка с цифрой
        btn[k]->BackColor =
            SystemColors::ButtonFace;
    }
    else {
        // кнопка операции
        btn[k]->BackColor =
            //System::Drawing::Color::Red;
            SystemColors::ControlLight;
    }

    x = x + bw + dx;
}
else // кнопка "ноль" шире остальных
{
    // создать и настроить кнопку
    btn[k] = gcnew
              System::Windows::Forms::Button;
    btn[k]->SetBounds(x, y, bw * 2 + dx, bh);
    btn[k]->Tag = btnTag[k];
    btn[k]->Text = btnText[k];//.ToString();

    x = x + 2*bw + 2*dx;

    j++;
}

// поместить кнопку на форму
this->Controls->Add(btn[k]);

// задать функцию обработки события Click
btn[k]->Click +=
```

```
            gcnew System::EventHandler(this, &Form1::btn_Click);

                k++; // увеличить номер кнопки
            }
            y = y + bh + dy;
        }

        ac = 0;         // аккумулятор
        fd = true;  // ждем первую цифру

    }

// щелчок на кнопке
private: System::Void btn_Click(System::Object^  sender,
                                System::EventArgs^  e)
{
    System::Windows::Forms::Button^  btn;

    btn = (Button^)sender;

    // кнопки 1..9
    if ( Convert::ToInt32(btn->Tag) > 0)
    {
        if ( fd )
        {
            // на индикаторе ноль, т. е.
            // это первая цифра
            label1->Text = btn->Text;
            fd = false;
        }
        else
            label1->Text += btn->Text;
            return;
        }
```

```
// ноль
if (Convert::ToInt32(btn->Tag) == 0)
{
    if (fd) label1->Text = btn->Text;
    if (label1->Text != "0")
                label1->Text += btn->Text;
    return;
}

// запятая
if ( Convert::ToInt32(btn->Tag) == -1 )
{
    if (fd)
    {
        label1->Text = "0,";
        fd = false;
    }
    else
        if (label1->Text->IndexOf(",") == -1)
                label1->Text += btn->Text;
        return;
}

// "c" — очистить
if (Convert::ToInt32(btn->Tag) == -5 )
{
        ac = 0; // очистить аккумулятор
        op = 0;
        label1->Text = "0";

        fd = true; // снова ждем первую цифру
        return;
}

// кнопки "+","-","="
if (Convert::ToInt32(btn->Tag) < -1)
```

```
    {
        double n; // число на индикаторе

        n = Convert::ToDouble(label1->Text);

        // Нажатие клавиши операции означает, что
        // пользователь ввел операнд. Если в
        // аккумуляторе есть число, то выполним
        // операцию. Если это не так, запомним
        // операцию, чтобы выполнить ее при следующем
        // нажатии клавиши операции.
        if (ac != 0)
        {
                switch (op)
                {
                    case -3: ac += n;
                        break;
                    case -4: ac -= n;
                        break;
                    case -2: ac = n;
                        break;
                }
                label1->Text = ac.ToString("N");
        }
        else
            ac = n;

        op = Convert::ToInt32(btn->Tag);
        fd = true; // ждем следующее число
    }
}
```

Просмотр иллюстраций

Программа **Просмотр иллюстраций** (рис. 1.8) демонстрирует использование компонентов `ListBox`, `PictureBox` и `FolderBrowserDialog`. Выбор папки, в которой находятся иллю-

страции, выполняется в стандартном окне **Обзор папок** (`FolderBrowserDialog`).

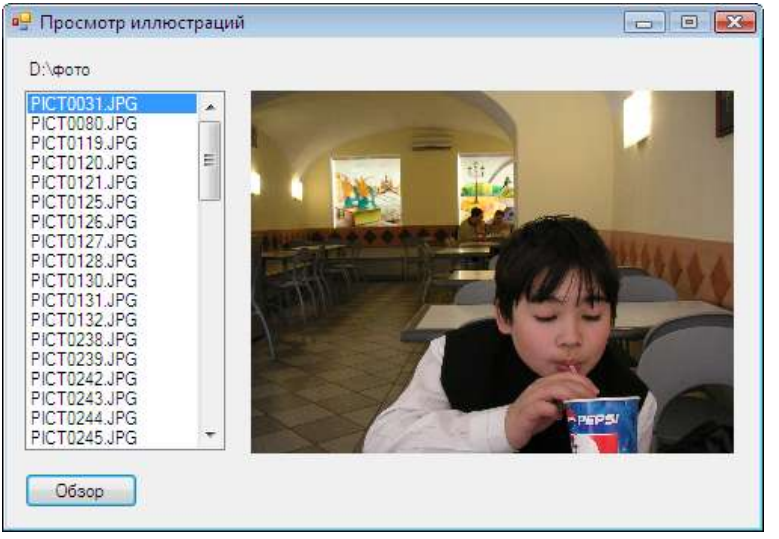


Рис. 1.8. Окно программы **Просмотр иллюстраций**

```
private:
        String^ aPath;      // путь к файлам картинок
        int pbw, pbh;       // первоначальный размер
        int pbX, pbY;       // и положение pictureBox

Form1(void)
{
      InitializeComponent();

      // запомнить размер
      // и положение pictureBox1
      pbh = pictureBox1->Height;
      pbw = pictureBox1->Width;
      pbX = pictureBox1->Location.X;
```

```
        pbY = pictureBox1->Location.Y;

        // настройка listBox1:
        // элементы списка сортируются
        // в алфавитном порядке
        listBox1->Sorted = true;

        // получить имя каталога "Мои рисунки"
        DirectoryInfo^ di; // каталог
        di = gcnew DirectoryInfo(Environment::GetFolderPath(
            Environment::SpecialFolder::MyPictures));
        aPath = di->FullName;
        label1->Text = aPath;

        // сформировать и отобразить в поле компонента
        // listBox список иллюстраций
        FillListBox(aPath);
}

// формирует список файлов иллюстраций
private:  System::Boolean FillListBox(String^ aPath) {

    // информация о каталоге
    DirectoryInfo^ di =
    gcnew DirectoryInfo(aPath);

    // информация о файлах
    array<FileInfo^> ^fi = di->GetFiles("*.jpg");

    // очистить список listBox
    listBox1->Items->Clear();

    // добавляем в listBox1 имена jpg-файлов,
    // содержащихся в каталоге aPath
    for each (FileInfo^ fc in fi)
```

```
    {
        listBox1->Items->Add(fc->Name);
    }

    label1->Text = aPath;

    if (fi->Length == 0) return false;
        else
        {
            // выбираем первый файл из полученного списка
            listBox1->SelectedIndex = 0;
            return true;
         }
}

// щелчок на кнопке Обзор
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    // FolderBrowserDialog — окно Обзор папок
    FolderBrowserDialog^ fb
            = gcnew FolderBrowserDialog();

    fb->Description =
                "Выберите папку,\n" +
                "в которой находятся иллюстрации";
    fb->ShowNewFolderButton = false;

    fb->SelectedPath = aPath;

    // отобразить окно Обзор папок
    // и проверить, щелчком на какой кнопке
    // пользователь закрыл его
    if ( fb->ShowDialog() ==
           System::Windows::Forms::DialogResult::OK )
```

```
    {
        // пользователь выбрал каталог и
        // щелкнул на кнопке OK
        aPath = fb->SelectedPath;
        label1->Text = aPath;

        if ( !FillListBox(fb->SelectedPath))
           // в каталоге нет файлов иллюстраций
           pictureBox1->Image = nullptr;
     }
}

// пользователь выбрал другой элемент списка
// щелчком кнопкой мыши или перемещением
// по списку при помощи клавиатуры
private: System::Void listBox1_
SelectedIndexChanged(System::Object^  sender,
                              System::EventArgs^  e) {



    double mh, mw;   // коэффициенты масштабирования
                     // по ширине и высоте

    pictureBox1->Visible = false;
    pictureBox1->Left = pbX;

    // загружаем изображение в pictureBox1
    pictureBox1->SizeMode =
        PictureBoxSizeMode::AutoSize;

    pictureBox1->Image =
               gcnew Bitmap( aPath + "\\" +
                        listBox1->SelectedItem->ToString());

    // масштабируем, если нужно
```

```
    if ((pictureBox1->Image->Width > pbw) ||
        (pictureBox1->Image->Height > pbh))
    {
        pictureBox1->SizeMode =
            PictureBoxSizeMode::StretchImage;

        mh = (double)pbh /(double)pictureBox1->Image->Height;
        mw = (double)pbw / double)pictureBox1->Image->Width;

        if (mh < mw)
        {
            // масштабируем по ширине
            pictureBox1->Width = Convert::ToInt16(
                            pictureBox1->Image->Width * mh);
            pictureBox1->Height = pbh;
        }
        else
        {
            // масштабируем по высоте
            pictureBox1->Width = pbw;
            pictureBox1->Height = Convert::ToInt16(
                            pictureBox1->Image->Height * mw);
         }
    }

    // разместить картинку в центре области
    // отображения иллюстраций
    pictureBox1->Left =
                pbX + (pbw — pictureBox1->Width) / 2;
    pictureBox1->Top =
                pbY + (pbh — pictureBox1->Height) / 2;

    pictureBox1->Visible = true;
}
```

Слайд-шоу

Программа **Слайд-шоу** позволяет просмотреть фотографии. Она демонстрирует использование компонента `PictureBox`, показывает, как превратить командную кнопку в графическую, а также как с помощью компонента `ToolTip` обеспечить отображение подсказки при позиционировании указателя мыши на командной кнопке. Форма программы приведена на рис. 1.9. Чтобы на кнопке появилась картинка, надо присвоить значение свойству `Image`, указав `png`-файл с "прозрачным" фоном. Значения свойств, обеспечивающих отображение подсказок, приведены в табл. 1.2.

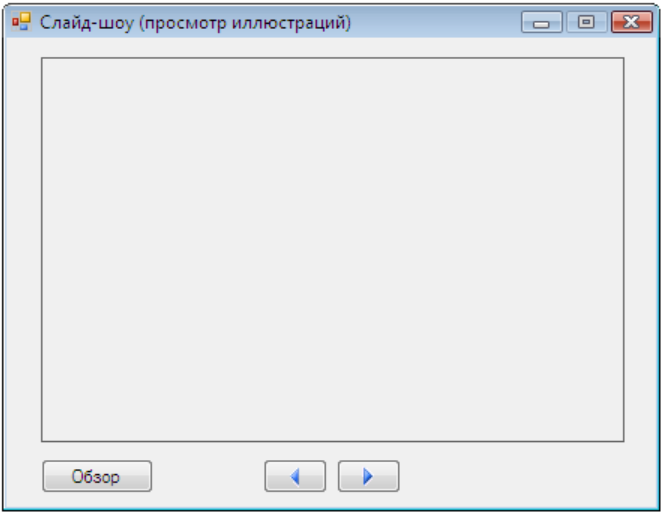


Рис. 1.9. Форма программы **Слайд-шоу**

Таблица 1.2. *Свойства, обеспечивающие отображение подсказок*

Компонент	Свойство	Значение
button1	ToolTip on toolTip1	Выбор папки
button2	ToolTip on toolTip1	Назад
button3	ToolTip on toolTip1	Далее

```
// конструктор формы
public:
    Form1(void)
    {
      InitializeComponent();
      //
      // запомнить размер
      // и положение pictureBox1
      pbh = pictureBox1->Height;
      pbw = pictureBox1->Width;
      pbX = pictureBox1->Location.X;
      pbY = pictureBox1->Location.Y;

      // получить имя каталога "Мои рисунки"
      DirectoryInfo^ di; // каталог
      di = gcnew DirectoryInfo(Environment::GetFolderPath(
          Environment::SpecialFolder::MyPictures));
      aPath = di->FullName;
      this->Text = aPath;

      imgList = gcnew
System::Collections::Generic::List<String^>;

      // сформировать список иллюстраций
      FillList(aPath);
    }

private:
    String^ aPath;     // путь к файлам картинок
    int pbw, pbh;      // первоначальный размер
    int pbX, pbY;      // и положение pictureBox

    // список иллюстраций
    System::Collections::Generic::List<String^> ^imgList;
```

```
    int cImg; // номер отображаемой иллюстрации

private: void ShowPicture(String^ aPicture)
{
    double mh, mw;  // коэффициенты масштабирования

    pictureBox1->Visible = false;
    pictureBox1->Left = pbX;

    // загружаем изображение в pictureBox1
    pictureBox1->SizeMode =
        PictureBoxSizeMode::AutoSize;

    pictureBox1->Image =
                gcnew Bitmap( aPath + "\\" +
                            imgList[cImg]);

    // масштабируем, если нужно
    if ((pictureBox1->Image->Width > pbw) ||
        (pictureBox1->Image->Height > pbh))
    {
        pictureBox1->SizeMode =
            PictureBoxSizeMode::StretchImage;

        mh = (double)pbh /
                    (double)pictureBox1->Image->Height;
        mw = (double)pbw /
                    (double)pictureBox1->Image->Width;

        if (mh < mw)
        {
            // масштабируем по ширине
            pictureBox1->Width = Convert::ToInt16(
                            pictureBox1->Image->Width * mh);
            pictureBox1->Height = pbh;
```

```
        }
        else
        {
            // масштабируем по высоте
            pictureBox1->Width = pbw;
            pictureBox1->Height = Convert::ToInt16(
                        pictureBox1->Image->Height * mw);
         }
    }

    // разместить картинку в центре области
    // отображения иллюстраций
    pictureBox1->Left =
                    pbX + (pbw — pictureBox1->Width) / 2;
    pictureBox1->Top =
                    pbY + (pbh — pictureBox1->Height) / 2;

    pictureBox1->Visible = true;
}

private: System::Boolean FillList(String^ aPath)
{
    // информация о каталоге
    DirectoryInfo^ di =
    gcnew DirectoryInfo(aPath);

    // информация о файлах
    array<FileInfo^> ^fi = di->GetFiles("*.jpg");

    // очистить список
    imgList->Clear();

    button2->Enabled = true;
    button3->Enabled = true;

    // добавляем в список имена jpg-файлов,
```

```
    // содержащихся в каталоге aPath
    for each (FileInfo^ fc in fi)
    {
        imgList->Add(fc->Name);
    }

    this->Text = aPath;

    if ( imgList->Count == 0)
          return false;
    else
    {
        cImg = 0;
        ShowPicture(aPath + "\\" + imgList[cImg]);

        // сделать недоступной кнопку Предыдущая
        button2->Enabled = false;

        // если в каталоге один jpg-файл,
        // сделать недоступной кнопку Следующая
        if (imgList->Count == 1)
            button3->Enabled = false;

        this->Text = aPath;

        return true;
    }
}

// щелчок на кнопке Обзор
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e) {
    // FolderBrowserDialog — окно Обзор папок
    FolderBrowserDialog^ fb = gcnew FolderBrowserDialog();
```

```
    fb->Description =
                "Выберите папку,\n" +
                "в которой находятся иллюстрации";
    fb->ShowNewFolderButton = false;

    fb->SelectedPath = aPath;

    // отобразить окно Обзор папок
    // и проверить, щелчком на какой кнопке
    // пользователь закрыл его
    if ( fb->ShowDialog() ==
              System::Windows::Forms::DialogResult::OK )
    {
        // пользователь выбрал каталог и
        // щелкнул на кнопке OK
        aPath = fb->SelectedPath;
        this->Text = aPath;

        if ( !FillList(fb->SelectedPath))
           // в каталоге нет файлов иллюстраций
           pictureBox1->Image = nullptr;
     }
}

// щелчок на кнопке Предыдущая картинка
private: System::Void button2_Click(System::Object^  sender,
                                    System::EventArgs^  e) {

    // если кнопка "Следующая" недоступна,
    // сделаем ее доступной
    if (!button3->Enabled)
        button3->Enabled = true;

    if (cImg > 0)
    {
```

```
        cImg--;
        ShowPicture(aPath + "\\" + imgList[cImg]);

        // отображается первая иллюстрация
        if (cImg == 0)
        {
            // теперь кнопка Предыдущая недоступна
            button2->Enabled = false;
         }
     }
}

// щелчок на кнопке Следующая картинка
private: System::Void button3_Click(System::Object^  sender,
                                    System::EventArgs^  e) {
  if (!button2->Enabled)
       button2->Enabled = true;

   if (cImg < imgList->Count)
   {
        cImg++;
        ShowPicture(aPath + "\\" + imgList[cImg]);
        if (cImg == imgList->Count -1)
        {
            button3->Enabled = false;
        }
   }
}
```

Финансовый калькулятор

Программа **Финансовый калькулятор** (рис. 1.10) позволяет рассчитать величину ежемесячных платежей по кредиту (предполагается, что сумма кредита выплачивается равными долями, а процент начисляется на сумму остатка долга). Форма програм-

мы приведена на рис. 1.11. Для отображения результата (таблицы платежей) используется компонент `ListView` (значения его свойств приведены в табл. 1.3).

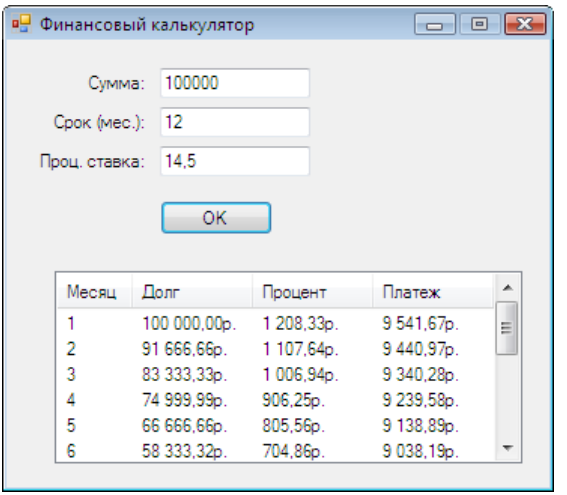


Рис. 1.10. Финансовый калькулятор

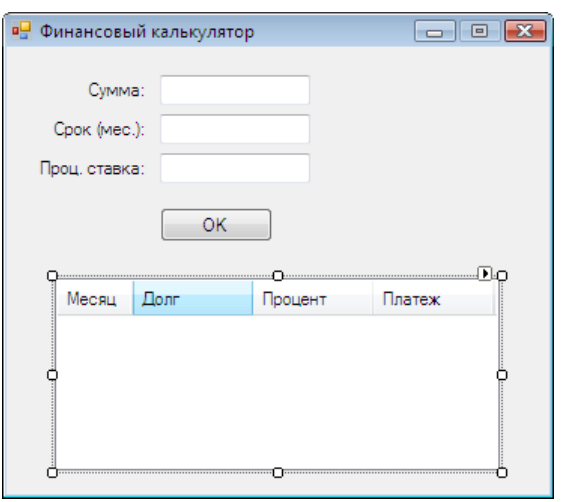


Рис. 1.11. Форма программы **Финансовый калькулятор**

Таблица 1.3. *Значения свойств компонента ListView*

Свойство	Значение
View	Details
Columns[0].Text	Месяц
Columns[0].Width	50
Columns[1].Text	Долг
Columns[1].Width	80
Columns[2].Text	Процент
Columns[2].Width	80
Columns[3].Text	Платеж
Columns[3].Width	80

```
public:
        Form1(void)
        {
            InitializeComponent();

            // настройка listView1 — увеличить
            // ширину компонента на ширину вертикальной
            // полосы прокрутки
            int w = 0;
            for ( int i=0; i < listView1->Columns->Count; i++)
                w += listView1->Columns[i]->Width;
            if (listView1->BorderStyle ==
                          BorderStyle::Fixed3D)
                w +=4;
            listView1->Width = w + 17;

            listView1->FullRowSelect = true;

        }

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
```

```
                                        System::EventArgs^  e)
{
    float value; // сумма кредита
    int period;  // срок
    float rate;  // процентная ставка

    float debt; // долг на начало текущего месяца
    float interest; // плата за кредит (проценты на долг)
    float paying;   // сумма платежа
    float suminterest; // общая плата за кредит

    String ^ st; // буфер

    // сумма
    value = System::Convert::ToSingle(textBox1->Text);

    // срок
    period = System::Convert::ToInt32(textBox2->Text);

    // процентная ставка
    rate = System::Convert::ToSingle(textBox3->Text);

    debt = value;
    suminterest = 0;

    // расчет для каждого месяца
    for ( int i = 1; i <= period; i++)
    {
        interest = debt * (rate/12/100);
        suminterest += interest;
        paying = value/period + interest;

        st = i.ToString() + "  " + debt.ToString("c") + "  " +
             interest.ToString("c") + "  " +
```

```
            paying.ToString("c");

        // добавить в listView элемент -
        // строку (данные в первый столбец)
        listView1->Items->Add(i.ToString());

        // добавить в добавленную строку подэлементы -
        // данные во второй, третий и четвертый столбцы
        listView1->Items[i-1]->
              SubItems->Add(debt.ToString("c"));
        listView1->Items[i-1]->
              SubItems->Add(interest.ToString("c"));
        listView1->Items[i-1]->
              SubItems->Add(paying.ToString("c"));

        debt = debt — value/period;
    }
}

// обрабатывает событие TextChanged компонентов
// textBox1—textBox3
private: System::Void textBox1_
TextChanged(System::Object^  sender, System::EventArgs^  e)
{
    // если хотя бы в одном из полей нет данных,
    // сделать кнопку OK недоступной
    if ( (textBox1->Text->Length == 0) ||
         (textBox2->Text->Length == 0) ||
         (textBox3->Text->Length == 0)
        )
        button1->Enabled = false;
    else
        button1->Enabled = true;
}
```

ОСАГО

Программа **ОСАГО**, ее форма приведена на рис. 1.12, позволяет рассчитать размер страховой премии, подлежащей уплате по договору обязательного страхования гражданской ответственности. Для ввода исходной информации используются компоненты `ComboBox`. Следует обратить внимание: чтобы пользователь мог ввести данные в поле компонента только путем выбора значения из списка, свойству `DropDowsStyle` необходимо присвоить значение `DropDown`.

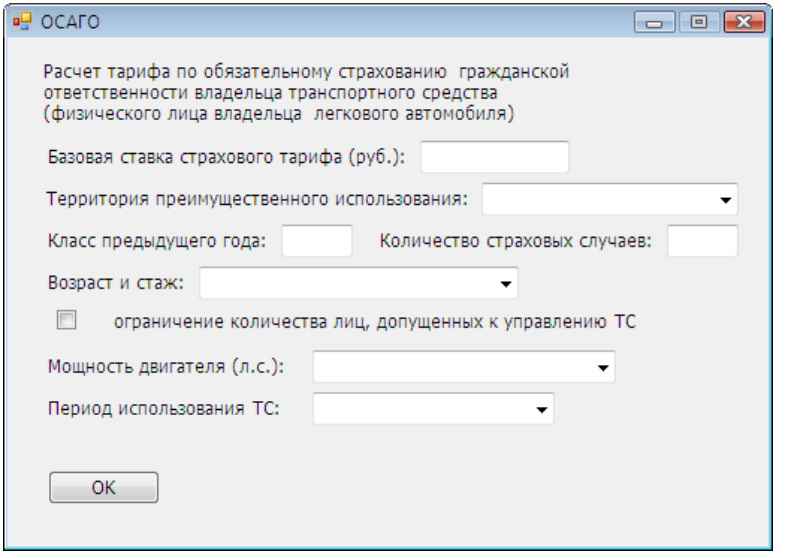


Рис. 1.12. Форма программы **ОСАГО**

```
// конструктор
Form1(void)
{
      InitializeComponent();
      //
      // Настройка компонентов
      //
```

```
// базовый тариф
textBox1->Text = "1980";

checkBox1->Checked = true;

// список регионов:
// загрузка из массива
for ( int i = 0; i < reg->Length; i++)
{
    comboBox1->Items->Add(reg[i]);
}
comboBox1->SelectedItem = 2;
comboBox1->DropDownStyle =
  System::Windows::Forms::ComboBoxStyle::DropDownList;

/* можно и так:
comboBox1->Items->Add("Ленинградская обл.");
comboBox1->Items->Add("Москва");
comboBox1->Items->Add("Московская обл.");
comboBox1->Items->Add("Мурманск");
comboBox1->Items->Add("Нижний Новгород");
comboBox1->Items->Add("Ростов-на-Дону");
comboBox1->Items->Add("Самара");
comboBox1->Items->Add("Санкт-Петербург");
*/

// возраст и стаж
comboBox2->Items->Add("до 22 лет, стаж менее 2-х лет");
comboBox2->Items->Add("до 22 лет, стаж свыше 2-х лет");
comboBox2->Items->Add("от 22 лет, стаж менее 2-х лет");
comboBox2->Items->Add("от 22 лет, стаж свыше 2-х лет");
comboBox2->DropDownStyle =
   System::Windows::Forms::ComboBoxStyle::DropDownList;

// мощность двигателя
comboBox3->Items->Add("до 50 включительно");
```

```
        comboBox3->Items->Add("свыше 50 до 70 включительно");
        comboBox3->Items->Add("свыше 70 до 95 включительно");
        comboBox3->Items->Add("свыше 95 до 120 включительно");
        comboBox3->Items->Add("свыше 120 до 160 включительно");
        comboBox3->Items->Add("свыше 160 до 200 включительно");
        comboBox3->Items->Add("свыше 200");
        comboBox3->DropDownStyle =
           System::Windows::Forms::ComboBoxStyle::DropDownList;

        // период использования
        comboBox4->Items->Add("6 месяцев");
        comboBox4->Items->Add("7 месяцев");
        comboBox4->Items->Add("8 месяцев");
        comboBox4->Items->Add("9 месяцев");
        comboBox4->Items->Add("более 9 месяцев");
        comboBox4->DropDownStyle =
           System::Windows::Forms::ComboBoxStyle::DropDownList;
}

private:

// регион
static array<String^>^ reg =
    {"Москва","Московская обл.","Санкт-Петербург",
     "Нижний Новгород","Ленинградская обл.","Ростов-на-Дону",
     "Самара","Мурманск"};

// коэффициент региона
static array<double>^ kt  = {1.8,1.6, 1.8,1.3,1,1,1,1};

/* таблица определения коэффициента страхового тарифа
Первый год — 3-й класс, второй год (если не было страховых
случаев) 4-й класс и т. д.). Если страховой случай был,
то класс в ячейке таблицы: строка — класс предыдущего
года, столбец — кол-во страховых случаев. */
```

```
// класс безаварийности — таблица 6×5 (6 строк, 5 столбцов)
static array<int,2>^ cb =
{
  {1,-1,-1,-1,-1},
  {2,-1,-1,-1,-1},
  {3, 1,-1,-1,-1},
  {4, 1,-1,-1,-1},
  {5, 2, 1,-1,-1},
  {6, 3, 1,-1,-1}
};

// коэффициент безаварийности
static array<double>^ kb = {2.3, 1.55, 1.4, 1, 0.95, 0.9};

// коэффициент водительского стажа
static array<double>^ kvs = {1.3, 1.2, 1.15, 1};

// коэффициент мощности двигателя
static array<double>^ km = {0.5, 0.7, 1.0, 1.3,
                            1.5, 1.7, 1.9};

// коэффициент периода использования ТС
static array<double>^ ks = {0.7, 0.8, 0.9, 0.95, 1.0};

// щелчок на кнопке ОК
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    // расчет страховой премии :)
    double aTb;  // базовая ставка тарифа
    double aKt;  // коэфф. тарифа
    double aKb;  // коэфф. безаварийности
    double aKvs; // коэфф. водительского стажа
    double aKo;  // коэфф., количества лиц, допущенных к
                 // управлению ТС
```

```
double aKm;  // коэфф. мощности двигателя
double aKs;  // коэфф., учитывающий период использования ТС

int pcb,ccb; // предыдущий и текущий класс безаварийности
int nss;     // кол-во страховых случаев предыдущего периода

// проверим, все ли поля формы заполнены
if ( (comboBox1->SelectedIndex == -1) ||
     (comboBox2->SelectedIndex == -1) ||
     (comboBox3->SelectedIndex == -1) ||
     (comboBox4->SelectedIndex == -1) ||
     (textBox2->Text->Length == 0) ||
     (textBox3->Text->Length == 0)
    )
{
    MessageBox::Show("Надо заполнить все поля формы",
        "ОСАГО",
        System::Windows::Forms::MessageBoxButtons::OK,
        System::Windows::Forms::MessageBoxIcon::Warning);

    return;
}

// Все поля формы заполнены. Расчет

aTb = System::Convert::ToDouble(textBox1->Text);
aKt = kt[comboBox1->SelectedIndex];

pcb = System::Convert::ToInt32(textBox2->Text);
nss = System::Convert::ToInt32(textBox3->Text);

if ( pcb > 5)
    pcb = 5; // см. определение таблицы cb

ccb =   cb[pcb-1,nss]; // текущий класс безаварийности
```

```
if ( ccb != -1)
    aKb = kb[ccb];
else aKb = 2.45;

// коэфф. водительского стажа
aKvs = kvs[comboBox2->SelectedIndex];

// количество лиц, допущенных к управлению
if (checkBox1->Checked)
    aKo = 1;
else
    aKo = 1.5;

// мощность двигателя
aKm = km[comboBox3->SelectedIndex];

// период использования ТС
aKs = ks[comboBox4->SelectedIndex];

// все коэффициенты определены — расчет

double T; // тариф
String ^st;

T = aTb * aKt * aKb * aKvs * aKo * aKm *aKs;

st = "Базовая ставка тарифа: " + aTb.ToString("c")+
    "\nКоэфф. тарифа: " + aKt.ToString()+
    "\nКоэфф. безаварийности: " + aKb.ToString()+
    "\nКоэфф. водительского стажа: " + aKvs.ToString() +
    "\nКоэфф. кол-ва лиц, допущенных к управлению: " +
              aKo.ToString() +
    "\nКоэфф. мощности двигател: " + aKm.ToString() +
    "\nКоэфф. периода использования ТС: " + aKs.ToString()+
```

```
        "\n\nТариф: " + T.ToString("c");

    MessageBox::Show(st,"Тариф ОСАГО");
}
```

Секундомер

Программа **Секундомер**, ее форма приведена на рис. 1.13, демонстрирует использование компонента `Timer`.

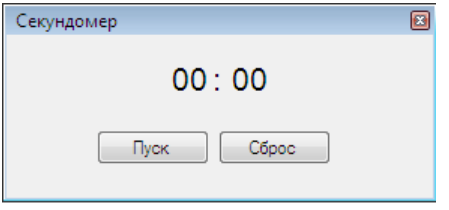


Рис. 1.13. Форма программы **Секундомер**

```
Form1(void)
  {
      InitializeComponent();
      //
      // настройка таймера:
      // сигнал от таймера — каждые полсекунды
      timer1->Interval = 500;

      // обнуление показаний
      m = 0;
      s = 0;

      label1->Text = "00";
      label2->Text = "00";
```

```
      label3->Visible = true;
  }

private:
        int m;  // количество минут
        int s;  // количество секунд

// щелчок на кнопке Пуск/Стоп
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e) {
  if (timer1->Enabled)
  {
      // таймер работает,
      // значит, щелчок на кнопке Стоп.
      // остановить таймер
      timer1->Enabled = false;

