РОСЖЕЛДОР
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)

Небаба А.Н.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

МДК.01.02 «РАЗРАБОТКА ИНТЕРФЕЙСОВ ПОЛЬЗОВАТЕЛЯ»

для специальности
09.02.09 Веб-разработка

Ростов-на-Дону
2025

Оглавление

Лабораторная работа 1. Структура проекта Windows Forms приложения в IDE MS Visual Studio. Компонент FORM

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Познакомиться со структурой проекта приложения Windows Forms (.NET Framework).

2. Разработать графический интерфейс и программный код приложения **Lab_1** в соответствии с представленной методикой.

3. Детально изучить основные свойства и события компонента **Form** (см. Приложение 1. Работа с формами).

Методика выполнения

Введение

C# (си шарп) – объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework.

C# относится к семейству языков с C-подобным синтаксисом. Переняв многое от своих предшественников – языков C++, Pascal, Модула, Smalltalk и, в особенности, Java – C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование.

Common Language Runtime (англ. CLR – общеязыковая исполняющая среда) – исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования (C#, Managed C++, Visual Basic .NET, F# и прочие).

CLR является одним из основных компонентов пакета Microsoft .NET Framework.

Создание проекта C# в среде Visual Studio

Хотя при создании нового проекта в среде MS Visual Studio предлагается довольно большой список типов проектов, но на самом деле есть всего три основные разновидности приложений – Windows-приложение, консольное приложение и библиотека классов. Все остальное – это их различные вариации за счет использования тех или иных шаблонов или мастеров, обеспечивающих автоматическое выполнение каких-то начальных действий.

Поле запуска Visual Studio[1] на экране отображается начальная страница среды (Рис. 1-1).

После выбора команды **Создание проекта** на экране отображается диалоговое окно с аналогичным названием (Рис. 1-2).

После установки фильтров как на Рис. 1-2 следует выбрать шаблон проекта **Приложение Windows Forms (.NET Framework)** и нажать кнопку **Далее**.

[1] В пособии рассматривается IDE MS Visual Studio 2022

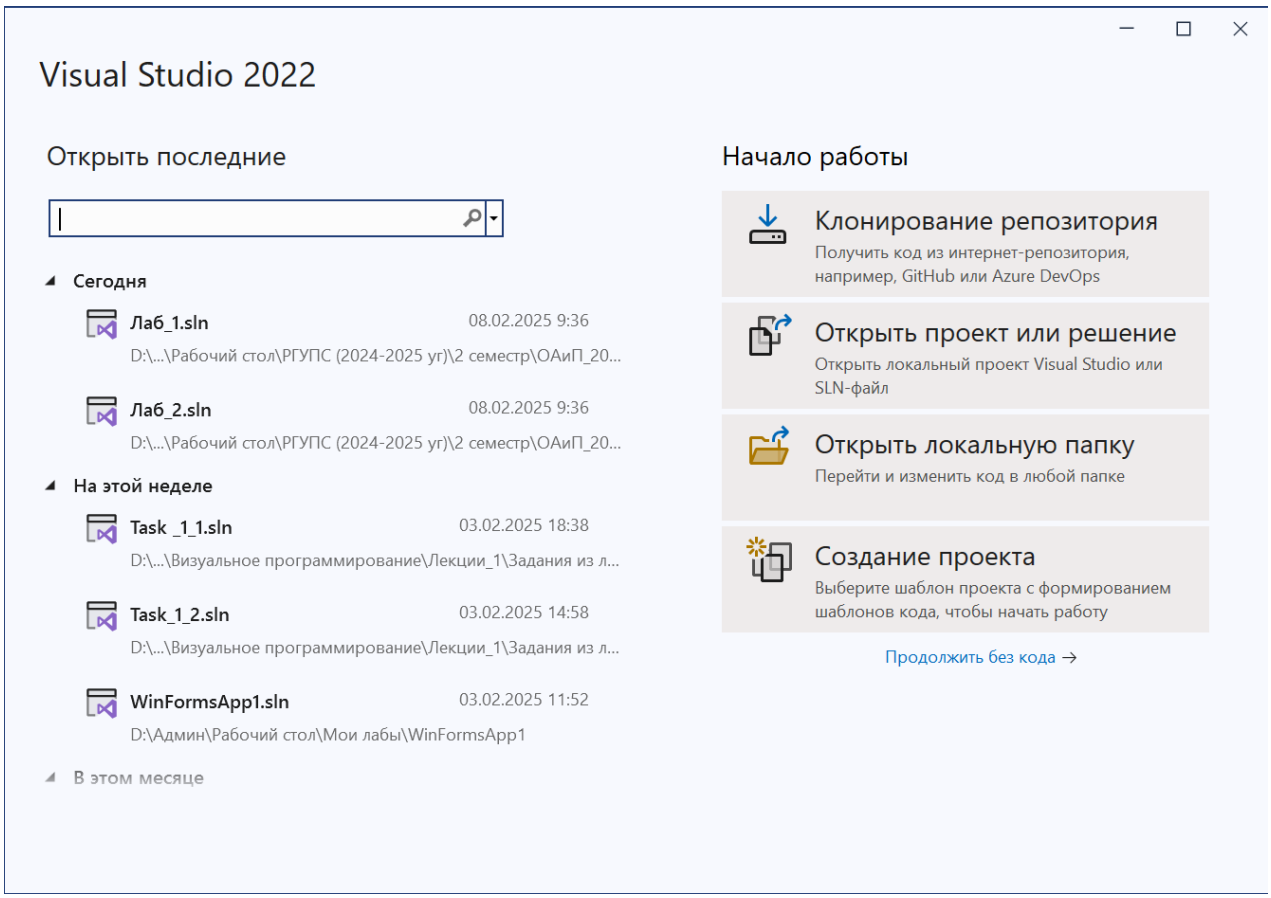

Рис. 1-1 – Начальная страница MS VS 2022

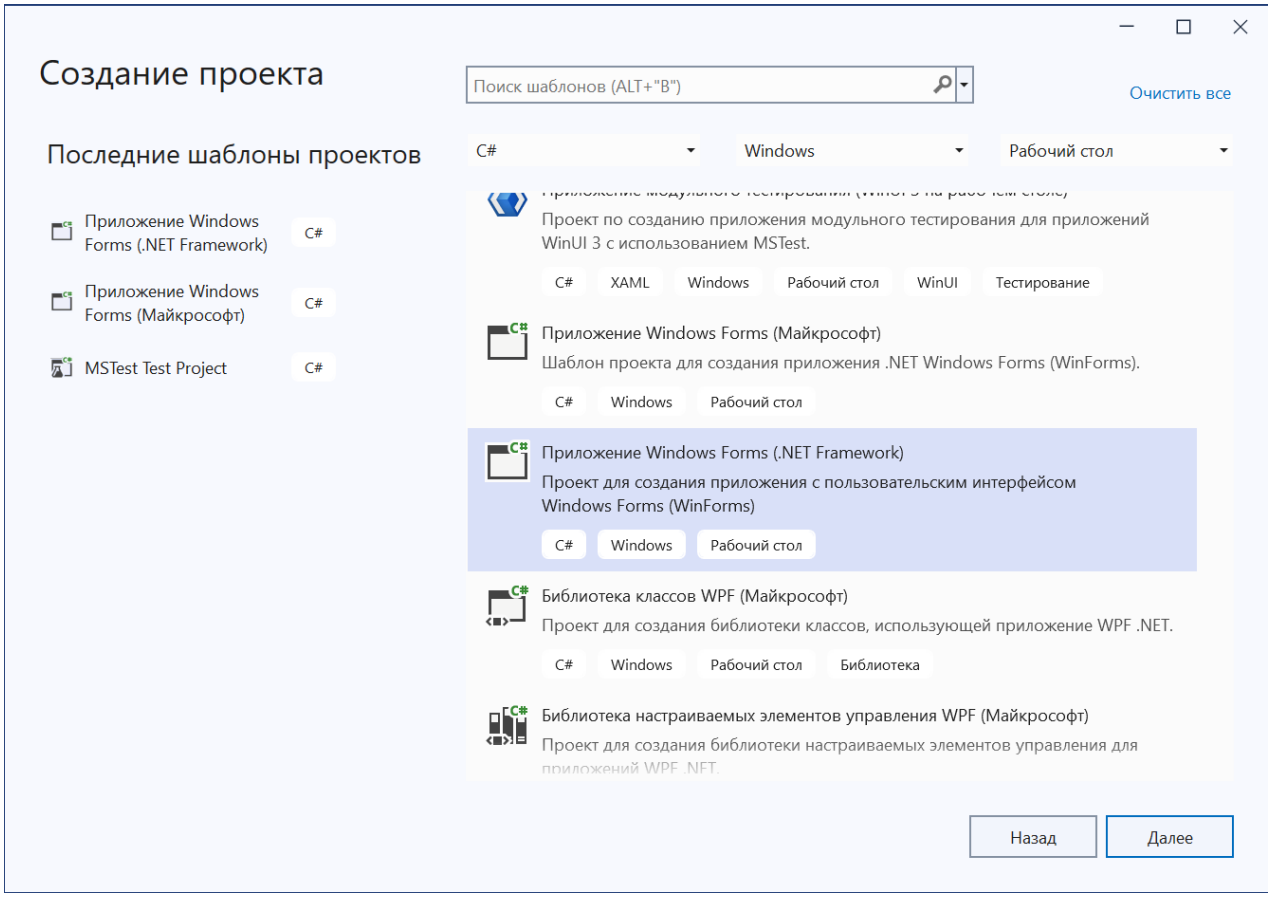

Рис. 1-2 – Окно **Создание проекта**

В окне **Настроить новый проект** (см. Рис. 1-3) нужно задать имя проекта, например, **Lab_1**, расположение проекта, платформу и нажать на кнопку **Создать.**

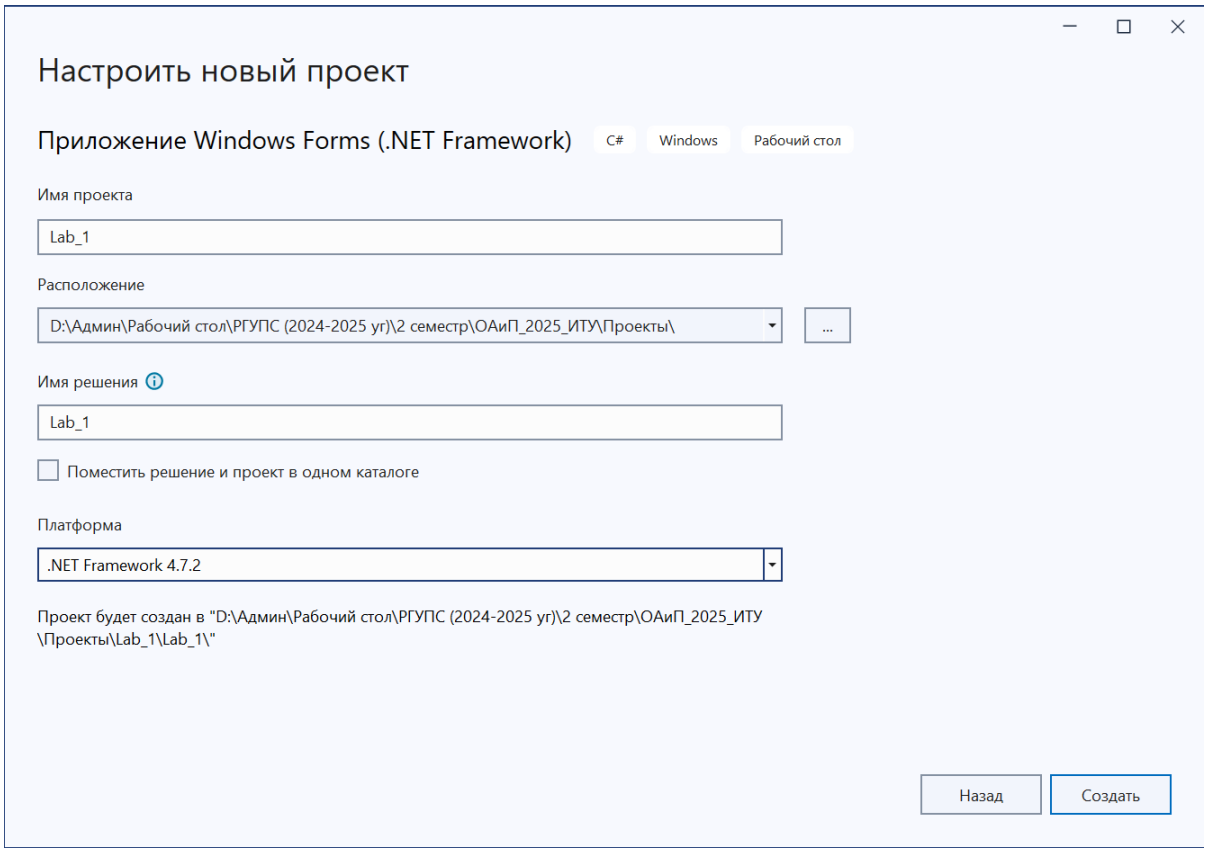

Рис. 1-3 – Настройка параметров проекта

После этого Visual Studio откроет наш проект с созданными по умолчанию файлами (Рис. 1-4).

