Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра теоретических основ информатики

# Разработка Windows Forms приложений на языке программирования C++

учебно-методическое пособие

Ижевск 2020

УДК 004.438(075.8)
ББК 32.973.22я73
Р177

*Рекомендовано к изданию Учебно-методическим советом УдГУ.*

**Составитель:** А.Ю. Сапаров, кандидат технических наук, доцент кафедры теоретических основ информатики ИМИТиФ

**Рецензент:** М.Н. Мокроусов, кандидат технических наук, доцент, заведующий кафедрой «Автоматизированные системы обработки информации и управлени» Ижевского государственного технического университета имени М.Т. Калашникова

Р177 Разработка Windows Forms приложений на языке программирования C++: учебно-методическое пособие / Сост.: А.Ю. Сапаров, Ижевск, 2020. 61 с.

Пособие посвящается вопросам разработки программных продуктов с графическим пользовательским интерфейсом. Содержит описание основных конструкций языка C++, связанных с использованием графических элементов, примеры решений, список задач по вариантам и указания по оформлению отчета. Знания и навыки, полученные при решении предложенных задач, будут полезны в дальнейшем на производственной практике, а также при подготовке курсовых и дипломных работ.

Рекомендуется студентам первого курса бакалавриата направления подготовки «Прикладная информатика», проходящим учебную (ознакомительную) практику, а также студентам других направлений подготовки, начинающим изучение основ проектирования и разработки программных продуктов с графическим пользовательским интерфейсом.

УДК 004.438(075.8)
ББК 32.973.22я73

# Введение

Данное учебно-методическое пособие предназначено для студентов первого курса направления подготовки «Прикладная информатика», проходящих учебную (ознакомительную) практику. Практика проходит на территории учебного заведения, в котором проходят обучение студенты. Для них в назначенное время выделяется отдельный компьютерный класс с установленным необходимым программным обеспечением. Продолжительность практики 2 недели. Каждому студенту назначается руководитель практики из числа сотрудников кафедры, к которой прикреплен студент. Цель практики заключается в ознакомлении с основами проектной деятельности, а именно изучение конкретных информационных технологий и систем информационного обеспечения для решения различных задач организационной, управленческой или научной деятельности, а также получении первичных навыков в оформлении отчета по выполненной работе, что будет полезно при подготовке курсовых и выпускных квалификационных работ.

Пособие содержит описание основных классов в языке программирования C++, используемых при построении оконных приложений для операционных систем семейства Windows, несколько задач на построение графического пользовательского интерфейса, указания по их решению с использованием среды Visual Studio и примерный план для составления отчета по проделанной работе.

Прежде, чем приступить к изучению данного пособия, студент должен пройти успешно курс программирования на ЭВМ, а именно часть курса, связанную с изучением языка C++, включая основы объектно-ориентированного программирования. Он должен знать основные понятия технологии объектно-ориентированного программирования, а именно понятия: «абстракция», «наследование», «инкапсуляция», «полиморфизм»; уметь описывать собственные классы, в том числе наследуемые от существующих; и обладать начальными навыками работы в среде Visual Studio, в том числе работы с проектами, состоящими из нескольких файлов с текстами модулей. Решение задач данного пособия должно поспособствовать получению более обширных знаний технологии объектно-ориентированного программирования, в том числе классов с

визуальным представлением; развитию умений описания предметной области с использованием классов и их использования на практике; и совершенствованию навыков работы в среде программирования с использованием проектов, предназначенных для создания приложений с графическим интерфейсом. Полученные знания, умения и навыки будет полезны в дальнейшем при изучении различных технологий программирования.

В результате решения поставленных задач у студента должны сформироваться некоторые общепрофессиональные и универсальные компетенции, в том числе способность разрабатывать алгоритмы и программы, пригодные для практического применения; способность осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач; способность управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни.

# 1 Справочная информация по созданию пользовательского интерфейса на языке программирования C++

Пространство имен (библиотека) System.Windows.Forms [2] содержит классы (типы данных) для создания приложений Windows, позволяющие использовать все возможности, предоставляемые операционной системой Microsoft Windows.

В основе графического пользовательского интерфейса лежит понятие формы (Form). Форма представляет окно или диалоговое окно, которое составляет пользовательский интерфейс приложения. Процесс разработки графического интерфейса заключается в создании формы и добавления на ней элементов управления. Элементы управления (Control) из себя представляют примитивные компоненты с визуальным представлением, имеющие стандартный вид и выполняющие стандартные действия.

Каждый элемент управления имеет свой набор свойств, методов и событий. *Свойство* элемента — это параметр, описывающий его состояние (внешний вид, координаты расположения на форме, доступность пользователю и т.д.). *Метод* — это процедура или функция, определяющая те действия, которые можно выполнить над этим элементом управления либо может выполнить сам элемент. *Событие* — это действие, распознаваемое элементом управления (манипуляции пользователя, действия операционной системы). Таким образом, для создания правильного элемента управления требуется задать нужные значения свойств, задать необходимые обработчики событий и корректно использовать методы.

Рассмотрим основные элементы управления, их некоторые свойства, методы и события, которые будут необходимы для решения практических задач.

## 1.1 Класс Form

Представляет окно или диалоговое окно, которое составляет пользовательский интерфейс приложения. Form — это представление любого окна, отображаемого в приложении (рис. 1). Класс Form можно использовать для создания

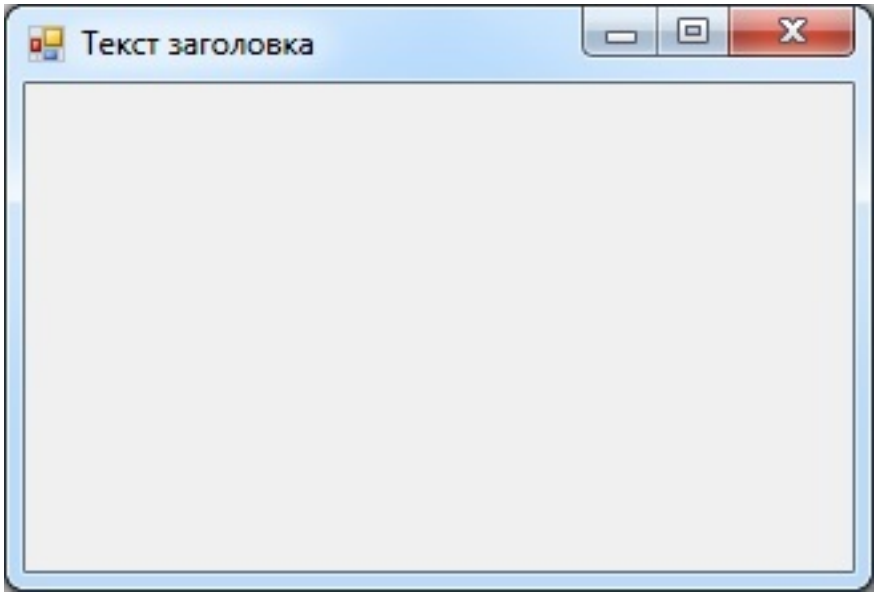


**Рис. 1.** Объект класса Form

стандартных окон, инструментальных, без границ и плавающих. Можно также использовать класс Form для создания модальных окон, например, диалоговых.

Объявление:

public ref class Form: System::Windows::Forms::ContainerControl

Свойства:

- *Text* — Возвращает или задает текст, связанный с этим элементом управления. Позволяет указать заголовок окна в строке заголовка.
- *Size* — Позволяет определить размер окна.
- *DesktopLocation* — Позволяет определить расположение окна при его отображении.
- *ForeColor* — Возвращает или задает цвет элемента управления.
- *FormBorderStyle* — Возвращает или задает стиль границы формы.
- *MinimizeBox* — Возвращает или задает значение, указывающее, отображается ли кнопка Свернуть в строке заголовка формы.
- *MaximizeBox* — Возвращает или задает значение, указывающее, отображается ли кнопка Развернуть в строке заголовка формы.

Методы:

- *ShowDialog* — Отображает форму в виде модального диалогового окна.

- *SetDesktopLocation* — Задает расположение формы в координатах рабочего стола.

Пример:

Разработка приложений с графическим интерфейсом основывается на описании формы. Форма и все остальные элементы представляются в виде объектов того или иного класса. В данном случае используется класс наследник для System::Windows::Forms::Form.

```
public ref class Form1 : public System::Windows::Forms::Form
{
    public:
    Form1(void)
    {
        InitializeComponent();
    }
    void InitializeComponent(void)
    {
        this->Text = L"Текст заголовка";
    }
}
```

В приведенном коде никаких особенностей нет, если сравнивать с любым другим использующим технологию объектно-ориентированного программирования. Описывается класс Form1, который содержит один конструктор без параметров. В конструкторе вызывается процедура инициализации компонент, в которой и выполняются основные действия по добавлению тех или иных элементов управления. В функции main() создается экземпляр этого класса и запускается приложение с указанием этой формы.

```
int main(array<System::String ^ > ^  args)
{
    Form1^  form=gcnew Form1();
    Application::Run(form);
    return 0;
}
```

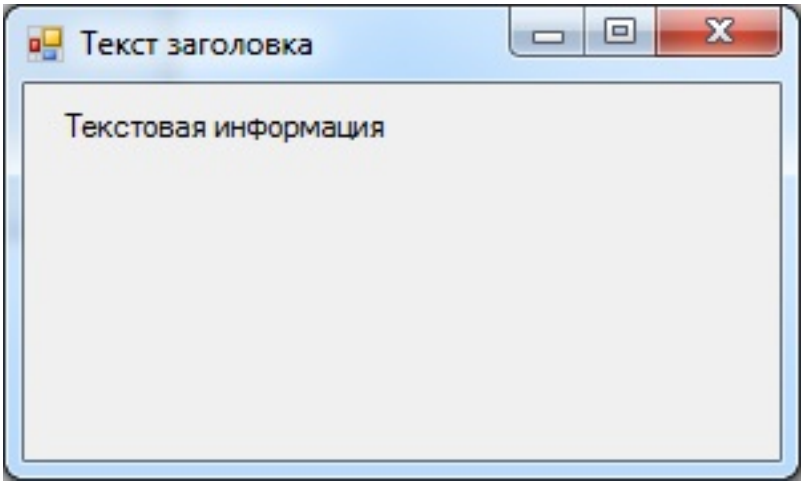


**Рис. 2.** Объект класса Label

## 1.2 Класс Label

Элементы управления Windows Forms Label предназначены для отображения текста (рис. 2) или изображений, которые пользователь только увидит, но не сможет изменить с клавиатуры. Они используются для идентификации объектов на форме — например, для описания того, что произойдет с элементом управления после выполнения на нем щелчка мышью, или для отображения сведений в ответ на процесс или событие во время выполнения приложения. Часто надписи используют для добавления описательных заголовков в текстовые поля, списки, поля со списком и т.д. Кроме того, возможно написание кода, который изменяет текст, отображаемый в надписи, в ответ на события во время выполнения. Если приложению требуется несколько минут на обработку изменения, можно отобразить в надписи сообщение о статусе обработки.

Объявление:

```
public ref class Label : System::Windows::Forms::Control,
                         System::Windows::Forms::Automation::IAutomationLiveRegion
```

Свойства:

- *Location* — Возвращает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла его контейнера (родительского элемента управления).
- *Size* — Возвращает или задает высоту и ширину элемента управления.
- *Text* — Возвращает или задает текст, связанный с этим элементом управления.