      // изменить текст на кнопке
      // и сделать доступной кнопку Сброс
      button1->Text = "Пуск";
      button2->Enabled = true;

     label3->Visible = true;
  }
  else
  {
      // таймер не работает,
      // значит, щелчок на кнопке Пуск
      // запускаем таймер
      timer1->Enabled = true;

      // изменить текст на кнопке
      button1->Text = "Стоп";

      // и сделать недоступной кнопку Сброс
```

```
      button2->Enabled = false;
   }
}

// щелчок на кнопке Сброс
private: System::Void button2_Click(System::Object^ sender,
                                    System::EventArgs^  e) {

    m = 0;
    s = 0;

    label1->Text = "00";
    label2->Text = "00";
}

// сигнал от таймера
private: System::Void timer1_Tick(System::Object^  sender,
                                  System::EventArgs^  e)
{
  // двоеточие мигает с периодом 0,5 с
  if (label3->Visible)
  {
      if (s < 59)
      {
          s++;
          if (s < 10)
             label2->Text = "0" + s.ToString();
          else
             label2->Text = s.ToString();
      }
      else
      {
           if (m < 59)
           {
              m++;
```

```
            if (m < 10)
                label1->Text = "0" + m.ToString();
            else
                label1->Text = m.ToString();
            s = 0;
            label2->Text = "00";
        }
        else
        {
            m = 0;
            label1->Text = "00";
        }
    }
    label3->Visible = false;
  }
  else
    label3->Visible = true;
}
```

Таймер

Программа **Таймер**, ее форма приведена на рис. 1.14, демонстрирует использование компонентов `NumericUpDown` и `Timer`.

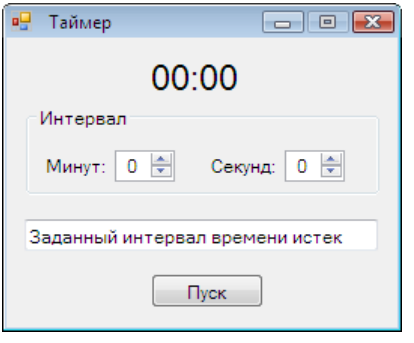


Рис. 1.14. Форма программы **Таймер**

```
public:
    Form1(void)
    {
        InitializeComponent();
        //
        // настройка компонентов numericUpDown
        numericUpDown1->Maximum = 59;
        numericUpDown1->Minimum = 0;

        // чтобы при появлении окна
        // курсор не мигал в поле редактирования
        numericUpDown1->TabStop = false;

        numericUpDown2->Maximum = 59;
        numericUpDown2->Minimum = 0;
        numericUpDown2->TabStop = false;

        // кнопка Пуск/Стоп недоступна
        button1->Enabled = false;
    }

private: DateTime^ t1; // время запуска таймера
private: DateTime^ t2; // время срабатывания таймера


// обработка события ValueChanged компонентов
// numericUpDown1 и numericUpDown1
private: System::Void numericUpDown1_ValueChanged
     (System::Object^  sender, System::EventArgs^  e)
{
    if ((numericUpDown1->Value == 0) &&
        (numericUpDown2->Value == 0))
        button1->Enabled = false;
    else
```

```
        button1->Enabled = true;
}

// щелчок на кнопке Пуск/Стоп
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    if (!timer1->Enabled)
    {
        // пуск таймера

        // t1 — текущее время
        // t2 = t1 + интервал


        t1 = gcnew DateTime(DateTime::Now.Year,
            DateTime::Now.Month, DateTime::Now.Day);

        t2 = t1->AddMinutes((double)numericUpDown1->Value);
        t2 = t2->AddSeconds((double)numericUpDown2->Value);

       groupBox1->Enabled = false;
       button1->Text = "Стоп";

       if (t2->Minute < 10)
          label1->Text = "0"+t2->Minute.ToString()+":";
       else
          label1->Text = t2->Minute.ToString() + ":";

       if (t2->Second < 10)
          label1->Text += "0" + t2->Second.ToString();
       else
          label1->Text += t2->Second.ToString();

       // период возникновения события Tick — 1 секунда
```

```
        timer1->Interval = 1000;

        // пуск таймера
        timer1->Enabled = true;

        groupBox1->Enabled = false;
    }
    else
    {
        // таймер работает, останавливаем
        timer1->Enabled = false;
        button1->Text = "Пуск";
        groupBox1->Enabled = true;
        numericUpDown1->Value = t2->Minute;
        numericUpDown2->Value = t2->Second;
     }
}

// обработка сигнала от таймера
private: System::Void timer1_Tick(System::Object^  sender,
                                  System::EventArgs^  e)
{
    t2 = t2->AddSeconds(-1);

    if (t2->Minute < 10)
       label1->Text = "0" + t2->Minute.ToString() + ":";
    else
       label1->Text = t2->Minute.ToString() + ":";

    if (t2->Second < 10)
        label1->Text += "0" + t2->Second.ToString();
    else
        label1->Text += t2->Second.ToString();

    if (Equals(t1, t2))
    {
```

```
        timer1->Enabled = false;

        this->Activate();

        MessageBox::Show(
            textBox1->Text,
            "Таймер",
            MessageBoxButtons::OK,
            MessageBoxIcon::Information);

        button1->Text = "Пуск";
        groupBox1->Enabled = true;
        numericUpDown1->Value = 0;
        numericUpDown2->Value = 0;
    }
}
```

Обработка исключения

Программа **Фунты-килограммы** (ее форма приведена на рис. 1.15) демонстрирует технологию обработки исключений (ошибок, возникающих во время работы программы). В отличие от других программ, например, программы **Мили-километры**, в поле редактирования пользователь может ввести любую строку символов, и поэтому во время работы программы возможно исключение `FormatException`.

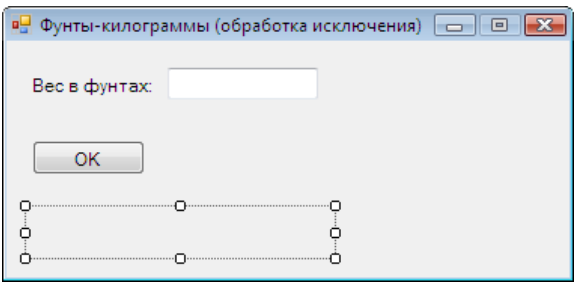


Рис. 1.15. Форма программы **Фунты-килограммы**

```
// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double funt; // вес в фунтах
    double kg;   // вес в килограммах

    try
    {
        funt = Convert::ToDouble(textBox1->Text);

        // 1 фунт = 409,5 грамма
        kg = funt * 0.4095;

        label2->Text = funt.ToString("N") + " ф. = " +
                    kg.ToString("N") + " кг.";
    }
    catch (System::FormatException^ e)
    {
        MessageBox::Show(
              "Ошибка исходных данных. Проверьте, что
              исходные данные (дробное число) введены в
              поле редактирования, а также, что в качестве
              десятичного разделителя используется запятая.",
              "Фунты-килограммы",
              MessageBoxButtons::OK,MessageBoxIcon::Error);
    }
}
```

Справочная информация

Программа **Чистый дисконтированный доход** (ее форма приведена на рис. 1.16) демонстрирует различные способы отображения справочной информации. Окно справочной информации (рис. 1.17) появляется на экране в результате нажатия клавиши <F1> или щелчка на кнопке **Справка**.

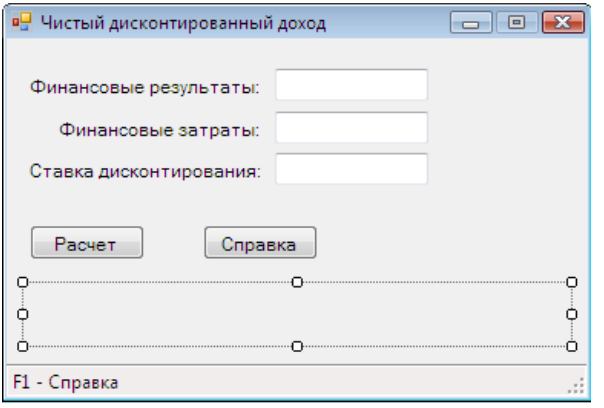


Рис. 1.16. Форма программы **Чистый дисконтированный доход**

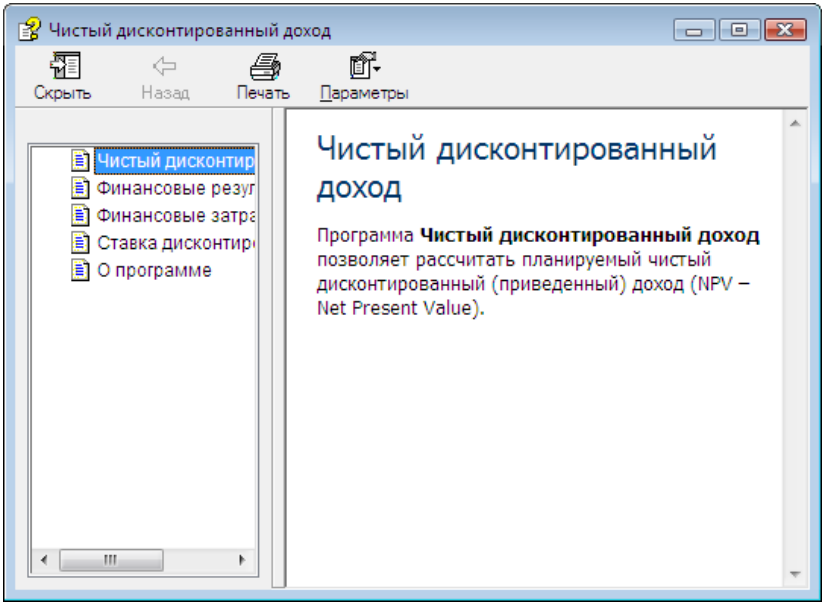


Рис. 1.17. Окно справочной информации программы **Чистый дисконтированный доход**

```
// конструктор формы
Form1(void)
  {
      InitializeComponent();

     // chm-файл получается путем компиляции htm-файлов,
     // в которых находится справочная информация.
     // Обычно каждый раздел справочной информации
     // помещают в отдельный файл. В дальнейшем
     // имя htm-файла используется в качестве
     // раздела идентификатора справочной информации.

      // файл справки
      helpProvider1->HelpNamespace = "npv.chm";
      helpProvider1->SetHelpNavigator(this,
          HelpNavigator::Topic);
      helpProvider1->SetShowHelp(this, true);

      // задать раздел справки
      // для textBox1 — Финансовые результаты
      helpProvider1->SetHelpKeyword(textBox1,
                                    "npv_02.htm");
      helpProvider1->SetHelpNavigator(textBox1,
          HelpNavigator::Topic);
      helpProvider1->SetShowHelp(textBox1, true);

      // задать раздел справки
      // для textBox2 — Финансовые затраты
      helpProvider1->SetHelpKeyword(textBox2,
                                  "npv_03.htm");
      helpProvider1->SetHelpNavigator(textBox2,
          HelpNavigator::Topic);
      helpProvider1->SetShowHelp(textBox2, true);

      // задать раздел справки
      // для textBox3 — Ставка дисконтирования
```

```
      helpProvider1->SetHelpKeyword(textBox3,
                                    "npv_04.htm");
      helpProvider1->SetHelpNavigator(textBox3,
          HelpNavigator::Topic);
      helpProvider1->SetShowHelp(textBox3, true);
  }

// щелчок на кнопке Справка
private: System::Void button2_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
     Help::ShowHelp(this, helpProvider1->HelpNamespace,
                    "npv_01.htm");
}

// щелчок на кнопке Расчет
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{
    double p = 0; // поступления от продаж
    double r = 0; // расходы
    double d = 0; // ставка дисконтирования

    double npv = 0; // чистый дисконтированный доход

    try
    {
        p = Convert::ToDouble(textBox1->Text);
        r = Convert::ToDouble(textBox2->Text);
        d = Convert::ToDouble(textBox3->Text) / 100;

        npv = (p — r) / (1.0 + d);

        label4->Text =
               "Чистый дисконтированный доход (NPV) =  " +
                npv.ToString("c");
    }
```

```
    catch ( System::FormatException^ e)
    {
        //MessageBox::Show( e->Message);
        MessageBox::Show("Ошибка исходных данных.\nУбедитесь,
                что все поля заполнены, а также в том, что в
                качестве десятичного разделителя используется
                запятая.",
           "Чистый дисконтированный доход",

           MessageBoxButtons::OK,MessageBoxIcon::Exclamation);
    }
}
```

Операции с файлами

В этом разделе приведены программы, демонстрирующие выполнение операций с файлами.

Курс

Программа **Курс** добавляет в базу данных, представляющую собой текстовый файл, информацию о текущем курсе доллара. Если файла данных в каталоге приложения нет, программа его создаст. Кнопка **Добавить** становится доступной только после того, как пользователь введет данные в поле редактирования. Форма программы приведена на рис. 1.18. Для ввода даты используется компонент `DateTimePicker`.

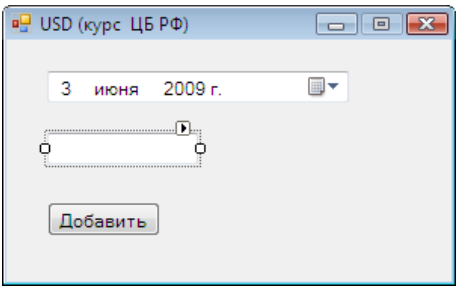


Рис. 1.18. Форма программы **Курс**

```
// конструктор формы
public:
  Form1(void)
  {
      InitializeComponent();

      button1->Enabled = false;
  }

// нажатие клавиши в поле редактирования
private: System::Void textBox1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^  e)
{
    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
                return;

    if (e->KeyChar == '.')
        e->KeyChar = ',';

    if (e->KeyChar == ',')
    {
        // в поле редактирования не может
        // быть больше одной запятой и запятая
        // не может быть первым символом
        if ((textBox1->Text->IndexOf(',') != -1) ||
             (textBox1->Text->Length == 0))
        {
            e->Handled = true;
        }
        return;
     }

    if (Char::IsControl(e->KeyChar))
    {
        // <Enter>, <Backspace>, <Esc>
```

```
        if (e->KeyChar == (char)Keys::Enter)
            // установить курсор на кнопку OK
            button1->Focus();
            return;
    }

    // остальные символы запрещены
    e->Handled = true;
}

// изменилось содержимое поля редактирования
private: System::Void textBox1_TextChanged
          (System::Object^  sender, System::EventArgs^  e)
{
    if (textBox1->Text->Length == 0)
       button1->Enabled = false;
    else
        button1->Enabled = true;
}

// щелчок на кнопке Добавить
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    double kurs;   // курс
    DateTime date; // дата

    date = dateTimePicker1->Value;
    kurs =  System::Convert::ToDouble(textBox1->Text);

    // получить информацию о файле
    System::IO::FileInfo^ fi =
        gcnew System::IO::FileInfo(
        Application::StartupPath + "\\usd.txt");
```

```
    // поток для записи
    System::IO::StreamWriter^ sw;

    if (fi->Exists) // файл данных существует?
        // откроем поток для добавления
        sw = fi->AppendText();
    else
        // создать файл и открыть поток для записи
        sw = fi->CreateText();

    // запись в файл
    sw->WriteLine(date.ToShortDateString());
    sw->WriteLine(kurs.ToString("N"));

    // закрыть поток
    sw->Close();

    // чтобы по ошибке не записать данные
    // второй раз, сделаем недоступным поле
    // ввода и кнопку
    button1->Enabled = false;
    textBox1->Enabled = false;
}

// пользователь выбрал другую дату
private: System::Void
dateTimePicker1_ValueChanged(System::Object^  sender,
                             System::EventArgs^  e)
{
    // очистить и сделать доступным
    // поле ввода
    textBox1->Enabled = true;
    textBox1->Clear();

    // установить курсор в поле ввода
    textBox1->Focus();
}
```

Котировки

Программа **Котировки** демонстрирует чтение данных из текстового файла, а также использование компонентов `MonthCalendar` и `ListBox`. Данные, удовлетворяющие критерию запроса (относящиеся к диапазону дат, выделенному в календаре), считываются из файла, сформированного программой **Курс**. Окно программы приведено на рис. 1.19.

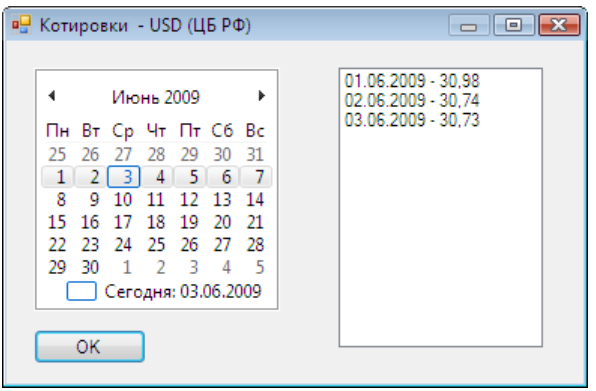


Рис. 1.19. Окно программы **Котировки**

```
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
  System::IO::StreamReader^ sr; // поток для чтения

  try
  {
      // создать поток для чтения
      sr = gcnew System::IO::StreamReader(
          Application::StartupPath + "\\usd.txt",
          System::Text::Encoding::GetEncoding(1251));

      // начало и конец интервала, выделенного
```

```
    // на календаре
    DateTime dateStart =
                    monthCalendar1->SelectionStart;
    DateTime dateEnd =
                    monthCalendar1->SelectionEnd;

    String^ st1;
    String^ st2;

    DateTime date;

    listBox1->Items->Clear();

    // читаем данные из файла
    while ( ! sr->EndOfStream )
    {
        st1 = sr->ReadLine(); // дата как строка
        date = System::Convert::ToDateTime(st1);
        st2 = sr->ReadLine();

        if ((date >= dateStart) &&
                            (date <= dateEnd))
        {
            listBox1->Items->Add(st1 + " — " + st2);
         }
     }
     sr->Close();

     if (listBox1->Items->Count == 0)
     {
        listBox1->Items->Add("--- нет данных ---");
     }
}
catch(System::IO::FileNotFoundException^ e)
{
```

```
        MessageBox::Show
                    ("Нет файла данных\n" +
                     Application::StartupPath + "\\usd.txt",
                     "Котировки",
                     MessageBoxButtons::OK,
                     MessageBoxIcon::Error);
        button1->Enabled = false;
    }
}
```

Редактор текста

Программа **NkEdit** (редактор текста) демонстрирует использование компонентов `MenuStrip`, `TollStrip`, `OpenFileDialog`, `SaveFileDialog`, `FontDialog`, `PrintDialog`, а также отображение дочернего окна — диалога **О программе**. Информация о программе отображается в окне, которое появляется на экране в результате выбора соответствующей команды в меню **Справка**. Главная форма приведена на рис. 1.20, форма **О программе** — на рис. 1.21. Следует обратить внимание на то, что свойству `DialogResult` кнопки **OK** надо присвоить значение `OK`.

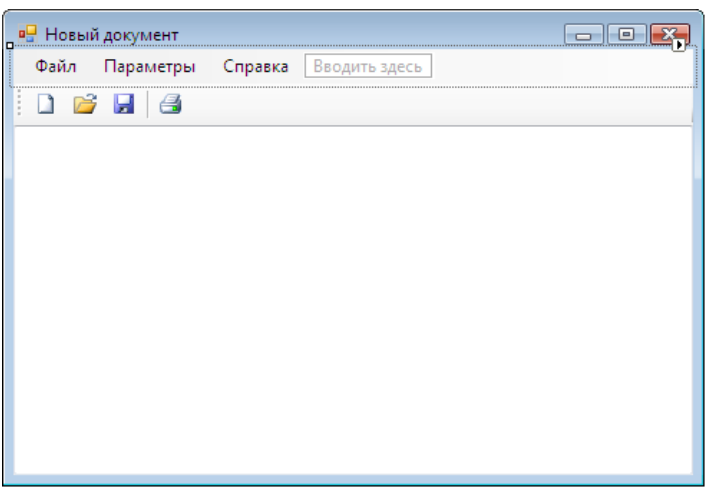


Рис. 1.20. Главная форма программы **NkEdit**

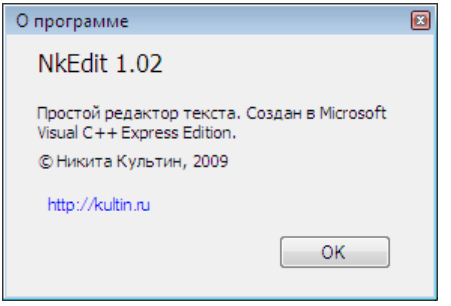


Рис. 1.21. Форма **О программе**

```
// конструктор формы
Form1(void)
  {
      InitializeComponent();

      textBox1->ScrollBars = ScrollBars::Vertical;
      textBox1->Text = "";

      this->Text = "NkEdit — Новый документ";

      // отобразить панель инструментов
      toolStrip1->Visible = true;
      toolStripMenuItem6->Checked = true;

      // назначаем файл справки
      // helpProvider1.HelpNamespace = "nkedit.chm";

      // настройка компонента openDialog1
      openFileDialog1->DefaultExt = "txt";
      openFileDialog1->Filter = "текст|*.txt";
      openFileDialog1->Title = "Открыть документ";
      openFileDialog1->Multiselect = false;

      // настройка компонента saveDialog1
      saveFileDialog1->DefaultExt = "txt";
```

```
        saveFileDialog1->Filter = "текст|*.txt";
        saveFileDialog1->Title = "Сохранить документ";

        fn = String::Empty;
        textChanged = false;
    }

private:
    String^ fn;          // имя файла
    bool textChanged;    // true — в текст внесены изменения


// изменился текст в поле компонента textBox
private: System::Void textBox1_TextChanged(System::Object^
sender, System::EventArgs^  e)
{
    textChanged = true; // текст изменен
}

// Записывает текст в файл.
// Возвращает 0 или -1, если пользователь
// в окне Сохранить нажмет кнопку Отмена
private: int TextToFile()
{
    System::Windows::Forms::DialogResult dr;
    int r = 0;

    if (fn == String::Empty)
    {
        // это новый документ
        // запросить у пользователя имя файла

        // отобразить диалог Сохранить
        dr = saveFileDialog1->ShowDialog();
        if (dr ==
```

```
            System::Windows::Forms::DialogResult::OK)
        {

            fn = saveFileDialog1->FileName;
            r = 0;
        }
        else
            // в окне Сохранить пользователь
            // сделал щелчок на кнопке Отмена
            r = -1;
    }

    // сохранить файл
    if ( r == 0)
    {
        try
        {
            // получим информацию о файле fn
            System::IO::FileInfo^ fi =
                gcnew System::IO::FileInfo(fn);

            // поток для записи
            System::IO::StreamWriter^ sw =
                      fi->CreateText();
            sw->Write(textBox1->Text);

            sw->Close(); // закрываем поток
            textChanged = false;
            r = 0;
        }
        catch ( System::IO::IOException^ e)
        {
            MessageBox::Show(e->ToString(),
                  "NkEdit",
                  MessageBoxButtons::OK,
```

```
                    MessageBoxIcon::Error);
        }
    }
    return r;
}

// Проверяет, есть ли изменения в тексте, и
// сохраняет текст в файле.
// Возвращает:
//   0 или -1, если пользователь
//   отказался от выполнения операции (нажал Cancel)
private: int SaveText()
{
    System::Windows::Forms::DialogResult dr;

    int r;

    r = 0;
    if (textChanged)
    {
        dr = MessageBox::Show(
                 "В текст внесены изменения. Сохранить
                  измененный текст?", "NkEdit",
                 MessageBoxButtons::YesNoCancel,
                 MessageBoxIcon::Warning);
        switch (dr)
        {
            case System::Windows::Forms::DialogResult::Yes:
                r = TextToFile();
                break;

            case System::Windows::Forms::DialogResult::No:
                r = 0;
                break;
```

```
            case System::Windows::Forms::DialogResult::Cancel:
                r = -1;
                break;
        };
    }
    return r;
}

// Открыть документ
private: void OpenDocument()
{
    System::Windows::Forms::DialogResult dr;
    int r;

    r = SaveText();

    if ( r == 0)
    {
        openFileDialog1->FileName = String::Empty;

        // отобразить диалог Открыть

        dr = openFileDialog1->ShowDialog();
        if (dr == System::Windows::Forms::DialogResult::OK)
        {
            fn = openFileDialog1->FileName;

            // отобразить имя файла в заголовке окна
            this->Text = fn;

            try
            {
                // считываем данные из файла
                System::IO::StreamReader^ sr =
                    gcnew System::IO::StreamReader(fn);
```

```
                textBox1->Text = sr->ReadToEnd();
                textBox1->SelectionStart =
                                textBox1->TextLength;
                sr->Close();
                textChanged = false;
            }
            catch (  System::IO::FileLoadException^ e)
            {
                MessageBox::Show("Ошибка:\n" + e->Message,
                         "NkEdit",
                        MessageBoxButtons::OK,
                        MessageBoxIcon::Error);
            }
        }
    }
}

// Сохранить документ
private: void SaveDocument()
{
    int r;

    r = SaveText();

    if ( r == 0)
    {
        this->Text = fn;
        textChanged = false;
    }
}

// Создать документ
private: void NewDocument()
{
    int r;
```

```
    r = SaveText();

    if ( r == 0)
    {
        this->Text = "Новый документ";
        textBox1->Clear();
        textChanged = false;
        fn = String::Empty;
    }
}

// выбор в меню Файл команды Открыть
private: System::Void toolStripMenuItem2_Click
        (System::Object^  sender, System::EventArgs^  e)
{
    OpenDocument();
}

// выбор в меню Файл команды Сохранить
private: System::Void toolStripMenuItem3_Click
           (System::Object^  sender, System::EventArgs^  e)
{
    SaveDocument();
}

// выбор в меню Файл команды Создать
private: System::Void toolStripMenuItem1_Click
           (System::Object^  sender, System::EventArgs^  e)
{
    NewDocument();
}

// выбор в меню Файл команды Печать
private: System::Void toolStripMenuItem4_Click
             (System::Object^  sender, System::EventArgs^  e)
```

```
{
    printDialog1->ShowDialog();
}

// выбор в меню Файл команды Выход
private: System::Void toolStripMenuItem5_Click
           (System::Object^  sender, System::EventArgs^  e)
{
    this->Close();
}

// выбор в меню Параметры команды Панель инструментов
private: System::Void toolStripMenuItem6_Click
            (System::Object^  sender, System::EventArgs^  e)
{
    // отобразить/скрыть панель инструментов
    toolStrip1->Visible = ! toolStrip1->Visible;
    toolStripMenuItem6->Checked =
                         ! toolStripMenuItem6->Checked;
}

// выбор в меню Параметры команды Шрифт
private: System::Void toolStripMenuItem7_Click
           (System::Object^  sender, System::EventArgs^  e)
{
    fontDialog1->Font = textBox1->Font;
    if ( fontDialog1->ShowDialog() ==
             System::Windows::Forms::DialogResult::OK )
    {
        textBox1->Font = fontDialog1->Font;
    }
}

// щелчок на кнопке Создать
private: System::Void toolStripButton1_Click
```

```
            (System::Object^  sender, System::EventArgs^  e)
{
    NewDocument();
}

// щелчок на кнопке Открыть
private: System::Void toolStripButton2_Click
            (System::Object^  sender, System::EventArgs^  e)
{
    OpenDocument();
}

// щелчок на кнопке Сохранить
private: System::Void toolStripButton3_Click
           (System::Object^  sender, System::EventArgs^  e)
{
    SaveDocument();
}

// щелчок на кнопке Печать
private: System::Void toolStripButton4_Click
            (System::Object^  sender, System::EventArgs^  e)
{
    printDialog1->ShowDialog();
}

// попытка закрыть окно программы
private: System::Void Form1_FormClosing
            (System::Object^  sender,
             System::Windows::Forms::FormClosingEventArgs^ e)
{
    int r;

    r = SaveText();

    if ( r != 0)
```

```
    {
        e->Cancel = true;
    }
}

// выбор в меню Справка команды О программе
private: System::Void toolStripMenuItem9_Click
             (System::Object^  sender, System::EventArgs^  e)
{
    // чтобы класс Form2 стал доступен,
    // в начало программы надо вставить
    // директиву #include "Form2.h",
    // где Form2.h — модуль, в котором определен
    // класс Form2
    Form2^ about = gcnew Form2();

    about->ShowDialog();
}
```

Графика

В этом разделе собраны программы, демонстрирующие работу с графикой.

Общие замечания

- Вывод графики осуществляется на графическую поверхность объекта (графическая поверхность формы, компонента или заданная вручную поверхность).
- Для того чтобы на поверхности объекта появился графический элемент (линия, окружность, прямоугольник и т. д.) или изображение, необходимо применить к графической поверхности соответствующий метод.
- Методы `DrawLine`, `DrawRectangle`, `DrawPie` и т. д. выполняют прорисовку контура объекта. Заполнение внутренней области

объекта осуществляется при помощи методов `FillRectangle`, `FillPie` и т. д.

- Цвет закраски внутренних областей геометрических фигур, вычерчиваемых методами `FillRectangle`, `FillPie`, определяется параметрами вызываемых методов.
- Характеристики шрифта текста, выводимого методом `DrawString`, определяются параметрами вызова метода. Если они не указаны, используются характеристики шрифта (тип, цвет, размер), указанные для основной формы программы.
- Основную работу по выводу графики на поверхность формы выполняет функция обработки события `Paint`.

Прямоугольники

Программа **Прямоугольники** (рис. 1.22) демонстрирует принцип формирования графики, рисует на поверхности формы итальянский флаг.

Рис. 1.22. Окно программы **Прямоугольники**

```
// обработка события Paint
System::Void Form1_Paint(System::Object^  sender,
                System::Windows::Forms::PaintEventArgs^  e)
{
    int w,h;    // размер флага
```

```
int ws;     // ширина полосы
int x0,y0; // координаты левого верхнего угла флага

int x,y;    // координаты левого верхнего угла полосы

w = 83;
h = 60;

// разместим флаг в центре формы:
// ClientSize.Width — ширина внутренней области формы;
// ClientSize.Height — высота внутренней области формы;

x0 = (this->ClientSize.Width — w) / 2;
y0 = (this->ClientSize.Height — h) /2;

ws = w / 3; // флаг из трех вертикальных полос

// рисуем флаг
x = x0;
y = y0;

// зеленая полоса
e->Graphics->FillRectangle
              (System::Drawing::Brushes::Green,x,y,ws,h);

// белая полоса
x = x + ws+1;
e->Graphics->FillRectangle
              (System::Drawing::Brushes::White,x,y,ws,h);

// красная полоса
x = x + ws+1;
e->Graphics->FillRectangle
              (System::Drawing::Brushes::Red,x,y,ws,h);
```

```
    // окантовка
    e->Graphics->DrawRectangle
                    (System::Drawing::Pens::Black,x0,y0,w,h);
}