Результат работы в среде Visual Studio – совокупность файлов. При этом уже существующие в этом каталоге файлы могут быть заменены вновь созданными. Это может привести к порче ранее сохранённых файлов из других проектов. *Поэтому следует каждый проект сохранять в отдельном предварительно созданном каталоге* (например, рассматриваемую лабораторную работу рекомендуется сохранить в каталоге **Lab_1**).

Файлы проекта Visual Studio

- ***.sln** – Microsoft Visual Studio Solution File. В этом файле содержится информация из чего состоит приложение, как называется и какой файл проект относится к нему. При открытии ранее созданного проекта следует запустить именно этот файл.
- ***.cs** – C Sharp. В этом файле храниться исходный код программы. Имя по умолчанию – Program.cs.
- ***.csproj** – XML-файл, содержащий все необходимые характеристики проекта. В частности, информация о платформе, имя корневого пространства имен.
- папка **bin** – содержит откомпилированные файлы.

• ***.pdb** – файл содержит отладочную информацию о приложении: локальные переменные, функции, разметка исходного кода и т.д.

• папка **obj** содержит временные и промежуточные файлы.

• **Properties/AssemblyInfo.cs** – содержит в себе информацию о сборке.

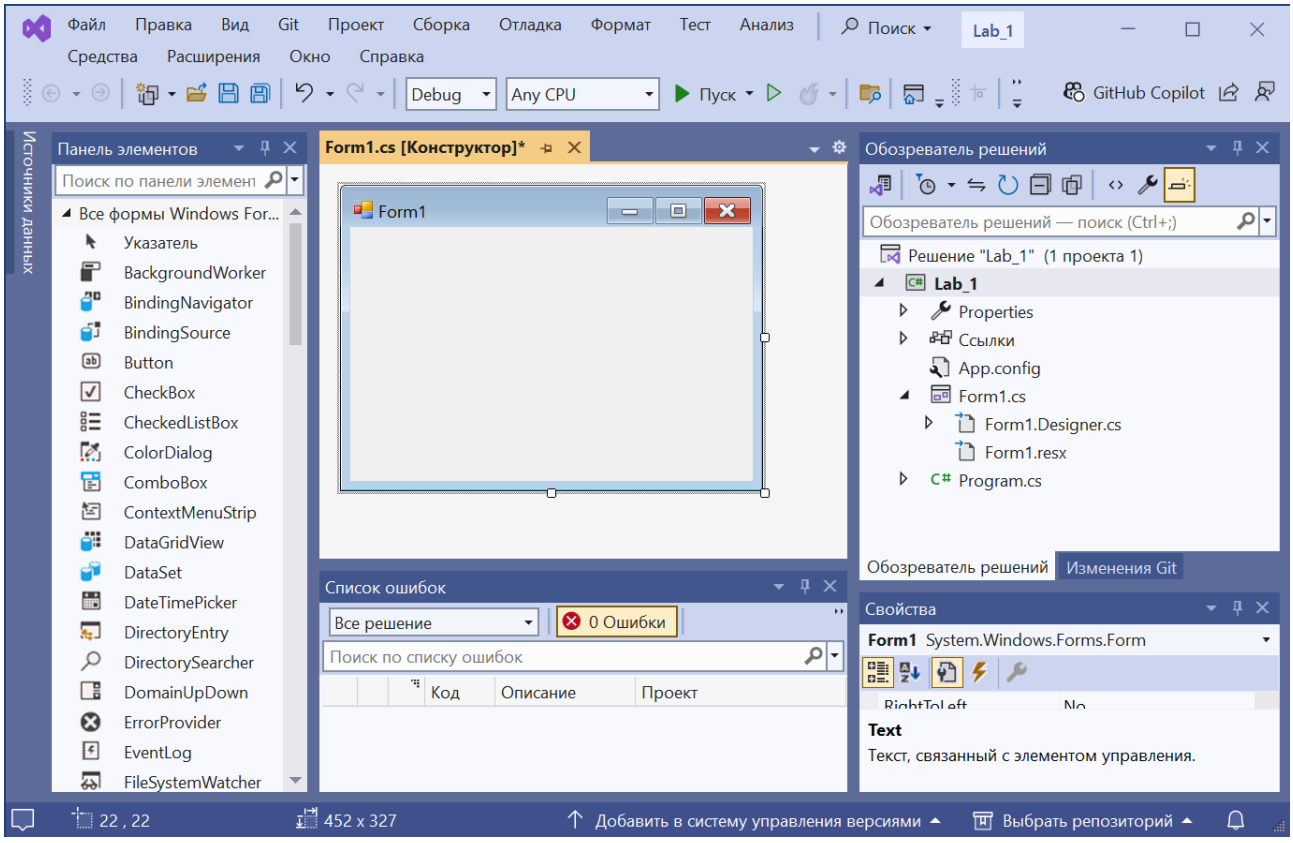

Рис. 1-4 – Окно нашего проекта в MS Visual Studio 2022

Задание 1

1. После создания проекта **Lab_1** необходимо проверить наличие в его окне панелей: **Панель элементов**, **Обозреватель решений**, **Свойства** и **Список ошибок** (см. Рис. 1-4). В случае отсутствия некоторых из них отобразить их с помощью меню **Вид** и разместить в соответствии с Рис. 1-4 (желательно).

2. Приступим к созданию интерфейса нашего приложения. В окне **Панель элементов** найдите компонент с надписью **Button** и щелкните на нём мышью. Затем щелкните мышью в окне **Form1**. Появится прямоугольник с надписью «*button1*». Вновь щелкните на значке **Button** и щелкните на **Form1** ниже от объекта **button1** – появится прямоугольник с надписью «*button2*» (Рис. 1-5).

3. Размер формы, расположение и размер кнопок настройте по образцу (Рис. 1-5).

4. Теперь щелкните на **button1,** чтобы выбрать ее. Окно свойств отобразит свойства объекта **button1**. В поле свойства с именем **Text** значение «*button1*» замените на «*Нажми меня*!» (см. Рис. 1-6). Если вы нажмете клавишу **Enter** или щелкнете в другой строке, то увидите, что надпись на первой кнопке формы изменилась на «*Нажми меня*!» (см. Рис. 1-7).

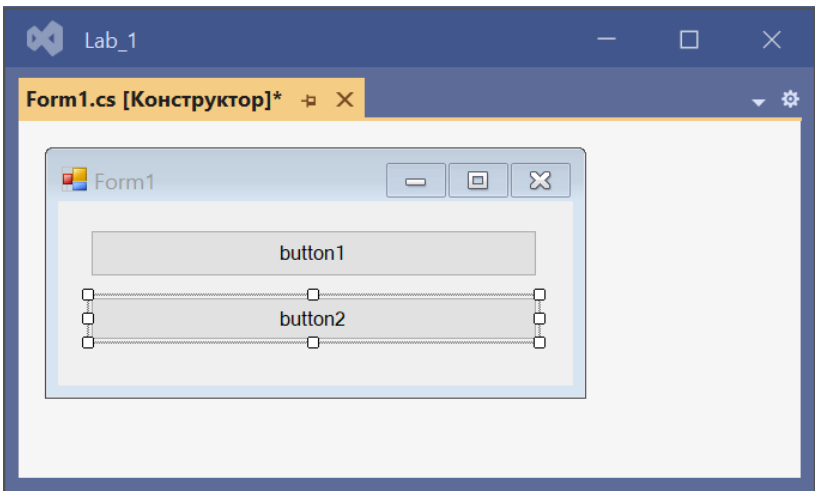

Рис. 1-5 – Конструктор формы приложения

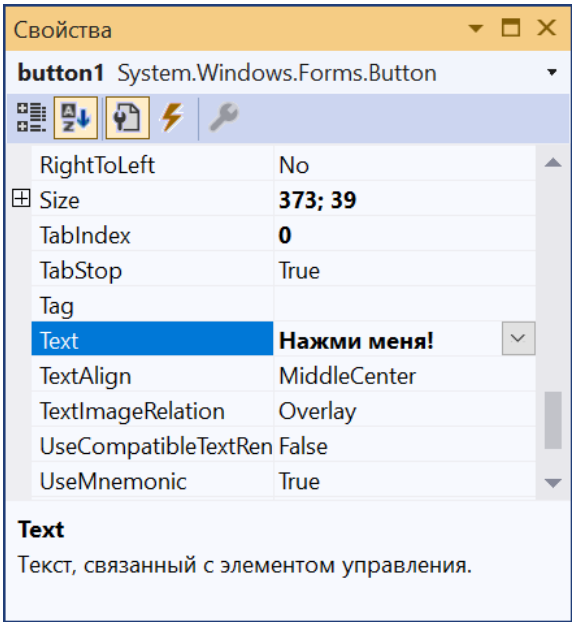

Рис. 1-6 – Настройка свойств кнопки **button1**

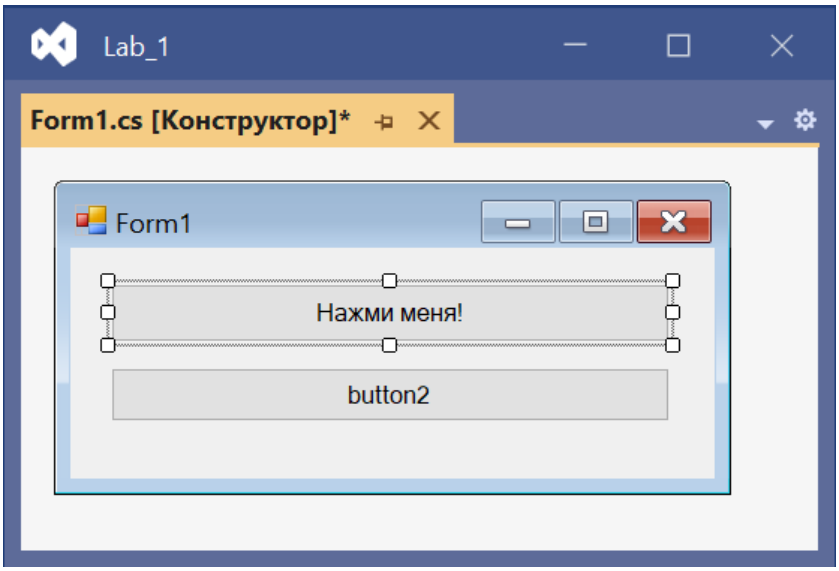

Рис. 1-7 – Конструктор формы приложения

5. В окне **Свойства** имеется также вкладка **События**, в которой отражены различные события (Рис. 1-8), на которые может реагировать кнопка. Среди них **Click**, **MouseCaptureChanged**, **MouseClick** и др. В настоящий момент нас интересует событие **Click**, которое возникает, при щелчке основной кнопкой мыши по элементу интерфейса.

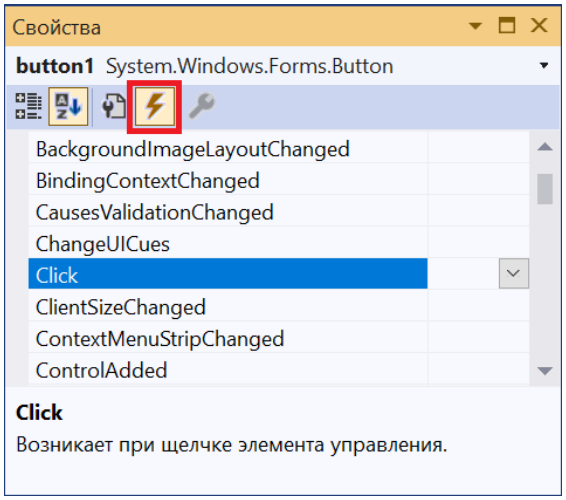

Рис. 1-8 – Вкладка **События** в окне свойств

Двойной щелчок правее выбранного события откроет редактор программного кода в окне **Form1.cs** и установит курсор в теле обработчика события **Click** объекта **button1** (Рис. 1-9).