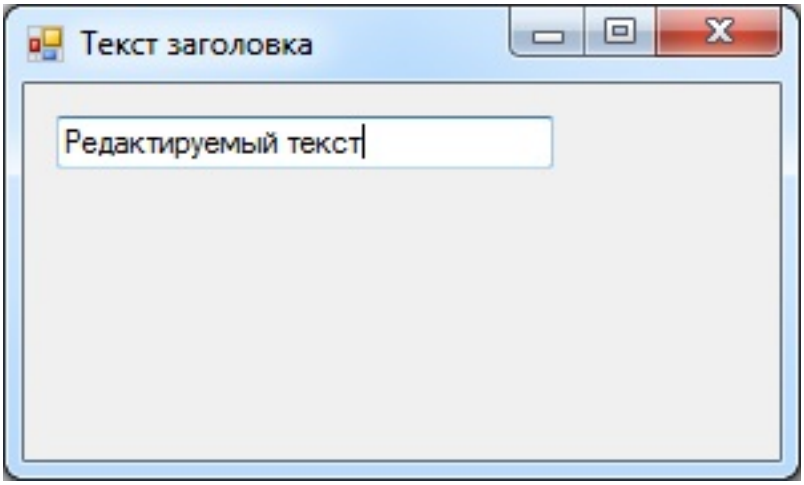


**Рис. 3.** Объект класса TextBox

## 1.3 Класс TextBox

Текстовые поля используются для приема данных, вводимых пользователем, или для отображения текста (рис. 3). Элемент управления TextBox обычно используется для редактируемого текста, хотя его можно также сделать доступным только для чтения. В текстовых полях можно выводить несколько строк текста, размещать текст в соответствии с размером элемента управления и применять основные элементы форматирования. В элементе управления TextBox можно вводить или отображать текст только в одном формате (т.е. весь текст форматируется одинаково).

Объявление:

```
public ref class TextBox : System::Windows::Forms::TextBoxBase
```

Свойства:

- *Location* — Возвращает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла его контейнера.
- *ReadOnly* — Возвращает или задает значение, указывающее, является ли текст в текстовом поле доступным только для чтения.
- *Size* — Возвращает или задает высоту и ширину элемента управления.
- *Text* — Возвращает или задает текст, связанный с этим элементом управления.

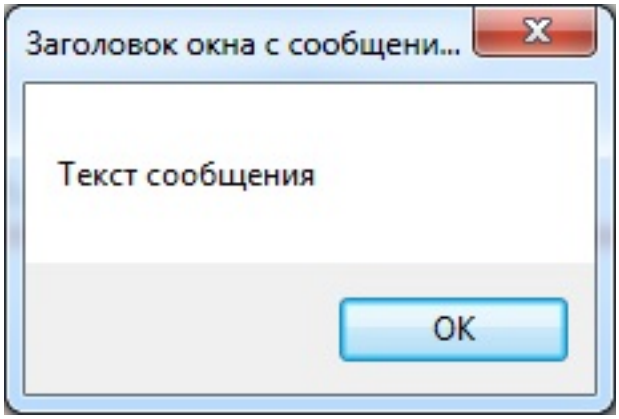


**Рис. 4.** Объект класса MessageBox

## 1.4 Класс MessageBox

Отображает окно сообщения (диалоговое окно) с текстом для пользователя (рис. 4). Это модальное окно, блокирующее другие действия в приложении, пока пользователь не закроет его. MessageBox может содержать текст, кнопки и символы, с помощью которых информируется и инструктируется пользователь.

Объявление:

```
public ref class MessageBox sealed
```

Методы:

- *Show(String, String)* — Отображает окно сообщения с заданным текстом и заголовком.

## 1.5 Класс Button

Представляет элемент управления «кнопка Windows». Элемент управления Windows Forms Button позволяет пользователю щелкнуть его для выполнения какого-либо действия (рис. 5). На элементе управления Button могут отображаться текст и изображение. При щелчке кнопки мышью элемент управления меняет свой вид так, как будто его нажимают и отпускают.

Объявление:

```
public ref class Button : System::Windows::Forms::ButtonBase,
                                    System::Windows::Forms::IButtonControl
```

Свойства:

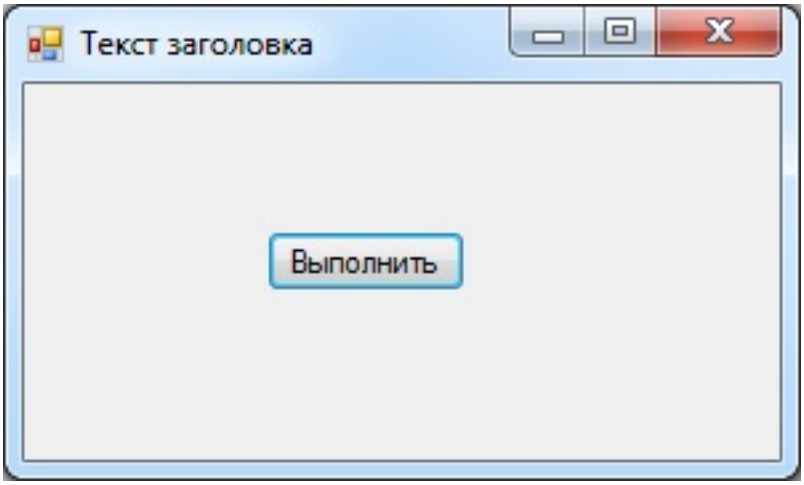


**Рис. 5.** Объект класса Button

- *Location* — Возвращает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла его контейнера.
- *Size* — Возвращает или задает высоту и ширину элемента управления.
- *Text* — Возвращает или задает текст, связанный с этим элементом управления.
- *AcceptButton* — Возвращает или задает кнопку в форме, которая срабатывает при нажатии клавиши ВВОД (ENTER).
- *CancelButton* — Возвращает или задает кнопку, которая срабатывает при нажатии клавиши Отмена (ESC).
- *DialogResult* — Возвращает или задает значение, возвращаемое в родительскую форму при нажатии кнопки.
- *FlatStyle* — Возвращает или задает плоский внешний вид для кнопки.

События:

- *Click* — Происходит при щелчке элемента управления.

Пример:

```
void InitializeComponent(void)
{
    this->Text = L"Текст заголовка";
    Button^  button1 = gcnew Button();
    button1->Location = System::Drawing::Point(214, 172);
    button1->Size = System::Drawing::Size(75, 23);
```

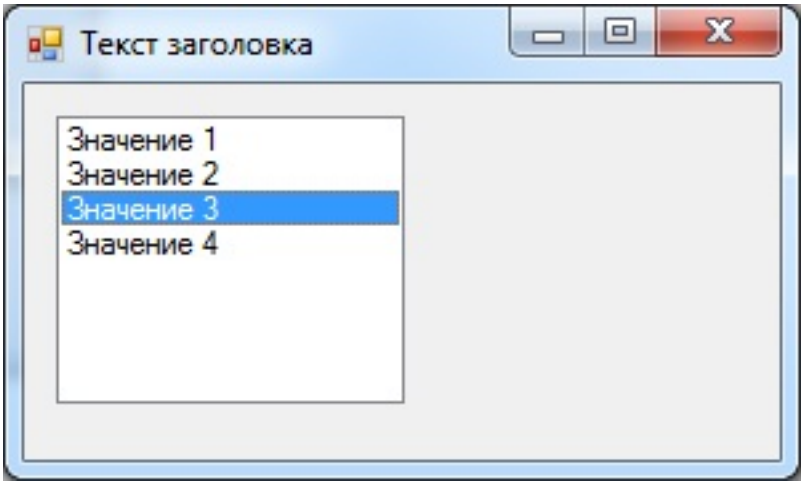


**Рис. 6.** Объект класса ListBox

```
    button1->Text = L"Выполнить";
    button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
    this->Controls->Add(button1);
}
private: System::void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    System::Windows::Forms::MessageBox::Show("Текст сообщения",
    "Заголовок окна с сообщением");
}
```

В данном примере создается элемент управления кнопка. Задаются координаты расположения (button1->Location = System::Drawing::Point(214, 172)), размеры (button1->Size = System::Drawing::Size(75, 23)), текст (button1->Text = L"Выполнить"), обработчик нажатия на кнопку (button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click)) и кнопка добавляется в список элементов управления формы (this->Controls->Add(button1)). При нажатии на кнопку будет вызываться процедура button1_Click, которая показывает окно с информационным сообщением.

## 1.6 Класс ListBox

Представляет элемент управления Windows для отображения списка элементов (рис. 6). Элемент управления ListBox позволяет отобразить список эле-

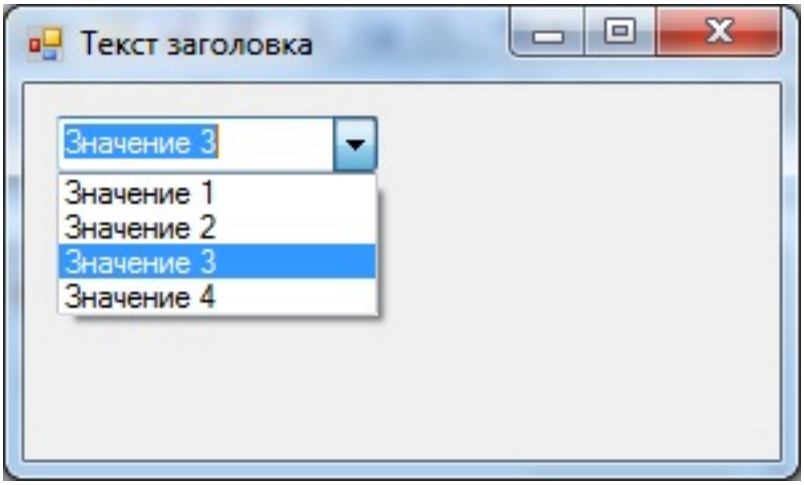


**Рис. 7.** Объект класса ComboBox

ментов, из которого пользователь может выбрать элемент, щелкнув на него.

Объявление:

public ref class ListBox : System::Windows::Forms::ListControl

Свойства:

- *Items* — Получает позиции элемента управления ListBox.
- *SelectedItems* — Получает коллекцию, которая содержит выделенные в настоящий момент позиции элемента управления ListBox.
- *SelectedIndices* — Получает коллекцию, содержащую индексы всех выделенных в настоящий момент позиций в элементе управления ListBox (индексы, начинающиеся с нуля).

Методы:

- *FindString* — Производит поиск первого элемента в ListBox, которая начинается с указанной строки.
- *FindStringExact* — Ищет первую позицию в ListBox, точно соответствующую указанной строке.

## 1.7 Класс ComboBox

Представляет элемент управления «поле со списком» (рис. 7). ComboBox отображает текстовое поле, объединенное с ListBox, которое позволяет пользователю выбрать элементы из списка или ввести новое значение.

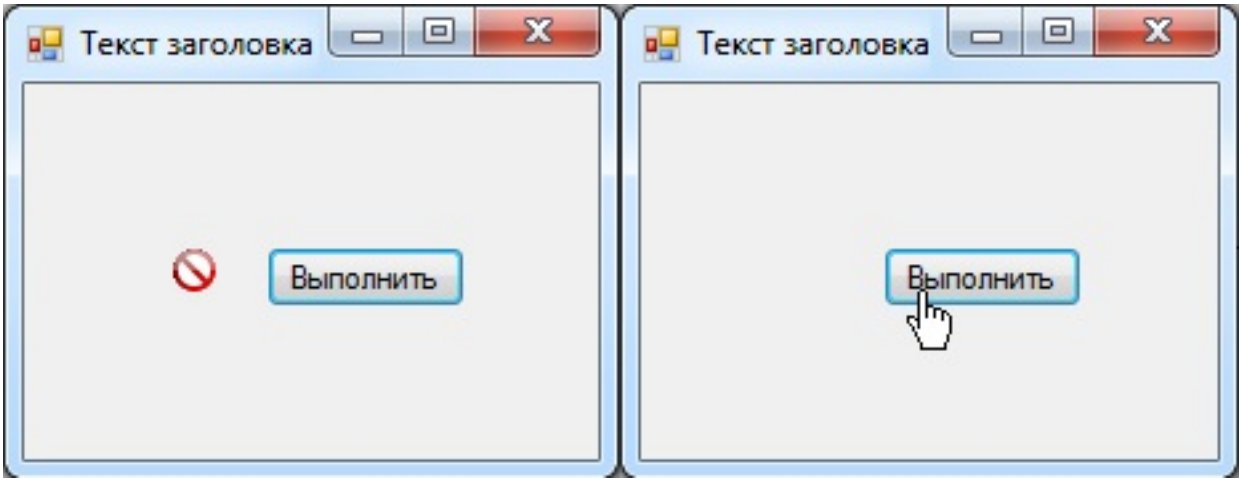


**Рис. 8.** Объект класса Cursor

Объявление:

public ref class ComboBox : System::Windows::Forms::ListControl

Свойства:

- *DropDownStyle* — Возвращает или задает значение, определяющее стиль поля со списком.
- *AddItemsCore(Object[])* — Добавляет указанные элементы в поле со списком.