// событие SizeChanged возникает при изменении размера формы
private: System::Void Form1_SizeChanged
             (System::Object^  sender, System::EventArgs^  e)
{
    // метод Refresh информирует систему о необходимости
    // перерисовать (обновить) форму. В результате
    // генерируется событие Paint
    this->Refresh(); // обновить форму
}
```

Вывод текста

Программа **DrawString** (рис. 1.23) показывает, как вывести текст на поверхность формы.

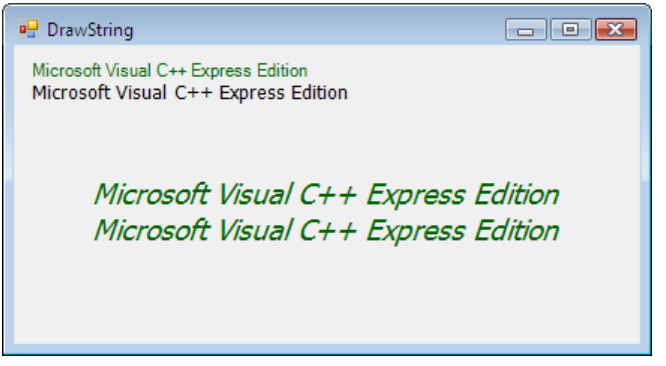


Рис. 1.23. Окно программы **DrawString**

```
// обработка события Paint
private: System::Void Form1_Paint(System::Object^  sender,
```

```
System::Windows::Forms::PaintEventArgs^  e)
{
   int x,y;
   int w,h; // размер области отображения текста

   x = 10;
   y = 10;

   String^ st =
           "Microsoft Visual C++ Express Edition";

   // вывод текста шрифтом, заданным значением
   // свойства Font формы. Brushes — коллекция
   // стандартных кистей.
   e->Graphics->DrawString(st,
       this->Font, Brushes::DarkGreen, x, y);

   // определить размер области отображения
   // выведенной строки

    h = (int)e->Graphics->MeasureString
                          (st, this->Font).Height;

    // вычислить Y-координату области отображения
    // второй строки
    y = y + h;

   // вывод текста шрифтом, заданным программистом:
   System::Drawing::Font^ aFont =
                gcnew System::Drawing::Font("Tahoma", 10,
                                    FontStyle::Regular);

    e->Graphics->DrawString(st,
                          aFont, Brushes::Black, x, y);
```

```
    // вывод текста в центре формы
    System::Drawing::Font^ hFont =
              gcnew System::Drawing::Font("Tahoma", 14,
                          FontStyle::Italic);


    // размер области отображения текста зависит от
    // характеристик шрифта, которым он отображается

    // определить размер области отображения текста
    w = (int)e->Graphics->
                MeasureString(st, hFont).Width;
    h = (int)e->Graphics->
                MeasureString(st, hFont).Height;

    // вычислить координаты левого верхнего угла
    // области отображения текста, так чтобы текст
    // был размещен в центре формы по горизонтали и
    // вертикали. ClientSize.Width — размер внутренней
    // области формы
    x = (this->ClientSize.Width — w) / 2;
    y = (this->ClientSize.Height — h) / 2;

    // вывести текст в центре формы
    e->Graphics->DrawString(st, hFont,
        System::Drawing::Brushes::DarkGreen, x, y);

    // выведем текст еще раз
    e->Graphics->DrawString(st, hFont,
        System::Drawing::Brushes::DarkGreen, x, y + h);
}

// изменился размер формы
private: System::Void Form1_Resize(System::Object^  sender,
System::EventArgs^  e)
```

```
{
    // сообщить системе о необходимости
    // перерисовать окно. В результате будет
    // сгенерировано событие Paint.
    this->Refresh();
}
```

Диаграмма

В окне программы **Диаграмма** (рис. 1.24) отображается диаграмма изменения курса доллара (данные загружаются из файла). Программа демонстрирует работу с массивами, чтение данных из файла, работу с методами `DrawString`, `DrawRectangle`, `FillRectangle`.

Рис. 1.24. Окно программы **Диаграмма**

```
public:
Form1(void)
{
    InitializeComponent();

    // чтение данных из файла в массив
```

```
System::IO::StreamReader^ sr; // поток для чтения
try
{
    sr = gcnew System::IO::StreamReader(
        Application::StartupPath + "\\usd.dat");

    // создаем массив
    d = gcnew array<double>(10);

    // читаем данные из файла
    int i = 0;
    String^ t = sr->ReadLine();
    while ((  t != String::Empty) && (i < d->Length))
    {
        // записываем считанное число в массив
        d[i++] = Convert::ToDouble(t);
        t = sr->ReadLine();
    }

    // закрываем поток
    sr->Close();

    // данные загружены
    // задаем функцию обработки события Paint
    this->Paint +=
     gcnew System::Windows::Forms::PaintEventHandler
                 (this, &Form1::drawDiagram);
}

// обработка исключений:
// — файл данных не найден
catch (System::IO::FileNotFoundException^ ex)
{
    MessageBox::Show( ex->Message + "\n",
```

```
            "График",
            MessageBoxButtons::OK,
            MessageBoxIcon::Error);
    }
}

private:

// данные
array<double>^ d;

// строит график
void drawDiagram(System::Object^ sender,
      System::Windows::Forms::PaintEventArgs^  e)
{
    // графическая поверхность
    System::Drawing::Graphics^ g = e->Graphics;

    // шрифт подписей данных
    System::Drawing::Font^ dFont = gcnew
                System::Drawing::Font("Tahoma", 9);



    // шрифт заголовка
    System::Drawing::Font^ hFont = gcnew
              System::Drawing::Font("Tahoma", 13,
                                    FontStyle::Regular);
    String^ header = "Изменение курса доллара";

    // ширина области отображения текста
    int wh =(int)g->MeasureString(header, hFont).Width;

    int x = (this->ClientSize.Width — wh) / 2;

    g->DrawString(header,hFont, Brushes::DarkGreen, x, 5);
```

```
/* Область построения графика.
   Отступы:
       сверху — 100;
       снизу — 20;
       слева — 20;
       справа — 20.

    ClientSize — размер внутренней области окна

    График строим в отклонениях от минимального
    значения ряда данных так, чтобы он занимал
    всю область построения
*/

double max = d[0]; // максимальный эл-т массива
double min = d[0]; // минимальный эл-т массива

for (int i = 1; i < d->Length; i++)
{
    if (d[i] > max) max = d[i];
    if (d[i] < min) min = d[i];
}

// рисуем график
int x1, y1, x2, y2;

// расстояние между точками графика (шаг по X)
int sw = (int)((this->ClientSize.Width — 40) /
                        (d->Length — 1));

// первая точка
x1 = 20;
y1 = this->ClientSize.Height — 20 -
            (int)((this->ClientSize.Height — 100) *
```

```
            (d[0] — min) / (max — min));

// маркер первой точки
g->DrawRectangle(Pens::Black,
            x1 — 2, y1 — 2, 4, 4);

// подпись численного значения первой точки
g->DrawString(Convert::ToString(d[0]),
    dFont, Brushes::Black,
            x1 — 10, y1 — 20);

// остальные точки
for (int i = 1; i < d->Length; i++)
{
    x2 = 8 + i * sw;
    y2 = this->ClientSize.Height — 20 -
                (int)((this->ClientSize.Height — 100) *
                (d[i] — min) / (max — min));

    // маркер точки
    g->DrawRectangle(Pens::Black,
                 x2 — 2, y2 — 2, 4, 4);

    // соединим текущую точку с предыдущей
    g->DrawLine(Pens::Black,
                x1, y1, x2, y2);

    // подпись численного значения
    g->DrawString(Convert::ToString(d[i]),
        dFont, Brushes::Black,
                x2 — 10, y2 — 20);

    x1 = x2;
    y1 = y2;
}
```

```
}

private: System::Void Form1_Resize(System::Object^  sender,
System::EventArgs^  e)
{
    this->Refresh();
}
```

График

В окне программы **График** (рис. 1.25) отображается график изменения курса доллара. Данные загружаются из файла.

Рис. 1.25. Окно программы **График**

```
public:
Form1(void)
{
    InitializeComponent();

    // чтение данных из файла в массив
```

```
System::IO::StreamReader^ sr; // поток для чтения
try
{
    sr = gcnew System::IO::StreamReader(
        Application::StartupPath + "\\usd.dat");

    // создаем массив
    d = gcnew array<double>(10);

    // читаем данные из файла
    int i = 0;
    String^ t = sr->ReadLine();
    while ((  t != String::Empty) && (i < d->Length))
    {
        // записываем считанное число в массив
        d[i++] = Convert::ToDouble(t);
        t = sr->ReadLine();
    }

    // закрываем поток
    sr->Close();

    // данные загружены
    // задаем функцию обработки события Paint
    this->Paint += gcnew
           System::Windows::Forms::PaintEventHandler
          (this, &Form1::drawDiagram);
}

// обработка исключений:
// — файл данных не найден
catch (System::IO::FileNotFoundException^ ex)
{
    MessageBox::Show( ex->Message + "\n",
```

```
            "График",
            MessageBoxButtons::OK,
            MessageBoxIcon::Error);
    }
}

private:

// данные
array<double>^ d;

// строит график
void drawDiagram(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e)
{
    // графическая поверхность
    System::Drawing::Graphics^ g = e->Graphics;

    // шрифт подписей данных
    System::Drawing::Font^ dFont = gcnew
            System::Drawing::Font("Tahoma", 9);

    // ** заголовок **
    // шрифт заголовка
    System::Drawing::Font^ hFont = gcnew
              System::Drawing::Font("Tahoma", 14,
              FontStyle::Regular);
    String^ header = "Изменение курса доллара";

    // ширина области отображения текста
    int wh =(int)g->MeasureString(header, hFont).Width;

    int x = (this->ClientSize.Width — wh) / 2;

    g->DrawString(header,hFont, Brushes::DarkGreen, x, 5);
```

```
/* Область построения диаграммы.
   Отступы:
       сверху — 100;
       снизу — 20;
       слева — 20;
       справа — 20.

    ClientSize — размер внутренней области окна

    График строим в отклонениях от минимального
    значения ряда данных так, чтобы он занимал
    всю область построения
*/

double max = d[0]; // максимальный эл-т массива
double min = d[0]; // минимальный эл-т массива

for (int i = 1; i < d.Length; i++)
{
    if (d[i] > max) max = d[i];
    if (d[i] < min) min = d[i];
}

// рисуем диаграмму
int x1, y1; // координаты левого верхнего
            //  угла столбика
int w, h;   // размер столбика

// ширина столбиков диаграммы
// 5 — расстояние между столбиками
// d.Length — кол-во рядов данных (столбиков)

w = (ClientSize.Width — 40 — 5 * (d.Length — 1))
```

```
                                        / d.Length;
    x1 = 20;
    for (int i = 0; i < d.Length; i++)
    {
        y1 = ClientSize.Height — 20 -
                 (int)((ClientSize.Height — 100) *
                    (d[i] — min) / (max — min));

        // подпись численного значения первой точки
        g->DrawString(Convert::ToString(d[i]),
                    dFont, Brushes::Black,
                    x1, y1 — 20);

        // рисуем столбик — зеленый прямоугольник
        h = ClientSize.Height — y1 — 20 + 1;

        g->FillRectangle(Brushes::ForestGreen,
                                x1, y1, w, h);

        // контур прямоугольника
        g->DrawRectangle(Pens::Black,
                                x1, y1, w, h);

         x1 = x1 + w +5;
    }
}


private: System::Void Form1_Resize(System::Object^ sender,
                                   System::EventArgs^ e)
             {
                 this->Refresh();
             }
```

Круговая диаграмма

В окне программы **Круговая диаграмма** (рис. 1.26) отображаются результаты социологического опроса. Исходные данные (вопрос, варианты ответа и количество ответов) загружаются из файла. Программа обрабатывает данные, вычисляет долю каждой категории и строит диаграмму.

Рис. 1.26. Окно программы **Круговая диаграмма**

```
private:
    String^ header;  // заголовок диаграммы

    // количество элементов данных
    int N;

    array<double>^ dat; // ряд данных
    array<double>^ p;   // доля категории в общей сумме

    // подписи данных
   array<String^>^ title;
```

```
public:
    Form1(void)
    {
      InitializeComponent();

      System::IO::StreamReader^ sr;


      try
      {

        sr = gcnew System::IO::StreamReader(
            Application::StartupPath + "\\date.dat",
            System::Text::Encoding::GetEncoding(1251));

        // считываем заголовок диаграммы
        header = sr->ReadLine();

        // считываем данные о количестве записей
        N = Convert::ToInt16(sr->ReadLine());

        // и инициализируем массивы
        dat = gcnew array<double>(N);
        p = gcnew array<double>(N);
        title = gcnew array<String^>(N);

        // читаем данные
        int i = 0;
        String^ st;
        st = sr->ReadLine();
        while ((st != String::Empty) && (i < N))
        {
            title[i] = st;

            st = sr->ReadLine();
```

```
            dat[i++] = Convert::ToDouble(st);

            st = sr->ReadLine();
        }

        // закрываем поток
        sr->Close();

        // данные загружены
        // задаем функцию обработки события Paint
        this->Paint += gcnew
            System::Windows::Forms::PaintEventHandler
                (this, &Form1::drawDiagram);

        double sum = 0;
        int j = 0;

        // вычислить сумму
        for (j = 0; j < N; j++)
            sum += dat[j];

        // вычислить долю каждой категории
        for (j = 0; j < N; j++)
                p[j] = (double)(dat[j] / sum);
    }
    catch (System::IO::FileNotFoundException^ ex)
        {
            MessageBox::Show(ex->Message,
                "Диаграмма",
                MessageBoxButtons::OK,
                MessageBoxIcon::Error);
        }
     }

// рисует круговую диаграмму
void drawDiagram(System::Object^  sender,
```

```
        System::Windows::Forms::PaintEventArgs^  e)
{
    // графическая поверхность
    System::Drawing::Graphics^ g = e->Graphics;

    // шрифт заголовка
            System::Drawing::Font^ hFont =
               gcnew System::Drawing::Font("Tahoma", 12);

    // выводим заголовок
    int w = (int)g->MeasureString(header, hFont).Width;
    int x = (ClientSize.Width — w) / 2;

    g->DrawString(header,hFont, Brushes::Black, x, 10);

    // шрифт легенды
    System::Drawing::Font^ lFont =
            gcnew System::Drawing::Font("Tahoma", 9);

    // диаметр диаграммы
    int d = ClientSize.Height — 80;

    int x0 = 30;
    int y0 = (ClientSize.Height — d) / 2 + 10;

    // координаты верхнего левого угла
    // области легенды
    int lx = 60 + d;
    int ly = y0 + (d — N * 20 + 10) / 2;

    int swe; // длина дуги сектора

    // кисть для заливки сектора диаграммы
    System::Drawing::Brush^  fBrush = Brushes::White;
```

```
// начальная точка дуги сектора
int sta = -90;

// рисуем диаграмму
for (int i = 0; i < N; i++)
{
    // длина дуги
    swe = (int)(360 * p[i]);

    // задать цвет сектора
    switch (i)
    {
        case 0:
            fBrush = Brushes::YellowGreen;
            break;
        case 1:
            fBrush = Brushes::Gold;
            break;
        case 2:
            fBrush = Brushes::Pink;
            break;
        case 3:
            fBrush = Brushes::Violet;
            break;
        case 4:
            fBrush = Brushes::OrangeRed;
            break;
        case 5:
            fBrush = Brushes::RoyalBlue;
            break;
        case 6:
            fBrush = Brushes::SteelBlue;
            break;
        case 7:
```

```
            fBrush = Brushes::Chocolate;
            break;
        case 8:
            fBrush = Brushes::LightGray;
            break;
    }

    // из-за округления возможна ситуация,
    // при которой будет промежуток между
    // последним и первым секторами
    if (i == N — 1)
        // последний сектор
        swe = 270 — sta;

    // рисуем сектор
    g->FillPie(fBrush, x0, y0, d, d, sta, swe);

    // рисуем границу сектора
    g->DrawPie(Pens::Black, x0, y0,
                    d, d, sta, swe);

    // прямоугольник легенды
    g->FillRectangle(fBrush,
                     lx, ly + i * 20, 20, 10);
    g->DrawRectangle(Pens::Black,
                     lx, ly + i * 20, 20, 10);

    // подпись
    g->DrawString( title[i] + " — " +
                p[i].ToString("P"),
                lFont, Brushes::Black,
                     lx + 24, ly + i * 20 — 3);

    // начальная точка дуги для следующего сектора
    sta = sta + swe;
```

```
    }
}

private: System::Void Form1_Resize(System::Object^ sender,
                                   System::EventArgs^ e)
         {
             this->Refresh();
         }
```

Кисти

Программа **Кисти** (рис. 1.27) демонстрирует работу с кистями различных типов. Следует обратить внимание: для того чтобы градиентные и текстурные кисти стали доступны, в программу надо добавить ссылку на пространство имен `System::Drawing::Drawing2D`.

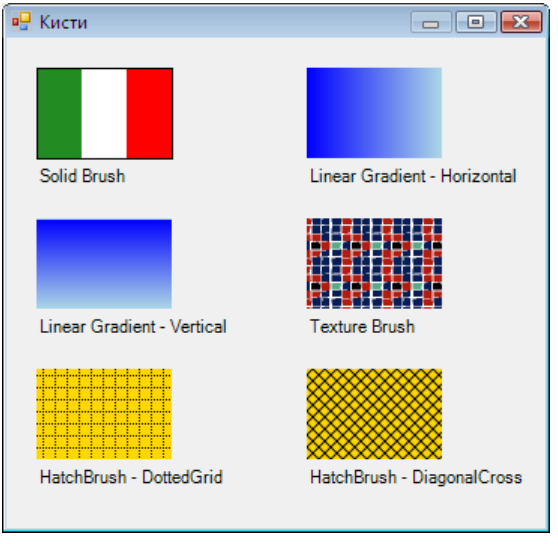


Рис. 1.27. Окно программы **Кисти**

```
private:
System::Void Form1_Paint(System::Object^  sender,
                System::Windows::Forms::PaintEventArgs^  e)
```

```
{

    // чтобы не писать e->Graphics, определим
    // графическую поверхность g
    System::Drawing::Graphics^ g = e->Graphics;

    // Solid Brush
    g->FillRectangle(Brushes::ForestGreen, 20, 20, 30, 60);
    g->FillRectangle(Brushes::White, 50, 20, 30, 60);
    g->FillRectangle(Brushes::Red, 80, 20, 30, 60);
    g->DrawRectangle(Pens::Black, 20, 20, 90, 60);
    g->DrawString("Solid Brush", this->Font,
        Brushes::Black, 20, 85);

   // градиентная кисть LinearGradientBrush

    LinearGradientBrush^ lgBrush_1 =
                gcnew LinearGradientBrush(
                    RectangleF(200, 20, 90, 10),
                    Color::Blue, Color::LightBlue,
                    LinearGradientMode::Horizontal);

    g->FillRectangle(lgBrush_1, 200, 20, 90, 60);
    g->DrawString( "Linear Gradient — Horizontal",
               this->Font, Brushes::Black, 200, 85);

    // градиентная кисть LinearGradientBrush
    LinearGradientBrush^ lgBrush_2 =
                gcnew LinearGradientBrush(
                    RectangleF(0, 0, 10, 60),
                    Color::Blue, Color::LightBlue,
                    LinearGradientMode::Vertical);

    g->FillRectangle(lgBrush_2, 20, 120, 90, 60);
    g->DrawString("Linear Gradient — Vertical",
```

```
        this->Font, Brushes::Black, 20, 185);

// текстурная кисть
try
{
    TextureBrush^ tBrush =
        gcnew TextureBrush(Image::FromFile
           ( Application::StartupPath+"\\brush_1.bmp"));

    g->FillRectangle(tBrush, 200, 120, 90, 60);
}
catch (System::IO::FileNotFoundException^ e)
{
    g->DrawRectangle(Pens::Black, 200, 120, 90, 60);
    g->DrawString("Source image",
         this->Font, Brushes::Black, 200, 125);
    g->DrawString(" not found",
         this->Font, Brushes::Black, 200, 140);
}

g->DrawString("Texture Brush", this->Font,
            Brushes::Black, 200, 185);

// штриховка ( HatchBrush кисть)
HatchBrush^ hBrush =
            gcnew HatchBrush(HatchStyle::DottedGrid,
            Color::Black, Color::Gold);

g->FillRectangle(hBrush, 20, 220, 90, 60);
g->DrawString("HatchBrush – DottedGrid",
            this->Font, Brushes::Black, 20, 285);

HatchBrush^ hBrush_2 =
            gcnew HatchBrush(HatchStyle::DiagonalCross,
```

```
                Color::Black, Color::Gold);

    g->FillRectangle(hBrush_2, 200, 220, 90, 60);
            g->DrawString(
                 "HatchBrush — DiagonalCross", this->Font,
                 Brushes::Black, 200, 285);
}
```

Бегущая строка

Программа **Бегущая строка** (рис. 1.28) демонстрирует использование битового образа для отображения баннера в стиле бегущей строки. При наведении курсора мыши на баннер процесс прокрутки приостанавливается.

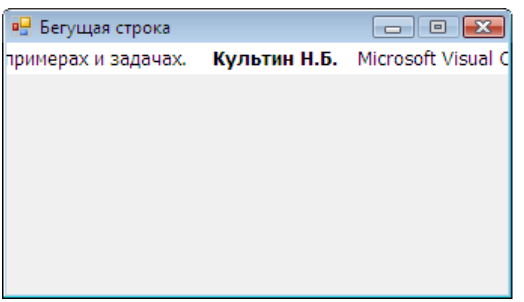


Рис. 1.28. Окно программы **Бегущая строка**

```
public:
    Form1(void)
    {
        InitializeComponent();
        //
        try
        {
            bm = gcnew
```

```
                System::Drawing::Bitmap
                (Application::StartupPath + "\\banner.png");
        }
        catch (System::Exception^ e)
        {
            MessageBox::Show(
               "Ошибка загрузки баннера (" +
                Application::StartupPath + "\\banner.png)",
               "Бегущая строка",
               MessageBoxButtons::OK,
               MessageBoxIcon::Error);
            this->Close();// закрываем форму
            return;
        }

        // определяем графическую поверхность
        g = this->CreateGraphics();

        // определяем область отображения баннера
        rct.X = 0;
        rct.Y = 0;
        rct.Width = bm->Width;
        rct.Height = bm->Height;

        // настройка таймера
        timer1->Interval = 25;
        timer1->Enabled = true;
    }

private:
    // баннер
    System::Drawing::Bitmap^  bm;

    // графическая поверхность
    Graphics^ g;
```

```
    // область вывода баннера
    Rectangle rct;

// сигнал от таймера
private: System::Void timer1_Tick(System::Object^ sender,
                                  System::EventArgs^  e)
{
    // сместить область отображения
    // баннера влево
    rct.X -= 1;

    // если область отображения целиком
    // "уехала" за левую границу формы,
    // вернем ее обратно
    if (Math::Abs(rct.X) > rct.Width)
        rct.X += rct.Width;

    // т. к. область отображения едет влево, то после
    // вывода в "съехавшую" область, выведем баннер
    // еще раз (как минимум один, если ширина формы
    // равна ширине баннера)
    for (int i = 0; i <= Convert::ToInt16(
                this->ClientSize.Width / rct.Width) + 1; i++)
        g->DrawImage(bm, rct.X + i * rct.Width, rct.Y);
}

// перемещение указателя мыши
private: System::Void Form1_MouseMove(System::Object^ sender,
                  System::Windows::Forms::MouseEventArgs^ e)
{
    // при наведении указателя мыши на баннер,
    // его прокрутка (движение) приостанавливается
    if ((e->Y < rct.Y + rct.Height) && (e->Y > rct.Y))
    {
```

```
        // указатель в области отображения баннера
        if (timer1->Enabled )
            // строка "бежит"
            timer1->Enabled = false;
    }
    else
        // указатель не на баннере
        {
            if ( ! timer1->Enabled )
                // строка "стоит",
                // запустим ее
                timer1->Enabled = true;
        }
}
```

Часы

Программа **Часы** (ее окно приведено на рис. 1.29) демонстрирует принцип формирования динамической графики.

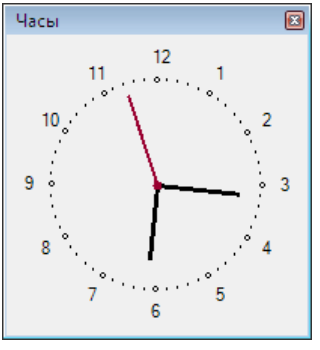


Рис. 1.29. Окно программы **Часы**

```
public:
  Form1(void)
  {
```

```
    InitializeComponent();

    // g — графическая поверхность, на которой
    // будем формировать рисунок
    // определяем графическую поверхность
    g = this->CreateGraphics();

    R =70;

    // установим размер формы
    // в соответствии с размером циферблата

    this->ClientSize =
        System::Drawing::Size((R + 30)*2,(R + 30)*2);

    x0  =  R + 30;
    y0  =  R + 30;

    // Определить положение стрелок.
    // Угол между метками часов (цифрами) — 30 градусов.
    // Угол между метками минут — 6 градусов.
    // Угол отсчитываем от 12-ти часов
    ahr  =   90 —  DateTime::Now.Hour *30 -
                  ( DateTime::Now.Minute / 12) *6;
    amin  =  90 — DateTime::Now.Minute *6;
    asec  =  90 — DateTime::Now.Second *6;

    timer1->Interval = 1000; // период сигнала от таймера 1 с
    timer1->Enabled  =  true; // пуск таймера
}

private:
    // графическая поверхность
    Graphics^ g;
```

```
    int R;              // радиус циферблата
    int  x0, y0;        // центр циферблата
    int ahr,amin,asec;  // положение стрелок (угол)

private:
// сигнал от таймера
System::Void timer1_Tick(System::Object^  sender,
                         System::EventArgs^  e)
{
    // нарисовать стрелки
    DrawClock();
}

private:
System::Void Form1_Paint(System::Object^  sender,
                  System::Windows::Forms::PaintEventArgs^ e)
{

    int x, y;  // координаты маркера на циферблате
    int a;     // угол между OX и прямой (x0,yo) (x,y)
    int h;     // метка часовой риски

    a = 0; // метки ставим от 3-х часов против часовой стрелки
    h = 3; // угол 0 градусов — это 3 часа

#define TORAD 0.0174532

    // циферблат
    while ( a < 360 )
    {
        x = x0 +  R * Math::Cos(a * TORAD);
        y = x0 —  R * Math::Sin(a * TORAD);
        //   Form1->Canvas->MoveTo(x,y);
        if ( (a % 30) == 0 )
```

```
        {
            g->DrawEllipse(Pens::Black,x-2,y-2,3,3);
            // цифры по большему радиусу
            x = x0 +  (R+15) * Math::Cos(a * TORAD);
            y = x0 -  (R+15) * Math::Sin(a * TORAD);
            g->DrawString(h.ToString(),
                   this->Font,Brushes::Black,x-5,y-7);
            h--;
            if ( h == 0 ) h = 12;
        }
        else
            g->DrawEllipse(Pens::Black,x-1,y-1,1,1);

        a = a + 6; // 1 минута — 6 градусов
    }

    // стрелки
    DrawClock();
}

// Рисует вектор из точки (x0,y0) под
// углом a относительно оси X
// L — длина вектора
void Vector(float x0, float y0, float a, float l,
                System::Drawing::Pen^ aPen)
{
    // x0,y0 — начало вектора
    // a — угол между осью x и вектором
    // l — длина вектора

// коэфф. пересчета угла из градусов в радианы
#define TORAD 0.0174532

  float x, y;       // координаты конца вектора
```

```
  x  =  x0 + l * Math::Cos(a*TORAD);
  y  =  y0 — l * Math::Sin(a*TORAD);

  g->DrawLine(aPen,x0,y0,x,y);
}

// рисует стрелки
void DrawClock(void)
{
    // карандаши для рисования стрелок
    Pen^ hPen = gcnew Pen(Color::LightBlue,3);
    Pen^ mPen = gcnew Pen(Color::LightBlue,3);
    Pen^ sPen = gcnew Pen(Color::Black,2);

    // карандаш для стирания стрелок
    Pen^ cPen = gcnew Pen(SystemColors::Control,4);

    // шаг секундной и минутной стрелок 6 градусов,
    // часовой — 30.