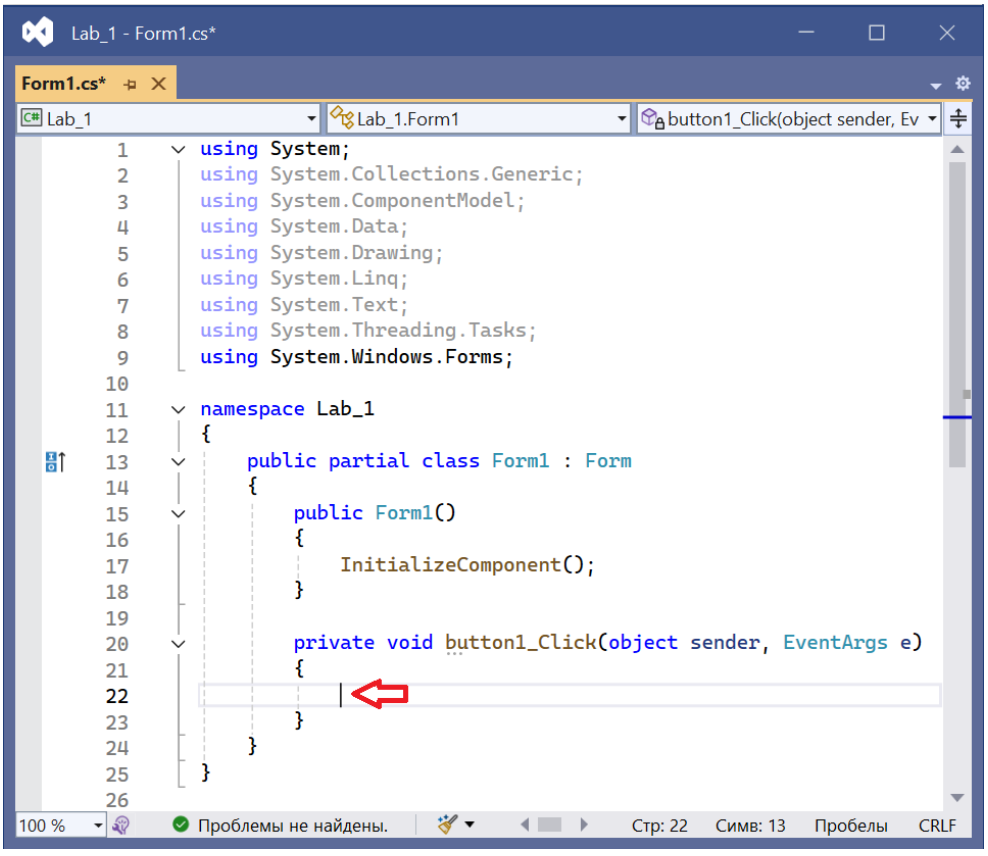

Рис. 1-9 – Редактор программного кода

Сделайте вставку следующего программного кода (выделен зелёным цветом) в тело обработчика события:

```
private void button1_Click(object sender, EventArgs e)
    {
       button1.Text = "Нажми меня ещё раз!";
    }
```

6. Отредактируйте свойства кнопки **button2**, изменив свойство **Text** на «*Выход*» вместо «*button2*».

7. Добавьте для этой кнопки обработчик события **Click** аналогично пункту 5. Сделайте вставку программного кода (выделен зелёным цветом) в тело обработчика события:

```
private void button2_Click(object sender, EventArgs e)
    {
       Close();
    }
```

8. Отредактируйте свойства формы **Form1**, изменив свойство **Text** на «*Лабораторная работа №1*» вместо «*Form1*».

9. Теперь необходимо сохранить проект, выбрав команду **Файл ▸ Сохранить всё**, и провести компиляцию, выбрав команду **Отладка ▸ Начать отладку**. Появится окно формы **Лабораторная работа №1** – это и есть главное окно нашего приложения (Рис. 1-10), и оно ожидает нажатия кнопок или любого другого действия пользователя. Попробуйте щелкнуть на кнопке с надписью: «*Нажми меня*!». Вы увидите, что надпись на ней сменится на «*Нажми меня еще раз*!» (Рис. 1-11). Если вы нажмете повторно, то на кнопке так и останется надпись: «*Нажми меня еще раз*!».

Теперь щелкните на кнопке с надписью «*Выход*». Окно закроется, и программа завершится.

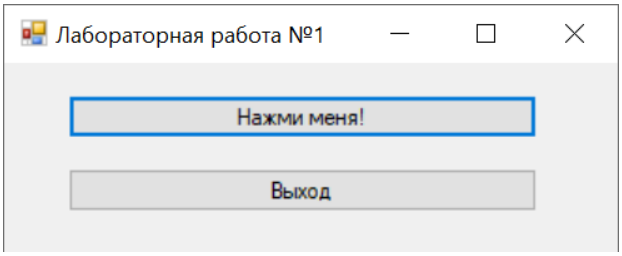

Рис. 1-10 – Вид приложения во время отладки

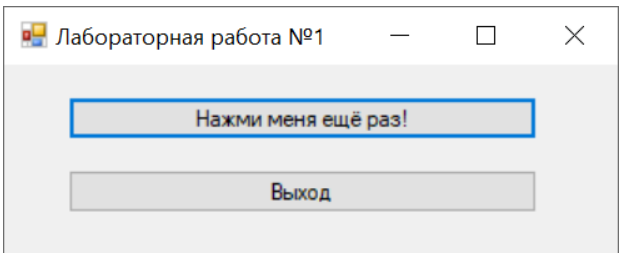

Рис. 1-11 – Надпись на кнопке изменилась!

Задание 2

1. Вновь откройте сохраненный проект. Откройте окно кода, нажав функциональную клавишу F7 или выбрав команду **Вид ▸ Код**.

2. Измените программный код (выделен зелёным цветом) обработчика события **Click** для кнопки **button1**:

```
private void button1_Click(object sender, EventArgs e)
    {
      if (button1.Text == "Нажми меня!") { button1.Text = "Нажми меня ещё раз!"; }
      else { button1.Text = "Нажми меня!"; };
    }
```

3. Сохраните проект и запустите его отладку, например, с помощью функциональной клавиши F5. Левая кнопка теперь циклически меняет свой текст с одной надписи на другую.

Задание 3

Изучите, используя Приложение 1. Работа с формами, основные свойства и события объекта **Form**.

Лабораторная работа 2. Панель элементов Visual Studio. Компоненты GroupBox, TextBox и Label

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Изучить основные свойства и события компонентов **GroupBox**, **TextBox** и **Label** (см. Приложение 2. Элементы управления).

2. Разработать графический интерфейс и программный код приложения в соответствии с представленной методикой.

3. Детально изучить свойства компонентов **TextBox**, **Label**, **GroupBox** (см. Приложение 2. Элементы управления).

Методика выполнения

Задание 1

1. Запустите Visual Studio и создайте новый проект **Lab_2** приложения **Windows Forms (.NET Framework)**.

2. Разместите на форме элемент управления **groupBox1** и измените его свойство **Text** на «*Задание 1*» (Рис. 2-1).

3. Добавьте компонент **textBox1** так, как показано на Рис. 2-2 (текстовое окно должно оказаться в рамке **Задание 1**). Внесите произвольный текст в свойство **Text** объекта **textBox1**.

4. Изменяя свойства **Font** и **ForeColor**, измените вид текста в объекте **textBox1**.

5. Изменяя свойство **TextAlign** объекта **textBox1**, убедитесь в изменении выравнивания надписи.

6. Изменяя свойство **Enabled** (значение по умолчанию – **True**) на **False**,

убедитесь, что при запущенном проекте отсутствует возможность изменения текста в объекте **textBox1**. **Вернитесь к значению по умолчанию!**

7. Аналогичный результат можно получить, изменив свойство **ReadOnly** (значение по умолчанию – **False**) на **True**.

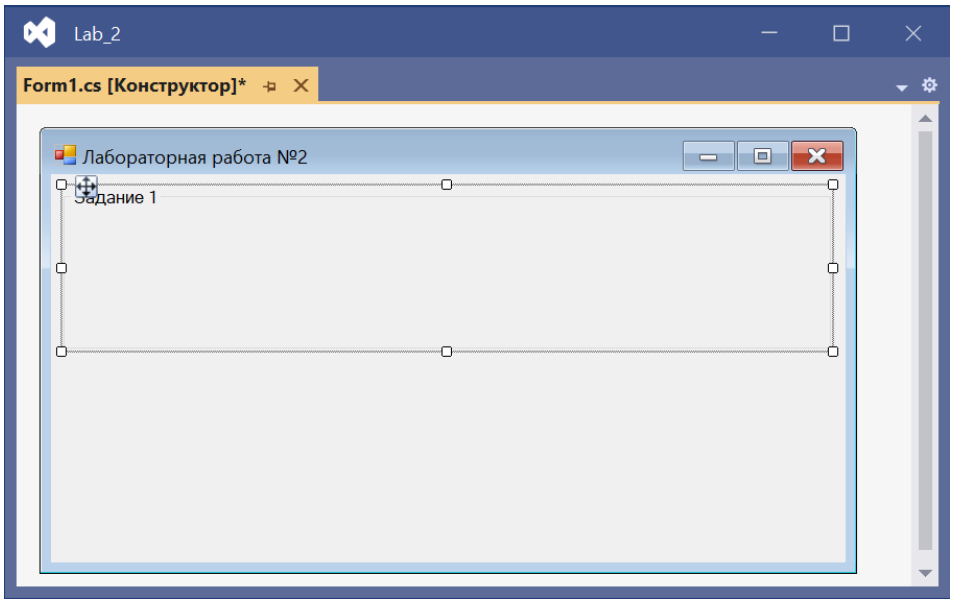

Рис. 2-1 – Конструктор формы приложения

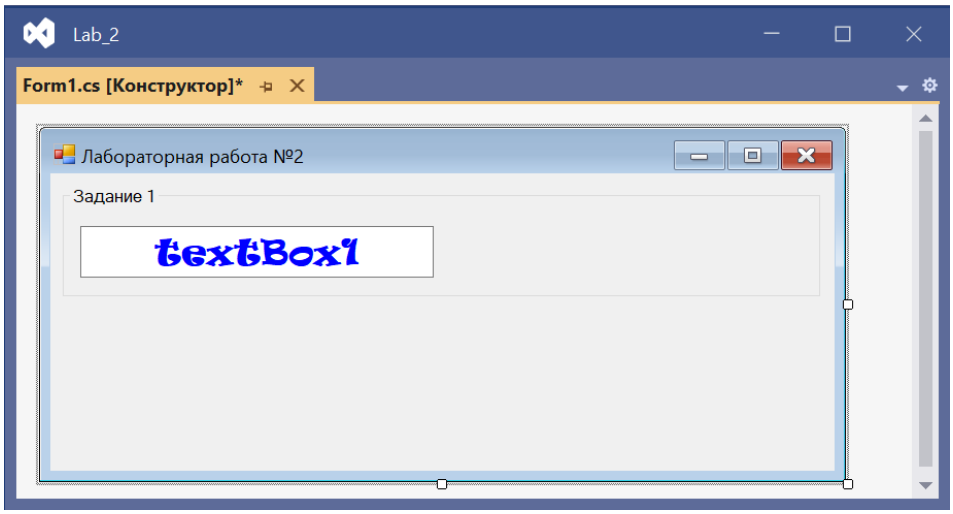

Рис. 2-2 – Настройка **textBox1**

8. Добавьте в форму объект **textBox2** (Рис. 2-3).

9. Сделав двойной щелчок мышью по объекту **TextBox1**, добавьте следующий программный код в обработчик события **TextChange** объекта **textBox1** (вставляемый код выделен зелёным цветом):

```
private void textBox1_TextChanged(object sender, EventArgs e)
    {
       textBox2.Text = textBox1.Text;
    }
```

10. Запустив проект, убедитесь, что изменения текста в объекте **textBox1** сразу же отображаются в объекте **textBox2** (Рис. 2-4).

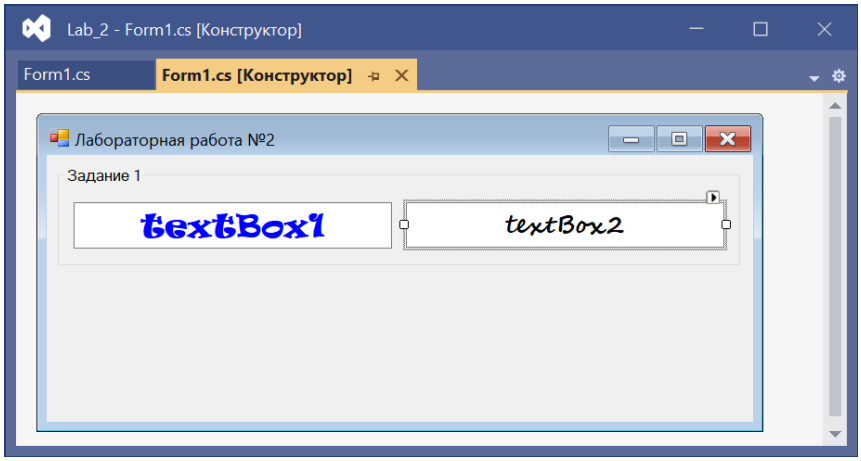

Рис. 2-3 – Конструктор формы приложения

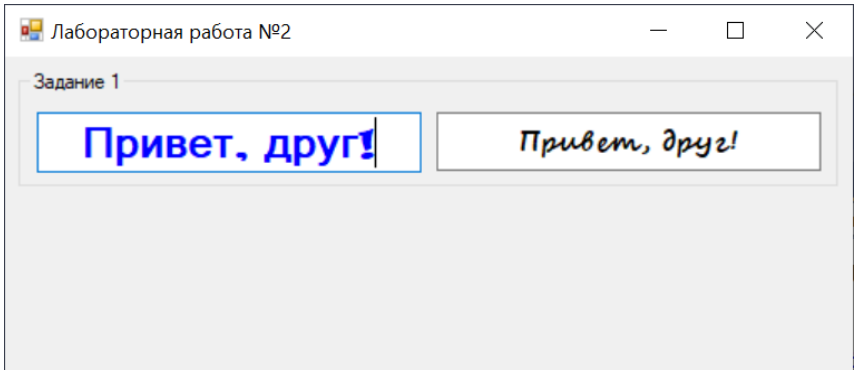

Рис. 2-4 – *Задание 1* готово!