## 1.8 Класс Cursor

Представляет изображение, используемое для задания внешнего вида указателя мыши (рис. 8). Курсор представляет собой небольшое изображение, положение которого на экране управляется указывающим устройством, например мышью, пером или другим способом. Когда пользователь перемещает указывающее устройство, операционная система соответствующим образом перемещает курсор.

Объявление:

public ref class Cursor sealed : IDisposable,
System::Runtime::Serialization::ISerializable

Конструкторы:

- *Cursor(Type, String)* — Инициализирует новый экземпляр класса Cursor из указанного ресурса, используя указанный тип ресурса.

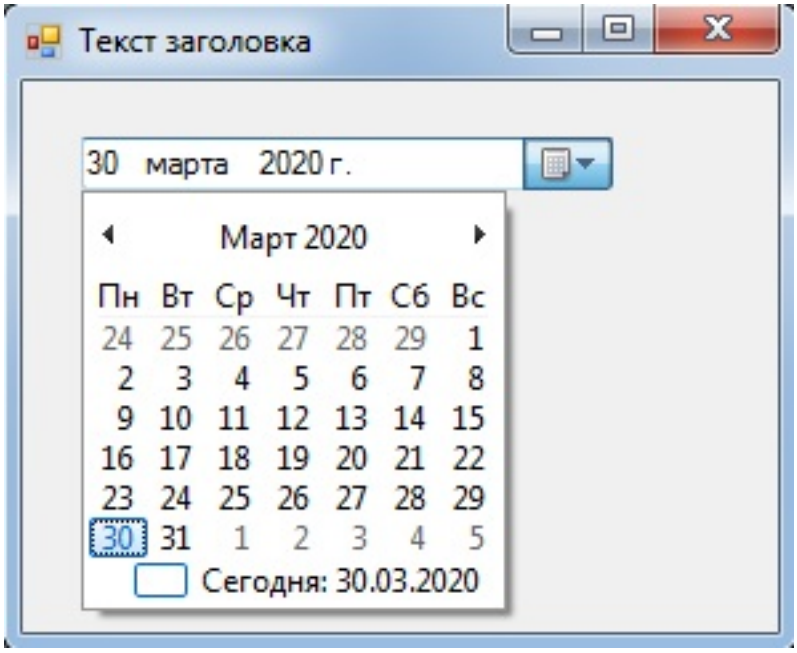


**Рис. 9.** Объект класса DateTimePicker

Свойства:

- *Position* — Возвращает или задает положение курсора.

## 1.9 Класс DateTimePicker

Представляет элемент управления Windows, который позволяет выбрать дату и время (рис. 9) и отобразить их в указанном формате. Элемент управления DateTimePicker используется для предоставления пользователю возможности выбора даты и времени, а также для вывода даты и времени в указанном формате. Элемент управления DateTimePicker упрощает работу с датами и временем, так как он автоматически обрабатывает большую часть проверки данных.

Объявление:
public ref class DateTimePicker : System::Windows::Forms::Control Свойства:

- *Value* — Возвращает или задает значение даты/времени, назначаемое элементу управления.

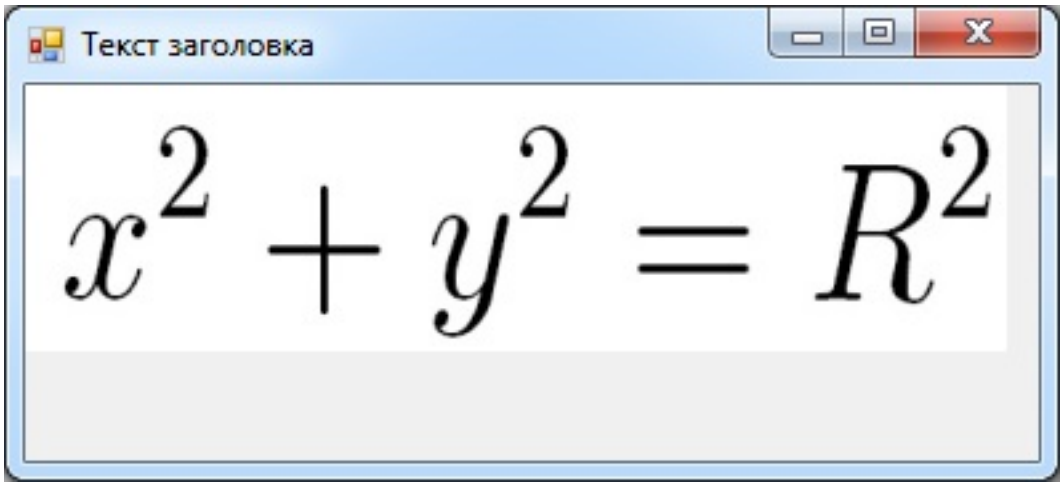


**Рис. 10.** Объект класса PictureBox

## 1.10 Класс PictureBox

Представляет элемент управления Windows «поле рисунка», предназначенный для отображения рисунков (рис. 10). Обычно PictureBox используется для вывода изображений из файла точечного рисунка, метафайла, значка, JPEG, GIF или PNG.

Объявление:

```
public ref class PictureBox : System::Windows::Forms::Control,
                              System::ComponentModel::ISupportInitialize
```

Свойства:

- *Image* — Получает или задает изображение, отображаемое элементом управления PictureBox.
- *SizeMode* — Указывает способ отображения изображения.

## 1.11 Класс CheckBox

Представляет флажок Windows CheckBox (рис. 11).

Объявление:

```
public ref class CheckBox : System::Windows::Forms::ButtonBase
```

Свойства:

- *Appearance* — Возвращает или задает значение, определяющее внешний вид элемента управления CheckBox.

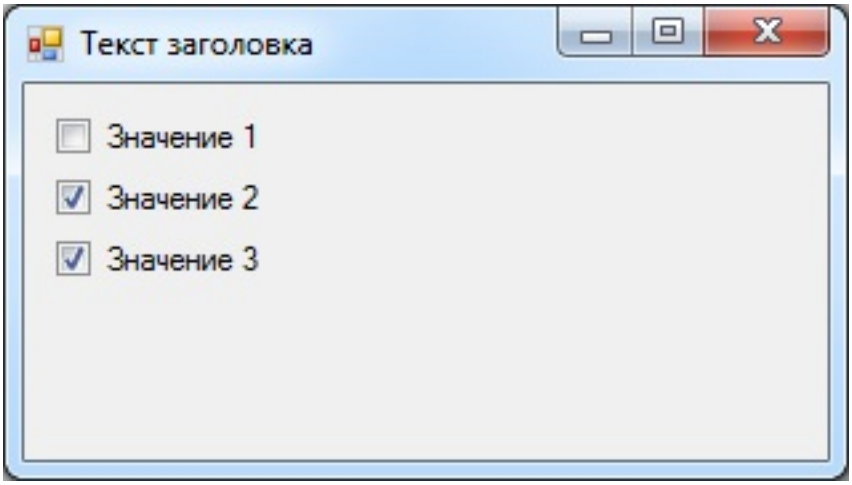


**Рис. 11.** Объект класса CheckBox

- *ThreeState* — Возвращает или задает значение, определяющее, будет ли три состояния у CheckBox.
- *Checked* — Возвращает или задает значение, определяющее, находится ли CheckBox в выбранном состоянии.
- *CheckState* — Возвращает или задает состояние CheckBox.

События:

- *Click* — Происходит при щелчке элемента управления. (Унаследовано от Control)

## 1.12 Класс RadioButton

Позволяет пользователю выбрать единственный вариант из нескольких доступных (рис. 12), когда используется вместе с другими элементами управления RadioButton. Элементы управления RadioButton и CheckBox имеют аналогичную функцию: они предоставляют возможность выбора. Различие состоит в том, что в CheckBox одновременно можно выбрать несколько элементов, а в RadioButton при выборе одного элемента отменяется предыдущий выбор.

Объявление:

```
public ref class RadioButton : System::Windows::Forms::ButtonBase
```

Свойства:

- *Checked* — Возвращает или задает значение, указывающее, выбран ли данный элемент управления.

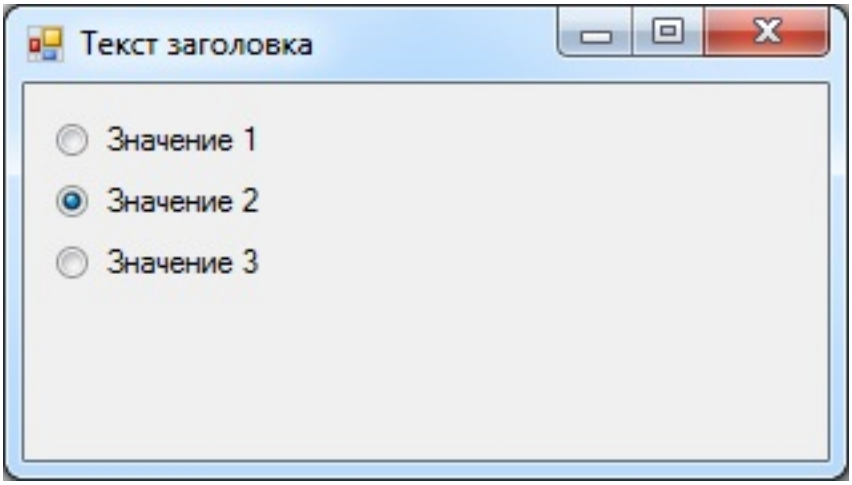


**Рис. 12.** Объект класса RadioButton

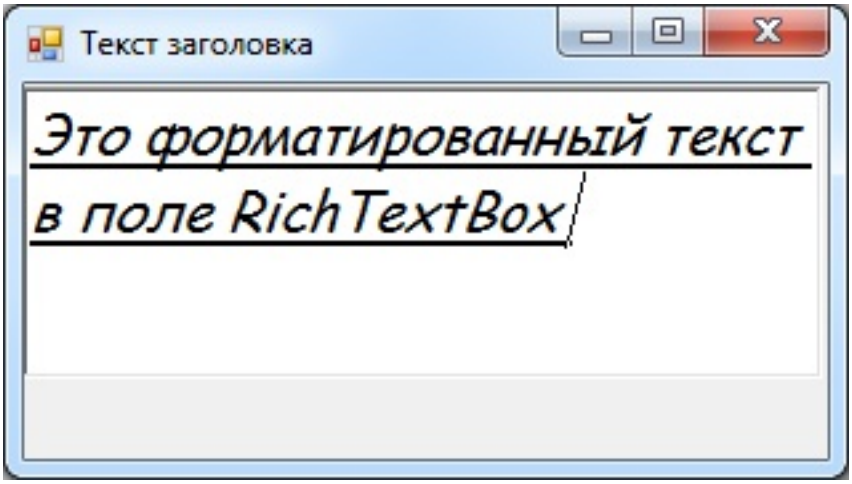


**Рис. 13.** Объект класса RichTextBox

- *Appearance* — Возвращает или задает значение, определяющее внешний вид переключателя RadioButton.

## 1.13 Класс RichTextBox

Предоставляет элемент управления полем текста с возможностью форматирования (рис. 13).

Объявление:

public ref class RichTextBox : System::Windows::Forms::TextBoxBase

Свойства:

- *SelectionFont* — Возвращает или задает цвет текущего текстового выделения или места вставки.

- *SelectionColor* — Возвращает или задает цвет текста, который будет применен к текущему выделению или положению курсора.
- *SelectionBullet* — Возвращает или задает значение, определяющее, будет ли применен к текущему выделению или положению курсора стиль маркированного списка.
- *SelectionIndent* — Возвращает или задает длину отступа первой строки выделенного в пикселях.
- *SelectionRightIndent* — Возвращает или задает расстояние в пикселях между правым краем элемента управления RichTextBox и правым краем текущего текстового выделения или текста, добавленного после места вставки.
- *SelectionHangingIndent* — Возвращает или задает расстояние между левым краем первой строки текста выделенного абзаца и левым краем последующих строк того же абзаца.

Методы:

- *LoadFile* — Загружает содержимое файла в элемент управления RichTextBox.
- *SaveFile* — Сохраняет в файл содержимое элемента управления RichTextBox.
- *Find* — Осуществляет поиск текста в содержимом RichTextBox.

События:

- *LinkClicked* — Происходит при щелчке пользователем ссылки, расположенной в тексте элемента управления.