    // стереть изображение стрелок:
    Vector(x0,y0, ahr, R-20, cPen);  // часовую
    Vector(x0,y0, amin, R-15, cPen); // минутную
    Vector(x0,y0, asec, R-7, cPen);  // секундную

    // новое положение стрелок
    ahr  =   90 —  DateTime::Now.Hour *30 -
              ( DateTime::Now.Minute / 12) *6;
    amin  =  90 — DateTime::Now.Minute *6;
    asec  =  90 — DateTime::Now.Second *6;

    // нарисовать стрелки:
    // часовую
    Vector(x0,y0, ahr, R-20, hPen);
    // минутную
```

```
    Vector(x0,y0, amin, R-15, mPen);

    // секундную
    Vector(x0,y0, asec, R-7, sPen);

    //g->DrawEllipse(Pens::Black,x0-2,y0-2,4,4);
    g->FillEllipse(Brushes::Black,x0-3,y0-3,6,6);
}
```

Полет

Программа **Полет** (рис. 1.30) демонстрирует принцип реализации так называемой спрайтовой анимации (спрайт — небольшая картинка). Кадры анимации формируются путем вывода в нужную область окна изображения объекта (спрайта). Фоновый рисунок и изображение объекта (рис. 1.31) загружаются из ресурса.

Рис. 1.30. Окно программы **Полет**

Рис. 1.31. Фоновый рисунок и объект

```
public:
  Form1(void)
  {
    InitializeComponent();
    //

    try
    {
        sky = gcnew  System::Drawing::Bitmap
   (Application::StartupPath + "\\sky.bmp");
        plane = gcnew  System::Drawing::Bitmap
    (Application::StartupPath + "\\plane.bmp");
    }
    catch (System::Exception^ e)
    {
        MessageBox::Show(
            "Ошибка загрузки битового образа: "
              + e->Message,
            "Полет",
            MessageBoxButtons::OK,
            MessageBoxIcon::Error);
        this->Close();// закрываем форму
        return;
    }

    // сделать прозрачным фон
    plane->MakeTransparent();

    // установить размер формы равным
    // размеру фонового рисунка
    this->ClientSize = sky->Size;

     // задать фоновый рисунок формы
     this->BackgroundImage = gcnew Bitmap(sky);
```

```
    // g — графическая поверхность, на которой
    // будем формировать рисунок
     // определяем графическую поверхность
    g = this->CreateGraphics();

    // инициализация генератора случ. чисел
    rnd = gcnew System::Random();

    // исходное положение самолета
    rct.X = -40;
    rct.Y = 20 + rnd->Next(20);

    rct.Width = plane->Width;
    rct.Height = plane->Height;

    /*
    скорость полета определяется периодом
    следования сигналов от таймера и величиной
    приращения координаты X
   */

    dx = 2; // скорость полета — 2 пиксела/тик_таймера

    timer1->Interval = 20;
    timer1->Enabled = true;
  }

private:
        System::Drawing::Bitmap^  sky;
        System::Drawing::Bitmap^ plane;

        // рабочая графическая поверхность
        Graphics^ g;

        // приращение координаты X,
```

```
        // определяет скорость полета
        int dx;

        // область, в которой находится объект
        Rectangle rct;

        // генератор случайных чисел
        System::Random^ rnd;

private:
System::Void timer1_Tick(System::Object^  sender,
                         System::EventArgs^  e)
{
    // Стереть изображение объекта — вывести
    // фрагмент фона в ту область
    // графической поверхности, в которой
    // сейчас находится объект
    g->DrawImage(sky,
                 Rectangle(rct.X,rct.Y,rct.Width,rct.Height),
                 Rectangle(rct.X,rct.Y,rct.Width,rct.Height),
                 GraphicsUnit::Pixel);

    // вычислить новое положение объекта
    if (rct.X < this->ClientRectangle.Width)
          rct.X += dx;
    else
    {
        // объект достиг правой границы,
        // перемещаем его к левой границе
        rct.X = -40;
        rct.Y = 20 + rnd->Next(40);

        // скорость полета от 2 до 5
        // пикселов/тик_таймера
```

```
        dx = 2 + rnd->Next(4);
    }

    g->DrawImage(plane,rct.X,rct.Y);
}
```

Базы данных

В Microsoft Visual C++ есть компоненты работы с базами данных Microsoft Access и Mictosoft SQL Server. Также есть возможность работы с базой данных Microsoft SQL Server Compact Edition.

Общие замечания

- Работу с базами данных Microsoft Access обеспечивают компоненты `oleDbConnection` и `oleDbDataAdapter`. Для того чтобы они стали доступны (отображались на панели элементов), надо в меню **Сервис** выбрать команду **Выбрать элементы панели элементов** и на вкладке **Элементы .NET Framework** установить в выбранное состояние флажки, находящиеся рядом с их именами.
- Соединение с базой данных (сервером) обеспечивает компонент `oleDbConnection`.
- Взаимодействие с базой данных путем отправки соответствующих SQL-команд серверу обеспечивает компонент `oleDbDataAdapter`.
- Хранение информации, полученной из базы данных, обеспечивает компонент `dataSet`.
- Отображение информации, полученной из базы данных, а также выполнение операций редактирования, добавления и удаления записей обеспечивает компонент `dataGridView`.

Контакты

Программа **Контакты** является примером приложения работы с базой данных Microsoft Access. Демонстрирует использование компонентов `oleDbConnection`, `oleDbDataAdapter`, `dataSet`

и `dataGridView`. База данных **Контакты** (`contacts.mdb`) состоит из одной-единственной таблицы `contacts` (табл. 1.4). Форма и окно программы приведены на рис. 1.32 и рис. 1.33, значения свойств компонентов — в табл. 1.5−1.8. Выполнять настройку компонентов и устанавливать значения их свойств следует в той последовательности, в которой приведены таблицы и значения свойств в таблицах.

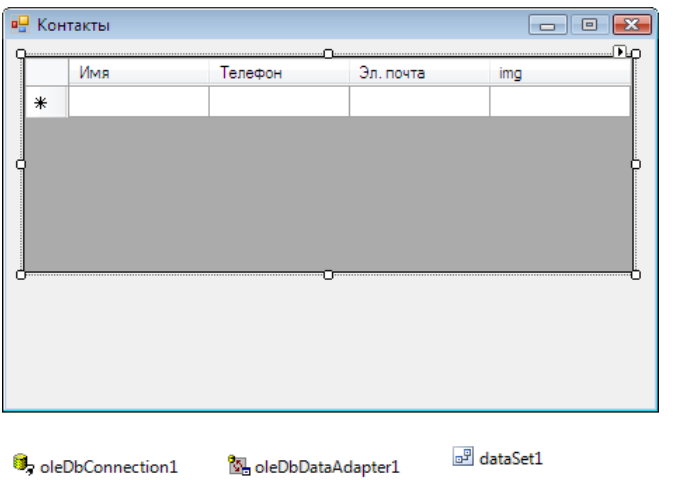


Рис. 1.32. Форма программы **Контакты**

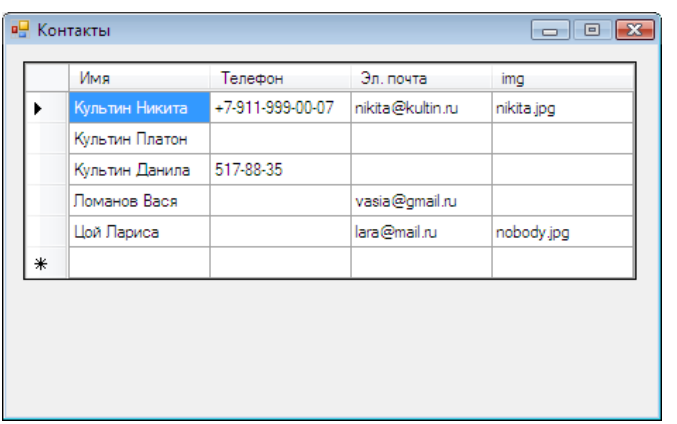


Рис. 1.33. Окно программы **Контакты**

Таблица 1.4. *Поля таблицы* `contacts`

Поле	Тип	Размер	Информация
cid	Целое, авто-увеличение		Идентификатор контакта
name	Текстовый	50	Имя
phone	Текстовый	50	Телефон
email	Текстовый	50	Эл. почта
img	Текстовый	50	Файл иллюстрации

Таблица 1.5. *Значения свойств компонента oleDbConnection1*

Свойство	Значение
Name	oleDbConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\Contacts.mdb

Таблица 1.6. *Значения свойств компонента oleDbDataAdapter1*

Свойство	Значение
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM contacts
InsertCommand.Connection	oleDbConnection1
InsertCommand.CommandText	INSERT INTO contacts (name, phone, email, img) VALUES (?, ?, ?, ?)
InsertCommand.Parameters[0].ParameterName	name
InsertCommand.Parameters[0].SourceColumn	name

Таблица 1.6 *(продолжение)*

Свойство	**Значение**
InsertCommand.Parameters[1].ParameterName	phone
InsertCommand.Parameters[1].SourceColumn	phone
InsertCommand Parameters[2].ParameterName	email
InsertCommand.Parameters[2].SourceColumn	email
InsertCommand.Parameters[3].ParameterName	img
InsertCommand.Parameters[3].SourceColumn	img
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE contacts SET name = ?, phone = ?, email = ?, img = ? WHERE (cid = ?)
UpdateCommand.Parameters[0].ParameterName	name
UpdateCommand.Parameters[0].SourceColumn	name
UpdateCommand.Parameters[1].ParameterName	phone
UpdateCommand.Parameters[1].SourceColumn	phone
UpdateCommand.Parameters[2].ParameterName	email
UpdateCommand.Parameters[2].SourceColumn	email
UpdateCommand.Parameters[3].ParameterName	img
UpdateCommand.Parameters[3].SourceColumn	img
UpdateCommand.Parameters[4].ParameterName	Original_cid
UpdateCommand.Parameters[4].SourceColumn	cid
UpdateCommand.Parameters[4].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1
DeleteCommand.CommandText	DELETE FROM contacts WHERE (cid = ?)

Таблица 1.6 *(окончание)*

Свойство	Значение
DeleteCommand.Parameters[0].ParameterName	cid
DeleteCommand.Parameters[0].SourceColumn	cid
DeleteCommand.Parameters[0].SourceVersion	Original

Таблица 1.7. *Значения свойств компонента* `dataSet1`

Свойство	Значение
Name	dataSet1
Tables[0].TableName	contacts
Tables[0].Columns[0].ColumnName	cid
Tables[0].Columns[0].Caption	cid
Tables[0].Columns[1].ColumnName	name
Tables[0].Columns[1].Caption	name
Tables[0].Columns[2].ColumnName	phpone
Tables[0].Columns[2].Caption	phone
Tables[0].Columns[3].ColumnName	email
Tables[0].Columns[3].Caption	email
Tables[0].Columns[4].ColumnName	img
Tables[0].Columns[4].Caption	img

Таблица 1.8. *Значения свойств компонента* `dataGridView1`

Свойство	Значение
DataSource	dataSet1
DataMember	contacts
AutoSizeColumnsMode	Fill

Таблица 1.8 *(окончание)*

Свойство	**Значение**
BorderStyle	None
Columns[0].DataPropertyName	cid
Columns[0].HeaderText	cid
Columns[0].Visible	False
Columns[1].DataPropertyName	name
Columns[1].HeaderText	Имя
Columns[2].DataPropertyName	email
Columns[2].HeaderText	Эл. почта
Columns[3].DataPropertyName	phone
Columns[3].HeaderText	Телефон
Columns[4].DataPropertyName	img
Columns[4].HeaderText	img

```
// начало работы программы
private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e)
       {
           // загрузить данные
           oleDbDataAdapter1->Fill(dataTable1);
       }
// пользователь выделил строку и нажал <Delete>
private:System::Void dataGridView1_UserDeletingRow
   (System::Object^  sender,
    System::Windows::Forms::DataGridViewRowCancelEventArgs^ e)
       {
           System::Windows::Forms::DialogResult dr =
              MessageBox::Show("Удалить запись?",
                          "Удаление записи",
                          MessageBoxButtons::OKCancel,
                          MessageBoxIcon::Warning,
                          MessageBoxDefaultButton::Button2);
```

```
            if ( dr ==
                  System::Windows::Forms::DialogResult::Yes )
            {
                e->Cancel = true;
            }

// завершение работы программы
private: System::Void Form1_FormClosing
             (System::Object^  sender,
              System::Windows::Forms::FormClosingEventArgs^  e)
     {
         // обновить данные
         oleDbDataAdapter1->Update
               (dataSet1->Tables["contacts"]);
     }
```

Контакты-2

Программа **Контакты-2**, ее форма приведена на рис. 1.34, показывает технологию извлечения нужной информации из базы данных. Компоненты `oleDbConnection`, `oleDbDataAdapter`, `dataSet` и `dataGridView` настраиваются точно так же, как и в предыдущем примере, за исключением настройки свойства `SelectCommand` компонента `oleDbDataAdapter` (табл. 1.9). Окно программы (результат выполнения запроса на поиск информации) приведено на рис. 1.35.

Таблица 1.9. *Значения свойств компонента oleDbDataAdapter1*

Свойство	Значение
SelectCommand.CommandText	SELECT * FROM contacts WHERE (name LIKE ?) ORDER BY name
SelectCommand.Parameters[0].ParameterName	name
SelectCommand.Parameters[0].SourceColumn	name

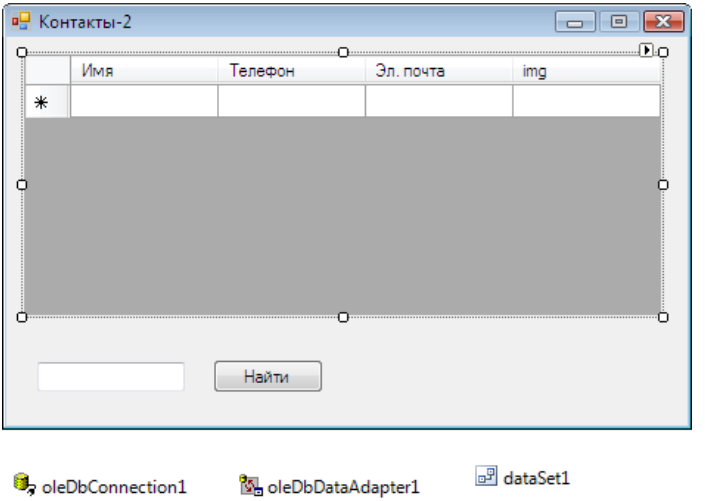


Рис. 1.34. Форма программы **Контакты-2**

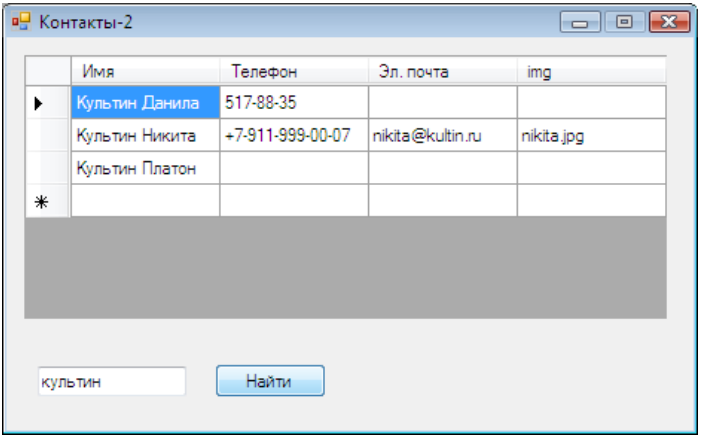


Рис. 1.35. Окно программы **Контакты-2**

\

```
public:
   Form1(void)
   {
       InitializeComponent();
```

```
      //
      //TODO: добавьте код конструктора
      //

     // т. к. у команды SELECT есть параметр,
     // который задает критерий отбора записей,
     // то надо задать его значение
     oleDbDataAdapter1->
         SelectCommand->Parameters[0]->Value = "%%";

    // прочитать данные из БД — выполнить команду SELECT
    oleDbDataAdapter1->Fill(dataTable1);
  }

// щелчок на кнопке Найти
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    dataSet1->Clear(); // удалить старые данные

   // для получения информации из базы данных
   // используется команда SELECT с параметром:
   // SELECT *FROM contacts WHERE (name Like ? )
   // где: ? — параметр.
   // Доступ к параметру можно получить
   // по номеру или по имени
   oleDbDataAdapter1->
            SelectCommand->Parameters["name"]->Value =
                  "%" + textBox1->Text + "%";

   // выполнить команду
   oleDbDataAdapter1->Fill(dataTable1);
}

// пользователь выделил строку и нажал <Delete>
```

```
private: System::Void dataGridView1_
UserDeletingRow(System::Object^  sender,
System::Windows::Forms::DataGridViewRowCancelEventArgs^  e)
     {
         System::Windows::Forms::DialogResult dr =
            MessageBox::Show("Удалить запись?",
                        "Удаление записи",
                        MessageBoxButtons::OKCancel,
                        MessageBoxIcon::Warning,
                        MessageBoxDefaultButton::Button2);

         if (dr == System::Windows::Forms::DialogResult::Yes
)
        {
            e->Cancel = true;
        }
     }

// завершение работы программы
private: System::Void Form1_FormClosing
          (System::Object^  sender,
           System::Windows::Forms::FormClosingEventArgs^ e)
     {
         // обновить данные
         oleDbDataAdapter1->
             Update(dataSet1->Tables["contacts"]);
     }
```

Контакты-3

Программа **Контакты-3** (ее окно приведено на рис. 1.36) демонстрирует отображение данных в режиме формы (данные записи, выбранной в таблице, отображаются в полях редактирования). Помимо текстовой информации в окне программы отображается иллюстрация, связанная с текущей записью (имя файла иллюстрации хранится в поле img). Необходимо обратить внимание

на то, как формируется значение поля img: имя файла иллюстрации, выбранной пользователем во время работы программы в окне просмотра каталогов (окно появляется в результате щелчка кнопкой мыши в области отображения иллюстрации), записывается в поле `img`.

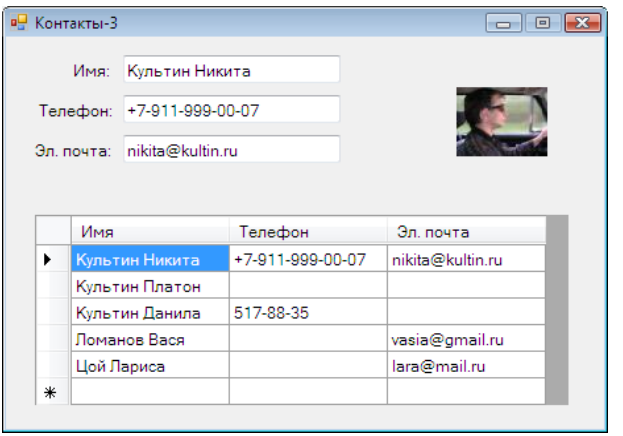


Рис. 1.36. Окно программы **Контакты-3**

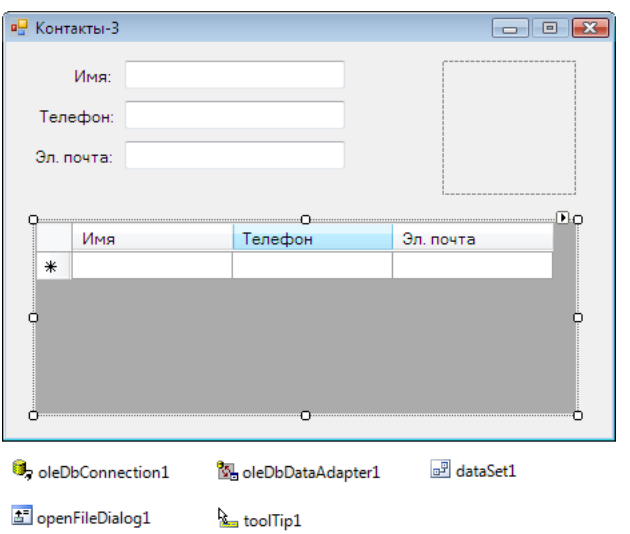


Рис. 1.37. Форма программы **Контакты-3**

Форма программы Контакты-3 приведена на рис. 1.37. В полях `textBox1`, `textBox2` и `textBox3` отображается соответственно содержимое полей `name`, `phone` и `email` текущей записи. Следует обратить внимание на то, что за компонентом `pictureBox` находится компонент `textBox4`, предназначенный для хранения значения поля `img` текущей записи.

Значения свойств компонентов приведены в таблицах 1.10–1.15.

Таблица 1.10. *Значения свойств компонента oleDbConnection1*

Свойство	**Значение**
Name	oleDbConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\Contacts.mdb

Таблица 1.11. *Значения свойств компонента oleDbDataAdapter1*

Свойство	**Значение**
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM contacts
InsertCommand.Connection	oleDbConnection1
InsertCommand.CommandText	INSERT INTO contacts (name, phone, email, img) VALUES (?, ?, ?, ?)
InsertCommand.Parameters[0].ParameterName	name
InsertCommand.Parameters[0].SourceColumn	name
InsertCommand.Parameters[1].ParameterName	phone
InsertCommand.Parameters[1].SourceColumn	phone
InsertCommand Parameters[2].ParameterName	email
InsertCommand.Parameters[2].SourceColumn	email

Таблица 1.11 *(окончание)*

Свойство	Значение
InsertCommand.Parameters[3].ParameterName	img
InsertCommand.Parameters[3].SourceColumn	img
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE contacts SET name = ?, phone = ?, email = ?, img = ? WHERE (cid = ?)
UpdateCommand.Parameters[0].ParameterName	name
UpdateCommand.Parameters[0].SourceColumn	name
UpdateCommand.Parameters[1].ParameterName	phone
UpdateCommand.Parameters[1].SourceColumn	phone
UpdateCommand.Parameters[2].ParameterName	email
UpdateCommand.Parameters[2].SourceColumn	email
UpdateCommand.Parameters[3].ParameterName	img
UpdateCommand.Parameters[3].SourceColumn	img
UpdateCommand.Parameters[4].ParameterName	Original_cid
UpdateCommand.Parameters[4].SourceColumn	cid
UpdateCommand.Parameters[4].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1
DeleteCommand.CommandText	DELETE FROM contacts WHERE (cid = ?)
DeleteCommand.Parameters[0].ParameterName	cid
DeleteCommand.Parameters[0].SourceColumn	cid
DeleteCommand.Parameters[0].SourceVersion	Original

Таблица 1.12. *Значения свойств компонента* `dataSet1`

Свойство	**Значение**
Name	dataSet1
Tables[0].TableName	contacts
Tables[0].Columns[0].ColumnName	cid
Tables[0].Columns[0].Caption	cid
Tables[0].Columns[1].ColumnName	name
Tables[0].Columns[1].Caption	name
Tables[0].Columns[2].ColumnName	phpone
Tables[0].Columns[2].Caption	phone
Tables[0].Columns[3].ColumnName	email
Tables[0].Columns[3].Caption	email
Tables[0].Columns[4].ColumnName	img
Tables[0].Columns[4].Caption	img

Таблица 1.13. *Значения свойств компонента* `dataGridView1`

Свойство	**Значение**
DataSource	dataSet1
DataMember	contacts
AutoSizeColumnsMode	Fill
BorderStyle	None
Columns[0].DataPropertyName	cid
Columns[0].HeaderText	cid
Columns[0].Visible	False
Columns[1].DataPropertyName	name
Columns[1].HeaderText	Имя
Columns[2].DataPropertyName	email

Таблица 1.13 *(окончание)*

Свойство	Значение
Columns[2].HeaderText	Эл. почта
Columns[3].DataPropertyName	phone
Columns[3].HeaderText	Телефон

Таблица 1.14. *Значения свойств компонентов* `textBox`

Компонент	Свойство	Значение
textBox1	dataBindings.Text	dataSet1 — contacts.name
textBox2	dataBindings.Text	dataSet1 — contacts.phone
textBox3	dataBindings.Text	dataSet1 — contacts.email
textBox4	dataBindings.Text	dataSet1 — contacts.img
textBox4	Text	nodody

Таблица 1.15. *Значения свойств компонента picture*`Box`

Свойство	Значение
SizeMode	Zoom
Tool Tip on toolTip1	Щелкните, чтобы задать или изменить иллюстрацию

```
public:
  Form1(void)
  {
      InitializeComponent();
      //
      //TODO: добавьте код конструктора
      //

      dbPath = Application::StartupPath;
      imFolder =  dbPath + "\\Images\\";
      oleDbDataAdapter1->Fill(dataTable1);
```

```
      // связать поля отображения текста с записями dataSet
      textBox1->DataBindings->Add((gcnew
         System::Windows::Forms::Binding(L"Text",
         dataSet1, L"contacts.name", true)));

      textBox2->DataBindings->Add((gcnew
         System::Windows::Forms::Binding(L"Text",
         dataSet1, L"contacts.phone", true)));

      textBox3->DataBindings->Add((gcnew
         System::Windows::Forms::Binding(L"Text",
         dataSet1, L"contacts.email", true)));

      textBox4->DataBindings->Add((gcnew
         System::Windows::Forms::Binding(L"Text",
         dataSet1, L"contacts.img", true)));
  }

private:
    String^ dbPath;   // папка приложения
    String^ imFolder; // папка иллюстраций

// начало работы программы
private:
System::Void Form1_Load(System::Object^  sender,
                        System::EventArgs^  e)
{
/*
    // файл базы данных находится в папке приложения
    dbPath = Application::StartupPath;

    // открыть соединение с БД
    //oleDbConnection1->ConnectionString =
    //       "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
```

```
+
    //       dbPath + "\\Contacts.mdb";

    // папка иллюстраций
    imFolder =  dbPath + "\\Images\\";

    // получить информацию из БД
    oleDbDataAdapter1->Fill(dataTable1);
    //oleDbDataAdapter1->Fill(dataSet1->Tables[0]);
*/
}

// изменилось содержимое поля textBox4 —
// отобразить иллюстрацию, имя файла которой
// находится в этом поле
private: System::Void textBox4_TextChanged
           (System::Object^  sender, System::EventArgs^  e)
{
    String^ imageFile;
    String^ msg; // сообщение об ошибке

    if (textBox4->Text == String::Empty )
    {
          imageFile = imFolder + "nobody.jpg";
    }
    else
          imageFile = imFolder + textBox4->Text;

    // отобразить иллюстрацию
    try
    {
          msg = "";
          pictureBox1->Image =
            System::Drawing::Bitmap::FromFile(imageFile);
    }
```

```
    catch (System::IO::FileNotFoundException^ e)
    {
          // вывести сообщение об ошибке в поле
          // компонента pictureBox1
          msg = "File nof found: " + imageFile;
          pictureBox1->Image = nullptr;
          pictureBox1->Refresh();
    }
}

// щелчок на компоненте pictureBox
private:
System::Void pictureBox1_Click(System::Object^  sender,
                               System::EventArgs^  e)
{
    openFileDialog1->Title = "Выберите иллюстрацию";
    openFileDialog1->InitialDirectory = imFolder;
    openFileDialog1->Filter = "фото|*.jpg|все файлы|*.*";
    openFileDialog1->FileName = "";

    if ( openFileDialog1->ShowDialog() ==
             System::Windows::Forms::DialogResult::OK )
    {
        // пользователь указал файл иллюстрации
        // проверим, находится ли выбранный файл
        // в каталоге imFolder
        bool r =
            openFileDialog1->FileName->ToLower()->Contains(
              openFileDialog1->InitialDirectory->ToLower());
        if ( r == true )
        {
              // копировать не надо т. к. пользователь
              // указал иллюстрацию, которая
              // находится в imFolder
```

```
        textBox4->Text =
                openFileDialog1->SafeFileName;
}
else
{
    // Скопировать файл иллюстрации в папку Images
    // Если в каталоге-приемнике есть файл
    // с таким же именем, что и копируемый,
    // возникает исключение
    try
    {
      // копировать файл
      System::IO::File::Copy(
          openFileDialog1->FileName,
          imFolder + openFileDialog1->SafeFileName);
      textBox4->Text = openFileDialog1->SafeFileName;
    }
    catch (System::Exception^ e)
    {
       System::Windows::Forms::DialogResult dr;
       dr = MessageBox::Show(e->Message +
                " Заменить его?", "",
                MessageBoxButtons::OKCancel,
                MessageBoxIcon::Warning,
                MessageBoxDefaultButton::Button2);
       if (dr ==
           System::Windows::Forms::DialogResult::OK)
        {
          // перезаписать файл
            System::IO::File::Copy(
            openFileDialog1->FileName,
            imFolder + openFileDialog1->SafeFileName,
            true);
          textBox4->Text =
                    openFileDialog1->SafeFileName;
```

```
                }
            }
        }
    }
}

// пользователь выделил строку и нажал <Delete>
private: System::Void dataGridView1_
UserDeletingRow(System::Object^ sender,
   System::Windows::Forms::DataGridViewRowCancelEventArgs^ e)
    {
        System::Windows::Forms::DialogResult dr =
           MessageBox::Show("Удалить запись?",
                       "Удаление записи",
                       MessageBoxButtons::OKCancel,
                       MessageBoxIcon::Warning,
                       MessageBoxDefaultButton::Button2);

        if (dr == System::Windows::Forms::DialogResult::Yes
)
        {
            e->Cancel = true;
        }
    }

// завершение работы программы
private: System::Void Form1_FormClosing
   (System::Object^  sender,
    System::Windows::Forms::FormClosingEventArgs^  e)
   {
       // обновить данные
       oleDbDataAdapter1->Update
              (dataSet1->Tables["contacts"]);
   }
```

Ежедневник

Программа **Ежедневник** (ее окно приведено на рис. 1.38) обеспечивает работу с базой данных Microsoft Access **Задачи**. Демонстрирует использование компонентов `oleDbConnection`, `oleDbDataAdapter`, `dataSet` и `dataGridView`, а также выполнение операций над датами. База данных **Задачи** (todo.mdb) состоит из одной-единственной таблицы `tasks` (табл. 1.16).

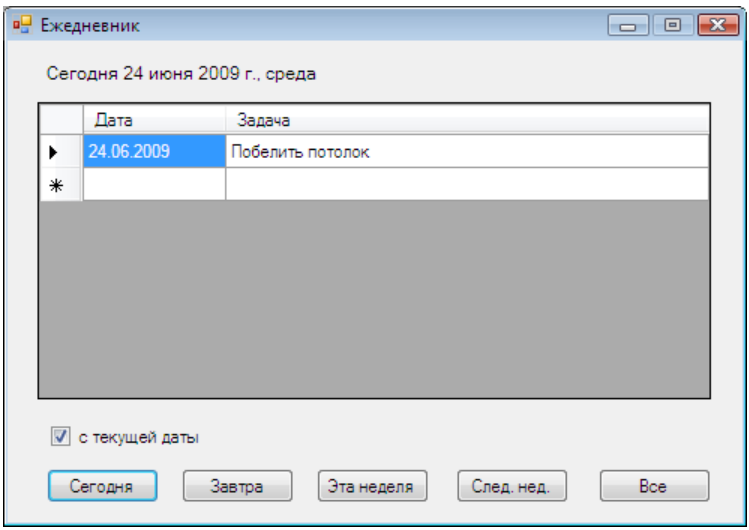


Рис. 1.38. Окно программы **Ежедневник**

Таблица 1.16. *Поля таблицы* `tasks`

Поле	Тип	Размер	Информация
TID	Целое, автоувеличение	–	Идентификатор задачи
aTask	Текстовый	50	Задача
aDate	Дата/Время	–	Дата

Форма программы приведена на рис. 1.39, значения свойств компонентов — в табл. 1.17—1.20. Выполнять настройку компонен-

тов и устанавливать значения их свойств следует в той последовательности, в которой приведены таблицы и значения свойств в таблицах.

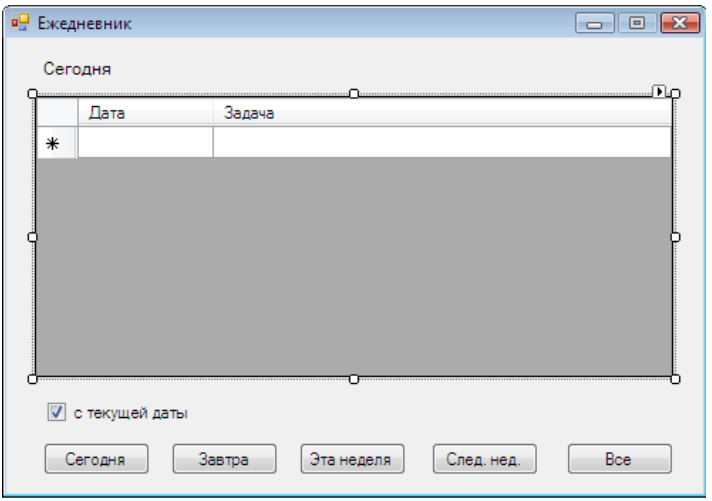


Рис. 1.39. Форма программы **Ежедневник**

Таблица 1.17. *Значения свойств компонента oleDbConnection1*

Свойство	**Значение**
Name	oleDbConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\todo.mdb

Таблица 1.18. *Значения свойств компонента oleDbDataAdapter1*

Свойство	**Значение**
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM tasks WHERE (aData >= ?) AND (aDate <= ?) ORDER BY aDate

Таблица 1.18 *(продолжение)*

Свойство	Значение
SelectCommand.Parameters[0].ParameterName	aDateFrom
SelectCommand.Parameters[0].SourceColumn	aDate
SelectCommand.Parameters[1].ParameterName	aDateTo
SelectCommand.Parameters[1].SourceColumn	aDate
InsertCommand.Connection	oleDbConnection1
InsertCommand.CommandText	INSERT INTO tasks (aDate, aTask) VALUES (?, ?)
InsertCommand.Parameters[0].ParameterName	aDate
InsertCommand.Parameters[0].SourceColumn	aDate
InsertCommand.Parameters[1].ParameterName	aTask
InsertCommand.Parameters[1].SourceColumn	aTask
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE tasks SET aDate = ?, aTask = ? WHERE (tid = ?)
UpdateCommand.Parameters[0].ParameterName	aDate
UpdateCommand.Parameters[0].SourceColumn	aDate
UpdateCommand.Parameters[1].ParameterName	aTask
UpdateCommand.Parameters[1].SourceColumn	aTask
UpdateCommand.Parameters[2].ParameterName	Original_tid
UpdateCommand.Parameters[2].SourceColumn	tid
UpdateCommand.Parameters[2].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1

***Таблица 1.18** (окончание)*

Свойство	Значение
DeleteCommand.CommandText	DELETE FROM tasks WHERE (tid = ?)
DeleteCommand.Parameters[0].ParameterName	tid
DeleteCommand.Parameters[0].SourceColumn	tid
DeleteCommand.Parameters[0].SourceVersion	Original

***Таблица 1.19.** Значения свойств компонента `dataSet1`*

Свойство	Значение
Name	dataSet1
Tables[0].TableName	tasks
Tables[0].Columns[0].ColumnName	tid
Tables[0].Columns[0].Caption	tid
Tables[0].Columns[1].ColumnName	aDate
Tables[0].Columns[1].Caption	aDate
Tables[0].Columns[2].ColumnName	aTask
Tables[0].Columns[2].Caption	aTask

***Таблица 1.20.** Значения свойств компонента `dataGridView1`*

Свойство	Значение
DataSource	dataSet1
DataMember	tasks
AutoSizeColumnsMode	Fill
BorderStyle	None
Columns[0].DataPropertyName	tid
Columns[0].HeaderText	tid

Таблица 1.20 *(окончание)*

Свойство	Значение
Columns[0].Visible	False
Columns[1].DataPropertyName	aDate
Columns[1].HeaderText	Дата
Columns[2].DataPropertyName	aTask
Columns[2].HeaderText	Задача