Задание 2

1. Разместите на форме элемент управления **groupBox2** и измените его свойство **Text** на «*Задание 2*» (см. Рис. 2-5).

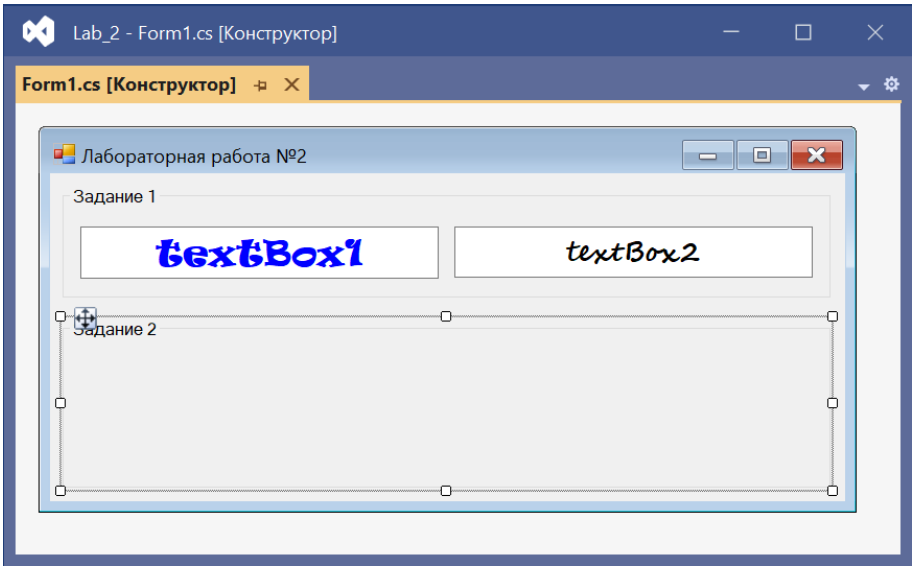

Рис. 2-5 – Конструктор формы приложения

2. Добавьте на форму компоненты **label1**, **label2**, **label3**, **textBox3**, **textBox4** и **button1** так, как показано на Рис. 2-6 (в рамку **Задание 2**).

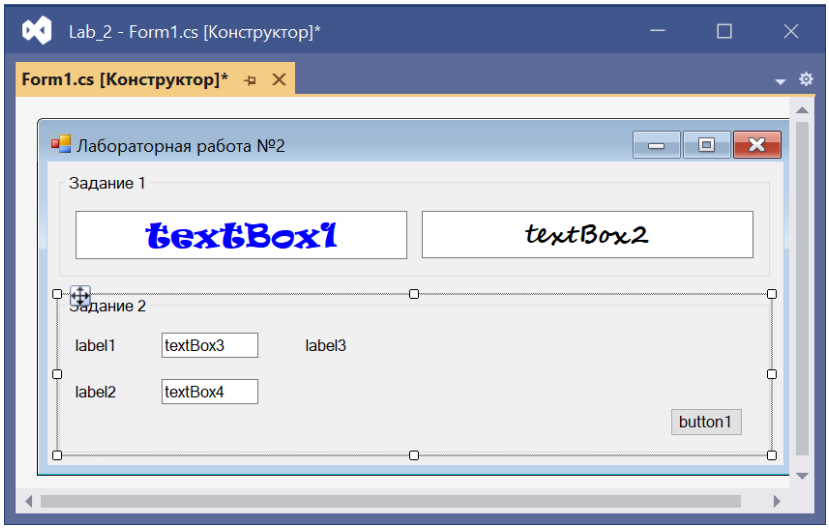

Рис. 2-6 – Конструктор формы приложения

3. Измените значения свойств **Text** и **Font**, подберите размер и положение объектов **label1**, **label2**, **label3**, **textBox3**, **textBox4** и **button1** так, как показано на Рис. 2-7.

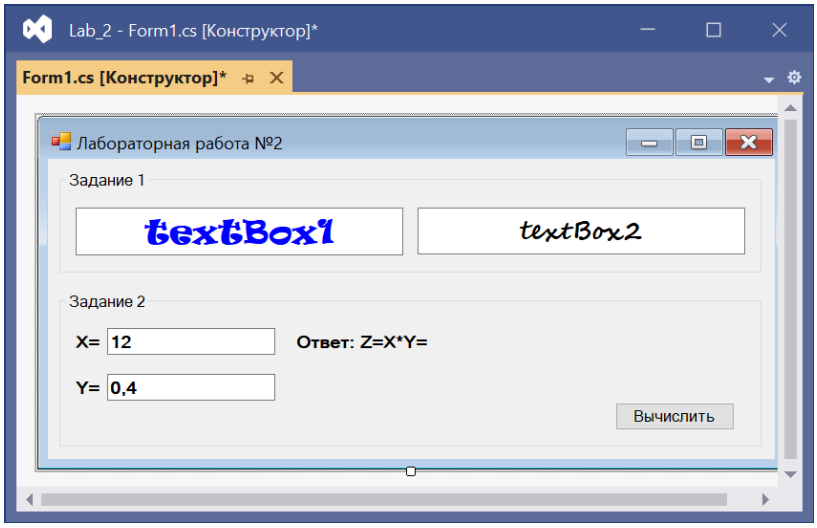

Рис. 2-7 – Интерфейс приложения готов!

4. Сделав двойной щелчок мышью по объекту **button1**, добавьте следующий программный код (выделен зелёным цветом) в обработчик события **Click** для кнопки **button1**:

```
private void button1_Click(object sender, EventArgs e)
    {
        double x = Convert.ToDouble(textBox3.Text);
        double y = Convert.ToDouble(textBox4.Text);
        double z = x * y;
        label3.Text = "Ответ: Z=X*Y=" + z.ToString();
    }
```

Для справки: методы класса **Convert** преобразуют значение одного базового типа данных в другой базовый тип данных, в данном случае строковый тип в вещественный тип данных. **ToString()** – это распространенный метод форматирования в .NET. Он преобразует объект в его строковое представление, чтобы оно подходило для отображения.

5. После запуска проекта, ввода исходных данных и нажатия кнопки **Вычислить** форма примет вид, представленный на Рис. 2-8.

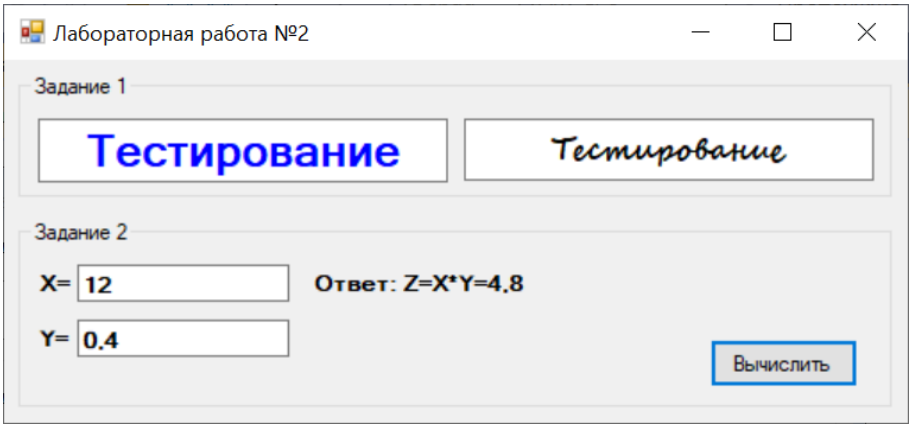

Рис. 2-8 – Отладка проекта **Lab_2**

Задание 3

Изучите, используя Приложение 2. Элементы управления, основные свойства и события объектов **TextBox**, **Label**, **GroupBox**.

Лабораторная работа 3. Математические функции и константы класса Math

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения в соответствии с представленной методикой.

2. Познакомиться со встроенными типами данных C# (см. Приложение 5. Встроенные типы данных в C#) и операциями над ними (см. Приложение 3. Описание некоторых операций C#). Особое внимание обратить на реализацию операции деления чисел.

3. Освоить математические функции и константы класса **Math** (см. Приложение 4. Математические вычисления и класс Math).

Методика выполнения

1. Запустите MS Visual Studio и создайте новый проект **Lab_3** приложения **Windows Forms (.NET Framework)**.

2. Разработайте в конструкторе интерфейс будущего приложения, разместив на форме компоненты так, как показано на Рис. 3-1 (тип компонента и его имя указаны на рисунке с помощью желтых подсказок). **Содержимое текстовых окон задать на стадии конструирования!**

Все поясняющие надписи выполнены с использованием компонента **Label**.

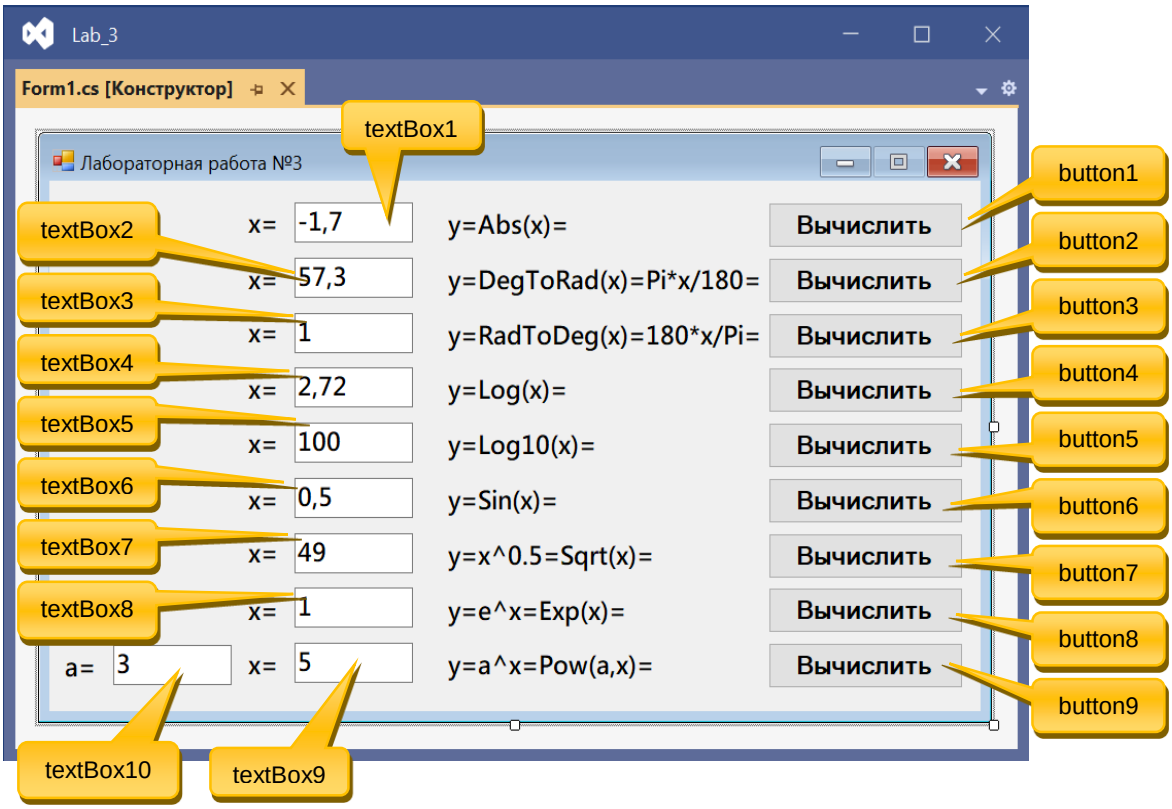

Рис. 3-1 – Конструктор формы приложения

3. Создайте программный код обработчиков события **Click** для соответствующих кнопок:

```
private void button1_Click(object sender, EventArgs e)
    {
        double x, y;
        x = Convert.ToDouble(textBox1.Text);
        y = Math.Abs(x);
        button1.Text = y.ToString();
    }

private void button2_Click(object sender, EventArgs e)
    {
        double x, y;
        x = Convert.ToDouble(textBox2.Text);
        y = Math.PI*x/180;
        button2.Text = y.ToString();
    }

private void button3_Click(object sender, EventArgs e)
    {
        double x, y;
        x = Convert.ToDouble(textBox3.Text);
        y = 180*x/Math.PI;
        button3.Text = y.ToString();
    }
```

```
private void button4_Click(object sender, EventArgs e)
        {
            double x, y;
            x = Convert.ToDouble(textBox4.Text);
            y = Math.Log(x);
            button4.Text = y.ToString();
        }

private void button5_Click(object sender, EventArgs e)
        {
            double x, y;
            x = Convert.ToDouble(textBox5.Text);
            y = Math.Log10(x);
            button5.Text = y.ToString();
        }

private void button6_Click(object sender, EventArgs e)
        {
            double x, y;
            x = Convert.ToDouble(textBox6.Text);
            y = Math.Sin(x);
            button6.Text = y.ToString();
        }

private void button7_Click(object sender, EventArgs e)
        {
            double x, y;
            x = Convert.ToDouble(textBox7.Text);
            y = Math.Sqrt(x);
            button7.Text = y.ToString();
        }

private void button8_Click(object sender, EventArgs e)
        {
            double x, y;
            x = Convert.ToDouble(textBox8.Text);
            y = Math.Exp(x);
            button8.Text = y.ToString();
        }

private void button9_Click(object sender, EventArgs e)
        {
            double a,x, y;
            a = Convert.ToDouble(textBox10.Text);
            x = Convert.ToDouble(textBox9.Text);
            y = Math.Pow(a,x);
            button9.Text = y.ToString();
        }
```

4. Результат запуска приложения на отладку представлен на Рис. 3-2.