## 1.14 Класс SplitContainer

Объявление:

```
public ref class SplitContainer : System::Windows::Forms::ContainerControl,
                                   System::ComponentModel::ISupportInitialize
```

Представляет элемент управления, состоящий из подвижной строки, которая разделяет область отображения контейнера на две панели с изменяемыми размерами (рис. 14).

Свойства:

- *Orientation* — Возвращает или задает значение, указывающее ориентацию (горизонтальную или вертикальную) панелей SplitContainer.

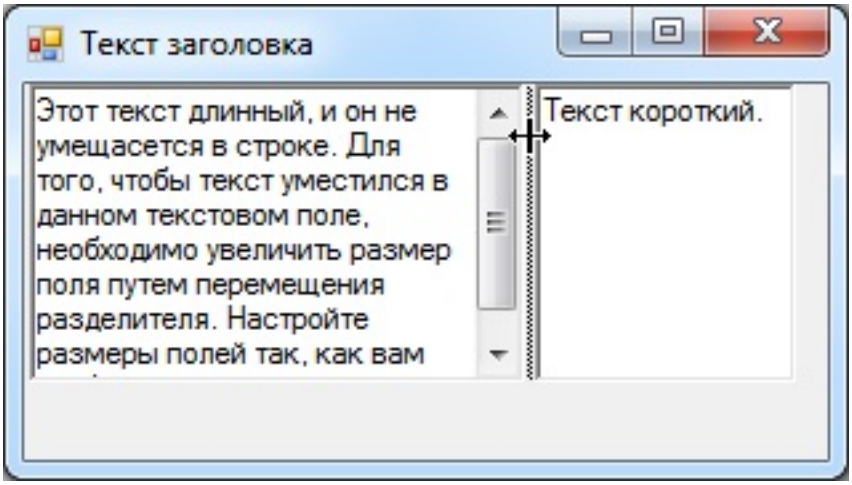


**Рис. 14.** Объект класса SplitContainer

- *Panel1* — Получает левую или верхнюю панель объекта SplitContainer в зависимости от значения Orientation.
- *Panel1Collapsed* — Возвращает или задает значение, определяющее, свернута панель Panel1 или развернута.
- *Panel1MinSize* — Возвращает или задает минимальное расстояние (в пикселях) разделителя от левого или верхнего края панели Panel1.
- *Panel2* — Получает правую или нижнюю панель объекта SplitContainer в зависимости от значения Orientation.
- *Panel2Collapsed* — Возвращает или задает значение, определяющее, свернута панель Panel2 или развернута.
- *SplitterDistance* — Возвращает или задает местоположение разделителя (в пикселях) от левого или верхнего края объекта SplitContainer.
- *PreferredSize* — Возвращает размер прямоугольной области, в которую может поместиться элемент управления.

События:

- *SplitterMoved* — Происходит при перемещении элемента управления разделителем.
- *SplitterMoving* — Происходит, когда элемент управления разделителя находится в процессе перемещения.

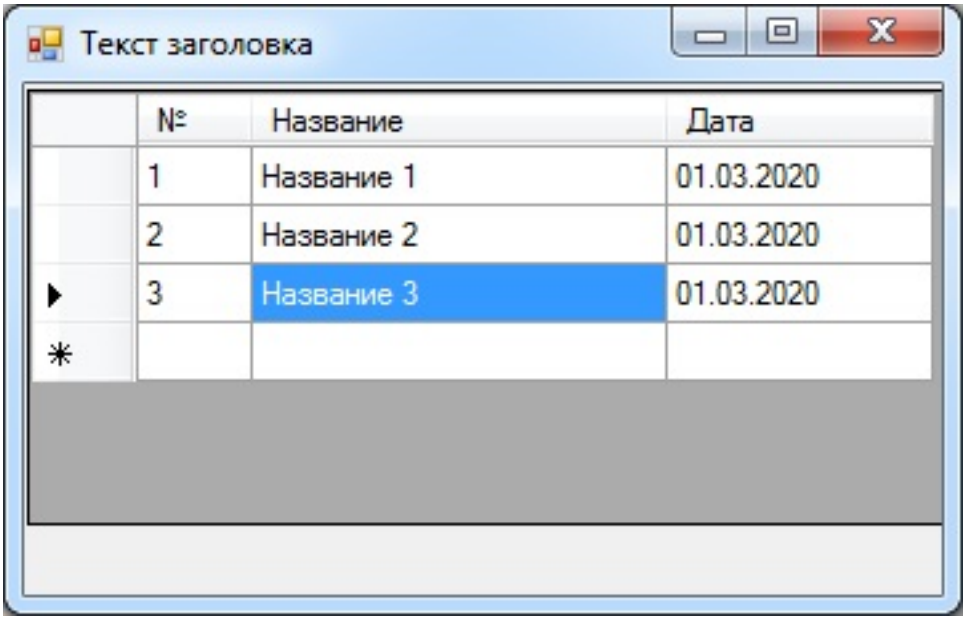


**Рис. 15.** Объект класса DataGridView

## 1.15 Класс DataGridView

Элемент управления DataGridView предоставляет настраиваемую таблицу для отображения данных (рис. 15). Класс DataGridView позволяет настраивать ячейки, строки, столбцы и границы таблицы с помощью таких свойств, как DefaultCellStyle, ColumnHeadersDefaultCellStyle, CellBorderStyle и GridColor.

Элемент управления DataGridView можно использовать для вывода данных с базовым источником данных или без него. Без указания источника данных можно создавать столбцы и строки, содержащие данные, и добавлять их непосредственно в DataGridView с помощью свойств Rows и Columns. Можно также использовать коллекцию Rows для доступа к DataGridViewRow объектам и свойство DataGridViewRow.Cells для прямого чтения или записи значений ячеек. Item[String, Int32] индексатор также предоставляет прямой доступ к ячейкам.

В качестве альтернативы ручному заполнению элемента управления можно задать свойства DataSource и DataMember, чтобы привязать DataGridView к источнику данных и автоматически заполнить его данными.

Объявление:

```
public ref class DataGridView : System::Windows::Forms::Control,
                                System::ComponentModel::ISupportInitialize
```

Свойства:

- *Rows* — Возвращает коллекцию, содержащую все строки в элементе управления DataGridView.
- *Columns* — Возвращает коллекцию, содержащую все столбцы элемента управления.
- *DataSource* — Возвращает или задает источник, данные для которого отображаются в таблице.
- *DataMember* — Возвращает или задает определенный список в свойстве DataSource, для которого элемент управления DataGrid отображает таблицу.

  Допустимы следующие источники данных:

  – DataTable
  – DataView
  – DataSet
  – DataViewManager
  – Одномерный массив
  – Любой компонент, реализующий интерфейс IListSource
  – Любой компонент, реализующий интерфейс IList

- *Cells* — Возвращает коллекцию ячеек, заполняющих строку.
- *DefaultCellStyle* — Возвращает или задает стиль ячейки по умолчанию, который будет применяться к ячейкам в объекте DataGridView, если не заданы какие-либо другие свойства стиля ячейки.
- *ColumnHeadersDefaultCellStyle* — Возвращает или задает стиль заголовка столбца по умолчанию.
- *CellBorderStyle* — Возвращает или задает стиль границы ячейки для объекта DataGridView.
- *GridColor* — Возвращает или задает цвет линий сетки, разделяющих ячейки объекта DataGridView.

Методы:

- *SetDataBinding(Object, String)* — Задает свойства DataSource и DataMember во время выполнения.

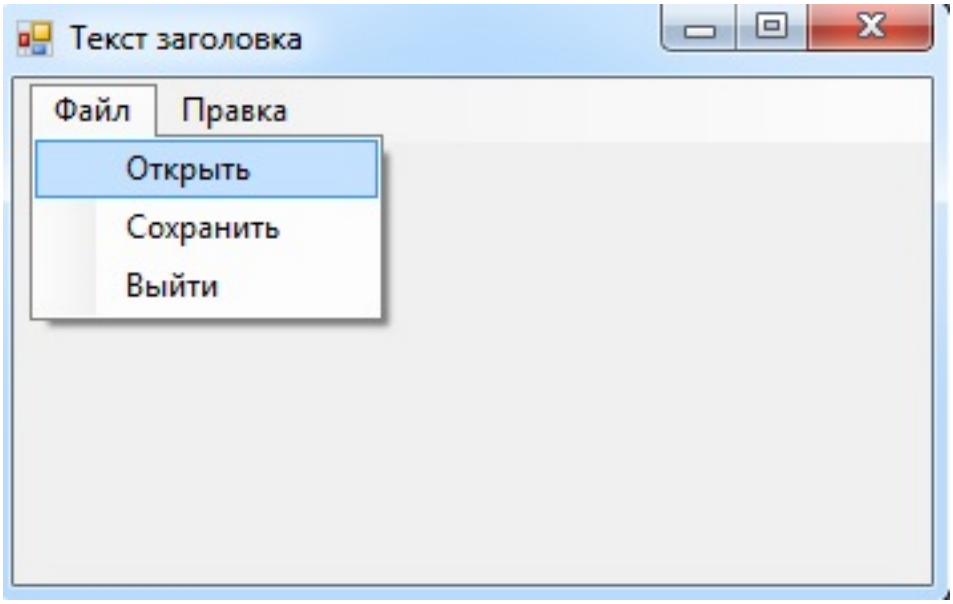


**Рис. 16.** Объект класса MenuStrip

## 1.16 Класс MenuStrip

Объект MenuStrip элемент управления группирует команды приложения и предоставляет быстрый доступ к ним (рис. 16).

Объявление:

```
public ref class MenuStrip : System::Windows::Forms::ToolStrip
```

Свойства:

- *Enabled* — Указывает, включен ли элемент управления.
- *Items* — Возвращает все элементы, которые принадлежат объекту.

## 1.17 Класс OpenFileDialog

Отображает диалоговое окно, позволяющее пользователю выбрать файл для открытия (рис. 17).

Объявление:

```
public ref class OpenFileDialog sealed : System::Windows::Forms::FileDialog
```

Свойства:

- *FileName* — Возвращает или задает строку, содержащую имя файла, выбранного в диалоговом окне.

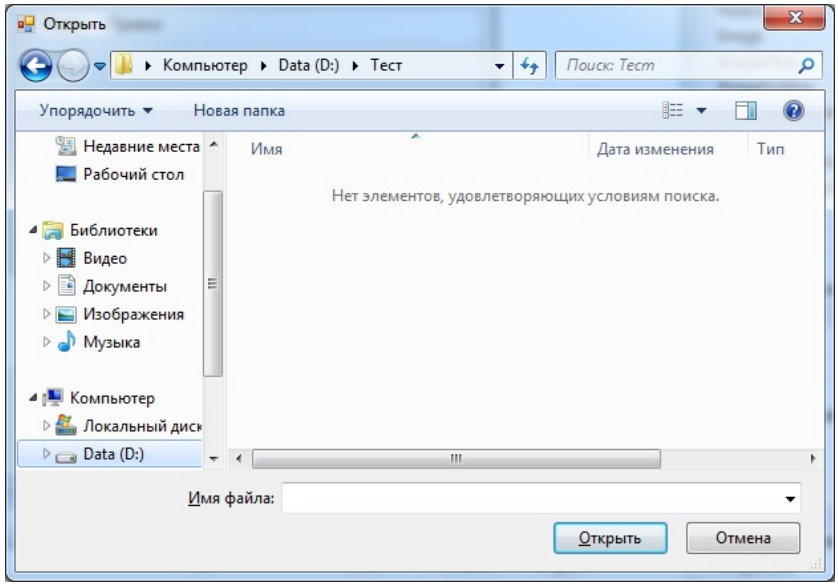


**Рис. 17.** Объект класса OpenFileDialog

- *FileNames* — Возвращает имена всех выбранных файлов в диалоговом окне.
- *Filter* — Возвращает или задает текущую строку фильтра имен файлов, которая определяет варианты, появляющиеся в поле «тип файла» в диалоговом окне.

Методы:

- *ShowDialog()* — Запускает общее диалоговое окно с заданным по умолчанию владельцем.

События:

- *FileOk* — Происходит, когда пользователь щелкает кнопку «Открыть» в диалоговом окне выбора файла.