```
// начало работы программы
private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e)
{
    DateTime d = DateTime::Now;

    oleDbDataAdapter1->SelectCommand->CommandText =
        "SELECT TID, aTask, aDate FROM Tasks WHERE aDate = ?";

    // задать значение параметра команды SELECT
    oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
                DateTime::Now.ToShortDateString();

    // У команды SELECT два параметра. В данном случае
    // используется только один. Но чтобы
    // не возникло исключение, зададим значение
    // второго параметра.
    oleDbDataAdapter1->SelectCommand->Parameters[1]->Value =
                          DateTime::Now.ToShortDateString();

    // выполнить команду
    oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// щелчок на кнопке Сегодня
```

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
{
    int dow = System::Convert::ToInt16(DateTime::Now.DayOfWeek);

    // если пользователь внес изменения,
    // сохраним их
    if (dataSet1->HasChanges())
    {
        oleDbDataAdapter1->Update(dataSet1->Tables["Tasks"]);
    }

    label2->Text = "Сегодня " +
            DateTime::Now.ToLongDateString() + ", " +
            Application::CurrentCulture->
                        DateTimeFormat->DayNames[dow];

    oleDbDataAdapter1->SelectCommand->CommandText =
        "SELECT TID, aTask, aDate FROM Tasks WHERE aDate = ?";

    // задать значение параметра команды SELECT
    oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
        DateTime::Now.ToShortDateString();

    // удалить результат выполнения предыдущей
    // команды SELECT
    dataSet1->Clear();

    // выполнить команду
    oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// щелчок на кнопке Завтра
private: System::Void button2_Click(System::Object^  sender,
                                    System::EventArgs^  e)
```

```
{

    DateTime d = DateTime::Now.AddDays(1);

    // если пользователь внес изменения,
    // сохраним их
    if (dataSet1->HasChanges () )
        oleDbDataAdapter1->Update(dataSet1->Tables["Tasks"]);

    label2->Text = "Завтра (" +  d.ToShortDateString()+")";

    oleDbDataAdapter1->SelectCommand->CommandText =
        "SELECT TID, aTask, aDate FROM Tasks WHERE aDate = ?";

    // задать параметр команды SELECT
    oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
                          d.ToShortDateString();

    dataSet1->Clear();

    // выполнить команду
    oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// на этой неделе
private: System::Void button3_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
      DateTime d1; // первый день (дата) недели
      DateTime d2; // последний день (дата) недели

      int dow; // сегодня (день недели)

      // если пользователь внес изменения,
```

```
      // сохраним их
      if ( dataSet1->HasChanges() )
         oleDbDataAdapter1->
                 Update(dataSet1->Tables["Tasks"]);

      dow =
      System::Convert::ToInt16(DateTime::Now.DayOfWeek);

      d1 = DateTime::Now.AddDays(1- dow);
      d2 = DateTime::Now.AddDays(6);

      if ( checkBox1->Checked )
          // не отображать задачи предыдущих дней
          d1 = DateTime::Now;

      label2->Text =  "На этой неделе (с " +
             d1.ToShortDateString() +
             " по " + d2.ToShortDateString()+")";

      oleDbDataAdapter1->SelectCommand->CommandText =
        "SELECT TID, aTask, aDate FROM Tasks WHERE
        (aDate >= ?) AND (aDate <= ?) ORDER BY aDate";

      oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
                d1.ToShortDateString();
      oleDbDataAdapter1->SelectCommand->Parameters[1]->Value =
              d2.ToShortDateString();

      dataSet1->Clear();
      oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// на следующей неделе
private: System::Void button5_Click(System::Object^  sender,
                                    System::EventArgs^  e)
```

```
{
    DateTime d1; // первый день (дата) след. недели
    DateTime d2; // последний день (дата) след. недели

    int dow; // сегодня (день недели)

    // если пользователь внес изменения,
    // сохраним их
    if ( dataSet1->HasChanges() )
        oleDbDataAdapter1->Update(dataSet1->Tables["Tasks"]);

    dow= System::Convert::ToInt16(DateTime::Now.DayOfWeek);

   // d1= DateTime::Now.AddDays( -(dow -1) + 7 );
    d1 = DateTime::Now.AddDays( 8 — dow );
    d2 = d1.AddDays(6);

    label2->Text =  "На следующей неделе (с " +
                     d1.ToShortDateString() +
                   " по " + d2.ToShortDateString()+")";

    // задать параметры команды SELECT
    //oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
                  d1.ToShortDateString();
    //oleDbDataAdapter1->SelectCommand->Parameters[1]->Value =
                  d2.ToShortDateString();

    oleDbDataAdapter1->SelectCommand->CommandText =
           "SELECT TID, aTask, aDate FROM Tasks WHERE
           (aDate >= ?) AND (aDate <= ?) ORDER BY aDate";

    // задать параметры команды SELECT
    oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
             d1.ToShortDateString();
```

```
    oleDbDataAdapter1->SelectCommand->Parameters[1]->Value =
             d2.ToShortDateString();

    dataSet1->Clear();

    oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// Все, что намечено сделать
private: System::Void button4_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    label2->Text = "";

    // если пользователь внес изменения,
    // сохраним их
    if ( dataSet1->HasChanges() )
        oleDbDataAdapter1->Update(dataSet1->Tables["Tasks"]);

    if ( checkBox1->Checked )
    {
      // не отображать задачи предыдущих дней

      oleDbDataAdapter1->SelectCommand->CommandText =
            "SELECT TID, aTask, aDate FROM Tasks WHERE
             (aDate >= ?)ORDER BY aDate";

      // задать значение параметра команды SELECT
      oleDbDataAdapter1->SelectCommand->Parameters[0]->Value =
                  DateTime::Now.ToShortDateString();

      oleDbDataAdapter1->SelectCommand->CommandText =
             "SELECT TID, aTask, aDate FROM Tasks WHERE
             (aDate >= ?) AND (aDate <= ?) ORDER BY aDate";
    }
    else
```

```
        // все задачи
        oleDbDataAdapter1->SelectCommand->CommandText =
              "SELECT * FROM Tasks ORDER BY aDate";

    dataSet1->Clear();
    oleDbDataAdapter1->Fill(dataSet1->Tables["Tasks"]);
}

// завершение работы программы
private: System::Void Form1_FormClosing(System::Object^
sender, System::Windows::Forms::FormClosingEventArgs^  e)
{
    oleDbDataAdapter1->Update(dataSet1->Tables["Tasks"]);
}
```

SQL Server Compact Edition

Следующая программа демонстрирует работу с базой данных Microsoft SQL Server Compact Edition. Программа позволяет просматривать базу данных Контакты (contacts.sdf), а также вносить в нее изменения (добавлять, редактировать и удалять записи). Форма программы приведена на рис. 1.40.

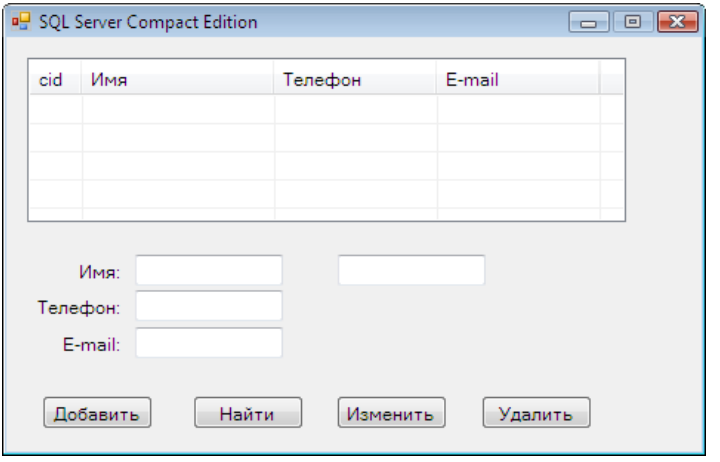


Рис. 1.40. Форма программы работы с базой данных Microsoft SQL Server Compact Edition

Для отображения записей в табличной форме используется компонент `listView` (значения его свойств приведены в табл. 1.21). Компоненты `textBox1`, `textBox2` и `textBox3` используются для ввода и редактирования полей `name`, `phone` и `emal`, а компонент `textBox4` (свойству `ReadOnly` которого следует присвоить значение `True`) — для хранения значения поля `cid`. Нетрудно заметить, что на форме нет компонентов, обеспечивающих доступа к базе данных. Все необходимые для работы с базой данных объекты создаются во время работы программы.

Таблица 1.21. *Значения свойств компонента* `listView`

Свойство	**Значение**
Columns[0].Text	cid
Columns[0].Width	35
Columns[1].Text	Имя
Columns[1].Width	130
Columns[2].Text	Телефон
Columns[2].Width	110
Columns[3].Text	E-mail
Columns[3].Width	110
View	Detals
HideSelection	False
MultiSelect	False
GridLines	True
FullRowSelect	True

```
public:Form1(void)
  {
      InitializeComponent();
      //
      // настройка компонента ListView:
      // увеличим ширину компонента
```

```
        // на ширину полосы прокрутки
        int w = 0;
        for (int i = 0; i < listView1->Columns->Count; i++)
        {
            w += listView1->Columns[i]->Width;
        }

        if (listView1->BorderStyle == BorderStyle::Fixed3D)
            w += 4;

        listView1->Width = w + 17;
        // 17 — ширина полосы прокрутки
    }

// отображает базу данных (таблицу)
private: void ShowDB()
{
    SqlCeEngine^ engine =
       gcnew SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection^ connection =
        gcnew SqlCeConnection(engine->LocalConnectionString);
    connection->Open();
    SqlCeCommand^ command = connection->CreateCommand();
    command->CommandText =
         "SELECT * FROM contacts ORDER BY name";
    SqlCeDataReader^ dataReader = command->ExecuteReader();

    String^ st;  // значение поля БД
    int itemIndex = 0;

    listView1->Items->Clear();

    while (dataReader->Read())
    {
```

```
      for (int i = 0; i < dataReader->FieldCount; i++)
      {
          st = dataReader->GetValue(i)->ToString();
          switch (i)
          {
            case 0:  // поле cid
                listView1->Items->Add(st);
                break;
            case 1:  // поле name
              listView1->Items[itemIndex]->SubItems->Add(st);
              break;
            case 2:  // поле phone
              listView1->Items[itemIndex]->SubItems->Add(st);
              break;
            case 3:  // поле email
              listView1->Items[itemIndex]->SubItems->Add(st);
               break;
          };
        }
        itemIndex++;
    }
    connection->Close();
  }

// начало работы программы — загрузка формы
private: System::Void Form1_Load(System::Object^ sender,
                                 System::EventArgs^ e)
{

    SqlCeEngine^ engine;

    engine =
       gcnew SqlCeEngine("Data Source='contacts.sdf';");

    // проверим, доступен ли файл БД
```

```
    if (!(File::Exists("contacts.sdf")))
    {
        // Создать БД SQL Server Compact Edition.
        // Путь к файлу не указан (см. ранее),
        // поэтому БД будет создана в каталоге приложения.
        // При запуске программы из среды разработки,
        // текущим является каталог Debug или Relese
        engine->CreateDatabase();
        SqlCeConnection^ connection =
        gcnew SqlCeConnection(engine->LocalConnectionString);
        connection->Open();
        SqlCeCommand^ command = connection->CreateCommand();
        command->CommandText =
            "CREATE TABLE contacts (cid int IDENTITY(1,1),
             name nvarchar(50) NOT NULL, phone nvarchar(50),
             email nvarchar(50))";
        command->ExecuteScalar();
        connection->Close();
    }
    else
    {
        ShowDB();
    }
}

// щелчок на кнопке Добавить
private: System::Void button1_Click(System::Object^ sender,
                                    System::EventArgs^  e)
{
    SqlCeConnection^ conn =
       gcnew SqlCeConnection("Data Source ='contacts.sdf'");
    conn->Open();
    SqlCeCommand^ command = conn->CreateCommand();
    command->CommandText =
      "INSERT INTO contacts(name, phone,email) VALUES(?,?,?)";
```

```
    command->Parameters->Add("name", textBox1->Text);
    command->Parameters->Add("phone", textBox2->Text);
    command->Parameters->Add("email", textBox3->Text);
    command->ExecuteScalar();
    conn->Close();

    // очистить поля ввода
    textBox1->Clear();
    textBox2->Clear();
    textBox3->Clear();

    ShowDB();

    // установить курсор в поле textBox1
    textBox1->Focus();
}

// щелчок на кнопке Найти
private: System::Void button2_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
  SqlCeEngine^ engine =
       gcnew SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection^ connection =
       gcnew SqlCeConnection(engine->LocalConnectionString);
    connection->Open();

    SqlCeCommand^ command = connection->CreateCommand();

    command->CommandText =
         "SELECT * FROM contacts WHERE (name LIKE ?)";
    command->Parameters->Add("name",
                           "%" + textBox1->Text + "%");

    SqlCeDataReader^ dataReader = command->ExecuteReader();
```

```
    String^ st;  // значение поля БД
    int itemIndex = 0;

    listView1->Items->Clear();

    while (dataReader->Read())
    {
      for (int i = 0; i < dataReader->FieldCount; i++)
      {
        st = dataReader->GetValue(i)->ToString();
        switch (i)
        {
          case 0:  // поле cid
            listView1->Items->Add(st);
            break;
          case 1:  // поле name
            listView1->Items[itemIndex]->SubItems->Add(st);
            break;
          case 2:  // поле phone
            listView1->Items[itemIndex]->SubItems->Add(st);
            break;
          case 3:  // поле email
            listView1->Items[itemIndex]->SubItems->Add(st);
            break;
        };
      }
      itemIndex++;
    }
    connection->Close();
}

// В поле компонента listView пользователь
// выбрал другую строку
private: System::Void
listView1_ItemSelectionChanged(System::Object^  sender,
```

```
System::Windows::Forms::ListViewItemSelectionChangedEventArgs^ e)
{
    /*
         При выборе строки событие ItemSelectionChanged
         возникает два раза: первый раз, когда выделенная в
         данный момент строка теряет фокус, второй — когда
         строка, в которой сделан щелчок, получает фокус.
         Нас интересует строка, которая получает фокус.
     */
     if (e->IsSelected)
     {
        // строка выбрана, т. е. она получила фокус
        textBox4->Text = listView1->Items[e->ItemIndex]->Text;

        // дублируем содержимое полей текущей
        // записи в поля textBox
        for (int i = 1;
          i < listView1->Items[e->ItemIndex]->SubItems->Count;
                       i++)
        {
           switch (i)
           {
            case 1:
                textBox1->Text =
           listView1->Items[e->ItemIndex]->SubItems[i]->Text;
                break;
            case 2:
                textBox2->Text =
            listView1->Items[e->ItemIndex]->SubItems[i]->Text;
                break;
            case 3:
                textBox3->Text =
            listView1->Items[e->ItemIndex]->SubItems[i]->Text;
                break;
```

```
            }
         }
      }
}

// щелчок на кнопке Удалить
private: System::Void button4_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
  if (listView1->SelectedItems->Count != 0)
  {
      SqlCeEngine^ engine =
          gcnew SqlCeEngine("Data Source='contacts.sdf';");

      SqlCeConnection^ connection =
         gcnew SqlCeConnection(engine->LocalConnectionString);

      connection->Open();

      SqlCeCommand^ command = connection->CreateCommand();

      command->CommandText =
                "DELETE FROM contacts WHERE (cid = ?)";
      command->Parameters->Add("cid", textBox4->Text);

      command->ExecuteScalar(); // выполнить команду

      ShowDB();

      textBox1->Clear();
      textBox2->Clear();
      textBox3->Clear();
      textBox4->Clear();
  }
}
```

```
// щелчок на кнопке Изменить
private: System::Void button3_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    if (listView1->SelectedItems->Count != 0)
    {
        SqlCeEngine^ engine =
          gcnew SqlCeEngine("Data Source='contacts.sdf';");

        SqlCeConnection^ connection = gcnew
          SqlCeConnection(engine->LocalConnectionString);

        connection->Open();

        SqlCeCommand^ command = connection->CreateCommand();

        command->CommandText =
           "UPDATE contacts " +
           "SET name = ?, phone =?, email=? " +
           "WHERE cid = ?";

        command->Parameters->Add("name", textBox1->Text);
        command->Parameters->Add("phone", textBox2->Text);
        command->Parameters->Add("email", textBox3->Text);
        command->Parameters->Add("cid", textBox4->Text);

        command->ExecuteScalar(); // выполнить команду

        ShowDB();

        textBox1->Clear();
        textBox2->Clear();
        textBox3->Clear();
        textBox4->Clear();
    }
```

```
}

// Обработка события TextChanged компонентов
// textBox1 — textBox3. Одна функция обрабатывает
// событие трех компонентов.
private: System::Void textBox_TextChanged
      (System::Object^  sender, System::EventArgs^ e)
{
    // кнопка Добавить становится доступной,
    //  если информация введена в поле Имя и в
    //  какое-либо из полей, Телефон или E-mail
    if ((textBox1->TextLength > 0) &&
        ((textBox2->TextLength > 0) ||
              (textBox3->TextLength > 0)))
        button1->Enabled = true;
    else
    {
        button1->Enabled = false;
    }
}
```

Игры и другие полезные программы

Парные картинки

Игра **Парные картинки** (рис. 1.41) развивает память. Правила игры следующие: игровое поле разделено на клетки, за каждой из которых скрыта картинка. Картинки парные, т. е. на поле есть две клетки с одинаковыми картинками. Задача игрока — найти все пары картинок. В начале игры все клетки закрыты. Щелчок мыши в клетке открывает картинку. Щелчок в другой клетке открывает вторую картинку. Если картинки в открытых клетках одинаковые, считается, что пара найдена, и клетки исчезают. Если картинки разные, то они остаются открытыми. Следующий

щелчок открывает клетку, в которой он сделан, и закрывает открытые, причем даже в том случае, если картинка в ней такая же, что и в одной из открытых. Игра заканчивается тогда, когда будут найдены все пары картинок. Картинки загружаются из файла (рис. 1.42).

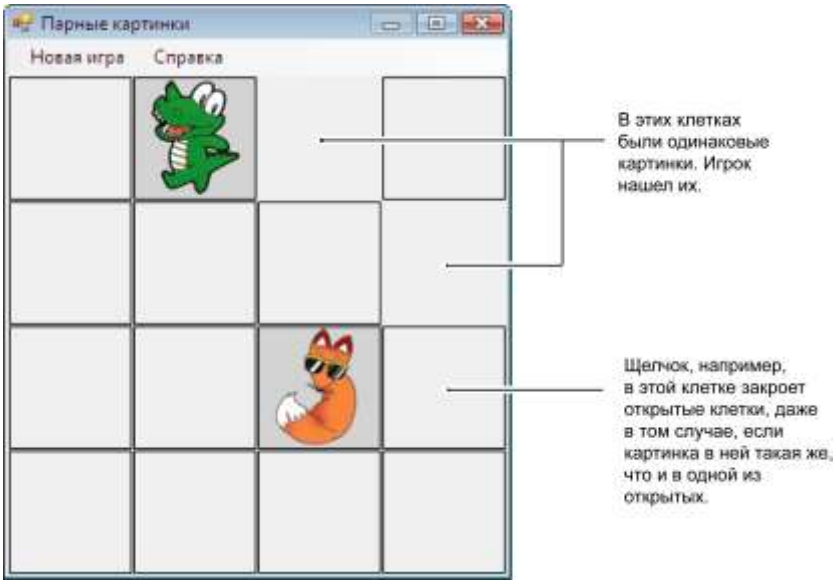


Рис. 1.41. Окно игры **Парные картинки**

Рис. 1.42. Файл картинок

В верхней части главного окна находится строка меню (компонент `MenuStrip`). В результате выбора в меню **Справка** команды **О программе** на экране появляется соответствующее окно (рис. 1.43).

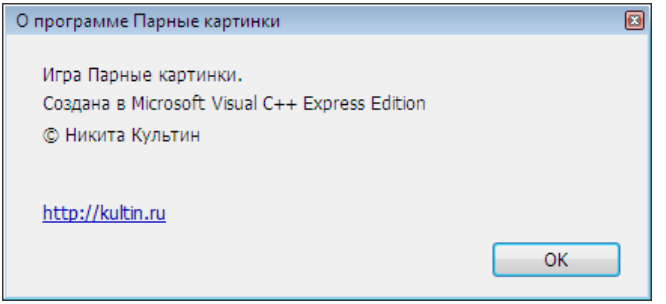


Рис. 1.43. Окно **О программе**

Щелчок на WEB-ссылке в этом окне запускает браузер и открывает страницу **http://kultin.ru**. Непосредственный запуск браузера осуществляет процедура обработки события `Click` на ссылке, которая представляет собой компонент `LinkLabel`. Следует обратить внимание: свойству `DialogResult` кнопки **OK** необходимо присвоить значение `OK`.

```
public:
Form1(void)
{
      InitializeComponent();
      //
  try
  {
      // загружаем файл с картинками
      pics = gcnew Bitmap(Application::StartupPath +
                          "\\pictures.bmp");
  }
  catch (Exception^ )
  {
      MessageBox::Show("Файл 'pictures.bmp' не найден.\n",
                       "Парные картинки",
                       MessageBoxButtons::OK,
                       MessageBoxIcon::Error);
```

```
    this->новаяИграToolStripMenuItem->Enabled = false;

  return;
}

// Зададим функцию обработки события Click на форме.
// Если функцию обработки задать обычным образом,
// то в процедуре drawCell надо
// контролировать, загружена ли картинка.
// Так проще.
this->MouseClick += gcnew
    System::Windows::Forms::MouseEventHandler(this,
      &Form1::Form1_MouseClick);

// определяем размер картинки и устанавливаем
// размер клеток игрового поля
cw = (int)(pics->Width / np);
ch = pics->Height;

// установить размер клиентской области формы
// в соответствии с размером картинок и их количеством
// (см. определения констант cw и ch)
this->ClientSize =
    System::Drawing::Size(nw * (cw + 2) + 1,
              nh * (ch + 2) + 1 + menuStrip1->Height);

// рабочая графическая поверхность
g = this->CreateGraphics();

// создать объект timer1
timer1 = gcnew Timer();

timer1->Tick +=
    gcnew System::EventHandler(this, &Form1::timer1_Tick);
```

```
  timer1->Interval = 200;

  newGame();
}

private:

    static int nw = 4; // кол-во клеток по горизонтали
    static int nh = 4; // кол-во клеток по вертикали

    static int np = (nw * nh) / 2;  // кол-во пар картинок

    // рабочая графическая поверхность
    System::Drawing::Graphics^ g;

    // картинки (загружаются из файла)
    Bitmap^ pics;

    // размер (ширина и высота) клетки (картинки)
    int cw, ch;

    // игровое поле:
    static array<int,2>^ field  = gcnew array<int,2>(4,4);
    // field[i,j] == 1 ... k — клетка закрыта
    //                        (k-номер картинки);
    // field[i,j] == 101 ... (100+k) — клетка открыта
    //                       (игрок видит картинку);
    // field[i,j] == 201 ... (200+k) — в клетке картинка,
    //                        для которой найдена пара

    // кол-во открытых (найденных) пар картинок
    static int nOpened = 0;

    // количество открытых в данный момент клеток
```

```
    static int cOpened = 0;

    // координаты 1-й открытой клетки
    static array<int>^ open1 = gcnew array<int>(2);

    // координаты 2-й открытой клетки
    static array<int>^ open2 = gcnew array<int>(2);

    // таймер
    System::Windows::Forms::Timer^ timer1;



// Рисует клетку:
// — картинку, если клетка открыта;
// — границу, если клетка закрыта;
// — фон, если в клетке картинка, для которой найдена пара
void drawCell(int i, int j)
{
    int x,y;  // координаты левого верхнего угла клетки

    // между картинками
    // оставляем промежутки в 1 пиксел
    x = i * (cw + 2);
    y = j * (ch + 2) + menuStrip1->Height;

    if (field[i,j] > 200)
        // для этой клетки найдена пара,
        // картинку отображать не надо
        g->FillRectangle(SystemBrushes::Control,
            x, y, cw + 2, ch + 2);

    if ((field[i, j] > 100) && (field[i, j] < 200))
    {
        // клетка открыта — отобразить картинку:
        // скопировать фрагмент с одной графической
```

```
        // поверхности на другую
        Rectangle r1 = Rectangle(x + 1, y + 1, cw, ch);
        Rectangle r2 = Rectangle
                    ((field[i, j] — 101) * cw, 0, cw, ch);

        g->DrawImage (pics, r1,r2,
          GraphicsUnit::Pixel);

        g->DrawRectangle(Pens::Black,
            x + 1, y + 1, cw, ch);
    }

    if ((field[i, j] > 0) && (field[i, j] < 100))
    {
        // клетка закрыта, рисуем контур
        g->FillRectangle(SystemBrushes::Control,
            x + 1, y + 1, cw, ch);
        g->DrawRectangle(Pens::Black,
            x + 1, y + 1, cw, ch);
    }
}

// нарисовать поле
void drawField()
{
    for (int i = 0; i < nw; i++)
        for (int j = 0; j < nh; j++)
            drawCell(i, j);
}

// новая игра
void newGame()
{
      // Раскидаем пары картинок по игровому полю:
      // запишем в массив field случайные числа
```

```
    // от 1 до k, где к — количество картинок.
    // Каждое число должно быть записано
    // в массив field два раза.

    Random^ rnd; // генератор случайных чисел
    int rndN;   // случайное число

    rnd = gcnew Random();

    array<int>^ buf = gcnew array<int>(np);
    // np — кол-во картинок;
    // в buf[i] записываем, сколько i чисел
    // (идентификаторов картинок) записали в массив field

    for (int i = 0; i < nw; i++)
        for (int j = 0; j < nh; j++)
        {
            do
            {
                rndN = rnd->Next(np) + 1;
            } while (buf[rndN - 1] == 2);

            field[i, j] = rndN;
            buf[rndN - 1]++;
        }

    nOpened = 0;
    cOpened = 0;

    this->drawField();
}

// обработка события таймера
private: System::Void timer1_Tick(System::Object^  sender,
                                  System::EventArgs^  e)
```

```
{
      // нарисовать (перерисовать) клетки
      drawCell(open1[0], open1[1]);
      drawCell(open2[0], open2[1]);

      // остановить таймер
      timer1->Enabled = false;

      if (nOpened == np)
      {
          MessageBox::Show
                  ("Вы справились с поставленной задачей!",
                   "Собери картинку");
      }
}

// обработка события Click на форме
private: System::Void Form1_MouseClick(System::Object^
sender, System::Windows::Forms::MouseEventArgs^  e)
{
        int i,j; // индексы элемента массива field
                 // клетки, в которой
                 // сделан щелчок

        i = e->X / (cw +3);
        j = (e->Y — menuStrip1->Height) / (ch+3);

        // если таймер работает, это значит, что в данный
        // момент открыты две клетки, в которых находятся
        // одинаковые картинки, но они еще не "стерты".
        // Если щелчок сделан в одной из этих картинок,
        // то ничего делать не надо.
        if ((timer1->Enabled) && (field[i,j] > 200))
       {
            return;
```

```
}

// щелчок на месте одной из двух уже найденных
// парных картинок
if (field[i, j] > 200) return;

// открытых клеток нет
if (cOpened == 0)
{
    cOpened++;

    // записываем координаты 1-й открытой клетки
    open1[0] = i; open1[1] = j;

    // клетка помечается как открытая
    field[i, j] += 100;

    // нарисовать (перерисовать) клетку
    drawCell(i, j);

    return;
}

// открыта одна клетка, надо открыть вторую
if (cOpened == 1)
{
    // записываем координаты 2-й открытой клетки
    open2[0] = i; open2[1] = j;

    // если открыта одна клетка, и щелчок сделан
    // в той же клетке, ничего не происходит
    if ((open1[0] == open2[0]) &&
                        (open1[1] == open2[1]))
        return;
    else
    {
```

```
        // теперь открыты две клетки
        cOpened++;

        // клетка помечается как открытая
        field[i, j] += 100;

        // отрисовать клетку
        drawCell(i, j);

        // открыты 2 одинаковые картинки
        if (field[open1[0], open1[1]] ==
            field[open2[0], open2[1]])
        {
          nOpened++;

          // пометим клетки как найденные
          field[open1[0], open1[1]] += 100;
          field[open2[0], open2[1]] += 100;

          cOpened = 0;

         // Запускаем таймер. Процедура обработки
         // сигнала от таймера "сотрет" открытые клетки
         // (с одинаковыми картинками)
         timer1->Enabled = true;
        }
      }
      return;
}

// открыты 2 клетки, но в них разные картинки,
// закроем их и откроем ту клетку, в которой
// сделан щелчок
if (cOpened == 2)
{
    // закрываем открытые клетки
```

```
            field[open1[0], open1[1]] -= 100;
            field[open2[0], open2[1]] -= 100;

            drawCell(open1[0], open1[1]);
            drawCell(open2[0], open2[1]);

            // записываем в open1 номер текущей клетки
            open1[0] = i; open1[1] = j;

            cOpened = 1;        // счетчик открытых клеток

            // открыть клетку, в которой сделан щелчок
            field[i, j] += 100;
            drawCell(i, j);
        }
}

// Выбор в меню команды "Новая игра"
private: System::Void новаяИграToolStripMenuItem_
Click(System::Object^  sender,
                                  System::EventArgs^  e)
{
        this->newGame();
}

// обработка события Paint
private: System::Void Form1_Paint(System::Object^  sender,
            System::Windows::Forms::PaintEventArgs^  e)
{
            this->drawField();
}

// выбор в меню Справка команды О программе
private: System::Void оПрограммеToolStripMenuItem_
Click(System::Object^  sender,
                                   System::EventArgs^  e)
{
```

```
    // в начало программы надо добавить ссылку на файл about.h
    // (директиву #include "about.h"),
    // в котором определен класс About — форма О программе

    About^ frm = gcnew About(); // создать окно О программе

    frm->ShowDialog(); // отобразить окно О программе
}

// щелчок в окне О программе на WEB-ссылке
private: System::Void linkLabel1_LinkClicked(
    System::Object^  sender,
    System::Windows::Forms::LinkLabelLinkClickedEventArgs^ e)
{
        String^ webRef = linkLabel1->Text;
        System::Diagnostics::Process::Start(webRef);

}
```

Собери картинку

Программа **Puzzle** (рис. 1.44) — графический вариант некогда популярной игры "15". Ее цель — расположить фрагменты картинки (фишки) в правильном порядке.