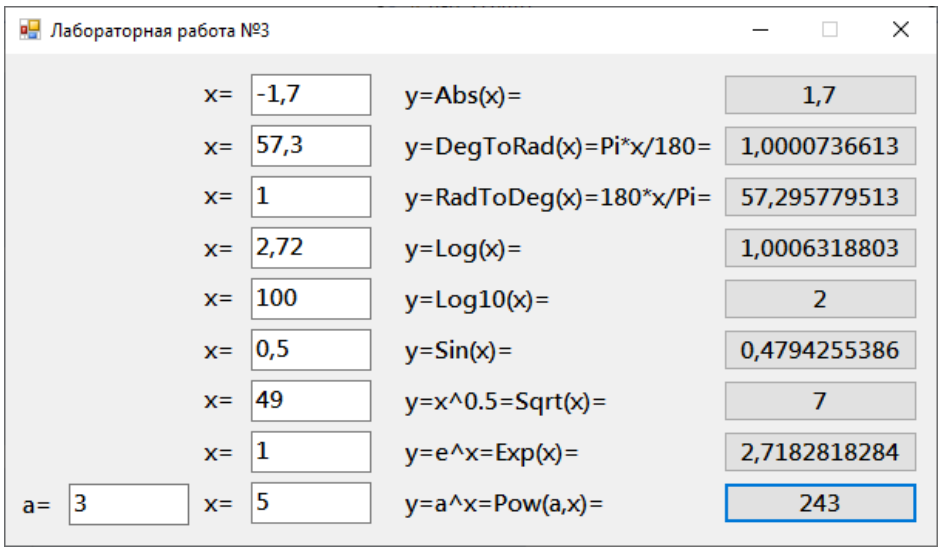

Рис. 3-2 – Отладка проекта **Lab_3**

5. Обратите внимание на тот факт, что методы класса **Convert** выдают ошибку при пустом текстовом окне. Для решения этой проблемы доработаем программный код для обработчиков наших кнопок.

Например, для **button1** код будет выглядеть следующим образом:

```
private void button1_Click(object sender, EventArgs e)
    {
      double x, y;
      if (textBox1.Text=="") return;
      x = Convert.ToDouble(textBox1.Text);
      y = Math.Abs(x);
      button1.Text = y.ToString();
    }
```

Теперь при пустом окне ввода вычисления будут просто прерываться.

6. Необходимо также обработать событие **TextChanged** для каждого тестового окна так, чтобы при попытке изменить его содержимое надпись на соответствующей кнопке изменялась на «*Вычислить*».

Например, для **textBox1** код будет выглядеть следующим образом:

```
private void textBox1_TextChanged(object sender, EventArgs e)
    {
      button1.Text = "Вычислить";
    }
```

Лабораторная работа 4. Использование диалогового окна сообщения MessageBox

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления значения функции $y = A + B\sin(x)$ в соответствии с

представленной методикой.

2. Вывести результат с помощью диалогового окна сообщения **MessageBox** (см. Приложение 2. Элементы управления) и проанализировать действия пользователя (какая кнопка окна оказалась нажатой):

- если нажата **Да** – фон **txtA** сделать зелёным;
- если **Нет** – фон **lblB** – красным;
- если **Отмена** – фон **cmdResult** – жёлтым.

3. Выполнить пп.1-2 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Вычислить значение функции при A, B и C – константах (задать значения в текстовых окнах) и произвольном x.

1) $y = \frac{A+B}{2A-C}x$
2) $y = 2B + x^{AC}$
3) $y = \frac{A+Bx-Cx^2}{\cos x}$
4) $y = \cos x^5 \cdot \sqrt{\frac{AC}{x^A}}$
5) $y = \frac{xA}{B}\sqrt{x^C+1}$
6) $y = A + C\sqrt{\sin B + x}$
7) $y = AB - 3C$
8) $y = tg\frac{A}{B} + C$
9) $y = B^x + \frac{A}{x^C}$
10) $y = \frac{4x^2}{A^2+B-C}$
11) $y = \frac{B+C}{\sin(x+B)}$
12) $y = \sqrt{\sin^2(x^2+A)+C}$
13) $y = x^2(A-BC)$
14) $y = Ax + \frac{x^B}{1+x^C}$
15) $y = A^{-\frac{x}{B}} + C$
16) $y = \sqrt{A+\sqrt{B+\sqrt{C+x}}}$
17) $y = A + Bx + Cx^2$
18) $y = x^3 + BC + A$
19) $y = \frac{A+B}{Cx}$
20) $y = A^x + \frac{B}{x^C}$
21) $y = \frac{ABC}{B+x}$
22) $y = \sqrt[5]{Ax} + \sqrt[3]{Bx} + C$
23) $y = \frac{1+\left(\frac{A}{C}\right)^3}{Bx}$
24) $y = \frac{\sqrt[A]{x}+Ax}{x+C}$
25) $y = \frac{1}{\cos x} + \frac{A}{B+Cx}$
26) $y = A + B + x^{A-C}$
27) $y = \sqrt[3]{(x-AB)} + C$
28) $y = x - AC \cdot \sqrt[3]{Bx}$
29) $y = B - \sqrt[4]{Ax+C}$
30) $y = \frac{B}{C} + \sqrt[3]{\cos Ax}.$

Методика выполнения

1. Расположение, вид и имена меток, текстовых окон и командной кнопки в форме **Form1** представлены на Рис. 4-1.

2. Следующий шаг – создание программного кода для обработчика события **Click** командной кнопки:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
      double a, b, x, y;
      a = Convert.ToDouble(txtA.Text);
      b = Convert.ToDouble(txtB.Text);
      x = Convert.ToDouble(txtX.Text);
      y= a + b * Math.Sin(x);
```