Пример:

```
private:
void button1_Click(Object^  sender, System::EventArgs^  e)
{
      Stream^  myStream;
      OpenFileDialog^  openFileDialog1 = gcnew OpenFileDialog();
      openFileDialog1->InitialDirectory = "c:\\";
      openFileDialog1->Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
```

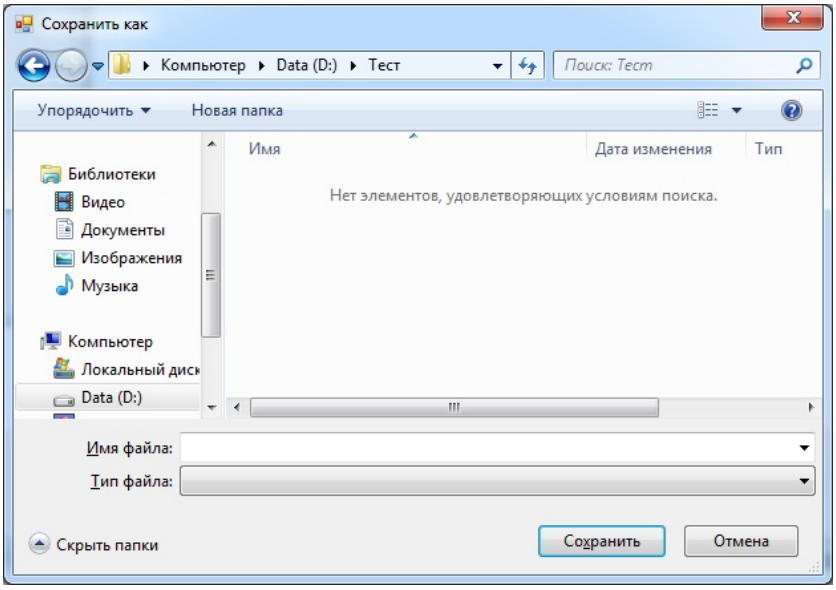


**Рис. 18.** Объект класса SaveFileDialog

```
    openFileDialog1->FilterIndex = 2;
    openFileDialog1->RestoreDirectory = true;
    if (openFileDialog1->ShowDialog() == DialogResult::OK)
    {
        if ( (myStream = openFileDialog1->OpenFile()) != nullptr )
        {
            // Insert code to read the stream here.
            myStream->Close();
        }
    }
}
```

## 1.18 Класс SaveFileDialog

Запрашивает у пользователя местоположение для сохранения файла (рис. 18).

Объявление:

```
public ref class SaveFileDialog sealed : System::Windows::Forms::FileDialog
```

Свойства:

- *FileName* — Возвращает или задает строку, содержащую имя файла, выбранного в диалоговом окне.

- *Filter* — Возвращает или задает текущую строку фильтра имен файлов, которая определяет варианты, появляющиеся в поле «тип файла» в диалоговом окне.

Методы:

- *ShowDialog()* — Запускает общее диалоговое окно с заданным по умолчанию владельцем.

События:

- *FileOk* — Происходит, когда пользователь щелкает кнопку «Сохранить» в диалоговом окне выбора файла.

Пример:

```
private:
void button1_Click(Object^  sender, System::EventArgs^  e)
{
    Stream^  myStream;
    SaveFileDialog^  saveFileDialog1 = gcnew SaveFileDialog();
    saveFileDialog1->InitialDirectory = "c:\\";
    saveFileDialog1->Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog1->FilterIndex = 2;
    saveFileDialog1->RestoreDirectory = true;
    if (saveFileDialog1->ShowDialog() == DialogResult::OK)
    {
        if ( (myStream = saveFileDialog1->OpenFile()) != nullptr )
        {
            // Code to write the stream goes here.
            myStream->Close();
        }
    }
}
```

# 2 Пример простейшей программы

Рассмотрим простейшую задачу вычисления длины гипотенузы по заданным значениям длин катетов. Составим две программы для решения данной задачи.

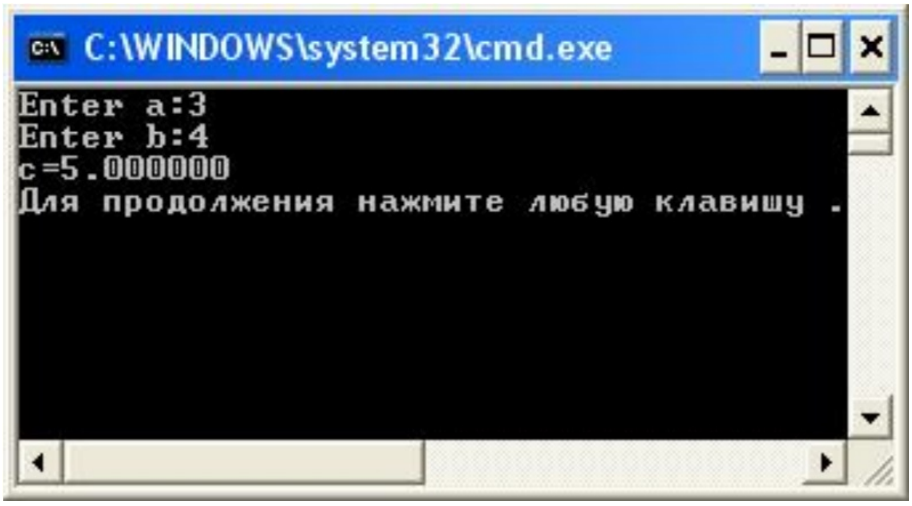


**Рис. 19.** Выполнение консольного приложения

Первая программа будет представлять собой консольное приложение, а второе —приложение с графическим интерфейсом.

## 2.1 Консольное приложение

Программа достаточно простая. С клавиатуры вводятся последовательно значения длин двух катетов, после чего на экран выводится найденное значение гипотенузы. При запуске увидим картину, изображенную на рис. 19. Полный текст программы выглядит следующим образом:

```
#include <iostream>
#include <math.h>
int main()
{
	float a,b,c;
	printf("Enter a:");
	scanf("%f", &a);
	printf("Enter b:");
	scanf("%f", &b);
	c=sqrt(a*a+b*b);
	printf("c=%f\n", c);
	return 0;
}
```

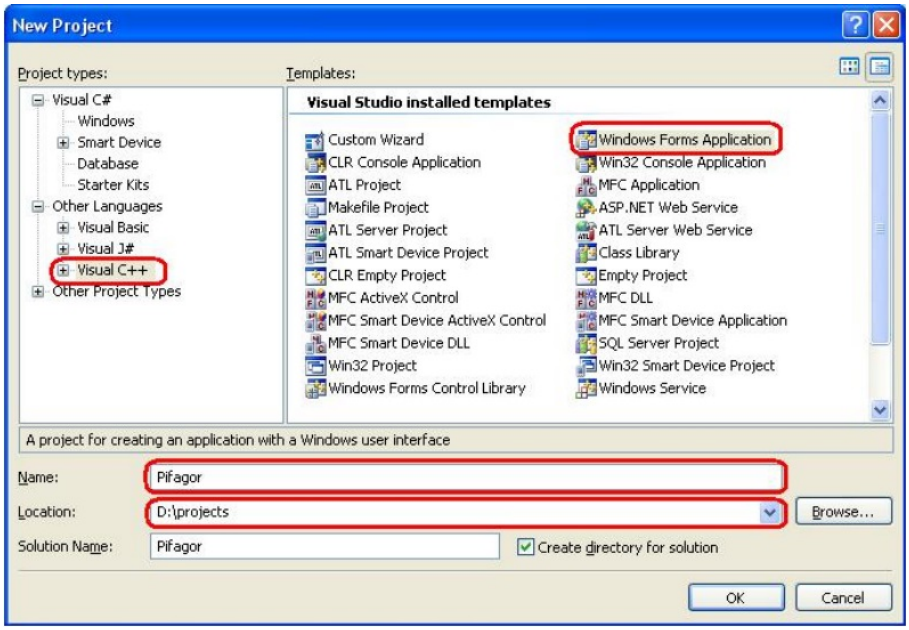


**Рис. 20.** Создание проекта

## 2.2 Оконное приложение

Рассмотрим пошагово создание оконного приложения на примере программы, вычисляющей длину гипотенузы по двум катетам.

Создадим в среде Visual Studio проект на языке «Visual C++» с типом «Windows Forms Application» (рис. 20). Укажем название проекта «Pifagor» и выберем каталог для размещения проекта.

Автоматически сформируется программа с одной пустой формой. Добавим на форму необходимые элементы. Это можно сделать при помощи специального списка элементов управления простым перетаскиванием на нашу форму (рис. 21). Нам потребуется два поля ввода (TextBox) для ввода значений катетов, одно поле ввода (TextBox) для вывода результата, кнопка (Button) для запуска процесса вычисления длины гипотенузы и три надписи (Label) для обозначений наших полей ввода.

Второй способ более сложный и требует внесения изменений в код программы. Для данной формы код добавления элементов управления будет выглядеть следующим образом:

```
...
//обявление переменных, хранящих элементы управления формы
private: System::Windows::Forms::Label^  label1;
```

```
private: System::Windows::Forms::Label^  label2;
private: System::Windows::Forms::Button^  button1;
private: System::Windows::Forms::Label^  label3;
private: System::Windows::Forms::TextBox^  textBox1;
private: System::Windows::Forms::TextBox^  textBox2;
private: System::Windows::Forms::TextBox^  textBox3;
...
//создание элементов управления
this->label1 = (gcnew System::Windows::Forms::Label());
this->label2 = (gcnew System::Windows::Forms::Label());
this->button1 = (gcnew System::Windows::Forms::Button());
this->label3 = (gcnew System::Windows::Forms::Label());
this->textBox1 = (gcnew System::Windows::Forms::TextBox());
this->textBox2 = (gcnew System::Windows::Forms::TextBox());
this->textBox3 = (gcnew System::Windows::Forms::TextBox());
...
//размещение элементов управления на форме
this->Controls->Add(this->textBox3);
this->Controls->Add(this->textBox2);
this->Controls->Add(this->textBox1);
this->Controls->Add(this->label3);
this->Controls->Add(this->button1);
this->Controls->Add(this->label2);
this->Controls->Add(this->label1);
...
```

Приведем нашу форму к более привлекательному виду. Для этого можно воспользоваться специальным окном редактирования свойств элементов формы (рис. 22).

В коде программы это выглядит следующим образом:

```
...
//
// button1
//
```

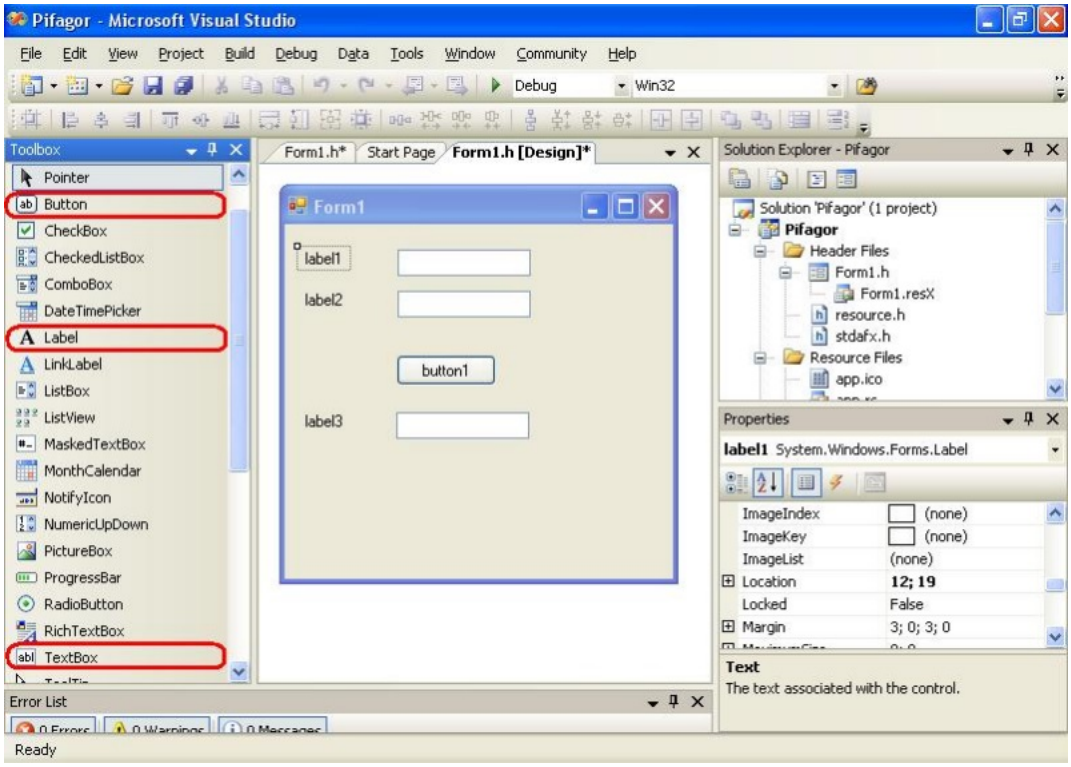


**Рис. 21.** Создание элементов формы

```
this->button1->Location = System::Drawing::Point(52, 67);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(75, 23);
this->button1->TabIndex = 2;
this->button1->Text = L"Calculate";
this->button1->UseVisualStyleBackColor = true;
...
```

Для единственной кнопки (Button) укажем функцию, которая будет выполняться при нажатии на эту кнопку. Для этого в тексте программы сформируем процедуру button1_Click и укажем ее в качестве события Click (рис. 23).