Рис. 1.44. Окно игры **Puzzle**

```
public:
Form1(void)
{
    InitializeComponent();
    //

    try
    {
        // загружаем файл картинки
        pics = gcnew Bitmap(Application::StartupPath +
                            "\\puzzle.bmp");
    }
    catch (Exception^ )
    {
        MessageBox::Show("Файл 'puzzle.bmp' не найден.\n",
            "Puzzle game",
            MessageBoxButtons::OK,
            MessageBoxIcon::Error);

        return;
    }

    // определяем высоту и ширину клетки (фишки)
    cw = (int)(pics->Width / nw);
    ch = (int)(pics->Height / nh);

    // установить размер клиентской области формы
    this->ClientSize =
        System::Drawing::Size(cw * nw + 1,
                    ch * nh + 1 + menuStrip1->Height);

    // рабочая графическая поверхность
    g = this->CreateGraphics();

    newGame();
```

```
}

private:
    // 4x4 — размер игрового поля
    static int nw = 4, nh = 4;

    // графическая поверхность
    System::Drawing::Graphics^ g;

    // картинка
    Bitmap^ pics;

    // размер (ширина и высота) клетки
    int cw, ch;

    // игровое поле — массив 4x4
    // хранит номера фрагментов картинки
    static array<int,2>^ field = gcnew array<int,2>(4,4);

    // координаты пустой клетки
    int ex, ey;

    // отображать номера фишек
    static Boolean showNumbers = false;


// новая игра
void newGame()
{
    // разместить фишки в правильном порядке
    for (int j = 0; j < nh; j++)
        for (int i = 0; i < nw; i++)
            field[i, j] = j * nw + i + 1;

    // последняя фишка — пустая
    field[nw — 1, nh — 1] = 0;
```

```
    ex = nw — 1; ey = nh — 1;

    this->mix();         // перемешиваем фишки
    this->drawField();   // рисуем игровое поле
}

// перемешиваем фишки
void mix()
{
    int d; // положение (относительно пустой) перемещаемой
           // фишки: 0 — слева; 1 — справа;
           //         2 — сверху; 3 — снизу.

    int x, y; // координаты перемещаемой фишки

    // генератор случайных чисел
    Random^ rnd = gcnew Random();


    // сделаем 160 сдвигов
    for (int i = 0; i < 160; i++)
    {
        x = ex;
        y = ey;

        // выберем фишку, которую будем
        // "двигать" в пустую клетку
        d = rnd->Next(4);
        switch (d)
        {
                case 0: if (x > 0) x--; break;
                case 1: if (x < nw — 1) x++; break;
                case 2: if (y > 0) y--; break;
                case 3: if (y < nh — 1) y++; break;
```

```
        }

        // здесь определили фишку, которую
        // нужно переместить в пустую клетку
        field[ex, ey] = field[x, y];
        field[x, y] = 0;

        // запоминаем координаты пустой клетки
        ex = x;
        ey = y;
    }
}

// рисуем поле
void drawField()
{
    // рисуем клетки
    for (int i = 0; i < nw; i++)
        for (int j = 0; j < nh; j++)
          {
            if (field[i, j] != 0)
            {
                // рисуем фрагмент картинки:
                // копируем фрагмент с одной графической
                // поверхности на другую

                // куда
                Rectangle r1 =
                       Rectangle(i * cw,
                              j * ch + menuStrip1->Height,
                              cw, ch);
                // откуда
                Rectangle r2 = Rectangle(
                        ((field[i, j] — 1) % nw) * cw,
                        ((field[i, j] — 1) / nw) * ch,
```

```
                    cw, ch);

                // копируем
                g->DrawImage(pics,r1,r2,GraphicsUnit::Pixel);
            }
            else
                // выводим пустую фишку
                g->FillRectangle(SystemBrushes::Control,
                    i * cw, j * ch + menuStrip1->Height,
                     cw, ch);

            // рисуем границу
            g->DrawRectangle(Pens::Black,
                i * cw, j * ch + menuStrip1->Height,
                cw, ch);

            // номер фишки
            if ((showNumbers) && field[i, j] != 0)
                g->DrawString(
                    Convert::ToString(field[i, j]),
                    gcnew System::Drawing::Font("Tahoma",
                                12, FontStyle::Regular),
                    Brushes::Black,
                    (float)i * cw + 5,
                    (float)j * ch + 5 + menuStrip1-
>Height);
        }
}

// проверяем, расположены ли фишки в правильном порядке
private: Boolean finish()
{
    // координаты клетки
    int i = 0;
    int j = 0;
```

```
    int c;        // число в клетке

    // фишки расположены правильно, если
    // числа в них образуют матрицу:
    //   1  2  3  4
    //   5  6  7  8
    //   9 10 11 12
    //  13 14 15

    for (c = 1; c < nw * nh; c++)
    {
        if (field[i, j] != c) return false;

            // к следующей клетке
            if (i < nw - 1) i++;
            else { i = 0; j++; }
    }
    return true;
}

// перемещаем фишку, на которой сделан щелчок,
// в соседнюю пустую клетку:
// (cx, cy) - клетка, в которой сделан щелчок,
// (ex, ey) - пустая клетка
private: void move(int cx, int cy)
{
    // проверим, возможен ли обмен
    if (!(((Math::Abs(cx - ex) == 1) && (cy - ey == 0)) ||
        ((Math::Abs(cy - ey) == 1) && (cx - ex == 0))))
            return;

     // обмен: переместим фишку из (x, y) в (ex, ey)
     field[ex, ey] = field[cx, cy];
     field[cx, cy] = 0;
```

```
    ex = cx; ey = cy;

    // перерисовать поле
    this->drawField();

    // проверить: возможно, фишки выстроены
    // в правильном порядке
    if (this->finish())
    {
        field[nw − 1, nh − 1] = nh * nw;
        this->drawField();

        // Игра закончена. Сыграть еще раз?
        // No  − завершить работу программы,
        // Yes − новая игра
        if (MessageBox::Show(
            "Вы справились с поставленной задачей!\n" +
            "Еще раз?", "Puzzle game",
            MessageBoxButtons::YesNo,
            MessageBoxIcon::Question)
            ==
            System::Windows::Forms::DialogResult::No )
          this->Close();
        else this->newGame();
      }
}

// обработка события paint
private: System::Void Form1_Paint(System::Object^  sender,
         System::Windows::Forms::PaintEventArgs^  e)
{
    drawField(); // нарисовать поле
}

// щелчок кнопкой мыши на игровом поле
```

```
private: System::Void Form1_MouseClick(System::Object^
sender, System::Windows::Forms::MouseEventArgs^  e)
{
        // преобразуем координаты мыши в координаты клетки
        move( e->X/cw, (e->Y-menuStrip1->Height)/ch);
}

// команда Новая игра
private: System::Void MenuItem1_Click(System::Object^  sender,
                                      System::EventArgs^  e)
{
            this->newGame();
}
```

Сапер

Игра **Сапер**, окно которой приведено на рис. 1.45 — аналог игры, хорошо знакомой всем пользователям Windows. Программа демонстрирует работу с графикой, массивами, вывод справочной информации. Главная форма программы и форма **О программе** приведены на рис. 1.46 и 1.47 соответственно.

Рис. 1.45. Окно игры **Сапер**

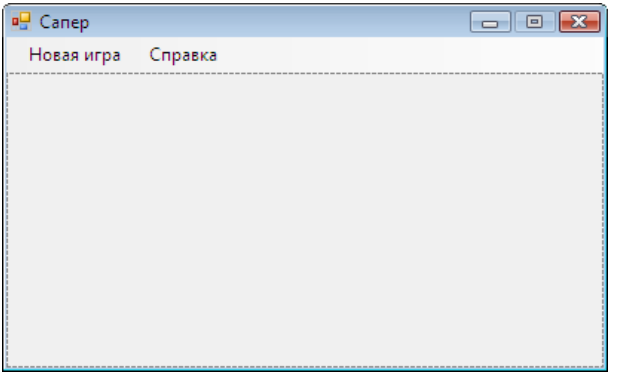


Рис. 1.46. Главная форма

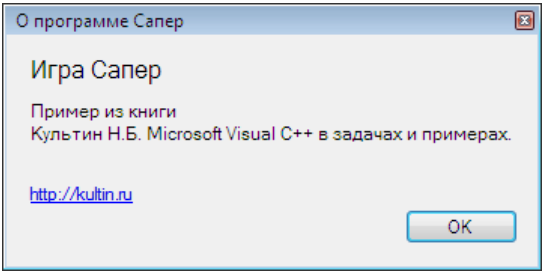


Рис. 1.47. Форма **О программе**

```
public:
  Form1(void)
  {
      InitializeComponent();

      // В неотображаемые эл-ты массива, соответствующие
      // клеткам границы игрового поля, запишем число -3.
      // Это значение используется процедурой open()
      // для завершения рекурсивного процесса открытия
      // соседних пустых клеток
      for(int row = 0; row <= MR+1; row++)
      {
```

```
            Pole[row,0] = -3;
            Pole[row,MC+1] = -3;
        }

        for(int col = 0; col <= MC+1; col++)
        {
            Pole[0,col] = -3;
            Pole[MR+1,col] = -3;
        }

        // устанавливаем размер формы в соответствии
        // с размером игрового поля
        this->ClientSize = System::Drawing::Size(W*MC + 1,
                        H*MR +  menuStrip1->Height +  1);

        newGame(); // новая игра

        // графическая поверхность
        g = panel1->CreateGraphics();

    }

private:

    static int
      MR = 10, // кол-во клеток по вертикали
      MC = 10, // кол-во клеток по горизонтали
      NM = 10, // кол-во мин
      W = 40,  // ширина клетки
      H = 40;  // высота клетки

    // игровое (минное) поле
    static array<int,2>^ Pole =
              gcnew array<int,2>(MR + 2,MC + 2);
```

```
  // значение элемента массива:
  // 0..8 — количество мин в соседних клетках
  // 9 — в клетке мина
  // 100..109 — клетка открыта
  // 200..209 — в клетку поставлен флаг

  int nMin;  // кол-во найденных мин
  int nFlag; // кол-во поставленных флагов

  // статус игры
  int status;
  // 0 — начало игры,
  // 1 — игра,
  // 2 — результат

  // графическая поверхность формы
  System::Drawing::Graphics^ g;

// новая игра
void newGame()
{
    int row, col;     // индексы клетки
    int n = 0;        // количество поставленных мин
    int k;            // кол-во мин в соседних клетках

    // очистить поле
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            Pole[row,col] = 0;

    // генератор случайных чисел
    Random^ rnd = gcnew Random();

    // расставим мины
    do
```

```
{
    row = rnd->Next(MR) + 1;
    col = rnd->Next(MC) + 1;

    if (Pole[row,col] != 9)
    {
        Pole[row,col] = 9;
        n++;
    }
}
while (n != NM);

// для каждой клетки вычислим кол-во
// мин в соседних клетках
for(row = 1; row <= MR; row++)
    for(col = 1; col <= MC; col++)
        if (Pole[row,col] != 9)
        {
            k = 0;

            if (Pole[row-1,col-1] == 9) k++;
            if (Pole[row-1,col]   == 9) k++;
            if (Pole[row-1,col+1] == 9) k++;
            if (Pole[row,col-1]   == 9) k++;
            if (Pole[row,col+1]   == 9) k++;
            if (Pole[row+1,col-1] == 9) k++;
            if (Pole[row+1,col]   == 9) k++;
            if (Pole[row+1,col+1] == 9) k++;

            Pole[row,col] = k;
        }

status = 0;       // начало игры
nMin   = 0;       // нет обнаруженных мин
```

```
    nFlag  = 0;       // нет поставленных флагов
}

// рисует поле
void showPole(Graphics^ g, int status)
{
    for(int row = 1; row <= MR; row++)
        for(int col = 1; col <= MC; col++)
            this->kletka(g, row, col, status);
}

// рисует клетку
void kletka(Graphics^ g,
    int row, int col, int status)
{
    int x,y;// координаты левого верхнего угла клетки

    x = (col — 1) * W + 1;
    y = (row-1)* H + 1;

    // не открытые клетки — серые
    if (Pole[row,col] < 100)
        g->FillRectangle(SystemBrushes::ControlLight,
            x-1, y-1, W, H);

    // открытые или помеченные клетки
    if (Pole[row,col] >= 100) {

        // открываем клетку, открытые — белые
        if (Pole[row,col] != 109)
            g->FillRectangle(Brushes::White,
                x-1, y-1, W, H);
        else
            // на этой мине подорвались!
            g->FillRectangle(Brushes::Red,
```

```
                x-1, y-1, W, H);

        // если в соседних клетках есть мины,
        // указываем их количество
        if ((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
            g->DrawString((Pole[row,col]-100).ToString(),
            gcnew System::Drawing::Font("Tahoma", 10,
                System::Drawing::FontStyle::Regular),
                Brushes::Blue, (float)x+3, (float)y+2);
    }

    // в клетке поставлен флаг
    if (Pole[row,col] >= 200)
        this->flag(g, x, y);

    // рисуем границу клетки
    g->DrawRectangle(Pens::Black,
        x-1, y-1, W, H);

    // если игра завершена (status = 2),
    // показываем мины
    if ((status == 2) && ((Pole[row,col] % 10) == 9))
        this->mina(g, x, y);
}

// открывает текущую и все соседние с ней клетки,
// в которых нет мин
void open(int row, int col)
{
    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    if (Pole[row,col] == 0)
    {
```

```
        Pole[row,col] = 100;

        // отобразить содержимое клетки
        this->kletka(g, row, col, status);

        // открыть примыкающие клетки
        // слева, справа, сверху, снизу
        this->open(row, col-1);
        this->open(row-1, col);
        this->open(row, col+1);
        this->open(row+1, col);

        //примыкающие диагонально
        this->open(row-1,col-1);
        this->open(row-1,col+1);
        this->open(row+1,col-1);
        this->open(row+1,col+1);
    }
    else
        if ((Pole[row,col] < 100) &&
             (Pole[row,col] != -3))
        {
            Pole[row,col] += 100;
            // отобразить содержимое клетки
            this->kletka(g, row, col, status);
        }
}

// рисует мину
void mina(Graphics^ g, int x, int y)
{
    // корпус
      g->FillRectangle(Brushes::Green,
        x+16, y+26, 8, 4);
```

```
      g->FillRectangle(Brushes::Green,
        x+8, y+30, 24, 4);
      g->DrawPie(Pens::Black,
        x+6, y+28, 28, 16, 0, -180);
      g->FillPie(Brushes::Green,
        x+6, y+28, 28, 16, 0, -180);

    // полоса на корпусе
      g->DrawLine(Pens::Black,
        x+12, y+32, x+28, y+32);

    // вертикальный "ус"
      g->DrawLine(Pens::Black,
        x+20, y+22, x+20, y+26);

    // боковые "усы"
      g->DrawLine(Pens::Black,
        x+8, y+30, x+6, y+28);
      g->DrawLine(Pens::Black,
        x+32, y+30, x+34, y+28);
}

// рисует флаг
void flag(Graphics^ g, int x, int y)
{
    array<Point>^ p = gcnew array<Point>(3);
    array<Point>^ m = gcnew array<Point>(5);

    // флажок
    p[0].X = x+4;   p[0].Y = y+4;
    p[1].X = x+30;  p[1].Y = y+12;
    p[2].X = x+4;   p[2].Y = y+20;
    g->FillPolygon(Brushes::Red, p);

    // древко
```

```
    g->DrawLine(Pens::Black,
        x+4, y+4, x+4, y+35);

    // буква М на флажке
    m[0].X = x+8;   m[0].Y = y+14;
    m[1].X = x+8;   m[1].Y = y+8;
    m[2].X = x+10;  m[2].Y = y+10;
    m[3].X = x+12;  m[3].Y = y+8;
    m[4].X = x+12;  m[4].Y = y+14;
    g->DrawLines(Pens::White, m);
}


// обработка события Paint
private: System::Void panel1_Paint(System::Object^ sender,
           System::Windows::Forms::PaintEventArgs^ e)
{
     showPole(g, status);
}

// щелчок в клетке игрового поля
private: System::Void panel1_MouseClick(System::Object^
       sender, System::Windows::Forms::MouseEventArgs^ e)
{
    if (status == 2)
        // игра завершена
        return;

    if (status == 0)
        // первый щелчок
        status = 1;

    // преобразуем координаты мыши в индексы
    // клетки поля, в которой был сделан щелчок;
    // (e.X, e.Y) — координаты точки формы,
```

```
// в которой была нажата кнопка мыши
int row,col;

row = e->Y/H + 1;
col = e->X/W + 1;

// координаты области вывода
int x = (col-1)* W + 1,
    y = (row-1)* H + 1;

// щелчок левой кнопки мыши
if ( e->Button ==
      System::Windows::Forms::MouseButtons::Left)
{
    // открыта клетка, в которой есть мина
    if (Pole[row,col] == 9)
    {
        Pole[row,col] += 100;

        // игра закончена
        status = 2;

        // перерисовать форму
        this->panel1->Invalidate();
    }
    else
        if (Pole[row,col] < 9)
            // открыть клетку
            this->open(row,col);
}

// щелчок правой кнопки мыши
if ( e->Button ==
      System::Windows::Forms::MouseButtons::Right)
{
```

```
    // в клетке не было флага, ставим его
    if (Pole[row,col] <= 9) {
        nFlag += 1;

        if (Pole[row,col] == 9)
            nMin += 1;

        Pole[row,col] += 200;

        if ((nMin == NM) && (nFlag == NM))
        {
            // игра закончена
            status = 2;

            // перерисовываем все игровое поле
            this->Invalidate();
        }
        else
            // перерисовываем только клетку
            this->kletka(g, row, col, status);
    }
    else
        // в клетке был поставлен флаг,
        // повторный щелчок правой кнопки мыши
        // убирает его и закрывает клетку
        if (Pole[row,col] >= 200)
        {
            nFlag -= 1;
            Pole[row,col] -= 200;

            // перерисовываем клетку
            this->kletka(g, row, col, status);
        }
}
```

```
}

// команда Новая игра
private: System::Void
новаяИграToolStripMenuItem_Click(System::Object^  sender,
                                 System::EventArgs^  e)
{
     newGame();
     showPole(g, status);
}

// Выбор в меню Справка команды Правила
private: System::Void
правилаToolStripMenuItem_Click(System::Object^  sender,
                               System::EventArgs^  e)
{
    Help::ShowHelp(this,
                   helpProvider1->HelpNamespace,
                   "saper_02.htm");
}

// Выбор в меню Справка команды О программе
private: System::Void оПрограммеСаперToolStripMenuItem_Click
      (System::Object^ sender, System::EventArgs^  e)
{
    AboutForm^ frm = gcnew AboutForm();
    frm->ShowDialog();
}
```

Будильник

Программа **Будильник** в установленный пользователем момент времени выводит на экран окно с сообщением. Особенность программы в том, что ее значок отображается не в рабочей, а в системной области панели задач. При наведении указателя мыши

на значок появляется окно подсказки (рис. 1.48), а в результате щелчка на значке правой кнопкой мыши — меню (рис. 1.49), в котором можно выбрать команду управления программой.

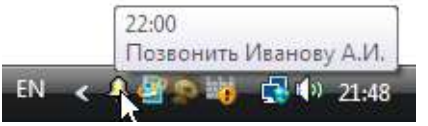


Рис. 1.48. Значок программы **Будильник** отображается в системной части панели задач

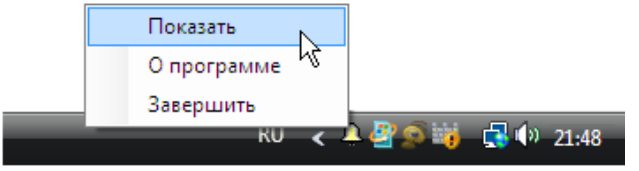


Рис. 1.49. В результате щелчка правой кнопкой мыши на значке программы появляется меню команд

Главная форма программы **Будильник** приведена на рис. 1.50, форма окна сообщения — на рис. 1.51.

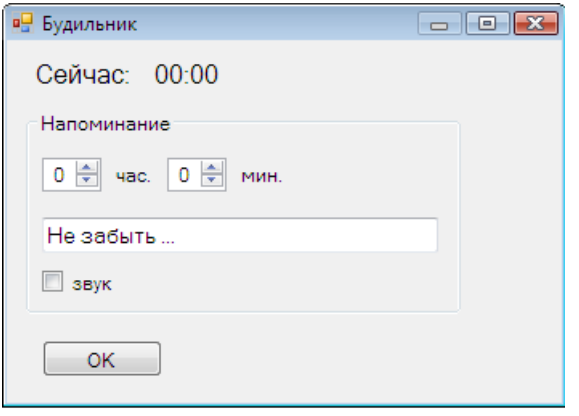


Рис. 1.50. Главная форма программы **Будильник**

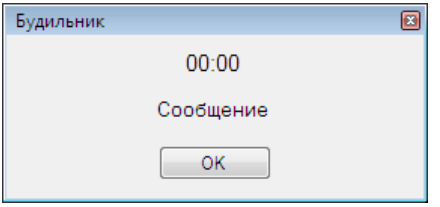


Рис. 1.51. Форма окна сообщения

```
// функция PlaySound, обеспечивающая воспроизведение
// wav-файлов, находится в библиотеке winmm.dll
// подключим эту библиотеку
// Приведенное далее объявление надо поместить в начало
// модуля, за директивами using
[System::Runtime::InteropServices::DllImport("winmm.dll")]
    extern bool PlaySound(String^ lpszName, int hModule,
                          int dwFlags);

public:Form1(void)
{
    InitializeComponent();
    //
    // настройка компонентов numericUpDown
    numericUpDown1->Maximum = 23;
    numericUpDown1->Minimum = 0;

    numericUpDown2->Maximum = 59;
    numericUpDown2->Minimum = 0;

    numericUpDown1->Value = DateTime::Now.Hour;
    numericUpDown2->Value = DateTime::Now.Minute;

    notifyIcon1->Visible = false;

    isSet = false;
```

```
    // настройка и запуск таймера
    timer1->Interval = 1000;
    timer1->Enabled = true;

    label4->Text = DateTime::Now.ToLongTimeString();
}

private:
    DateTime alarm; // время сигнала (отображения сообщения)
    bool isSet;     // true — будильник установлен

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^ sender,
                                    System::EventArgs^  e)
{
      // установить время сигнала
      alarm = DateTime(
          DateTime::Now.Year,
          DateTime::Now.Month,
          DateTime::Now.Day,
          Convert::ToInt16(numericUpDown1->Value),
          Convert::ToInt16(numericUpDown2->Value),
          0, 0);

      // если установленное время будильника меньше
      // текущего, нужно увеличить дату срабатывания
      // на единицу (+1 день)
      if (DateTime::Compare(DateTime::Now, alarm) > 0)
          alarm = alarm.AddDays(1);

      // текст подсказки
      // будет отображаться при
      // позиционировании указателя мыши
      // на значке программы
```

```
        notifyIcon1->Text =
                        alarm.ToShortTimeString() + "\n" +
                        textBox1->Text;

        isSet = true;

        this->Hide();
        notifyIcon1->Visible = true;
}

// сигнал от таймера
private: System::Void timer1_Tick(System::Object^  sender,
                                  System::EventArgs^  e)
{
    label4->Text = DateTime::Now.ToLongTimeString();

    // будильник установлен
    if ( isSet)
    {
        // время срабатывания будильника
        if (DateTime::Compare(DateTime::Now, alarm) > 0)
        {
            isSet = false;

            if ( checkBox1->Checked )
            {
                PlaySound(Application::StartupPath +
                "\\ring.wav", 0, 1);
            }

            Form2^ frm;

            // см. конструктор формы Form2
            frm = gcnew
```

```
                    Form2(DateTime::Now.ToShortTimeString(),
                        this->textBox1->Text);
            frm->ShowDialog();

            this->Show();
        }
    }
}

// команда Показать
private:
System::Void toolStripMenuItem1_Click(System::Object^  sender,
                                      System::EventArgs^  e)
{
        isSet = false;
        this->Show();
        notifyIcon1->Visible = false;
}

// команда О программе
private:
System::Void toolStripMenuItem2_Click(System::Object^  sender,
                                      System::EventArgs^  e)
{

}

// команда Завершить
private:
System::Void toolStripMenuItem3_Click(System::Object^  sender,
                                      System::EventArgs^  e)
{
     this->Close();
}
```

```
// щелчок на значке
private:
System::Void notifyIcon1_MouseClick(System::Object^ sender,
                    System::Windows::Forms::MouseEventArgs^ e)
{
    isSet = false;
    this->Show();
    notifyIcon1->Visible = false;
}
```

Экзаменатор

Программа **Экзаменатор** позволяет автоматизировать процесс тестирования. В окне программы (рис. 1.52) отображается тест — последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов.

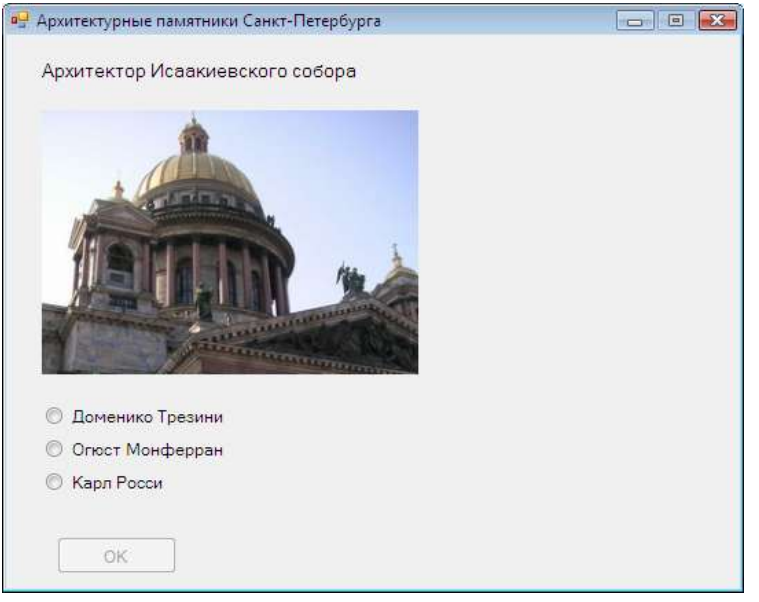


Рис. 1.52. Окно программы **Экзаменатор**. Испытуемый должен выбрать правильный ответ

В рассматриваемой программе вопросы загружаются из XML-файла (пример файла теста приведен на рис. 1.53). Имя файла теста передается программе при запуске — указывается в командной строке. Форма программы **Экзаменатор** приведена на рис. 1.54.

```
<?xml version="1.0" encoding="Windows-1251"?>
<test>
 <head>Архитектурные памятники Санкт-Петербурга</head>
 <description>Сейчас Вам будут предложены вопросы по архитектуре Петербурга. Вы
должны из предложенных нескольких вариантов ответа выбрать
правильный.</description>
 <qw>
  <q text="Архитектор Исаакиевского собора" src ="is.jpg" >
   <a right="no">Доменико Трезини</a>
   <a right="yes">Огюст Монферран</a>
   <a right="no">Карл Росси</a>
  </q>
  <q text="На фотографии" src ="marks.jpg">
   <a right="yes">Зимний дворец(Эрмитаж)</a>
   <a right="no">Мариинский дворец</a>
   <a right="no">Строгоновский дворец</a>
   </q>
 </qw>
 <levels>
  <level score="2" text = "На все вопросы вы ответили правильно. Оценка —
ОТЛИЧНО."/>
  <level score="1" text = "На некоторые вопросы вы ответили неправильно. Оценка —
ХОРОШО."/>
  <level score="0" text = "Вы плохо подготовились к испытанию. Оценка — ПЛОХО!"/>
 </levels>
</test>
```

Рис. 1.53. Файл теста "Архитектурные памятники Санкт-Петербурга"