```
        DialogResult res = MessageBox.Show("Y=" + y.ToString(),
                "Результат", MessageBoxButtons.YesNoCancel,
                MessageBoxIcon.Warning, MessageBoxDefaultButton.Button1);
        if (res == DialogResult.Yes) txtA.BackColor = Color.Green; else
                if (res == DialogResult.No) lblB.BackColor = Color.Red; else
                        if (res == DialogResult.Cancel) cmdResult.BackColor = Color.Yellow;
}
```

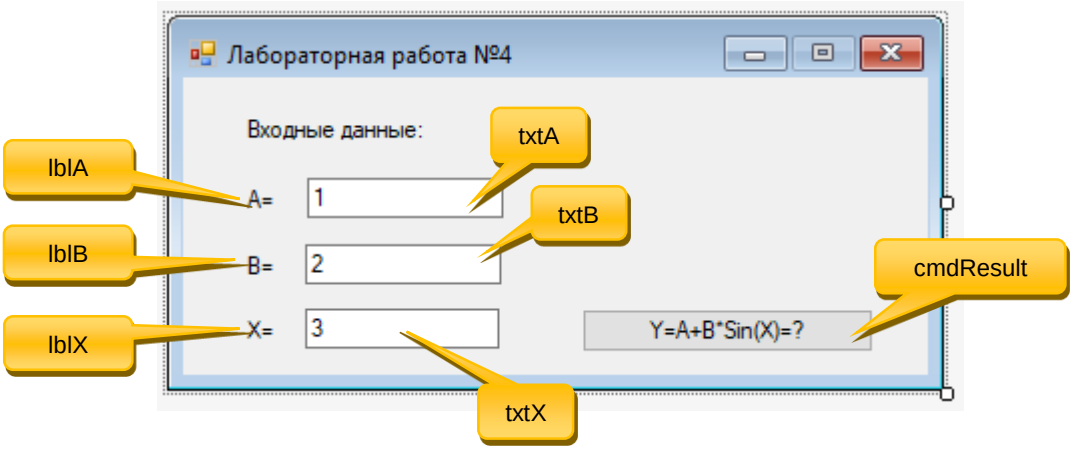

Рис. 4-1 – Окно конструктора приложения

3. На Рис. 4-2 представлен результат после запуска проекта и нажатия на кнопку **cmdResult**. На Рис. 4-3 представлено окно сообщения после щелчка по кнопкам окна сообщений с надписями: «*Да*», «*Нет*», «*Отмена*».

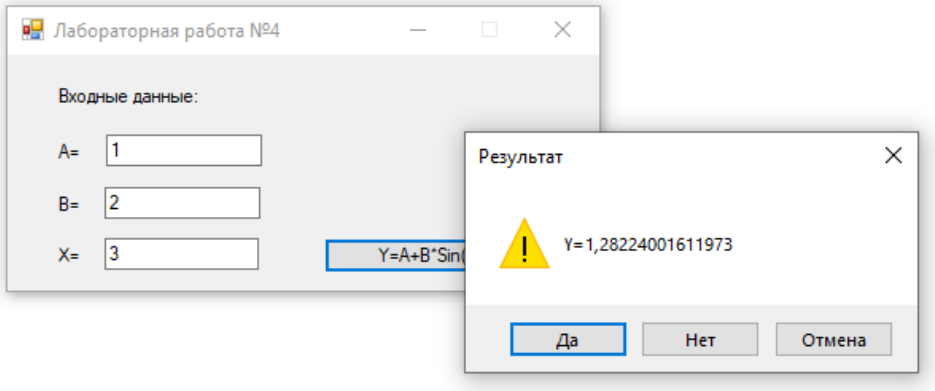

Рис. 4-2 – Результат расчёта

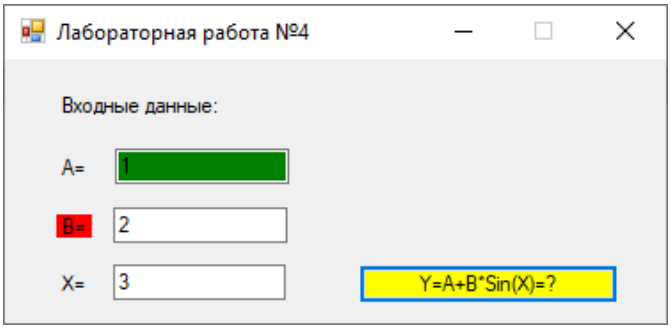

Рис. 4-3 – Реакция на нажатие кнопок **Да**, **Нет**, **Отмена**

Лабораторная работа 5. Использование диалогового окна ввода информации InputBox

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления значения функции

$$y=\begin{cases}\sin x & \textit{при } x>7,\\ \cos x & \textit{при } 2<x\le 6,\\ \sin x-\cos x & \textit{в остальных случаях}\end{cases}$$

в соответствии с представленной методикой.

2. Организовать ввод исходных данных с помощью диалогового окна ввода информации **InputBox** (см. Приложение 2. Элементы управления). Предусмотреть некорректный ввод данных при этом. В случае ошибки данных вывести соответствующее сообщение.

3. Для анализа исходных данных в программном коде использовать управляющие конструкции (Приложение 6. Управляющие конструкции языка C#).

4. Вывести развернутый результат, используя элемент **Label**.

5. Выполнить пп.1-4 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Вычислить функцию:

1. $y=\begin{cases}\dfrac{\sqrt{\sin^2 x_1+\cos^2 x_2}}{2{,}5\cdot b_1+b_2} & \textit{если } x_1 \textit{ или } x_2>0;\\ 1{,}5b_1^2+\sqrt[3]{\lvert b_2\rvert} & \textit{в остальных случаях};\end{cases}$	2. $r=\begin{cases}y^3-0{,}5b_1 & \textit{при } y-0{,}5b_1>0 \textit{ и } y>25;\\ \sin^2 b_2+\lvert b_3\rvert & \textit{при } y\le 25;\\ 2b_1 & \textit{в остальных случаях};\end{cases}$
3. $z=\begin{cases}\dfrac{3{,}5r+0{,}5\lvert d\rvert}{\sqrt{r^2+r^3}} & \textit{если } d=-2 \textit{ и } 0<R<1;\\ \sqrt[3]{r^d} & \textit{в остальных случаях};\end{cases}$	4. $y=\begin{cases}{z_2}^3+1{,}5 & \textit{при } z_1=0;\\ \dfrac{{z_1}^2+6{,}5\lvert z_3\rvert}{z_4} & \textit{при } z_1<0;\\ \sin(z_2) & \textit{в остальных случаях};\end{cases}$
5. $y=\begin{cases}\ln\lvert x+a^2\rvert & \textit{при } x>7;\\ \sqrt{\lvert b_1+{b_2}^2\rvert} & \textit{при } x=7;\\ \dfrac{3{,}5x}{0{,}5b_1+{b_2}^2} & \textit{в остальных случаях};\end{cases}$	6. $y=\begin{cases}a_1x^2+a_2x+a_3 & \textit{если } -15<x<21;\\ \sqrt{\dfrac{\ln\lvert x\rvert+5{,}5}{1{,}5a_1}} & \textit{в остальных случаях};\end{cases}$
7. $y=\begin{cases}\lvert 1{,}5-c_1+e^{x\,c_2}\rvert & \textit{если } x<10;\\ (x^2\ln c_1+0{,}5c_3)^2 & \textit{если } x>12.\end{cases}$	8. $w=\begin{cases}\dfrac{c_1x^3+c_3x+\sqrt{c_4}}{4x-3{,}2b} & \textit{если } 4x-3{,}2b\ne 0 \textit{ и } x<5;\\ \ln\lvert c_1x+2{,}5b\rvert & \textit{в остальных случаях}.\end{cases}$
9. $z=\begin{cases}\dfrac{3{,}5r_1+0{,}5\lvert d\rvert}{\sqrt{\lvert r_1+r_2\rvert}} & \textit{если } d=2 \textit{ и } r_2\ne 0;\\ d\cdot r_1\cdot r_2 & \textit{в остальных случаях}.\end{cases}$	10. $r=\begin{cases}\sqrt{y^3+0{,}5b_1} & \textit{при } y-0{,}5b_1>0 \text{ и } y>25;\\ \sin^2 b_2+\lvert b_3\rvert & \textit{при } y\le 27;\\ 2b_4 & \textit{в остальных случаях}.\end{cases}$

11. $w=\begin{cases}\ln\lvert a_2\rvert+\lvert a_1\rvert & \text{при } a_1+a_2+a_3>0;\\ \sqrt{\lvert c+3{,}5a_2\rvert} & \text{при } c>10 \text{ и } a_1+a_2+a_3\le 0;\\ 3{,}7a_1-a_3 & \text{в остальных случаях.}\end{cases}$	12. $z=\begin{cases}\dfrac{3{,}6d_1+0{,}5d_2+\lvert d_3\rvert}{\sqrt{\lvert a+d_3\rvert}} & \text{если } d_1=2 \text{ и } d_2=5;\\ 18a & \text{в остальных случаях.}\end{cases}$
13. $w=\begin{cases}a_1x^2-a_2x+a_3 & \text{если } 10<x<17;\\ \dfrac{\sqrt{\ln\lvert a_2\rvert+6{,}5}}{1{,}5a_1+{a_2}^2} & \text{в остальных случаях.}\end{cases}$	14. $y=\begin{cases}\lvert a_1+0{,}5a_2-z\rvert & \text{при } z>6;\\ \dfrac{6{,}5a_1z}{a_1+a_2+a_3} & \text{в остальных случаях;}\\ \ln\lvert z+a_1\rvert & \text{при } z<5.\end{cases}$
15. $y=\begin{cases}\ln\lvert z+a_1\rvert & \text{при } z<5;\\ \lvert a_1+0{,}5a_2-z\rvert & \text{при } z>6;\\ \text{"знач. не определено"} & \text{в ост. случаях.}\end{cases}$	16. $r=\begin{cases}y^3-0{,}5b_1 & \text{при } y-0{,}5b_1>0 \text{ и } y>25;\\ \sin^2 b_2+\lvert b_3\rvert & \text{при } y\le 25;\\ \text{"неопред. знач."} & \text{в остальных случаях.}\end{cases}$
17. $r=\begin{cases}\dfrac{w_1+\sqrt{w_2}}{4x-3{,}2c} & \text{если } 4x-3{,}2c\ne 0 \text{ и } w_2\ge 0;\\ \ln\lvert w_1x+2{,}5c\rvert & \text{в остальных случаях.}\end{cases}$	18. $r=\dfrac{\sqrt{d_1+d_2+d_3}}{\min(d_1,d_2,d_3)}$
19. $r=\dfrac{\sqrt{d_1+d_2+d_3}}{\max(d_1,d_2,d_3)}$	20. $w=\begin{cases}\dfrac{\lvert x_1-2{,}5x_2\rvert}{2{,}3} & \text{если } x_1 \text{ и } x_2>0;\\ 2{,}5x_1+\sqrt{\lvert x_2\rvert} & \text{в остальных случаях.}\end{cases}$
21. $r=\dfrac{\min(d_1,d_2,d_3)}{\sqrt{d_1+d_2+d_3}}$	22. $r=\dfrac{\max(d_1,d_2,d_3)}{\sqrt{d_1+d_2+d_3}}$
23. Определить максимальное из произвольных чисел A_1, A_2, A_3 и их сумму.	24. Определить максимальное из произвольных чисел A_1, A_2, A_3 и их среднее арифметическое значение.
25. Даны произвольные числа *a*, *b* и *c*. Если нельзя построить треугольник с такими длинами сторон, то напечатать соответствующее сообщение, иначе напечатать «равносторонний», «равнобедренный» или какой-либо иной треугольник.	26. $z=\begin{cases}\dfrac{1+x^2}{2\sqrt[3]{1+x^4}}, & x\le 0;\\ 2x+\dfrac{\sin^2(x)}{2+x+x^2}, & x>0.\end{cases}$
27. $u=\begin{cases}\dfrac{1+\lvert x\rvert}{\sqrt[3]{1+x+x^2}}, & x\le -1;\\ \dfrac{1+\cos^4(x)}{3(2+x)}, & -1<x<0;\\ (1+x)^{\frac{3}{5}}, & x\ge 0.\end{cases}$	28. $y=\begin{cases}\ln\lvert z+a_1\rvert & \text{при } z<5;\\ \lvert a_1+0{,}5a_2-z\rvert & \text{в ост. случаях;}\\ \dfrac{6{,}5a_1*z}{a_1+a_2+a_3} & \text{при } z>6.\end{cases}$
29. $y=\begin{cases}{z_2}^3+1{,}5 & \text{при } z_1+z_2>0;\\ \dfrac{{z_1}^2+6{,}5\lvert z_3\rvert}{z_4} & \text{при } z_4\ne 0;\\ \sin(z_2) & \text{в остальных случаях.}\end{cases}$	30. $y=\begin{cases}\dfrac{\sqrt{\sin^2 x_1+\cos^2 x_2}}{2{,}5b_1+b_2} & \text{если } x_1 \text{ и } x_2>0;\\ 1{,}5{b_1}^2+\sqrt[3]{\lvert b_2\rvert} & \text{в остальных случаях.}\end{cases}$

Методика выполнения

1. В C# нет готовой функции вызова диалогового окна ввода информации **InputBox**. Однако достаточно добавить в проект ссылку на *Microsoft.VisualBasic* командами **Проект ▸ Добавить ссылку** (Рис. 5-1) и вызывать **InputBox** через него.

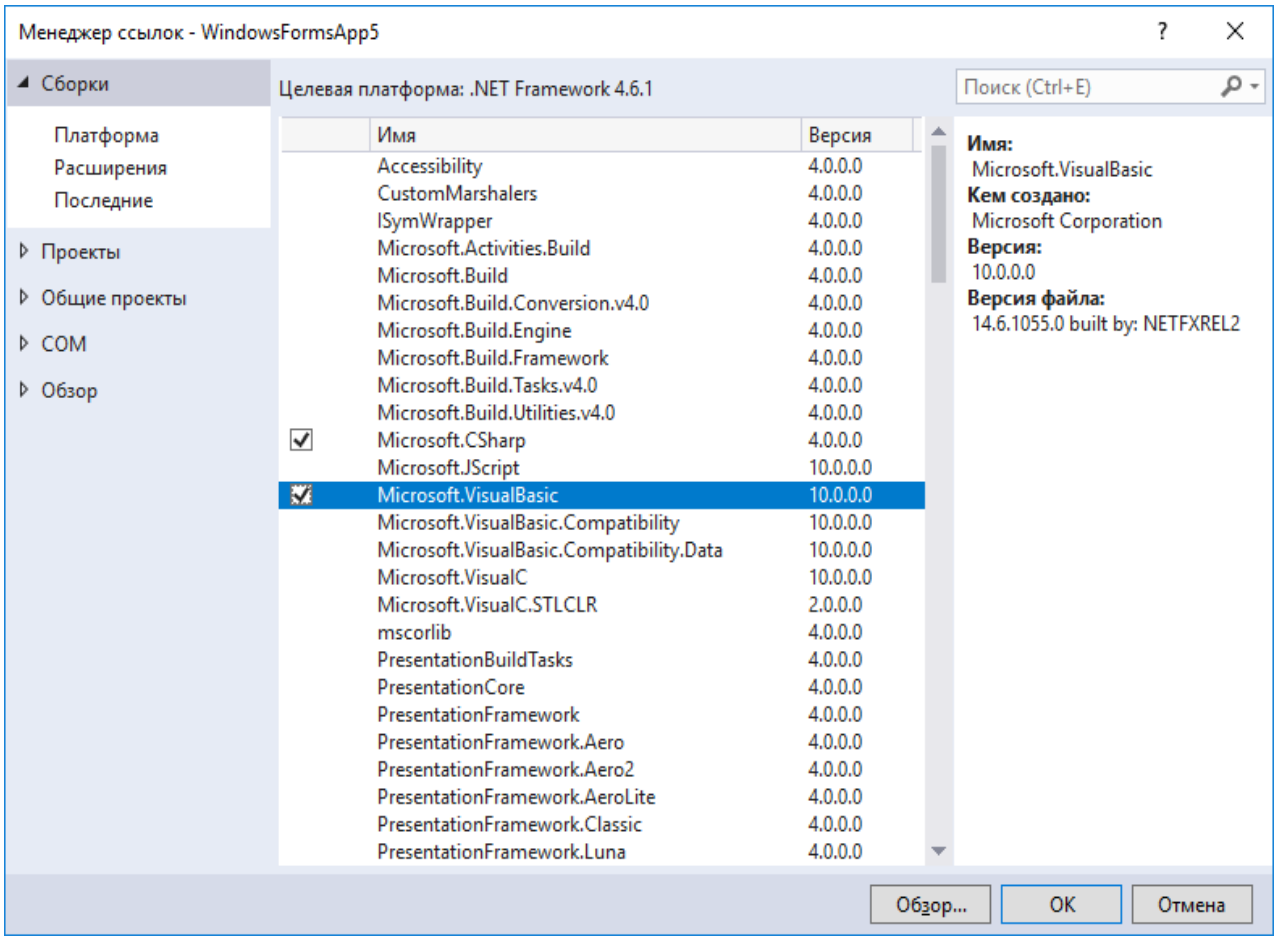

Рис. 5-1 – Добавление ссылки на Microsoft.VisualBasic

2. Расположение, вид и имена метки, картинки и командной кнопки на форме **Form1** представлены на Рис. 5-2.

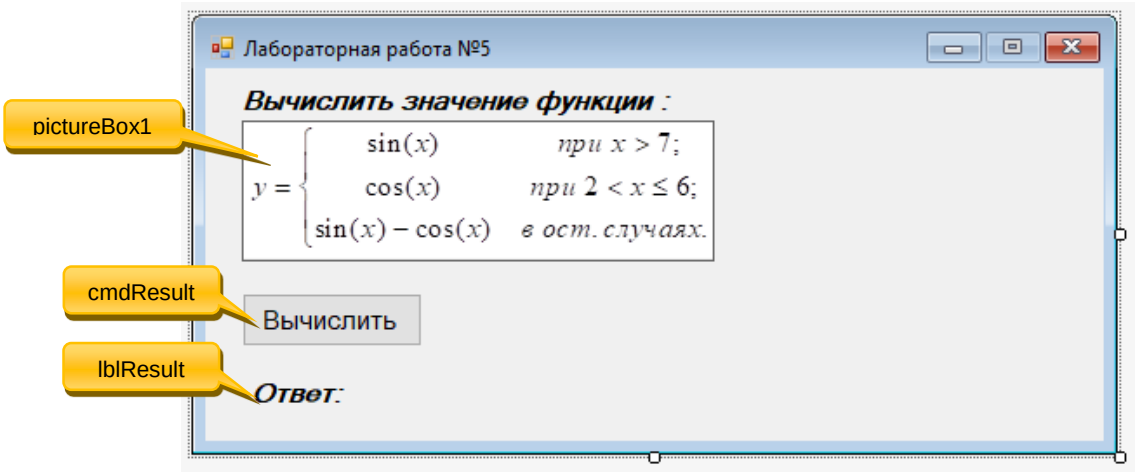

Рис. 5-2 – Конструктор **Form1**

3. Изображение формулы разместить на форме с помощью элемента **PicrureBox** (см. Приложение 2. Элементы управления), предварительно сохранив изображение формулы в файле *формула.png*, а затем указать этот файл как значение свойства **Image** для **picrureBox1**.

4. Программный код для **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
       double x,y;
       string s = Microsoft.VisualBasic.Interaction.InputBox("Введите значение
                                                    аргумента X", "Ввод данных", "0");
       if (s == "") { MessageBox.Show("Некоректный ввод данных!", "Ввод данных",
                            MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return; }
       x = Convert.ToDouble(s);
       if (x > 7) y = Math.Sin(x);
          else if ((x > 2) && (x <= 6)) y = Math.Cos(x);
             else y = Math.Sin(x) - Math.Cos(x);
       lblResult.Text = "Ответ: Y = " + y.ToString() + " при X = " + x.ToString();
    }
```

5. Вид проекта после нажатия на кнопку с надписью: «*Вычислить*» представлен на Рис. 5-3.