В коде программы это выглядит следующим образом:

```
...
this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
...
private: System::void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    float a=Convert::ToSingle(textBox1->Text);
```

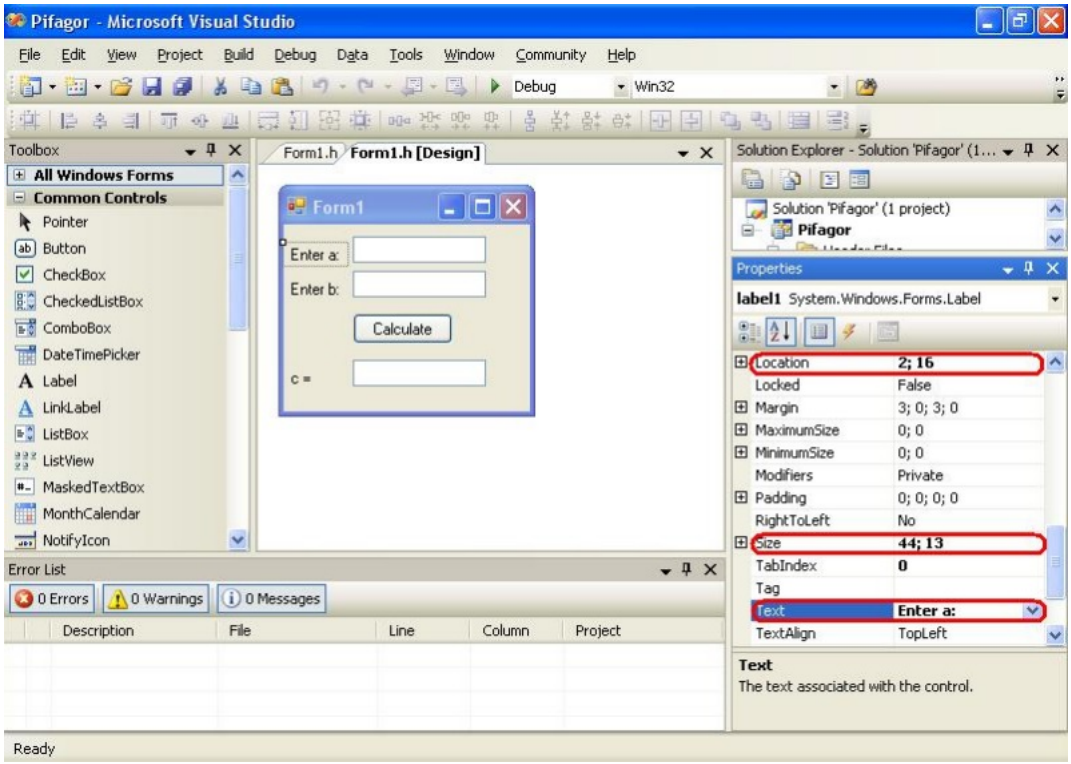


**Рис. 22.** Установка свойств элемента формы

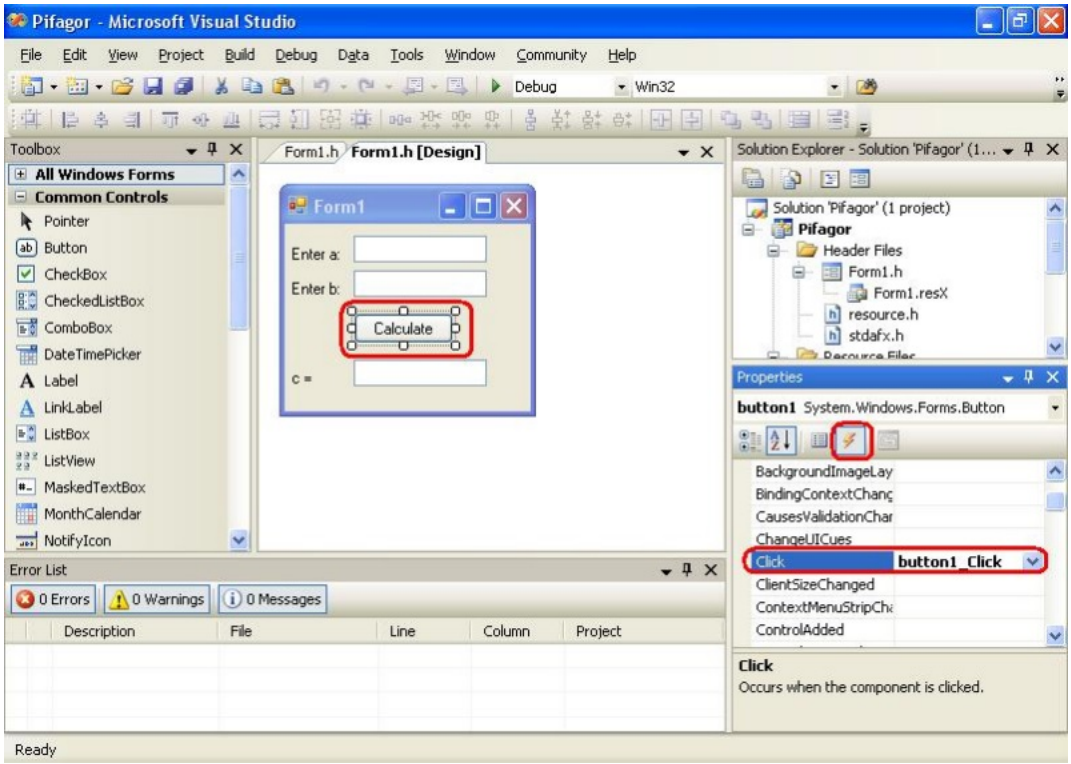


**Рис. 23.** Установка событий для элемента формы

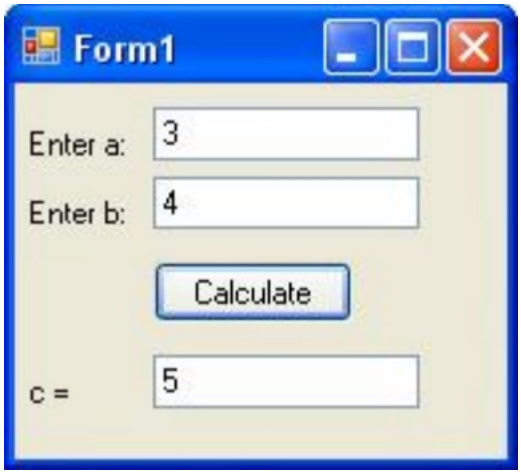


**Рис. 24.** Выполнение оконного приложения

```
    float b=Convert::ToSingle(textBox2->Text);
    float c=Math::Sqrt(a*a+b*b);
    textBox3->Text=c.ToString();
}
```

Простейшее оконное приложение готово, осталось его запустить и проверить корректность работы (рис. 24).

Отметим, что мы рассмотрели только минимально необходимые шаги для создания оконного приложения. Например, мы не учли тот факт, что пользователь может вводить некорректные данные в поля ввода, и тогда программа будет работать некорректно. Для решения таких проблем требуется правильное использование свойств элементов управления либо учитывать это в коде программы. В реальности задачи решаются более сложные и для этого могут быть использованы другие технологии создания оконных приложений.

# 3 Практические задания

Практическая часть состоит из трех задач (24 варианта в 1-й задаче и по 16 вариантов во 2-й и в 3-й) на построение графического пользовательского интерфейса. Номер и вариант для каждого студента выбирает руководитель практики. Прежде чем приступить к решению, стоит обратить внимание на то, что все задачи приводится в виде названия темы и некоторых требований. При желании требования к интерфейсу и к функциональности можно расширить. В связи с этим, каждый студент предварительно должен самостоятельно выписать требования к разрабатываемой программе, которые на его взгляд являются наиболее актуальными для конкретной задачи. Приступить к решению задачи допускается только после согласования формализованной постановки с руководителем практики.

## Задача 1

Требуется решить задачу из списка индивидуальных заданий. Необходимо разработать программу с графическим интерфейсом для решения простейшей геометрической задачи.

Требования:

- Интерфейс должен содержать поля для ввода данных, поле для вывода результата и кнопку для запуска вычисления.
- На форме должен быть чертеж (в виде рисунка PictureBox) с обозначением всех параметров, участвующих в вычислении.
- У формы должен быть соответствующий заголовок и иконка.
- При разворачивании формы на весь экран элементы управления не должны съезжать с места. Допускается растягивать элементы управления по ширине и по высоте.
- При разворачивании формы на весь экран картинка должна разворачиваться пропорционально размерам формы. При этом пропорции самой картинки должны сохраниться.
- При попытке закрытия окна пользователю должен выдаваться вопрос с текстом «Выйти из программы?»

- В полях ввода должна быть возможность ввода только корректных данных (Например, при попытке ввода текста вместо числа, должно выводиться соответствующее предупреждение).
- Поле вывода результата должно быть доступно только для чтения.

Индивидуальные задания:

1 вариант.

Площадь треугольника по трем сторонам.

2 вариант.

Площадь параллелограмма по двум смежным сторонам и углу между ними.

3 вариант.

Площадь ромба по заданному углу и заданной стороне.

4 вариант.

Площадь круга по заданному радиусу.

5 вариант.

Площадь правильного n-уголника по стороне и заданной n.

6 вариант.

Длину окружности по радиусу.

7 вариант.

Периметр параллелограмма по заданной площади, основанию и острому углу.

8 вариант.

Объем конуса по радиусу основания и высоте.

9 вариант.

Объем параллелепипеда по трем ребрам.

10 вариант.

Объем шара по заданному радиусу.

11 вариант.

Объем усеченного конуса по заданным радиусам и высоте.

12 вариант.

Объем цилиндра по заданной длине окружности основания и высоте.

13 вариант.

Высоту цилиндра по площади основания и площади боковой поверхности.

14 вариант.

Расстояние между точками на плоскости по заданным координатам.

15 вариант.

Расстояние между точками в пространстве по заданным координатам.

16 вариант.

Площадь треугольника по высоте и основанию.

17 вариант.

Площадь треугольника по заданным координатам вершин на плоскости.

18 вариант.

Площадь треугольника по заданным координатам вершин в пространстве.

19 вариант.

Площадь квадрата по заданным координатам вершин на плоскости (дополнительно проверить, что заданные координаты корректны).

20 вариант.

Объем куба по заданным координатам вершин (дополнительно проверить, что заданные координаты корректны).

21 вариант.

Объем шара по заданным координатам центра и точки поверхности.

22 вариант.

Объем цилиндра по заданным координатам центров оснований и радиусу основания.

23 вариант.

Объем конуса по заданным координатам центра основания и вершины, а также радиусу основания.

24 вариант.

Площадь параллелограмма по заданным координатам вершин на плоскости (дополнительно проверить, что заданные координаты корректны).

## Задача 2

Тема: работа с динамическими списками и динамическое создание элементов управления. Требуется рассмотреть и проанализировать проект ListProgram (код модуля формы приведен в приложении 1 на стр. 53).

1. Сделать краткое описание того, что делает программа.
2. Доработать проект, чтобы список считывался из файла.

3. Доработать проект согласно индивидуальным заданиям.