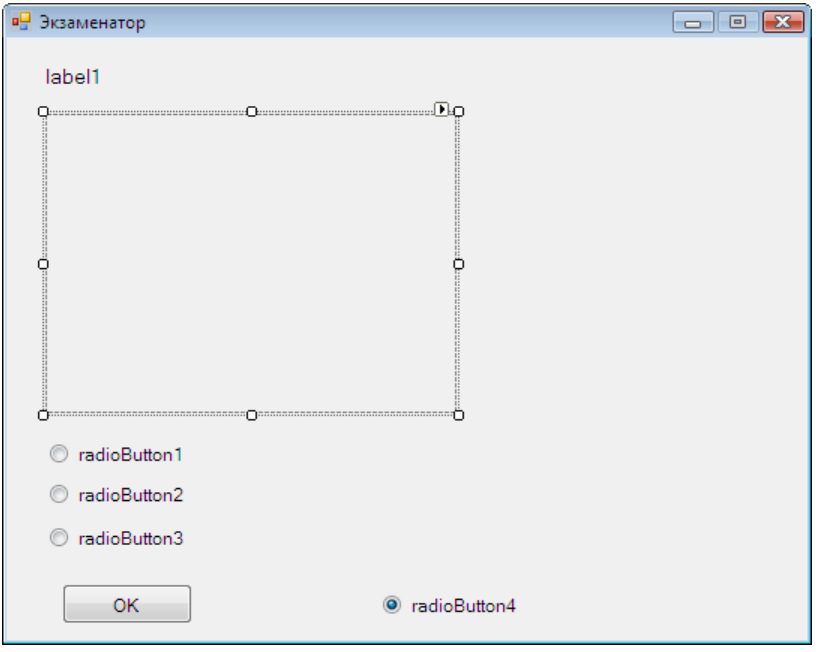


Рис. 1.54. Форма программы **Экзаменатор**

```
// функция main находится в главном модуле приложения
int main(array<System::String ^> ^args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);


    String^ fpath;
    String^ fname;

    // проверим, указано ли имя файла теста
    if (args->Length > 0)
    {
        // указано только имя файла теста
```

```
    if (args[0]->IndexOf(":") == -1)
    {
        fpath = Application::StartupPath + "\\";
        fname = args[0];
    }

    else
    {
       // указано полное имя файла теста
       fpath =
          args[0]->Substring(0,
          args[0]->LastIndexOf("\\")+1);
       fname =
           args[0]->Substring(
             args[0]->LastIndexOf("\\")+1);
    }

  // проверим, существует ли указанный файл
  if ( System::IO::FileInfo( fpath->ToString() +
          fname->ToString() ).Exists )
  {
    // Создать главное окно и запустить программу
    Application::Run(gcnew Form1(fpath,fname));
    return 0;
}
else
{
    MessageBox::Show(
        "Неправильно указано имя файла теста\n"+
        "(нет папки или файла в указанной папке).\n" +
        fpath->ToString() + fname->ToString(),
        "Экзаменатор",
        MessageBoxButtons::OK,
        MessageBoxIcon::Stop);
  return -1;
```

```
      }
    }
    else
    {
        MessageBox::Show(
          "Файл теста необходимо указать " +
          "в команде запуска программы.\n" +
          "Например: 'exam economics.xml' " +
          "или 'exam c:\\spb.xml'.",
          "Экзаменатор",
          MessageBoxButtons::OK,
          MessageBoxIcon::Stop);
        return -1;
    }
}

// конструктор главной формы
public:
   Form1(String^ fpath, String^ fname)
   {
      InitializeComponent();

      picpath = fpath; // путь к файлам иллюстраций
      radioButton1->Visible = false;
      radioButton2->Visible = false;
      radioButton3->Visible = false;

    try
    {
          // прочитать xml-файл
          xmlReader =
              gcnew System::Xml::XmlTextReader(fpath + fname);
          xmlReader->Read();

          mode = 0;
          n    = 0;
```

```
            // загрузить и показать заголовок теста
            this->showHead();

            // загрузить и показать описание теста
            this->showDescription();
        }
        catch(Exception^ e)
        {
           MessageBox::Show( e->Message,
                             "Экзаменатор",
                             MessageBoxButtons::OK,
                             MessageBoxIcon::Error);
            mode = 2;
         }
}

private:

    // fpath — путь к файлу теста
    // fname — файл теста

    // XmlReader обеспечивает чтение данных xml-файла
    System::Xml::XmlReader^ xmlReader;

    String^ qw;      // вопрос

    // варианты ответа — массив строк (указателей на строки)
    static array<String^>^ answ = gcnew array<String^>(3);

    String^ picpath; // путь к файлу иллюстрации
    String^ pic;     // файл иллюстрации

    int right; // правильный ответ (номер)
    int otv;   // выбранный ответ (номер)
```

```
    int n;      // количество правильных ответов
    int nv;     // общее количество вопросов
    int mode;   // состояние программы:
                // 0 — начало работы;
                // 1 — тестирование;
                // 2 — завершение работы

// выводит название (заголовок) теста
void showHead()
{
  // ищем узел <head>
  do
  {
    xmlReader->Read();
  }
  while(xmlReader->Name != "head");

  // считываем заголовок
  xmlReader->Read();

  // вывести название теста в заголовок окна
  this->Text = xmlReader->Value;

  // выходим из узла <head>
  xmlReader->Read();
}

// выводит описание теста
void showDescription()
{
      // ищем узел <description>
      do
      {
          xmlReader->Read();
```

```
    }
    while(xmlReader->Name != "description");

    // считываем описание теста
    xmlReader->Read();

    // выводим описание теста
    label1->Text = xmlReader->Value;

    // выходим из узла <description>
    xmlReader->Read();

    // ищем узел вопросов <qw>
    do
        xmlReader->Read();
    while(xmlReader->Name != "qw");

    // входим в узел "qw"
    xmlReader->Read();
}

// читает вопрос из файла теста
bool getQw()
{
    // считываем тэг <q>
    xmlReader->Read();

    if (xmlReader->Name == "q")
    {
        // здесь прочитан тэг <q>,
        // атрибут text которого содержит вопрос, а
        // атрибут src — имя файла иллюстрации.

        // извлекаем значение атрибутов:
        qw  = xmlReader->GetAttribute("text");
```

```
pic = xmlReader->GetAttribute("src");
if (!pic->Equals(String::Empty))
    pic = picpath + pic;

// входим внутрь узла
xmlReader->Read();
int i = 0;

// считываем данные узла вопроса <q>
while (xmlReader->Name != "q")
{
    xmlReader->Read();

    // варианты ответа
    if (xmlReader->Name == "a")
    {
        // если есть атрибут right, то это
        // правильный ответ
        if (xmlReader->GetAttribute("right") ==
                "yes")
            right = i;

        // считываем вариант ответа
        xmlReader->Read();
        if (i < 3) answ[i] = xmlReader->Value;

        // выходим из узла <a>
        xmlReader->Read();

          i++;
     }
}

// выходим из узла вопроса <q>
xmlReader->Read();
```

```
        return true;
    }
    // если считанный тэг не <q>
    else
          return false;
}

// выводит вопрос и варианты ответа
void showQw()
{
    // выводим вопрос
    label1->Text = qw;

    // иллюстрация
    if (pic->Length != 0)
    {
        try
        {
            pictureBox1->Image = gcnew Bitmap(pic);
            pictureBox1->Visible = true;
            radioButton1->Top = pictureBox1->Bottom + 16;
        }
        catch (Exception^ )
        {
              if (pictureBox1->Visible)
                  pictureBox1->Visible = false;

              label1->Text +=
                  "\n\n\nОшибка доступа к файлу " + pic + ".";

              radioButton1->Top = label1->Bottom + 8;
        }
      }
      else
```

```
    {
        if (pictureBox1->Visible)
            pictureBox1->Visible = false;

        radioButton1->Top = label1->Bottom + 16;
    }

    // показать варианты ответа
    radioButton1->Text = answ[0];
    radioButton2->Top  = radioButton1->Top + 24;;
    radioButton2->Text = answ[1];
    radioButton3->Top  = radioButton2->Top + 24;;
    radioButton3->Text = answ[2];

    radioButton4->Checked = true;
    button1->Enabled = false;
}

// Щелчок на кнопке выбора ответа.
// Функция обрабатывает событие Click
// компонентов radioButton1 — radioButton3
private: System::Void radioButton1_Click(System::Object^
                        sender, System::EventArgs^  e)
{
    if ((RadioButton^)sender == radioButton1) otv = 0;
    if ((RadioButton^)sender == radioButton2) otv = 1;
    if ((RadioButton^)sender == radioButton3) otv = 2;

     button1->Enabled = true;
}

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^  sender,
                                    System::EventArgs^  e)
```

```
{
    switch (mode)
    {
      case 0:
          // Щелчок на кнопке OK
          // когда в окне отображается
          // информация о тесте и задание
          radioButton1->Visible = true;
          radioButton2->Visible = true;
          radioButton3->Visible = true;

          getQw();
          showQw();

          mode = 1;

          button1->Enabled = false;
          radioButton4->Checked = true;
          break;

      case 1:
          // щелчок, подтверждающий
          // выбор ответа
          nv++;

          // правильный ли ответ выбран
          if (otv == right) n++;

          if (getQw())
              showQw();
          else {
              // больше вопросов нет
              radioButton1->Visible = false;
              radioButton2->Visible = false;
```

```
            radioButton3->Visible = false;

            pictureBox1->Visible  = false;

            // обработка и вывод результата
            showLevel();

            // следующий щелчок на кнопке Ok
            // должен закрыть окно программы
            mode = 2;
        }
        break;

    case 2:
        // завершение работы программы
        this->Close(); // закрыть окно
        break;
    }
}

// выводит оценку
void showLevel()
{
    // ищем узел <levels>
    do
        xmlReader->Read();
    while (xmlReader->Name != "levels");

    // входим внутрь узла
    xmlReader->Read();

    // читаем данные узла
    while (xmlReader->Name != "levels")
    {
```

```
        xmlReader->Read();

        if (xmlReader->Name == "level")
              // n — кол-во правильных ответов.
              // Для достижения уровня кол-во
              // правильных ответов должно быть
              // равно или превышать значение,
              // заданное атрибутом score

            if (n >= Convert::ToInt32(
                  xmlReader->GetAttribute("score")))
              break;
    }

    // выводим оценку
    label1->Text =
         "Тестирование завершено.\n" +
         "Всего вопросов: " + nv.ToString() + "\n" +
         "Правильных ответов: " + n.ToString() + "\n\n" +
         xmlReader->GetAttribute("text");
}
```

Задачи для самостоятельного решения

1. Напишите программу расчета сопротивления электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно. Если сопротивление цепи меньше 1000 Ом, то результат отображать в Ом, иначе — в кОм.
2. Напишите программу, при помощи которой можно рассчитать доход по вкладу. Исходные данные для расчета — сумма и срок вклада (1, 3, 6 или 12 месяцев). Величина процентной ставки определяется сроком вклада.

3. Напишите программу, при помощи которой можно подсчитать цену бензина на автозаправочной станции. Исходные данные для расчета — число литров, марка бензина (92, 95 или 98) и наличие дисконтной карты.
4. Напишите программу, при помощи которой можно пересчитать цену из долларов в рубли или из рублей в доллары.
5. Напишите программу, при помощи которой можно определить стоимость аренды автомобиля. Исходные данные для расчета — время аренды (целое число часов) и тип автомобиля (такси, микроавтобус или автобус).
6. Напишите программу, при помощи которой можно рассчитать платеж КАСКО. Исходные данные для расчета — цена автомобиля и его марка (Ford, BMW, Toyota, Renault, Volkswagen).
7. Напишите программу, при помощи которой можно подсчитать расходы на доставку мебели. Исходные данные для расчета — номер этажа и информация о наличии и типе лифта (грузовой или обычный).
8. Напишите программу, при помощи которой можно посчитать цену стеклопакета. Исходные данные для расчета — габаритные размеры (ширина и высота в миллиметрах), тип механизма открывания (поворотный или поворотно-откидной) и наличие дополнительных опций (фиксатор, микропроветривание, москитная сетка).
9. Напишите программу, при помощи которой можно посчитать стоимость заказа печати фотографий. Исходные данные для расчета — размер (9×12, 12×15 или 18×24) и количество фотографий. Если количество фотографий больше 20, то предоставляется скидка 10%.
10. Напишите программу, при помощи которой можно пересчитать расстояния из миль в километры. Исходные данные для расчета — расстояние в километрах и тип мили (морская или сухопутная).

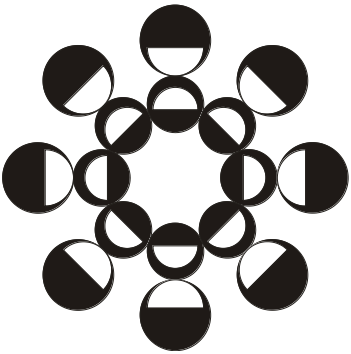
11. Напишите программу, при помощи которой можно пересчитать температуру из градусов Цельсия в градусы Фаренгейта или Кельвина.
12. Напишите программу, при помощи которой можно рассчитать цену аренды такси. Исходные данные для расчета — расстояние поездки и регион (в черте города, в радиусе до 70 км, в радиусе более 70 км).
13. Напишите программу, при помощи которой можно определить затраты на грузоперевозки. Исходные данные для расчета — расстояние и информация о предоставлении грузчиков (с грузчиками, без грузчиков).
14. Напишите программу, при помощи которой можно вычислить стоимость тиражирования материалов в типографии. Исходные данные для расчета — количество копий, формат (A5, A4, A2, A1, A0) и условие выполнения заказа (в присутствии заказчика или на следующий день).
15. Напишите программу, в окне которой отображается график изменения температуры воздуха за месяц. Предполагается, что возможны как положительные, так и отрицательные значения температуры.
16. Напишите программу, в окне которой отображается столбчатая диаграмма изменения температуры воздуха за месяц. Положительные температуры следует отображать красными столбиками, отрицательные — синими.
17. Напишите программу, в окне которой отображается график изменения цены бензина (92, 95, 98) за последние шесть месяцев.
18. Напишите программу, в окне которой отображается график изменения цены бензина одной марки (например, 95) у разных операторов (Лукойл, Shell, Neste) за последние шесть месяцев.
19. Напишите программу, в окне которой отображается в виде столбчатой диаграммы результат социологического опроса (5 вопро-

сов). Вопрос отобразить в заголовке диаграммы. Под каждым столбиком печатать ответ.

20. Напишите программу, в окне которой отображается в виде "горизонтальной" столбчатой диаграммы результат социологического опроса (5 вопросов).
21. Напишите программу, в окне которой отображается столбчатая диаграмма — результат сдачи экзамена (процент "5", "4", "3" и "2"). Исходные данные — количество пятерок, четверок, троек и двоек.
22. Напишите программу, в окне которой отображается в виде графика число посещений сайта. Исходные данные — ежедневные значения счетчика посещений (нарастающим итогом).
23. Напишите программу, в окне которой отображается в виде столбчатой диаграммы информация о продажах, например, книги. Исходные данные — информация об остатках на складе (на конец месяца).
24. Напишите программу, в окне которой в виде круговой диаграммы отображается информация о расходах среднестатистической семьи на питание, транспорт, одежду, образование.
25. Напишите программу, в окне которой в виде полигона отображается информация об изменении стоимости акций какой-либо компании (например, ОАО "Газпром") за последний месяц.
26. Напишите программу, в окне которой отображается информация о средней цене бензина (например, 95) в разных городах страны, например, в Санкт-Петербурге, Москве, Ростове, Новосибирске и Хабаровске.
27. Усовершенствуйте программу "Калькулятор" так, чтобы можно было выполнять операции умножения и деления.
28. Внесите изменения в программу "Сапер", чтобы изображения клеток (пустая клетка; флажок; мина; мина, помеченная флажком) загружались из файла.

29. Усовершенствуйте программу "Экзаменатор" так, чтобы результат тестирования сохранялся в файле. В начале работы программа должна запрашивать имя испытуемого, а в конце — записывать результат в файл.

30. Напишите программу, обеспечивающую работу с базой данных Microsoft Access "Расходы". В базе данных должны фиксироваться сумма, дата и на что потрачены деньги (по категориям, например, еда, транспорт, образование, развлечения, прочее). Программа должна обеспечивать фильтрацию данных по содержимому поля "Категория", а также выполнять статистическую обработку — выводить сумму затрат за период.



ЧАСТЬ II

Краткий справочник

Форма

Свойства формы (объекта `WinForm`) приведены в табл. 2.1.

Таблица 2.1. *Свойства* `формы`

Свойство	Описание
`Name`	Имя формы
`Text`	Текст в заголовке
`Size`	Размер формы. Уточняющее свойство `Width` определяет ширину, свойство `Heigh` — высоту
`StartPosition`	Положение формы в момент первого появления на экране. Форма может находиться в центре экрана (`CenterScreen`), в центре родительского окна (`CenterParent`). Если значение свойства равно `Manual`, то положение формы определяется значением свойства `Location`
`Location`	Положение формы на экране. Расстояние от верхней границы формы до верхней границы экрана задает уточняющее свойство `Y`, расстояние от левой границы формы до левой границы экрана — уточняющее свойство `X`
`FormBorderStyle`	Тип формы (границы). Форма может представлять собой обычное окно (`Sizable`), окно фиксированного размера (`FixedSingle`, `Fixed3D`), диалог (`FixedDialog`) или окно без кнопок **Свернуть** и **Развернуть** (`SizeableToolWindow`, `FixedToolWindow`). Если свойству присвоить значение `None`, у окна не будет заголовка и границы

Таблица 2.1 *(окончание)*

Свойство	Описание
`ControlBox`	Управляет отображением системного меню и кнопок управления окном. Если значение свойства равно `false`, то в заголовке окна кнопка системного меню, а также кнопки **Свернуть**, **Развернуть**, **Закрыть** не отображаются
`MaximazeBox`	Кнопка **Развернуть**. Если значение свойства равно `false`, то находящаяся в заголовке окна кнопка **Развернуть** недоступна
`MinimazeBox`	Кнопка **Свернуть**. Если значение свойства равно `false`, то находящаяся в заголовке окна кнопка **Свернуть** недоступна
`Icon`	Значок в заголовке окна
`Font`	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение значения свойства приводит к автоматическому изменению значения свойства `Font` всех компонентов формы (при условии, что значение свойства компонента не было задано явно)
`ForeColor`	Цвет, наследуемый компонентами формы и используемый ими для отображения текста. Изменение значения свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства `Font` компонента не было задано явно)
`BackColor`	Цвет фона. Можно задать явно (выбрать на вкладке **Custom** или **Web**) или указать элемент цветовой схемы (выбрать на вкладке **System**)
`Opacity`	Степень прозрачности формы. Форма может быть непрозрачной (100%) или прозрачной. Если значение свойства меньше 100%, то сквозь форму видна поверхность, на которой она отображается

Компоненты

В этом разделе кратко описаны основные (базовые) компоненты. Подробное описание этих и других компонентов можно найти в справочной системе.

Button

Компонент `Button` представляет собой командную кнопку. Свойства компонента приведены в табл. 2.2.

***Таблица 2.2.** Свойства компонента* `Button`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст на кнопке
`TextAlign`	Положение текста на кнопке. Текст может располагаться в центре кнопки (`MiddleCenter`), быть прижат к левой (`MiddleLeft`) или правой (`MiddleRight`) границе
	Можно задать и другие способы размещения надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`FlatStyle`	Стиль. Кнопка может быть стандартной (`Standard`), плоской (`Flat`) или "всплывающей" (`Popup`)
`Location`	Положение кнопки на поверхности формы. Уточняющее свойство `X` определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство `Y` — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)
`Size`	Размер кнопки
`Font`	Шрифт, используемый для отображения текста на кнопке

Таблица 2.2 *(окончание)*

Свойство	Описание
`ForeColor`	Цвет текста, отображаемого на кнопке
`Enabled`	Признак доступности кнопки. Кнопка доступна, если значение свойства равно `true`, и недоступна, если значение свойства равно `false` (в этом случае нажать кнопку нельзя, событие `Click` в результате щелчка на ней не возникает)
`Visible`	Позволяет скрыть кнопку (`false`) или сделать ее видимой (`true`)
`Cursor`	Вид указателя мыши при позиционировании указателя на кнопке
`Image`	Картинка на поверхности кнопки. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
`ImageAlign`	Положение картинки на кнопке. Картинка может располагаться в центе (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`ImageList`	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения нужно добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) изображения из набора `ImageList`, которое отображается на кнопке
`ToolTip`	Подсказка, появляющаяся рядом с указателем мыши при его позиционировании на кнопке. Чтобы свойство стало доступно, в форму приложения нужно добавить компонент `ToolTip`

ComboBox

Компонент ComboBox представляет собой комбинацию поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 2.3.

Таблица 2.3. *Свойства компонента* ComboBox

Свойство	Описание
DropDownStyle	Вид компонента: DropDown — поле ввода и раскрывающийся список; Simple — поле ввода со списком; DropDownList — раскрывающийся список
Text	Текст, находящийся в поле редактирования (для компонентов типа DropDown и Simple)
Items	Элементы списка — коллекция строк
Items->Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
Sorted	Признак автоматической сортировки (true) списка после добавления очередного элемента
MaxDropDownItems	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше, чем MaxDropDownItems, то появляется вертикальная полоса прокрутки
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDown и DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
DropDownWidth	Ширина области списка
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

ContextMenuStrip

Компонент ContextMenuStrip представляет собой контекстное меню — список команд, который отображается в результате щелчка правой кнопкой мыши на форме или в поле компонента. Команды меню определяет значение свойства Items, представляющего собой коллекцию объектов MenuItem. Свойства объекта MenuItem приведены в табл. 2.4. Чтобы задать контекстное меню компонента или формы, нужно указать имя контекстного меню (компонента ContextMenuStrip) в качестве значения свойства ContextMenuStrip.

Таблица 2.4. *Свойства объекта* `MenuItem`

Свойство	Описание
Text	Команда
Enabled	Признак доступности команды. Если значение свойства равно false, то команда недоступна (название команды отображается серым цветом)
Image	Картинка, которая отображается в строке команды
Checked	Признак того, что элемент меню выбран. Если значение свойства равно true, то элемент считается выбранным и помечается галочкой или (если значение свойства RadioCheck равно true) точкой. Свойство Checked обычно используется для тех элементов меню, которые предназначены для отображения параметров
RadioCheck	Признак того, что для индикации состояния свойства Checked используется точка (true), а не галочка (false)

CheckBox

Компонент CheckBox представляет собой переключатель, который может находиться в одном из двух состояний: выбранном или невыбранном (часто вместо "выбранный" говорят "установлен-

ный", а вместо "невыбранный" — "сброшенный"). Свойства компонента CheckBox приведены в табл. 2.5.

***Таблица 2.5.** Свойства компонента* `CheckBox`

Свойство	Описание
`Text`	Текст, отображаемый справа от кнопки
`Checked`	Состояние переключателя. Если переключатель выбран (в поле компонента отображается галочка), то значение свойства равно `true`. Если переключатель сброшен (галочка не отображается), то значение свойства равно `false`
`TextAllign`	Положение текста в поле отображения текста. Текст может располагаться в центре поля (`MiddleCenter`), быть прижат к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения текста надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`CheckAllign`	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (`TopLeft`), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (`MiddleLeft`). Есть и другие варианты размещения кнопки в поле компонента
`Enabled`	Управляет доступностью компонента. Позволяет сделать переключатель недоступным (`false`)
`Visible`	Управляет видимостью компонента. Позволяет скрыть, сделать невидимым (`false`) переключатель
`AutoCheck`	Определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно `true`
`FlatStyle`	Стиль (вид) переключателя. Переключатель может быть обычным (`Standard`), плоским (`Flat`) или "всплывающим" (`Popup`). Стиль определяет поведение переключателя при позиционировании указателя мыши на его изображении

***Таблица 2.5** (окончание)*

Свойство	Описание
`Appearance`	Определяет вид переключателя. Переключатель может выглядеть обычным образом (`Normal`) или как кнопка (`Button`)
`Image`	Картинка, которая отображается в поле компонента
`ImageAlign`	Положение картинки в поле компонента. Картинка может располагаться в центре (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`ImageList`	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения следует добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) картинки из набора `ImageList`, которая отображается в поле компонента

CheckedListBox

Компонент `CheckedListBox` представляет собой список, перед каждым элементом которого находится переключатель `CheckBox`. Свойства компонента `CheckedListBox` приведены в табл. 2.6.

***Таблица 2.6.** Свойства компонента `CheckedListBox`*

Свойство	Описание
`Items`	Элементы списка — коллекция строк
`Items->Count`	Количество элементов списка
`Sorted`	Признак необходимости автоматической сортировки (`true`) списка после добавления очередного элемента

Таблица 2.6 *(окончание)*

Свойство	Описание
`CheckOnClick`	Способ пометки элемента списка. Если значение свойства равно `false`, то первый щелчок выделяет элемент списка (строку), а второй устанавливает в выбранное состояние переключатель. Если значение свойства равно `true`, то щелчок на элементе списка выделяет элемент и устанавливает во включенное состояние переключатель
`CheckedItems`	Элементы, выбранные в списке
`CheckedItems->Count`	Количество выбранных элементов
`CheckedIndices`	Коллекция, элементы которой содержат номера выбранных элементов списка
`MultiColumn`	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка
`Location`	Положение компонента на поверхности формы
`Size`	Размер компонента без (для компонентов типа `DropDown` и `DropDownList`) или с учетом (для компонента типа `Simple`) размера области списка или области ввода
`Font`	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

GroupBox

Компонент `GroupBox` представляет собой контейнер для других компонентов. Обычно он используется для объединения в группы компонентов `RadioButton` по функциональному признаку. Свойства компонента `GroupBox` приведены в табл. 2.7.

Таблица 2.7. *Свойства компонента* `GroupBox`

Свойство	Описание
`Text`	Заголовок — текст, поясняющий назначение компонентов, которые находятся в поле компонента `GroupBox`
`Enabled`	Позволяет управлять доступом к компонентам, находящимся в поле (на поверхности) компонента `GroupBox`. Если значение свойства равно `false`, то все находящиеся в поле `GroupBox` компоненты недоступны
`Visible`	Позволяет скрыть (сделать невидимым) компонент `GroupBox` и все компоненты, которые находятся на его поверхности

ImageList

Компонент `ImageList` представляет собой коллекцию изображений и может использоваться другими компонентами (например, `Button` или `ToolBar`) как источник иллюстраций. Компонент является невизуальным, т. е. он не отображается в окне программы во время ее работы. Во время создания формы компонент отображается в нижней части окна редактора формы. Свойства компонента `ImageList` приведены в табл. 2.8.

Таблица 2.8. *Свойства компонента* `ImageList`

Свойство	Описание
`Images`	Коллекция изображений (объектов Bitmap)
`ImageSize`	Размер изображений коллекции. Уточняющее свойство `Width` определяет ширину изображений, `Height` — высоту
`Transparent Color`	Прозрачный цвет. Участки изображения, окрашенные этим цветом, не отображаются
`ColorDepht`	Глубина цвета — число байтов, используемых для кодирования цвета точки (пиксела)

Label

Компонент Label предназначен для отображения текстовой информации. Задать текст, отображаемый в поле компонента, можно как во время разработки формы, так и во время работы программы, присвоив нужное значение свойству Text. Свойства компонента приведены в табл. 2.9.

Таблица 2.9. *Свойства компонента* Label

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
Text	Отображаемый текст
Location	Положение компонента на поверхности формы
AutoSize	Признак автоматического изменения размера компонента. Если значение свойства равно true, то при изменении значения свойства Text (или Font) автоматически изменяется размер компонента
Size	Размер компонента (области отображения текста). Определяет (если значение свойства AutoSize равно false) размер компонента (области отображения текста)
MaximumSize	Если значение свойства AutoSize равно true, то задает максимально допустимый (макимально возможный) размер компонента (области отображения текста). Свойство MaximumSize.Width задает максимально возможную ширину области, свойство MaximumSize.Height — высоту
Font	Шрифт, используемый для отображения текста
ForeColor	Цвет текста, отображаемого в поле компонента
BackColor	Цвет закраски области вывода текста
TextAlign	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (TopLeft), посередине (TopCentre) и по центру (MiddleCenter)

Таблица 2.9 *(окончание)*

Свойство	Описание
`BorderStyle`	Вид рамки (границы) компонента. По умолчанию граница вокруг поля `Label` отсутствует (значение свойства равно `None`). Граница компонента может быть обычной (`Fixed3D`) или тонкой (`FixedSingle`)

ListBox

Компонент `ListBox` представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 2.10.

Таблица 2.10. *Свойства компонента* `ListBox`

Свойство	Описание
`Items`	Элементы списка — коллекция строк
`Items->Count`	Количество элементов списка
`SelectedIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
`Sorted`	Признак необходимости автоматической сортировки (`true`) списка после добавления очередного элемента
`SelectionMode`	Определяет режим выбора элементов списка: `One` — только один элемент; `MultiSimple` — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка; `MultiExtended` — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка при нажатой клавише <Ctrl>, или выделить диапазон, щелкнув при нажатой клавише <Shift> на первом и последнем элементе диапазона
`MultiColumn`	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка
`Location`	Положение компонента на поверхности формы

Таблица 2.10 *(окончание)*

Свойство	Описание
`Size`	Размер компонента без учета (для компонентов типа `DropDown` и `DropDownList`) или с учетом (для компонента типа `Simple`) размера области списка или области ввода
`Font`	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

MenuStrip

Компонент `MenuStrip` представляет собой главное меню программы. И сами меню, и команды, образующие меню, — это объекты `ToolStripMenuItem`. Свойства объекта `ToolStripMenuItem` приведены в табл. 2.11.

Таблица 2.11. Свойства объекта *`ToolStripMenuItem`*

Свойство	Описание
`Text`	Название меню (команды)
`Enabled`	Признак доступности меню (команды). Если значение свойства равно `false`, то меню (или команда) недоступно
`Image`	Картинка, отображаемая рядом с названием меню или команды. В качестве картинки следует использовать png-иллюстрацию с прозрачным фоном
`Checked`	Признак того, что элемент меню выбран. Если значение свойства равно `true`, то элемент помечается галочкой (если для него не задана картинка). Свойство `Checked` обычно используется для тех элементов меню, которые применяются для отображения параметров
`Shortcut`	Свойство определяет функциональную клавишу (или комбинацию клавиш), нажатие которой активизирует выполнение команды
`ShowShortcut`	Если значение свойства равно `true` и задано значение свойства `Shortcut`, то после названия команды отображается название функциональной клавиши, нажатие которой активизирует команду

NotifyIcon

Компонент NotifyIcon представляет собой значок, который отображается в системной области панели задач. Обычно он используется для изображения программ, работающих в фоновом режиме. При позиционировании указателя мыши на значке, как правило, появляется подсказка, а в результате щелчка правой кнопки — контекстное меню, команды которого позволяют управлять работой программы. Свойства компонента приведены в табл. 2.12.

***Таблица 2.12.** Свойства компонента NotifyIcon*

Свойство	Описание
Icon	Значок, который отображается на панели задач
Text	Подсказка (обычно название программы), которая отображается рядом с указателем мыши при позиционировании указателя на находящемся на панели задач значке
ContextMenuStrip	Ссылка на компонент ContextMenuStrip, обеспечивающий отображение контекстного меню
Visible	Свойство позволяет скрыть (false) значок (убрать с панели задач) или сделать его видимым

NumericUpDown

Компонент NumericUpDown предназначен для ввода числовых данных. Данные в поле редактирования можно ввести путем набора на клавиатуре или при помощи командных кнопок **Увеличить** и **Уменьшить**, которые находятся справа от поля редактирования. Свойства компонента NumericUpDown приведены в табл. 2.13.

***Таблица 2.13.** Свойства компонента NumericUpDown*

Свойство	Описание
Value	Значение, соответствующее содержимому поля редактирования

Таблица 2.13 *(окончание)*

Свойство	Описание
`Maximum`	Максимально возможное значение, которое можно ввести в поле компонента
`Minimum`	Минимально возможное значение, которое можно ввести в поле компонента
`Increment`	Величина, на которую увеличивается или уменьшается значение свойства `Value` при каждом щелчке мышью на кнопках **Увеличить** или **Уменьшить**
`TextAlign`	Расположение текста в поле редактирования (`Left` — прижат к левому краю; `Center` — в центре; `Right` — прижат к правому краю)

OpenFileDialog

Компонент `OpenFileDialog` представляет собой стандартное диалоговое окно **Открыть**. Свойства компонента приведены в табл. 2.14.