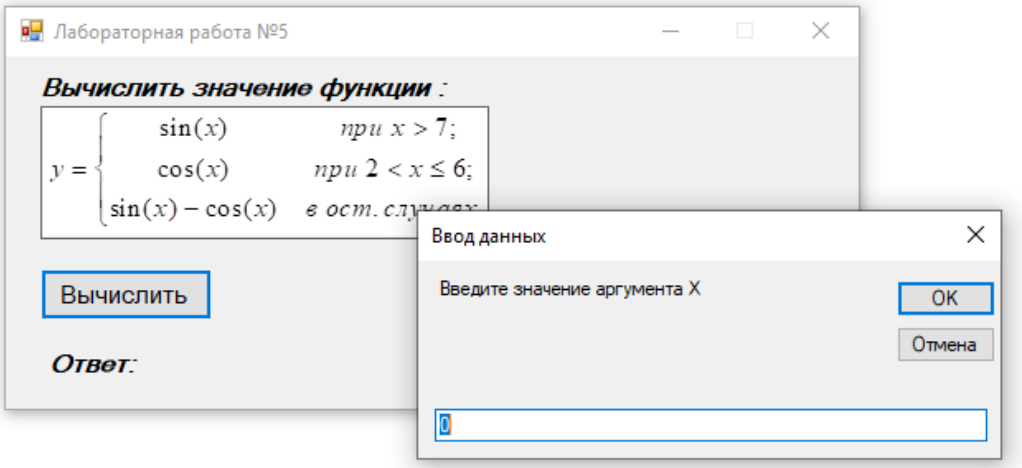

Рис. 5-3 – Окно **InputBox**

6. На Рис. 5-4 представлен вид проекта после ввода в диалоговое окно значения аргумента и нажатия на командную кнопку **OK**.

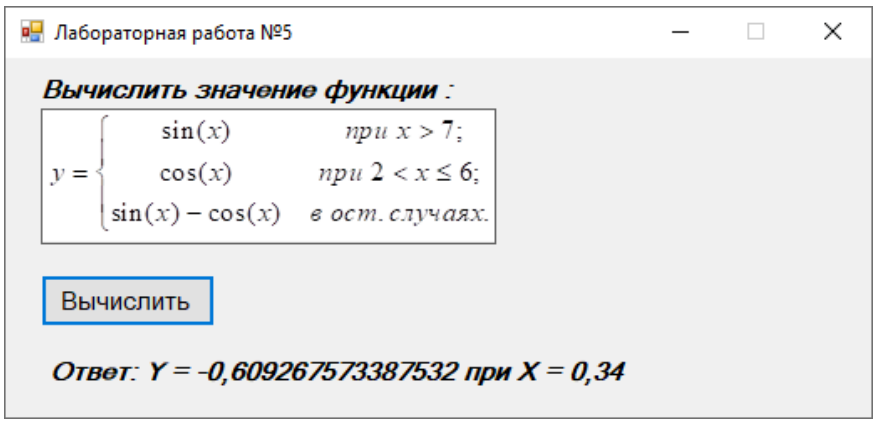

Рис. 5-4 – Результат

Лабораторная работа 6. Приложение для вычисления значения суммы ряда

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления значения функции $s = \sum_{i=0}^{n} \frac{1}{i!}$ при заданном значении параметра *n* в соответствии с представленной методикой.

2. Организовать ввод исходных данных с помощью диалогового окна ввода информации **InputBox**. Предусмотреть некорректный ввод данных при этом. В случае ошибки данных вывести соответствующее сообщение.

3. В программном коде использовать цикл **for** (см. Приложение 6. Управляющие конструкции языка C#).

4. Вывести развернутый результат, используя элемент **Label**.

5. Выполнить пп.1-4 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Вычислить функцию:

$1.\ s = \sum_{i=1}^{n} i^2$	$2.\ s = \sin(x) = \sum_{i=1}^{n} \frac{(-1)^{i-1} x^{2i-1}}{(2i-1)!}$	$3.\ s = \sum_{i=1}^{n} \frac{1}{i}$
$4.\ s = \sum_{i=m}^{n} (2i)$	$5.\ s = \sum_{i=10}^{n} i^3$	$6.\ p = \prod_{i=m}^{n} (2i)$
$7.\ p = n! = \prod_{i=1}^{n} i$	$8.\ s = \sum_{i=1}^{n} i$	$9.\ s = \sum_{i=1}^{n} i!$
$10.\ s = \cos(x) = \sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i)!}$	$11.\ s = e^x = \sum_{i=0}^{n} \frac{x^i}{i!}$	$12.\ s = \sum_{i=0}^{n} \frac{1}{2^i}$
$13.\ s = \sum_{i=1}^{n} \frac{(x+1)^3}{(4i)!}$	$14.\ s = \sum_{i=1}^{n} \frac{x^{i+1}}{(2i-1)^2}$	$15.\ s = \sum_{i=1}^{n} \frac{x^{2i-1}}{(2i-1)(2i-1)!}$
$16.\ s = \sum_{i=1}^{n} \frac{1}{(2i-1)^4}$	$17.\ s = \sum_{i=1}^{n} \frac{(-1)^{i-1}}{i^4}$	$18.\ s = \sum_{i=1}^{n} \frac{(-1)^i}{i^4}$
$19.\ s = \sum_{i=1}^{n} \frac{1}{i^4}$	$20.\ s = \sum_{i=1}^{n} \frac{(-1)^{i-1}}{(2i-1)^3}$	$21.\ s = \sum_{i=1}^{n} \frac{1}{i^2}$
$22.\ s = \sum_{i=1}^{n} \frac{(-1)^{i-1}}{i^2}$	$23.\ s = \sum_{i=1}^{n} \frac{1}{(2i-1)^2}$	$24.\ s = \sum_{i=1}^{n} \frac{1}{i(i+1)(i+2)}$
$25.\ s = \sum_{i=1}^{n} \frac{1}{(4i-1)(4i+1)}$	$26.\ s = \sum_{i=2}^{n} \frac{1}{(i-1)(i+1)}$	$27.\ s = \sum_{i=1}^{n} \frac{1}{(2i-1)(2i+1)}$

28. $s=\sum_{i=1}^{n}\frac{1}{i(i+1)}$	29. $s=\sum_{i=0}^{n}\frac{(-1)^i}{2^i}$	30. $s=\sum_{i=0}^{n}\frac{1}{2^{i+2}}$

Методика выполнения

1) Известно, что сумма данного бесконечного числового ряда

$$s=\sum_{i=0}^{\infty}\frac{1}{i!}=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots+\frac{1}{i!}+\ldots$$

в пределе стремится к значению константы $e\approx 2{,}71828182\ldots$

Требуется составить программу, вычисляющую эту константу по сумме числового ряда.

Если слагаемы в этом выражении обозначить следующим образом:

$$a_0=1,\qquad a_1=\frac{1}{1!},\qquad a_2=\frac{1}{2!},\qquad a_3=\frac{1}{3!},\qquad \ldots,$$

то обобщенная формула для *i*-го элемента будет такой:

$$a_i=\frac{1}{i!}$$

Поскольку

$$\frac{a_i}{a_{i-1}}=\frac{\frac{1}{i!}}{\frac{1}{(i-1)!}}=\frac{1}{i!}\cdot(i-1)!=\frac{(i-1)!}{i!}=\frac{1}{i},$$

то нетрудно видеть, что между элементами данной последовательности имеется зависимость:

$$a_i=\frac{1}{i}\cdot a_{i-1}=\frac{a_{i-1}}{i}.$$

Такая зависимость называется **рекуррентной зависимостью**, а соответствующая числовая последовательность – **рекуррентной последовательностью**. Данная рекуррентная последовательность может быть описана следующим образом:

$$a_i=\begin{cases}1 & при\ i=0;\\ \frac{a_{i-1}}{i} & при\ i>0.\end{cases}$$

Если в индивидуальном варианте отсутствует вычисление факториала, то рекуррентную последовательность строить нет необходимости, можно использовать обобщенную формулу для *i*-го элемента (см. п.7).

2) Расположение, вид и имена метки, картинки и командной кнопки на форме **Form1** представлены на Рис. 6-1.

3) Изображение формулы разместить на форме с помощью элемента **PicrureBox**, предварительно сохранив изображение формулы в файле *ряд.png*, а затем указать этот файл как значение свойства **Image** для **picrureBox1**.

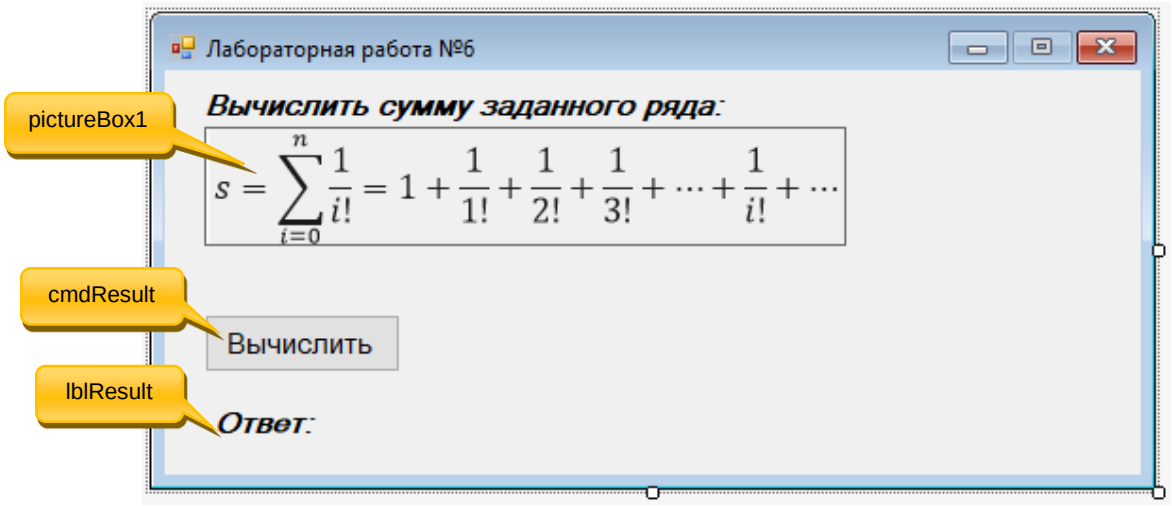

Рис. 6-1 – Конструктор **Form1**

4) Программный код для **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
      string s = Microsoft.VisualBasic.Interaction.InputBox("Введите количество
                                                элементов ряда n", "Ввод данных", "10");
      if (s == "") { MessageBox.Show("Некоректный ввод данных!", "Ввод данных",
                              MessageBoxButtons.OK,MessageBoxIcon.Error); return; }
      int n = Convert.ToInt32(s);
      double sl = 1;
      double sum = sl;
      for (int i = 1; i <= n; i++)
      {
            sl = sl / i;
            sum = sum + sl;
      }
      lblResult.Text = "Ответ: s = "+sum.ToString()+" при n = "+n.ToString();
    }
```

5) Вид проекта после нажатия на кнопку «*Вычислить*» представлен на Рис. 6-2.

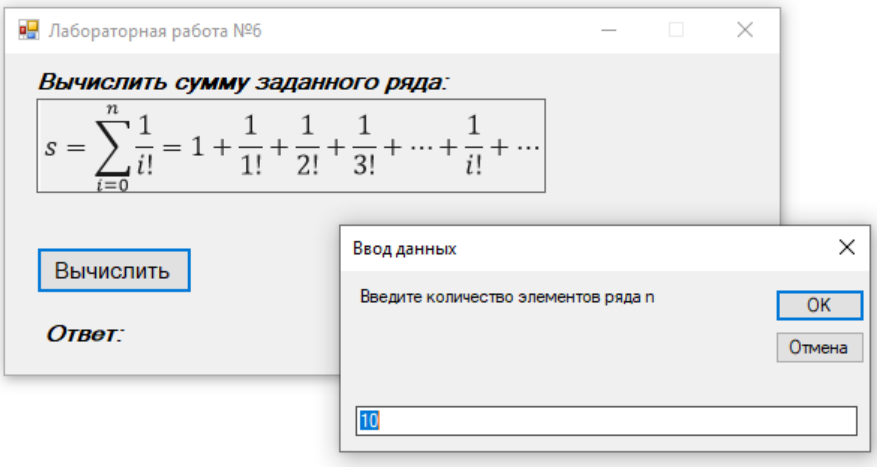

Рис. 6-2 – Окно **InputBox**

6) На Рис. 6-3 представлен вид проекта после ввода в диалоговое окно значения аргумента и нажатия на командную кнопку ***OK***.

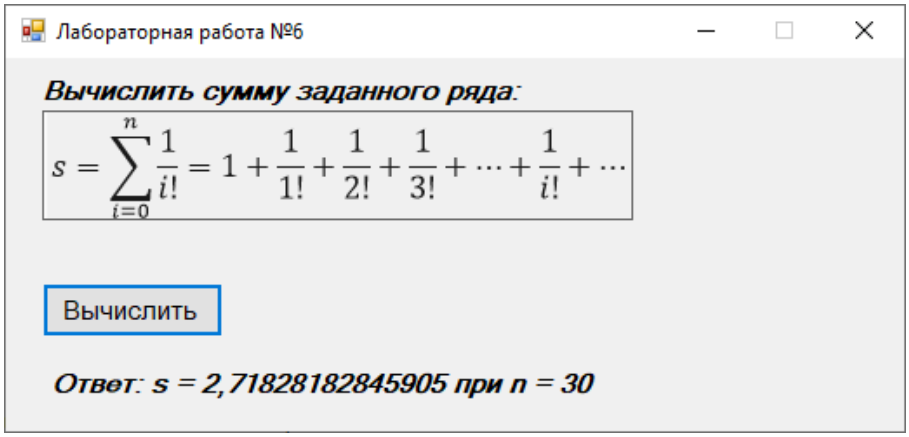

Рис. 6-3 – Результат

7) **Попробуем 2-ой способ (без построения рекуррентной зависимости).** Сделайте копию проекта и исправьте программный код для **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
      string s = Microsoft.VisualBasic.Interaction.InputBox("Введите количество
                                                элементов ряда n", "Ввод данных", "10");
      if (s == "") { MessageBox.Show("Некоректный ввод данных!", "Ввод данных",
                              MessageBoxButtons.OK,MessageBoxIcon.Error); return; }
      int n = Convert.ToInt32(s);
      double sum = 0;
      for (int i = 0; i <= n; i++)
      {
              double f = 1;
              for (int j = 1; j <= i; j++) f = f * j;
              sum = sum + 1/f;
      }
      lblResult.Text = "Ответ: s = "+ sum.ToString()+" при n = "+n.ToString();
    }
```

Мы здесь использовали вложенный цикл для вычисления факториала.