Требования:

- В предложенном проекте реализован односвязный список. Тип списка изменить согласно индивидуальному заданию.
- Программа должна быть с графическим интерфейсом. Список до и после обработки отображается в виде списка элементов управления формы («Поле ввода»).
- Для задач, типа «Добавить функционал вставки элемента в список», потребуется добавить поле ввода на форму, для того чтобы задать значение элемента. Необходимо ограничить диапазон возможных значений ввода. Например, если речь идет о целочисленном значении, то вводить можно только целое число.

Справочная информация:

- Циклическим списком называется такой список, в котором конец соединяется с началом.

Индивидуальные задания:

1 вариант.

Двухсвязный список с вещественным информационным полем.

1) Добавить функционал вставки элемента в список.
2) Отсортировать список по возрастанию.
3) Удалить все повторяющиеся элементы.

2 вариант.

Двухсвязный список с вещественным информационным полем.

1) Добавить функционал удаления элемента из списка.
2) Отсортировать список по убыванию.
3) Удалить все повторяющиеся 2 или более раза элементы.

3 вариант.

Двухсвязный список с вещественным информационным полем.

1) Добавить функционал поиска элемента в списке.
2) Отсортировать список по возрастанию.
3) Удалить элементы, повторяющиеся ровно 2 раза.

4 вариант.

Двухсвязный список с вещественным информационным полем.

1) Добавить функционал замены местами двух элементов списка.
2) Отсортировать список по убыванию.
3) Удалить все повторяющиеся элементы.

5 вариант.

Односвязный циклический список с символьным информационным полем.

1) Добавить функционал вставки элемента в список.
2) Отсортировать список по возрастанию.
3) Удалить все повторяющиеся 2 или более раза элементы.

6 вариант.

Односвязный циклический список с символьным информационным полем.

1) Добавить функционал удаления элемента из списка.
2) Отсортировать список по убыванию.
3) Удалить элементы, повторяющиеся ровно 2 раза.

7 вариант.

Односвязный циклический список с символьным информационным полем.

1) Добавить функционал поиска элемента в списке.
2) Отсортировать список по возрастанию.
3) Удалить все повторяющиеся элементы.

8 вариант.

Односвязный циклический список с символьным информационным полем.

1) Добавить функционал замены местами двух элементов списка.
2) Отсортировать список по убыванию.
3) Удалить все повторяющиеся 2 или более раза элементы.

9 вариант.

Двухсвязный циклический список с целочисленным информационным полем.

1) Добавить функционал вставки элемента в список.
2) Отсортировать список по возрастанию.
3) Удалить элементы, повторяющиеся ровно 2 раза.

10 вариант.

Двухсвязный циклический список с целочисленным информационным полем.

1) Добавить функционал удаления элемента из списка.

2) Отсортировать список по убыванию.

3) Удалить все повторяющиеся элементы.

11 вариант.

Двухсвязный циклический список с целочисленным информационным полем.

1) Добавить функционал поиска элемента в списке.

2) Отсортировать список по возрастанию.

3) Удалить все повторяющиеся 2 или более раза элементы.

12 вариант.

Двухсвязный циклический список с целочисленным информационным полем.

1) Добавить функционал замены местами двух элементов списка.

2) Отсортировать список по убыванию.

3) Удалить элементы, повторяющиеся ровно 2 раза.

13 вариант.

Односвязный список со строковым информационным полем.

1) Добавить функционал вставки элемента в список.

2) Отсортировать список по возрастанию.

3) Удалить все повторяющиеся элементы.

14 вариант.

Односвязный список со строковым информационным полем.

1) Добавить функционал удаления элемента из списка.

2) Отсортировать список по убыванию.

3) Удалить все повторяющиеся 2 или более раза элементы.

15 вариант.

Односвязный список со строковым информационным полем.

1) Добавить функционал поиска элемента в списке.

2) Отсортировать список по возрастанию.

3) Удалить элементы, повторяющиеся ровно 2 раза.

16 вариант.

Односвязный список со строковым информационным полем.

1) Добавить функционал замены местами двух элементов списка.

2) Отсортировать список по убыванию.

3) Удалить все повторяющиеся элементы.

## Задача 3

Требуется разработать программу с графическим интерфейсом, обрабатывающую согласно индивидуальному заданию прямоугольные матрицы. Исходные матрицы должны считываться из файла. Должна быть возможность просмотра матрицы на форме и запись результата в файл. Формат входных и выходных файлов произвольный. Если в задаче не говорится о типе элементов матрицы, то разрешено использовать целочисленный тип.

1 вариант.

Найти максимальные элементы в каждой строке, записать максимальные элементы в новом столбце справа.

2 вариант.

Найти максимальные элементы в каждом столбце, записать максимальные элементы в новой строке снизу.

3 вариант.

Найти минимальные элементы в каждой строке, записать минимальные элементы в новом столбце слева.

4 вариант.

Найти минимальные элементы в каждом столбце, записать минимальные элементы в новой строке сверху.

5 вариант.

Вычислить сумму двух матриц.

6 вариант.

Вычислить разность двух матриц.

7 вариант.

Вычислить произведение двух матриц.

8 вариант.

Вычислить сумму заданного (определяется параметром на форме) числа матриц.

9 вариант.

Вычислить транспонированную матрицу.

10 вариант.

Найти минимальный элемент в матрице, добавить новую строку снизу по следующему правилу: если есть минимальный элемент в соответствующем

столбце, то записать его; если нет минимального элемента, то записать 0.

11 вариант.

Найти максимальный элемент в матрице добавить новый столбец справа по следующему правилу: если есть минимальный элемент в соответствующей строке, то записать его; если нет минимального элемента, то записать 0.

12 вариант.

Отсортировать каждую строку матрицы по убыванию/возрастанию.

13 вариант.

Найти определитель матрицы.

14 вариант.

Матрица, состоящая из элементов строкового типа. Добавить новую строку снизу, записать в нее значение максимальной длины элемента из соответствующего столбца. Например, в столбце находятся значения «aa», «abb», «ab», «aa», тогда в соответствующей ячейке записывается «3», как длина строки «abb».

15 вариант.

Матрица, состоящая из элементов комплексного типа. Отсортировать матрицу в порядке возрастания модулей элементов.

16 вариант.

Матрица, каждым элементом которой является целочисленный массив. Разработать программу для удобного просмотра элементов матрицы.

# 4 План отчета

При составлении отчета по проделанной работе рекомендуется придерживаться следующего плана:

1. Титульный лист.
2. Оглавление.
3. Введение (не более 1 страницы).
4. Постановка задачи (не более 1 страницы).
5. Теоретический обзор (1-3 страницы).
6. Описание метода решения (2-6 страниц).

7. Заключение (не более 1 страницы).
8. Список литературы.
9. Приложения (0-10 страниц).

## 4.1 Титульный лист

На титульном листе указывается учебное заведение, в котором выполнена работа; вид работы (в данном случае это «Учебная практика»); название темы; кем выполнена работа (указывается номер группы, курс, фамилия, имя и отчество); кто является научным руководителем; город и год составления отчета. Образец титульного листа можно посмотреть в приложении (прил. 3 стр. 58).

## 4.2 Оглавление

В оглавлении указываются названия всех разделов с указанием номеров страниц.

## 4.3 Введение

Введение — одна из главных частей отчета по любой исследовательской работе. Из него должно быть понятно краткое содержание всей работы. Несмотря на то, что текст работы начинается с введения, рекомендуется его составить только после того, как будут готовы все остальные разделы. Это связано с тем, что некоторые сведения можно получить только после того, как будет готова вся основная часть работы. Введение отчета по учебной практике обязательно должно содержать следующую информацию:

1. **Актуальность работы.** Под актуальностью понимается важность, значимость, востребованность рассматриваемого вопроса в данный момент времени.

*Пример.*

Тема данной работы на сегодняшний день **актуальна** потому что, оконные приложения в настоящее время весьма популярны. Сложно представить современного пользователя, который предпочел бы консольное управление вместо

графического интерфейса. При разработке программных продуктов немалое внимание удаляется именно удобству его использования, поэтому умение правильно создавать графические интерфейсы очень востребовано. . .

2. **Цель работы.** Формулируется исходя из основной проблемы. Так как практика учебная, то и основную цель можно рассматривать как получение каких-либо знаний, умений и навыков.

*Пример.*

**Цель работы:** Научиться разрабатывать приложения с графическим интерфейсом пользователя в среде программирования Visual Studio.

3. **Задачи.** Перечисляются те задачи, при решении которых планируется достижение поставленной цели.

*Пример.*

Для достижения поставленной цели необходимо решить следующие **задачи:**

1) Изучение литературы.
2) Формулировка требований.
3) Разработка программы.
4) Тестирование программы.
5) Составление отчета по работе.

## 4.4 Постановка задачи

В постановке задачи приводится формальное описание поставленной задачи. Исходная постановка может быть не достаточно формализована либо описана не полностью, поэтому перед началом решения необходимо выписать все требования в виде списка и согласовать с преподавателем. В данный раздел включается весь список с подробными пояснениями при необходимости.

*Пример:*

Требуется разработать простейшее приложение с графическим пользовательским интерфейсом для вычисления длины гипотенузы по заданным катетам.

Требования к интерфейсу:

- Интерфейс должен содержать поля для ввода длин катетов, поле для вы-

вода длины гипотенузы и кнопку для запуска вычисления.

- На форме должна быть возможность просмотра чертежа треугольника с обозначенными катетами и гипотенузой.
- У формы должен быть соответствующий заголовок и иконка (в виде маленького треугольника).
- При разворачивании формы на весь экран элементы управления не должны съезжать с места. Допускается растягивать элементы управления по ширине и по высоте.
- При разворачивании формы на весь экран картинка должна разворачиваться пропорционально размерам формы. При этом пропорции самой картинки должны сохраниться.
- При попытке закрытия окна пользователю должен выдаваться вопрос с текстом «Выйти из программы?»
- В полях ввода должна быть возможность ввода только целых и вещественных, представленных десятичной дробью.
- Поле вывода результата должно быть доступно только для чтения.

Требование к функциональности:

- Значения длин катетов должны задаваться в вещественной форме с точностью 4 знака после запятой.
- Результат должен выводиться с точностью 4 знака после запятой.
- Программа должна содержать справочную информацию с кратким описанием ее функциональности.

## 4.5 Теоретический обзор

В теоретическом обзоре приводится краткое описание изученного материала в различных источниках. Главное, на что уделяется внимание, это описание конструкций языка программирования, которые потребовалось использовать при решении поставленной задачи. Необходимо описать все используемые классы. Можно руководствоваться примером в разделе 1.

*Пример.*

В основе графического пользовательского интерфейса лежит понятие формы (Form) [2]. Форма представлена в виде специального класса с заранее определенными свойствами, методами и событиями. К ним относятся, например, . . .

Для ввода/вывода текстовых или числовых данных принято использовать объект класса TextBox . . .

Для выполнения какого-либо действия используется объект класса Button . . .

Для вывода изображения используется объект класса PictureBox . . .

## 4.6 Описание метода решения задачи

В разделе приводится пошагово ход решения задачи. Приводятся описания всех задач, перечисленных во введении и не решенных в предыдущих разделал. Можно руководствоваться примером в разделе 2.2.

*Пример.*

**1 Разработка программы.**

Решение основной задачи, а именно разработка программы, выполняется в среде программирования Visual Studio с использованием проекта на языке «Visual C++» [1, 4, 5] с типом «Windows Forms Application». Проект данного типа уже по умолчанию содержит одну пустую форму, требуется добавить на нее все необходимые элементы управления. Для этого . . .

Ввод и вывод данных выполняется при помощи объектов типа TextBox. Свойства этих объектов содержат . . .

Основное действие выполняется при нажатии на кнопку «Calculate». Описание кнопки:

```
this->button1->Location = System::Drawing::Point(52, 67);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(75, 23);
this->button1->TabIndex = 2;
this->button1->Text = L"Calculate";
this->button1->UseVisualStyleBackColor = true;
```

Действие при нажатии кнопки:

```
this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
```

```
private: System::void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    float a=Convert::ToSingle(textBox1->Text);
    float b=Convert::ToSingle(textBox2->Text);
    float c=Math::Sqrt(a*a+b*b);
    textBox3->Text=c.ToString();
}
```

Для ограничения возможности ввода пользователем некорректных данных используется . . .