Таблица 2.14. Свойства компонента *`OpenFileDialog`*

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст **Открыть**
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов нужно отобразить только файлы с расширением txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

***Таблица 2.14** (окончание)*

Свойство	Описание
`FileNane`	Имя файла, введенное пользователем или выбранное в списке файлов
`InitialDirectory`	Каталог, содержимое которого отображается при появлении диалога на экране
`RestoreDirectory`	Признак необходимости отображать содержимое каталога, указанного в свойстве `InitialDirectory`, при каждом появлении окна. Если значение свойства равно `false`, то при следующем появлении окна на экране отображается содержимое каталога, выбранного пользователем в предыдущий раз

Panel

Компонент `Panel` представляет собой контейнер для других компонентов и позволяет легко управлять компонентами, которые находятся на панели. Например, для того чтобы сделать недоступными компоненты, находящиеся на панели, достаточно присвоить значение `false` свойству `Enabled` панели. Свойства компонента `Panel` приведены в табл. 2.15.

***Таблица 2.15.** Свойства компонента `Panel`*

Свойство	Описание
`Dock`	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (`Left`), правой (`Right`), верхней (`Top`) или нижней (`Bottom`) границе формы
`BorderStyle`	Вид границы панели: `FixedSingle` — рамка; `Fixed3D` — объемная граница; `None` — граница не отображается
`Enabled`	Свойство позволяет сделать недоступными (`false`) все компоненты, которые находятся на панели
`Visible`	Позволяет скрыть (`false`) панель

PictureBox

Компонент PictureBox обеспечивает отображение иллюстрации (рисунка, фотографии и т. п.). Свойства компонента приведены в табл. 2.16.

Таблица 2.16. *Свойства компонента* PictureBox

Свойство	Описание
Image	Иллюстрация, которая отображается в поле компонента
Size	Размер компонента. Уточняющее свойство Width определяет ширину компонента, Height — высоту
SizeMode	Метод отображения (масштабирования) иллюстрации, если ее размер не соответствует размеру компонента: Normal — масштабирование не выполняется (если размер компонента меньше размера иллюстрации, то отображается только часть иллюстрации); StretchImage — выполняется масштабирование иллюстрации так, чтобы она занимала всю область компонента (если размер компонента не пропорционален размеру иллюстрации, иллюстрация искажается)
	AutoSize — размер компонента автоматически изменяется в соответствии с размером иллюстрации; CenterImage — центрирование иллюстрации в поле компонента, если размер иллюстрации меньше размера компонента; Zoom — изменение размера иллюстрации таким образом, чтобы она занимала максимально возможную область компонента и при этом отображалась без искажения (с соблюдением пропорций)
Image->PhysicalDimension	Свойство содержит информацию об истинном размере картинки (иллюстрации), загруженной в поле компонента

***Таблица 2.16** (окончание)*

Свойство	Описание
`Location`	Положение компонента (области отображения иллюстрации) на поверхности формы. Уточняющее свойство `X` определяет расстояние от левой границы области до левой границы формы, уточняющее свойство `Y` — от верхней границы области до верхней границы клиентской области формы (нижней границы заголовка)
`Visible`	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
`BorderStyle`	Вид границы компонента: `None` — граница не отображается; `FixedSingle` — тонкая; `Fixed3D` — объемная

RadioButton

Компонент `RadioButton` представляет собой кнопку (переключатель), состояние которой зависит от состояния других кнопок (компонентов `RadioButton`). Обычно компоненты `RadioButton` объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента `GroupBox`). В каждый момент времени только одна из кнопок группы может находиться в выбранном состоянии (возможна ситуация, когда ни одна из кнопок не выбрана). Состояния компонентов, принадлежащих разным группам, независимы. Свойства компонента приведены в табл. 2.17.

***Таблица 2.17.** Свойства компонента `RadioButton`*

Свойство	Описание
`Text`	Текст, который находится справа от кнопки
`Checked`	Состояние, внешний вид кнопки. Если кнопка выбрана, то значение свойства `Checked` равно `true`; если кнопка не выбрана, то значение свойства `Checked` равно `false`

Таблица 2.17 *(продолжение)*

Свойство	Описание
`TextAllign`	Положение текста в поле отображения. Текст может располагаться в центре поля (`MiddleCenter`), прижат к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения текста надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`CheckAllign`	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (`TopLeft`), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (`MiddleLeft`). Есть и другие варианты размещения кнопки в поле компонента
`Enabled`	Свойство позволяет сделать кнопку недоступной (`false`)
`Visible`	Свойство позволяет скрыть (`false`) кнопку
`AutoCheck`	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно `true`
`FlatStyle`	Стиль кнопки. Кнопка может быть обычной (`Standard`), плоской (`Flat`) или "всплывающей" (`Popup`). Стиль кнопки определяет ее поведение при позиционировании указателя мыши на изображении кнопки
`Appearance`	Определяет вид переключателя. Переключатель может выглядеть обычным образом (`Normal`) или как кнопка (`Button`)
`Image`	Картинка, которая отображается в поле компонента
`ImageAlign`	Положение картинки в поле компонента. Картинка может располагаться в центре (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)

***Таблица 2.17** (окончание)*

Свойство	Описание
`ImageList`	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения нужно добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) картинки из набора `ImageList`, которая отображается в поле компонента

ProgressBar

Компонент `ProgressBar` — это индикатор, который обычно используется для наглядного представления протекания процесса (например, обработки или копирования файлов, загрузки информации из сети и т. п.). Свойства компонента `ProgressBar` приведены в табл. 2.18. Следует обратить внимание, что выход значения свойства `Value` за границы диапазона, заданного свойствами `Minimum` и `Maximum`, вызывает исключение.

***Таблица 2.18.** Свойства компонента `ProgressBar`*

Свойство	Описание
`Value`	Значение, которое отображается в поле компонента в виде полосы, длина которой пропорциональна значению свойства `Value`
`Minimum`	Минимально допустимое значение свойства `Value`
`Maximum`	Максимально допустимое значение свойства `Value`
`Step`	Приращение (шаг) изменения значения свойства `Value` при использовании метода `PerformStep`

SaveFileDialog

Компонент `SaveFileDialog` представляет собой стандартное диалоговое окно **Сохранить**. Свойства компонента приведены в табл. 2.19.

Таблица 2.19. *Свойства компонента* `SaveFileDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**
`FileName`	Полное имя файла, которое задал пользователь. В общем случае оно образуется из имени каталога, содержимое которого отображается в диалоговом окне, имени файла, которое пользователь ввел в поле **Имя файла**, и расширения, заданного значением свойства `DefaultExt`
`DefaultExt`	Расширение файла по умолчанию. Если пользователь в поле **Имя файла** не укажет расширение, то к имени файла будет добавлено расширение (при условии, что значение свойства `AddExtension` равно `true`), заданное значением этого свойства
`InitialDirectory`	Каталог, содержимое которого отображается при появлении диалога на экране
`RestoreDirectory`	Признак необходимости отображать содержимое каталога, указанного в свойстве `InitialDirectory`, при каждом появлении окна. Если значение свойства равно `false`, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз
`CheckPathExists`	Признак необходимости проверки существования каталога, в котором следует сохранить файл. Если указанного каталога нет, то выводится информационное сообщение
`CheckFileExists`	Признак необходимости проверки существования файла с заданным именем. Если значение свойства равно `true` и файл с указанным именем уже существует, то появляется окно запроса, в котором пользователь может подтвердить необходимость замены (перезаписи) существующего файла

Таблица 2.19 *(окончание)*

Свойство	Описание
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке файлов отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов нужно отобразить только файлы с расширением txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

TextBox

Компонент `TextBox` предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 2.20.

Таблица 2.20. *Свойства компонента* `TextBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст, который находится в поле редактирования
`Location`	Положение компонента на поверхности формы
`Size`	Размер компонента
`Font`	Шрифт, используемый для отображения текста в поле компонента
`ForeColor`	Цвет текста, находящегося в поле компонента
`BackColor`	Цвет фона поля компонента
`BorderStyle`	Вид рамки (границы) компонента. Граница компонента может быть обычной (`Fixed3D`), тонкой (`FixedSingle`) или отсутствовать (`None`)

Таблица 2.20 *(окончание)*

Свойство	Описание
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находиться по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
PasswordChar	Символ, который используется для отображения вводимых пользователем символов (введенная пользователем строка находится в свойстве Text)
Multiline	Разрешает (true) или запрещает (false) ввод нескольких строк текста
ReadOnly	Разрешает (true) или запрещает (false) редактирование отображаемого текста
Dock	Способ привязки положения и размера компонента к размеру формы. По умолчанию привязка отсутствует (None). Если значение свойства равно Top или Bottom, ширина компонента устанавливается равной ширине формы и компонент прижимается соответственно к верхней или нижней границе формы. Если значение свойства Dock равно Fill, а свойства Multiline — true, то размер компонента устанавливается максимально возможным
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме MultiLine. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
ScrollBars	Задает отображаемые полосы прокрутки: Horizontal — горизонтальная; Vertical — вертикальная; Both — горизонтальная и вертикальная; None — не отображать

ToolTip

Компонент ToolTip — вспомогательный, он используется другими компонентами формы для вывода подсказок при позициони-

ровании указателя мыши на компоненте. Свойства компонента приведены в табл. 2.21.

***Таблица 2.21.** Свойства компонента* `ToolTip`

Свойство	Описание
`Active`	Разрешает (`true`) или запрещает (`false`) отображение подсказок
`AutoPopDelay`	Время отображения подсказки
`InitialDelay`	Время, в течение которого указатель мыши должен быть неподвижным, чтобы появилась подсказка
`ReshowDelay`	Время задержки отображения подсказки после перемещения указателя мыши с одного компонента на другой

Timer

Компонент `Timer` генерирует последовательность событий `Tick`. Свойства компонента приведены в табл. 2.22.

***Таблица 2.22.** Свойства компонента* `Timer`

Свойство	Описание
`Interval`	Период генерации события `Tick`. Задается в миллисекундах
`Enabled`	Разрешение работы. Разрешает (значение `true`) или запрещает (значение `false`) генерацию события `Tick`

Графика

Графические примитивы

Вычерчивание графических примитивов (линий, прямоугольников, окружностей, дуг, секторов) на графической поверхности выполняют соответствующие методы объекта `Graphics` (табл. 2.23).

Таблица 2.23. *Некоторые методы вычерчивания графических примитивов*

Метод	Действие
`DrawLine (Pen, x1, y1, x2, y2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `x1`, `y1`, `x2`, `y2` — координаты точек начала и конца линии
`DrawLine(Pen, p1, p2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `p1` *и* `p2` (структуры `Point`) точки начала и конца линии
`DrawRectangle (Pen, x, y, w, h)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника
`DrawRectangle(Pen, rect)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметр `rect` (структура `Rectangle`) область, граница которой определяет контур прямоугольника
`FillRectangle (Brush, x, y, w, h)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника
`FillRectangle(Brush, rect)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметр `rect` (структура `Rectangle`) — закрашиваемую область
`DrawEllipse (Pen, x, y, w, h)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметры `x`, `y`, `w`, `h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс

Таблица 2.23 *(окончание)*

Метод	Действие
`DrawEllipse(Pen, rect)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметр `rect` — область, внутри которой рисуется эллипс
`FillEllipse (Brush, x, y, w, h)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметры `x`, `y`, `w`, `h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`FillEllipse (Brush, x, y, rect)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметр `rect` — область, внутри которой вычерчивается эллипс
`DrawPolygon(Pen, P)`	Рисует контур многоугольника. Параметр `Pen` определяет цвет, толщину и стиль линии границы многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника
`FillPolygon(Brush, P)`	Рисует закрашенный многоугольник. Параметр `Brush` определяет цвет и стиль закраски внутренней области многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника
`DrawString (str, Font, Brush, x, y)`	Выводит на графическую поверхность строку текста. Параметр `Font` определяет шрифт; `Brush` — цвет символов; `x` и `y` — точку, от которой будет выведен текст
`DrawImage(Image, x, y)`	Выводит на графическую поверхность иллюстрацию. Параметр `Image` определяет иллюстрацию; `x` и `y` — координату левого верхнего угла области вывода иллюстрации

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть набор цветных карандашей (всего их 141), при помоши которых можно рисовать сплошые линии толщиной в *один пиксел* (табл. 2.24).

***Таблица 2.24.** Некоторые карандаши из стандартного набора*

Карандаш	Цвет
`Pens::Red`	Красный
`Pens::Orange`	Оранжевый
`Pens::Yellow`	Желтый
`Pens::Green`	Зеленый
`Pens::LightBlue`	Голубой
`Pens::Blue`	Синий
`Pens::Purple`	Фиолетовый
`Pens::Black`	Черный
`Pens::LightGray`	Серый
`Pens::White`	Белый
`Pens::Transparent`	Прозрачный

Программист может создать свой собственный карандаш, объект типа `Pen` и, задав значения свойств (табл. 2.25), определить цвет, толщину и стиль линии, которую он будет рисовать.

***Таблица 2.25.** Свойства объекта `Pen`*

Свойство	Описание
`Color`	Цвет линии. В качестве значения свойства следует использовать одну из констант `Color` (например, `Color::Red`), определенных в пространстве имен `System::Drawing`
`Width`	Толщина линии (задается в пикселах)

Таблица 2.25 *(окончание)*

Свойство	Описание
`DashStyle`	Вид линии. В качестве значения свойства следует использовать одну из констант `DashStyle`, определенных в пространстве имен `System::Drawing::Drawing2D`): `DashStyle.Solid` — сплошная; `DashStyle::Dash` — пунктирная, длинные штрихи; `DashStyle::Dot` — пунктирная, короткие штрихи; `DashStyle::DashDot` — пунктирная, чередование длинного и короткого штрихов; `DashStyle::DashDotDot` — пунктирная, чередование одного длинного и двух коротких штрихов; `DashStyle::Custom` — пунктирная линия, вид которой определяет значение свойства `DashPattern`
`DashPattern`	Длина штрихов и промежутков пунктирной линии `DashStyle::Custom`

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. В распоряжении программиста есть четыре типа кистей: стандартные (`Brush`), штриховые (`HatchBrush`), градиентные (`LinearGradientBrush`) и текстурные (`TextureBrush`).

Стандартная кисть закрашивает область одним цветом (сплошная закраска). В стандартном наборе более 100 кистей, некоторые из них приведены в табл. 2.26.

Таблица 2.26. *Некоторые кисти из стандартного набора*

Кисть	Цвет
`Brushes::Red`	Красный
`Brushes::Orange`	Оранжевый
`Brushes::Yellow`	Желтый

Таблица 2.26 *(окончание)*

Кисть	Цвет
`Brushes::Green`	Зеленый
`Brushes::LightBlue`	Голубой
`Brushes::Blue`	Синий
`Brushes::Purple`	Фиолетовый
`Brushes::Black`	Черный
`Brushes::LightGray`	Серый
`Brushes::White`	Белый
`Brushes::Transparent`	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 2.27 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Таблица 2.27. *Некоторые стили штриховки областей*

Стиль	Штриховка
`HatchStyle::LightHorizontal`	Редкая горизонтальная
`HatchStyle::Horizontal`	Средняя горизонтальная
`HatchStyle::NarrowHorizontal`	Частая горизонтальная
`HatchStyle::LightVertical`	Редкая вертикальная
`HatchStyle::Vertical`	Средняя вертикальная
`HatchStyle::NarrowVertical`	Частая вертикальная
`HatchStyle::LageGrid`	Крупная сетка из горизонтальных и вертикальных линий
`HatchStyle::SmallGrid`	Мелкая сетка из горизонтальных и вертикальных линий

Таблица 2.27 *(окончание)*

Стиль	Штриховка
`HatchStyle::DottedGrid`	Сетка из горизонтальных и вертикальных линий, составленных из точек
`HatchStyle::ForwardDiagonal`	Диагональная штриховка "вперед"
`HatchStyle::BackwardDiagonal`	Диагональная штриховка "назад"
`HatchStyle::Percent05` — `HatchStyle::Percent90`	Точки (степень заполнения 5%, 10%, ..., 90%)
`HatchStyle::HorizontalBrick`	"Кирпичная стена"
`HatchStyle::LargeCheckerBoard`	"Шахматная доска"
`HatchStyle::SolidDiamond`	"Бриллиант" ("Шахматная доска", повернутая на 45°)
`HatchStyle::Sphere`	"Пузырьки"
`HatchStyle::ZigZag`	"Зигзаг"

Градиентная кисть (`LinearGradientBrush`) представляет собой прямоугольную область, цвет точек которой зависит от расстояния до границы. Обычно градиентные кисти двухцветные, т. е. цвет точек по мере удаления от границы постепенно меняется с одного на другой. Цвет может меняться вдоль горизонтальной или вертикальной границы области. Возможно также изменение цвета вдоль линии, образующей угол с горизонтальной границей.

Текстурная кисть (`TextureBrush`) представляет собой рисунок, который обычно загружается во время работы программы из файла (bmp, jpg или gif) или из ресурса. Закраска текстурной кистью выполняется путем дублирования рисунка внутри области.

Типы данных

Целый тип

Целые типы приведены в табл. 2.28.

Таблица 2.28. *Основные целые типы*

Тип	Диапазон	Формат
`System::SByte`	–128 ... 127	8 битов, со знаком
`System::Int16`	–32768 ... 32767	16 битов, со знаком
`System::Int32`	–2 147 483 648 ... 2 147 483 647	32 бита, со знаком
`System::Int64`	-2^{63} ... 2^{63}	64 бита, со знаком
`System::Byte`	0 ... 255	8 битов, без знака
`System::UInt16`	0 ... 65535	16 битов, без знака
`System::UInt32`	0 ... 4294967295	32 бита, без знака

Вещественный тип

Вещественные типы приведены в табл. 2.29.

Таблица 2.29. *Основные вещественные типы*

Тип	Диапазон	Значащих цифр	Байтов
`System::Single`	$-1.5 \cdot 10^{45}$ ... $3.4 \cdot 10^{38}$	7—8	4
`System::Double`	$-5.0 \cdot 10^{324}$ ... $1.7 \cdot 10^{308}$	15—16	8

Символьный и строковый типы

Основной символьный тип — `System::Char`.

Основной строковый тип — `System::String`.

Функции

В этом разделе описаны некоторые наиболее часто используемые функции.

Функции преобразования

В табл. 2.30 приведены функции, обеспечивающие преобразование строки символов в число, изображением которого является строка. Если строка не может быть преобразована в число (например, из-за того что содержит недопустимый символ), то возникает исключение типа `FormatException`. Функции преобразования принадлежат пространству имен `System::Convert`.

***Таблица 2.30.** Функции преобразования строки в число*

Функция	Значение
`ToSingle(s)` `ToDouble`	Дробное типа `Single`, `Double`

`ToByte(s)`
`ToInt16(s)`
`ToInt32(s)`
`ToInt64(s)`
	Целое типа `Byte`, `Int16`, `Int32`, `Int64`

`ToUInt16(s)`
`ToUInt32(s)`
`ToUInt64(s)`
	Целое типа `UInt16`, `UInt32`, `UInt64`

Преобразование числа в строку символов обеспечивает метод (функция) `ToString`. В качестве параметра функции `ToString` можно указать символьную константу (табл. 2.31), которая задает формат строки-результата.

Таблица 2.31. *Форматы отображения чисел*

Константа	Формат	Пример
c, C	Currency — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28 р.
e, E	Scientific (exponential) — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004
f, F	Fixed — число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
n, N	Number — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
g, G	General — универсальный формат. Похож на Number, но разряды не разделены на группы	55 055,275
r, R	Roundtrip — без округления. В отличие от формата N, этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55 055,2775

Функции манипулирования строками

Некоторые функции (методы) манипулирования строками приведены в табл. 2.32.

Таблица 2.32. *Функции манипулирования строками*

Функция (метод)	Значение
`Length`	Длина (количество) символов строки. Строго говоря, `Length` — это не метод, а свойство объекта `String`
`IndexOf(c)`	Положение (номер) символа `c` в строке `s` (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
`LastIndexOf(c)`	Положение (от конца) символа `c` в строке `s` (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
`IndexOf(st)`	Положение подстроки `st` в строке `s` (символы строки `s` нумеруются с нуля). Если указанной подстроки в строке нет, то значение функции равно минус один
`Trim()`	Строка, полученная из строки `s` путем "отбрасывания" пробелов, находящихся в начале и в конце строки
`Substring(n)`	Подстрока, выделенная из строки `s`, начиная с символа с номером `n` (символы строки `s` нумеруются с нуля). Если значение `n` больше, чем количество символов в строке, то возникает исключение `ArgumentOutOfRangeException`
`Substring(n,k)`	Подстрока длиной `k` символов, выделенная из строки `s`, начиная с символа с номером `n` (символы строки `s` нумеруются с нуля). Если значение `n` больше, чем количество символов в строке или если `k` больше, чем `len-n` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`Insert(pos,st)`	Строка, полученная путем вставки в строку `s` строки `st`. Параметр `pos` задает номер символа строки `s`, после которого вставляется строка `st`

Таблица 2.32 *(окончание)*

Функция (метод)	Значение
`Remove(pos,n)`	Строка, полученная путем удаления из строки `s` цепочки символов (подстроки). Параметр `pos` задает положение подстроки, параметр `n` — количество символов, которое нужно удалить. Если значение `pos` больше, чем количество символов в строке или если `n` больше, чем `len-pos` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`Replace(old,new)`	Строка, полученная из строки `s` путем замены всех символов `old` на символ `new`
`ToUpper()`	Строка, полученная из строки `s` путем замены строчных символов на прописные
`ToLower()`	Строка, полученная из строки `s` путем замены прописных символов на строчные
`Split(sep)`	Массив строк, полученный разбиением строки `s` на подстроки. Параметр `sep` (массив типа `Char`) задает символы, которые используются методом `Split` для обнаружения границ подстрок

Функции манипулирования датами и временем

Некоторые функции (методы) манипулирования датами приведены в табл. 2.33.

Таблица 2.33. *Функции манипулирования датами и временем*

Функция (метод)	Значение
`DateTime::Now`	Структура типа `DateTime`. Текущие дата и время
`DateTime::Today`	Структура типа `DateTime`. Текущая дата
`ToString()`	Строка вида dd.mm.yyyy mm:hh:ss (например, 05.06.2006 10:00)

Таблица 2.33 *(окончание)*

Функция (метод)	Значение
`ToString(f)`	Строка — дата и/или время. Вид строки определяет параметр `f`. Полному формату даты и времени соответствует строка `dd.MM.yyyy hh:mm:ss`
`Day`	День (номер дня месяца)
`Month`	Номер месяца (1 — январь, 2 — февраль и т. д.)
`Year`	Год
`Hour`	Час
`Minute`	Минуты
`DayOfYear`	Номер дня года (отсчет от 1 января)
`DayOfWeek`	День недели
`ToLongDateString()`	"Длинная" дата. Например: 5 июня 2009
`ToShortDateString()`	"Короткая" дата. Например: 05.06.2009
`ToLongTimeString()`	Время в формате hh:mm:ss
`ToShotrTimeString()`	Время в формате hh:mm
`AddDays(n)`	Дата, отстоящая от `d` на `n` дней. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`AddMonths(n)`	Дата, отстоящая от `d` на `n` месяцев. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`AddYears(n)`	Дата, отстоящая от `d` на `n` лет. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`AddHours(n)`	Дата (время), отстоящая от `d` на `n` часов. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`Minutes(т)`	Дата (время), отстоящая от `d` на `n` минут. Положительное `n` "сдвигает" дату вперед, отрицательное — назад

Функции манипулирования каталогами и файлами

Функции (методы) манипулирования каталогами и файлами (табл. 2.34) принадлежат пространству имен `System::IO`. При выполнении операций с каталогами и файлами возможны исключения (`di` — объект типа `DirectoryInfo`, `fi` — объект типа `FileInfo`, `sr` — объект типа `StreamReader`, `sw` — объект типа `StreamWriter`).

Таблица 2.34. *Функции (методы) манипулирования каталогами и файлами*

Функция (метод)	Результат (значение)
`DirectoryInfo (Path)`	Создает объект типа `DirectoryInfo`, соответствующий каталогу, заданному параметром `Path`
`di->GetFiles(fn)`	Формирует список файлов каталога — массив объектов типа `FileInfo`
	Каждый элемент массива соответствует файлу каталога, заданного объектом `di` (тип `DirectoryInfo`). Параметр `fn` задает критерий отбора файлов
`di->Exists`	Проверяет, существует ли в системе каталог. Если каталог существует, то значение функции равно `true`, в противном случае — `false`
`di->Create(dn)`	Создает каталог. Если путь к новому каталогу указан неправильно, возникает исключение
`fi->CopyTo(Path)`	Копирует файл, заданный объектом `fi` (тип `FileInfo`), в каталог и под именем, заданным параметром `Path`. Значение метода — объект типа `FileInfo`, соответствующий файлу, созданному в результате копирования
`fi->MoveTo(Path)`	Перемещает файл, заданный объектом `fi` (тип `FileInfo`), в каталог и под именем, заданным параметром `Path`

Таблица 2.34 *(продолжение)*

Функция (метод)	Результат (значение)
`fi->Delete()`	Уничтожает файл, соответствующий объекту `fi` (тип `FileInfo`)
`StreamReader(fn)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`. Поток открывается для чтения в формате UTF-8
`StreamReader (fn, encd)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`
	Поток открывается для чтения в кодировке, заданной параметром `encd` (объект типа `System::Text::Encoding`). Для чтения текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System::Text::Encoding:: GetEncoding(1251)`
`sr->Read()`	Читает символ из потока `sr` (объект типа `StremReader`). Значение метода — код символа
`sr->Read(buf, p, n)`	Читает из потока `sr` (объект типа `StremReader`) символы и записывает их в массив символов `buf` (тип `Char`). Параметр `p` задает номер элемента массива, в который будет помещен первый прочитанный символ, параметр `n` — количество символов, которое нужно прочитать
`sr->ReadToEnd()`	Читает весь текст из потока `sr` (объект типа `StremReader`). Значение метода — прочитанный текст
`sr->ReadLine()`	Читает строку из потока `sr` (объект типа `StremReader`). Значение метода — прочитанная строка

Таблица 2.34 *(окончание)*

Функция (метод)	Результат (значение)
`StreamWriter(fn)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`. Поток открывается для записи в формате (кодировке) UTF-8
`StreamWriter (fn,a,encd)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`
	Поток открывается для записи в кодировке, заданной параметром `encd` (объект типа `System::Text::Encoding`). Для записи текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System::Text::Encoding:: GetEncoding(1251)`. Параметр `a` задает режим записи: `true` — добавление в файл, `false` — перезапись
`sw->Write(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`. В качестве `v` можно использовать выражение строкового или числового типа
`sw->WriteLine(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`, и символ "новая строка". В качестве `v` можно использовать выражение строкового или числового типа
`sw->WriteLine`	Записывает в поток `sw` (объект типа `StreamWriter`) символ "новая строка"
`s->Close()`	Закрывает поток `s`

Математические функции

Некоторые математические функции, принадлежащие пространству имен `Math`, приведены в табл. 2.35.

Таблица 2.35. *Математические функции*

Функция (метод)	Значение
`Abs(n)`	Абсолютное значение `n`
`Sign(n)`	1 (если `n` — положительное) –1 (если `n` — отрицательное)
`Round(n,m)`	Дробное, полученное путем округления `n` по известным правилам. Параметр `m` задает количество цифр дробной части
`Ceiling(n)`	Дробное, полученное путем округления `n` "с избытком"
`Floor(n)`	Дробное, полученное путем округления `n` "с недостатком" (отбрасыванием дробной части)
`Log(n)`	Натуральный логарифм `n` (логарифм `n` по основанию `E`, где `E` — постоянная, значение которой равно 2.7182818284590452354)
`Log(n,m)`	Логарифм `n` по основанию `m`
`Log10(n)`	Десятичный логарифм `n`
`Exp(n)`	Экспонента `n` (число "`E`" в степени `n`)
`Cos(α)`	Косинус угла α, заданного в радианах
`Sin(α)`	Синус угла α, заданного в радианах
`Tan(α)`	Тангенс угла α, заданного в радианах
`ASin(n)`	Арксинус `n` — угол (в радианах), синус которого равен `n`
`ACos(n)`	Арккосинус `n` — угол (в радианах), косинус которого равен `n`
`ATan(n)`	Арктангенс `n` — угол (в радианах), тангенс которого равен `n`
`Sqrt(n)`	Квадратный корень из `n`
`r->Next(hb)`	Случайное число из диапазона 0 ... (`hb-1`). `r` — объект типа `System::Random`

Величина угла тригонометрических функций должна быть задана в радианах. Чтобы преобразовать величину угла из градусов в радианы, нужно значение, выраженное в градусах, умножить на π/180. Числовую константу 3.1415926 (значение числа "Пи") можно заменить именованной константой `PI`.

События

В табл. 2.36 приведено краткое описание некоторых событий.

Таблица 2.36. *События*

Событие	Происходит
`Click`	При щелчке кнопкой мыши
`Closed`	Возникает сразу за событием `Closing`
`Closing`	Возникает как результат щелчка на системной кнопке **Закрыть окно**
`DblClick`	При двойном щелчке кнопкой мыши
`Enter`	При получении элементом управления фокуса
`KeyDown`	При нажатии клавиши клавиатуры. Сразу за событием `KeyDown` возникает событие `KeyPress`. Если нажатая клавиша удерживается, то событие `KeyDown` возникает еще раз. Таким образом, пара событий `KeyDown — KeyPress` генерируется до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `KeyUp`)
`KeyPress`	При нажатии клавиши клавиатуры. Событие `KeyPress` возникает сразу после события `KeyDown`
`KeyUp`	При отпускании нажатой клавиши клавиатуры
`Leave`	При потере элементом управления фокуса
`Load`	В момент загрузки формы. Используется для инициализации программы
`MouseDown`	При нажатии кнопки мыши
`MouseMove`	При перемещении мыши

Таблица 2.36 *(окончание)*

Событие	Происходит
`MouseUp`	При отпускании кнопки мыши
`Paint`	При появлении окна (элемента управления) на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном, и в других случаях. Только процедура обработки события `Paint` имеет доступ к свойству `Graphics`, методы которого обеспечивают отображение графики
`Resize`	При изменении размера окна. Если размер формы увеличивается, то сначала происходит событие `Paint`, затем — `Resize`. Если размер формы уменьшается, то происходит только событие `Resize`

Исключения

В табл. 2.37 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 2.37. *Исключения*

Исключение	Возникает
`FormatException` — ошибка преобразования	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
`IndexOutOfRangeException` — выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
`ArgumentOutOfRangeException` — выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками

Таблица 2.37 *(окончания)*

Исключение	Возникает
`OverflowException` — переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю
`FileNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого файла (ошибка в имени файла или обращение к несуществующему или недоступному устройству)
`DirectoryNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого каталога (ошибка в имени каталога или обращение к несуществующему или недоступному устройству)

Предметный указатель