**2 Тестирование программы.**

Для проверки правильности работы программы выполнено тестирование на входных данных, приведенных в таблице 1. Первые две колонки таблицы содержат значения длин катетов, третья — правильное значение длины гипотенузы, округленное до 4 знаков после запятой, четвертая — результат вычисления в разработанной программе.

**Таблица 1.** Тестовые данные

| Катет a | Катет b | Ожидаемое значение c | Результат |
|---|---|---|---|
| 3 | 4 | 5 | 5.0 |
| 3.0 | 4.0 | 5.0 | 5.0 |
| 3 | 4.0 | 5.0 | 5.0 |
| 1 | 1 | 1.4142 | 1.4142 |
| 1..1 | 1 | Введены неверные данные | Введены неверные данные |
| . . . | | | |

При всех тестовых входных данных получен результат, совпадающий (с точностью до не значащих нулей) с ожидаемым результатом. Отсюда следует, что . . .

## 4.7 Заключение

В заключении приводятся основные выводы проделанной работы. Для учебной практики основной целью является научиться чему-либо, поэтому к основным результатам можно отнести достижение этой цели. Кроме того, необходимо выписать проблемы, с которыми пришлось встретиться в ходе выполнения

поставленной работы, как решали эти проблемы, и как оценивается работа самим студентом.

*Пример.*

В ходе учебной практики были решены все поставленные задачи, а именно . . .

Разработанная программа протестирована, ошибок не выявлено . . .

Итоговая цель достигнута. В ходе выполнения задач были получены знания в области программирования с использованием классов Windows Forms, умение описывать предметную область с использованием классов и развиты навыки работы в среде Visual Studio. Полученные знания, умения и навыки будут использованы в дальнейшем при подготовке курсовых работ, прохождении последующих практик и выполнении итоговой выпускной квалификационной работы.

## 4.8 Список литературы

В списке литературы выписываются все источники, которые были использованы в работе. На каждый источник в тексте отчета должна быть хотя бы одна ссылка, вида [3]. Пример оформления списка литературы можно посмотреть на странице 47.

## 4.9 Приложения

В приложении приводятся все материалы (тексты программ, схемы, скриншоты, инструкции и т.д.), которые не вошли в основной текст отчета, но необходимы для пояснения хода решения задачи. Пример оформления приложений можно посмотреть на странице 48.

# Список литературы


1. Пахомов Б. C/C++ и MS Visual C++ 2012 для начинающих. — Сбп.: БХВ-Петербург, 2015, — 518 с.

2. Пространство имен System.Windows.Forms [Электронный ресурс] / Microsoft — Электрон. дан. — Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.windows.forms(v=vs.110).aspx, свободный. — Загл. с экрана. Дата обращения: 30.03.2020

3. Сапаров А.Ю. Разработка программных продуктов с графическим интерфейсом на языке программирования C++: методическое пособие. / — Ижевск: «Удмуртский университет», 2016. — 36 с.

4. Страуструп Б. Программирование: принципы и практика использования C++. Пер. с англ. — М.: Вильямс, 2011. — 1248 с.

5. Хортон А. Visual C++ 2010. Полный курс. Пер. с англ. — М.: Вильямс, 2011. — 1216 с.

# Приложения

## 1 Код модуля формы из примера оконного приложения

```
#pragma once

namespace Pifagor2 {

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Summary for Form1
///
/// WARNING: If you change the name of this class, you will need to change the
/// 'Resource File Name' property for the managed resource compiler tool
/// associated with all .resx files this class depends on. Otherwise,
/// the designers will not be able to interact properly with localized
/// resources associated with this form.
/// </summary>
public ref class Form1 : public System::Windows::Forms::Form
{
    public:
    Form1(void)
    {
        InitializeComponent();
        //
        //TODO: Add the constructor code here
        //
    }
```

```
    protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~ Form1()
    {
        if (components)
        {
            delete components;
        }
    }
    private: System::Windows::Forms::Label^  label1;
    private: System::Windows::Forms::Label^  label2;
    private: System::Windows::Forms::Button^  button1;
    private: System::Windows::Forms::Label^  label3;
    private: System::Windows::Forms::TextBox^  textBox1;
    private: System::Windows::Forms::TextBox^  textBox2;
    private: System::Windows::Forms::TextBox^  textBox3;

    private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^  components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    void InitializeComponent(void)
    {
```

```
this->label1 = (gcnew System::Windows::Forms::Label());
this->label2 = (gcnew System::Windows::Forms::Label());
this->button1 = (gcnew System::Windows::Forms::Button());
this->label3 = (gcnew System::Windows::Forms::Label());
this->textBox1 = (gcnew System::Windows::Forms::TextBox());
this->textBox2 = (gcnew System::Windows::Forms::TextBox());
this->textBox3 = (gcnew System::Windows::Forms::TextBox());
this->SuspendLayout();
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(2, 16);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(44, 13);
this->label1->TabIndex = 0;
this->label1->Text = L"Enter a:";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(2, 42);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(44, 13);
this->label2->TabIndex = 1;
this->label2->Text = L"Enter b:";
//
// button1
//
this->button1->Location = System::Drawing::Point(52, 67);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(75, 23);
this->button1->TabIndex = 2;
```

```
this->button1->Text = L"Calculate";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click +=
                gcnew System::EventHandler(this, &Form1::button1_Click);
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(2, 109);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(22, 13);
this->label3->TabIndex = 3;
this->label3->Text = L"c =";
//
// textBox1
//
this->textBox1->Location = System::Drawing::Point(52, 9);
this->textBox1->Name = L"textBox1";
this->textBox1->Size = System::Drawing::Size(100, 20);
this->textBox1->TabIndex = 4;
//
// textBox2
//
this->textBox2->Location = System::Drawing::Point(52, 35);
this->textBox2->Name = L"textBox2";
this->textBox2->Size = System::Drawing::Size(100, 20);
this->textBox2->TabIndex = 5;
//
// textBox3
//
this->textBox3->Location = System::Drawing::Point(52, 102);
this->textBox3->Name = L"textBox3";
this->textBox3->Size = System::Drawing::Size(100, 20);
```

```
        this->textBox3->TabIndex = 6;
        //
        // Form1
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(185, 141);
        this->Controls->Add(this->textBox3);
        this->Controls->Add(this->textBox2);
        this->Controls->Add(this->textBox1);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->button1);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
        this->Name = L"Form1";
        this->Text = L"Form1";
        this->ResumeLayout(false);
        this->PerformLayout();

    }
#pragma endregion
    private: System::void button1_Click(System::Object^  sender,
                 System::EventArgs^  e)
    {
        float a=Convert::ToSingle(textBox1->Text);
        float b=Convert::ToSingle(textBox2->Text);
        float c=Math::Sqrt(a*a+b*b);
        textBox3->Text=c.ToString();
    }
};
}
```

## 2 Текст программы ListProgram

```
#pragma once

namespace ListProgram {

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Summary for Form1
///
/// WARNING: If you change the name of this class, you will need to change the
/// 'Resource File Name' property for the managed resource compiler tool
/// associated with all .resx files this class depends on. Otherwise,
/// the designers will not be able to interact properly with localized
/// resources associated with this form.
/// </summary>
public ref struct List
{
    String^  info;
    List^  next;
    System::Windows::Forms::TextBox^  textBox;
};

public ref class Form1 : public System::Windows::Forms::Form
{
    public:
    Form1(void)
    {
```

```
        InitializeComponent();
        //
        //TODO: Add the constructor code here
        //
    }

    protected:
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    ~ Form1()
    {
        if (components)
        {
            delete components;
        }
    }
    private: System::Windows::Forms::MenuStrip^  menuStrip1;
    protected:
    private: System::Windows::Forms::ToolStripMenuItem^  toolStripMenuItem1;
    private: System::Windows::Forms::ToolStripMenuItem^ startToolStripMenuItem;
    private: int curTop;

    private:
    /// <summary>
    /// Required designer variable.
    /// </summary>
    System::ComponentModel::Container ^  components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
```

```
/// </summary>
void InitializeComponent(void)
{
    this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
    this->toolStripMenuItem1 =
                        (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->startToolStripMenuItem =
                        (gcnew System::Windows::Forms::ToolStripMenuItem());
    this->menuStrip1->SuspendLayout();
    this->SuspendLayout();
    //
    // menuStrip1
    //
    this->menuStrip1->Items->AddRange(
              gcnew cli::array< System::Windows::Forms::ToolStripItem^  >(1)
                                                  {this->toolStripMenuItem1});
    this->menuStrip1->Location = System::Drawing::Point(0, 0);
    this->menuStrip1->Name = L"menuStrip1";
    this->menuStrip1->Size = System::Drawing::Size(292, 24);
    this->menuStrip1->TabIndex = 0;
    this->menuStrip1->Text = L"menuStrip1";
    //
    // toolStripMenuItem1
    //
    this->toolStripMenuItem1->DropDownItems->AddRange(
              gcnew cli::array<System::Windows::Forms::ToolStripItem^  >(1)
                                                  {this->startToolStripMenuItem});
    this->toolStripMenuItem1->Name = L"toolStripMenuItem1";
    this->toolStripMenuItem1->Size = System::Drawing::Size(25, 20);
    this->toolStripMenuItem1->Text = L"1";
    //
    // startToolStripMenuItem
    //
```

```
        this->startToolStripMenuItem->Name = L"startToolStripMenuItem";
        this->startToolStripMenuItem->Size = System::Drawing::Size(152, 22);
        this->startToolStripMenuItem->Text = L"Start";
        this->startToolStripMenuItem->Click += gcnew System::EventHandler
                                        (this, &Form1::startToolStripMenuItem_Click);
        //
        // Form1
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
        this->ClientSize = System::Drawing::Size(292, 266);
        this->Controls->Add(this->menuStrip1);
        this->MainMenuStrip = this->menuStrip1;
        this->Name = L"Form1";
        this->Text = L"Form1";
        this->menuStrip1->ResumeLayout(false);
        this->menuStrip1->PerformLayout();
        this->ResumeLayout(false);
        this->PerformLayout();
    }
#pragma endregion
    private: System::void startToolStripMenuItem_Click
                                    (System::Object^  sender, System::EventArgs^  e) {
        List^  head=gcnew List();
        head->info="First";
        head->next=gcnew List();
        head->next->info="Second";
        ShowTextBoxs(head);
        //head->textBox->ReadOnly=false;
    }

    private: void ShowTextBoxs(List^  head){
        curTop=29;
```

```
        List^  cur=head;
        while(cur!=nullptr)
        {
            cur->textBox=AddTextBox(cur->info, 5, curTop, 250, 20);
            cur=cur->next;
            curTop+=25;
        }
    }

    private: System::Windows::Forms::TextBox^  AddTextBox
                                (String^  info, int left, int top, int width, int height){
        System::Windows::Forms::TextBox^  pTextBox=
                                    (gcnew System::Windows::Forms::TextBox());
        pTextBox->Text=info;
        pTextBox->Location = System::Drawing::Point(left, top);
        pTextBox->Size = System::Drawing::Size(width, height);
        pTextBox->TabIndex = 1;
        pTextBox->ReadOnly=true;
        this->Controls->Add(pTextBox);
        return pTextBox;
    }
};
}
```

# 3 Образец оформления титульного листа отчета

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра теоретических основ информатики

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ
Срок прохождения практики: 13.07.2020 — 26.07.2020

**«Разработка приложения с графическим интерфейсом для вычисления сторон треуголника»**

Выполнил(а) студент(ка):
Иванов Иван Иванович
*(ФИО студента(ки))*

Направления подготовки/специальности
Прикладная информатика

группы ОАБ-09.03.03.д-11
*(наименование группы)*

Руководитель практики от кафедры:
Петров П.П., доцент кафедры ТОИ, к.т.н.
*(ФИО, должность, ученое звание, ученая степень)*

Итоговая оценка по учебной практике_____
*(оценка, подпись руководителя)*

Ижевск 2020 г.

# Содержание

**Учебное издание**

Сапаров Алексей Юрьевич

**Разработка Windows Forms приложений**
**на языке программирования C++**

Учебно-методическое пособие

Отпечатано с оригинал-макета заказчика

Подписано в печать 28.12.20. Формат 60x84 $^{1}/_{16}$.
Тираж 50 экз. Заказ № 2255.

Типография
Издательского центра «Удмуртский университет»
426034, Ижевск, ул. Университетская, 1, корп. 2.
Тел. 68-57-18