Лабораторная работа 7. Приложение для вычисления значения суммы ряда с заданной точностью

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления суммы бесконечного ряда $\sum_{n=0}^{\infty} \frac{1}{n!}$ с точностью до $\varepsilon > 0$ и определить, при каком значении параметра n заданная точность достигается.

2. Организовать ввод исходных данных с помощью диалогового окна ввода информации **InputBox**. Предусмотреть некорректный ввод данных при этом. В случае ошибки данных вывести соответствующее сообщение.

3. В программном коде использовать цикл с пред– или постусловием (см. Приложение 6. Управляющие конструкции языка C#).

4. Вывести развернутый результат, используя элемент **Label**.

5. Выполнить пп.1-4 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Найти сумму ряда с точностью до $\varepsilon > 0$:

1	$\sum_{n=0}^{\infty}(-1)^n \frac{x}{2n+1}$	11	$\sum_{n=0}^{\infty}\frac{3^n}{n!}$	21	$\sum_{n=1}^{\infty}\frac{(x+1)^3}{(4n)!}$
2	$\sum_{n=1}^{\infty}(-1)^{n+1}e^{-\frac{n}{x}}$	12	$\sum_{n=1}^{\infty}\frac{x^{n+1}}{(2n-1)^2}$	22	$\sum_{n=1}^{\infty}\frac{x^{2n-1}}{(2n-1)!}$
3	$\sum_{n=1}^{\infty}\frac{5^n x^{n-1}}{(2+n)!}$	13	$\sum_{n=1}^{\infty}\frac{1}{(2n-1)^4}$	23	$\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n^4}$
4	$\sum_{n=1}^{\infty}\frac{1}{n^4}$	14	$\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{(2n-1)^3}$	24	$\sum_{n=1}^{\infty}\frac{1}{n^2}$
5	$\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n^2}$	15	$\sum_{n=1}^{\infty}\frac{1}{(2n-1)^2}$	25	$\sum_{n=1}^{\infty}\frac{1}{n(n+1)(n+2)}$
6	$\sum_{n=1}^{\infty}\frac{1}{(4n-1)(4n+1)}$	16	$\sum_{n=2}^{\infty}\frac{1}{(n-1)(n+1)}$	26	$\sum_{n=1}^{\infty}\frac{1}{(2n-1)(2n+1)}$
7	$\sum_{n=1}^{\infty}\frac{1}{n(n+1)}$	17	$\sum_{n=0}^{\infty}\frac{(-1)^n}{2^n}$	27	$\sum_{n=0}^{\infty}\frac{1}{2^n}$
8	$\sum_{n=0}^{\infty}\frac{(-1)^n}{n!}$	18	$\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n}$	28	$\sum_{n=1}^{\infty}\frac{1}{n^{2k}}$
9	$\sum_{n=1}^{\infty}\frac{nx}{(2n+1)!}$	19	$\sum_{n=1}^{\infty}\frac{x^n}{(2n+1)!}$	29	$\sum_{n=1}^{\infty}\frac{(x+1)^3}{(4+n)!}$
10	$\sum_{n=2}^{\infty}\frac{1}{n(n-1)}$	20	$\sum_{n=1}^{\infty}\frac{1}{n(n+2)}$	30	$\sum_{n=1}^{\infty}\frac{x^{2n}}{(2n-1)^2}$

Методика выполнения

1) Расположение, вид и имена метки, картинки и командной кнопки на форме **Form1** представлены на Рис. 7-1.

2) Изображение формулы разместить на форме с помощью элемента **PicrureBox**, предварительно сохранив изображение формулы в файле *ряд.png*, а затем указать этот файл как значение свойства **Image** для **picrureBox1**.

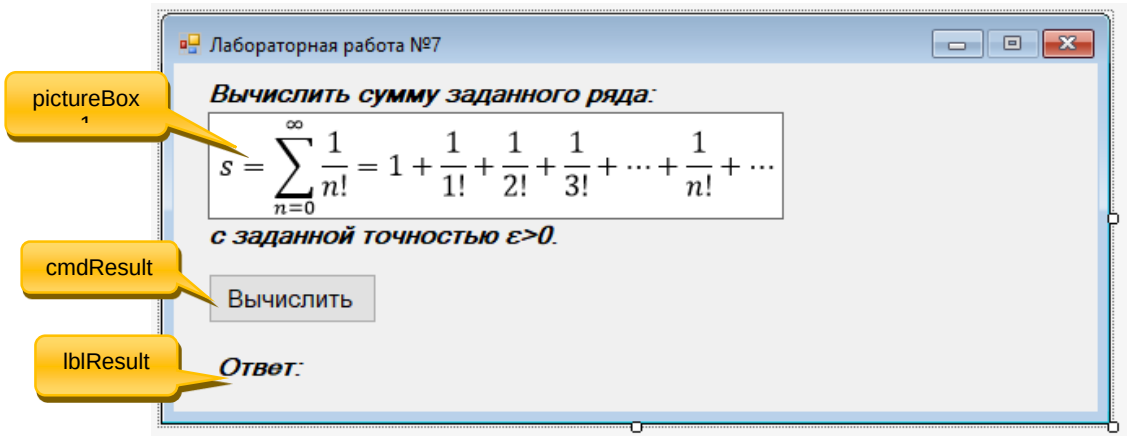

Рис. 7-1 – Конструктор **Form1**

3) Программный код для **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
{
        string s = Microsoft.VisualBasic.Interaction.InputBox("Введите точность
                                                вычислений", "Ввод данных", "0,001");
        if (s == "") { MessageBox.Show("Некоректный ввод данных!", "Ввод данных",
                                MessageBoxButtons.OK,MessageBoxIcon.Error); return; }
        double accuracy = Convert.ToDouble(s);
        double sum = 0;
        int n = 0;
        double sl = 1;
        while(Math.Abs(sl)>= accuracy)
        {
                sum = sum + sl;
                n++;
                sl = sl / n;
        }
        lblResult.Text = "Ответ: s = "+sum.ToString()+ " с точностью ε = "+
                                        accuracy+ " при n = "+ Convert.ToString(n-1);
}
```

4) Вид проекта после нажатия на кнопку «*Вычислить*» представлен на Рис. 7-2.

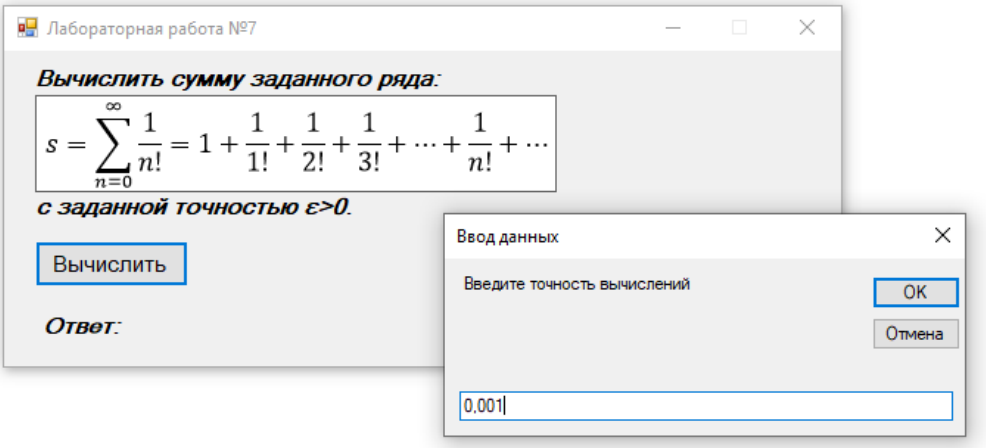

Рис. 7-2 – Окно **InputBox**

6) На Рис. 7-3 и Рис. 7-4 представлен вид проекта после ввода в диалоговое окно значения аргумента и нажатия на командную кнопку «***OK***».

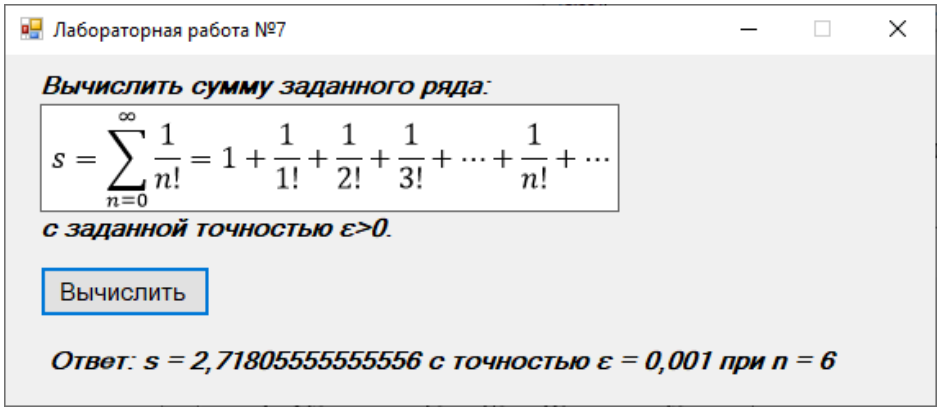

Рис. 7-3 – Результат с точностью ε = 0,001

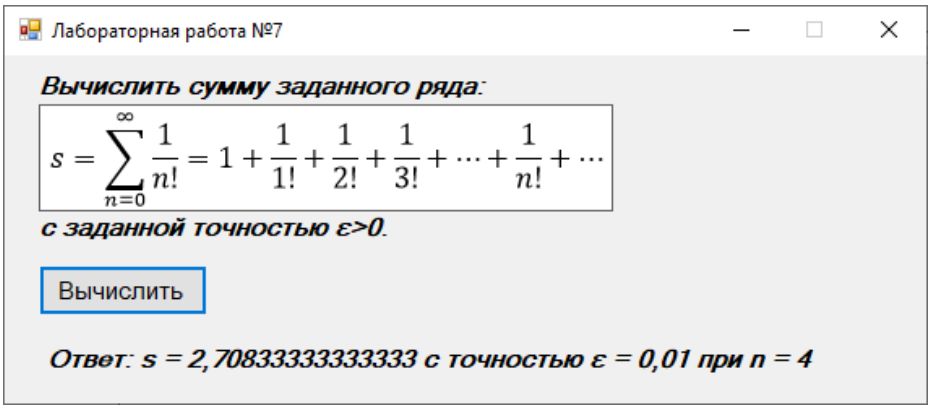

Рис. 7-4 – Результат с точностью ε = 0,01

Лабораторная работа 8. Одномерные массивы. Ввод с клавиатуры. Использование элемента NumericUpDown

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления суммы и среднего значения элементов заданного массива (см. Приложение 7. Массивы в C#).

2. Задать размерность одномерного массива с помощью элемента **NumericUpDown** (см. Приложение 2. Элементы управления).

3. Задать размерность и тип массива в программном коде.

4. Организовать ввод элементов массива с помощью диалогового окна ввода информации **InputBox**.

5. Вывести развернутый результат, используя элемент **TextBox.**

6. Выполнить пп.1-5 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Дан одномерный массив (вектор) $A(n)$. Вычислить:

1. сумму элементов вектора	2. произведение элементов вектора	3. сумму элементов, стоящих на четных местах
4. сумму элементов, стоящих на нечетных местах	5. произведение элементов, стоящих на четных местах	6. произведение элементов, стоящих на нечетных местах
7. среднюю величину положительных элементов	8. среднюю величину отрицательных элементов	9. число положительных элементов вектора
10. число отрицательных элементов вектора	11. число нулевых элементов вектора	12. сумму положительных элементов вектора
13. сумму отрицательных элементов вектора	14. произведение положительных элементов вектора	15. произведение отрицательных элементов вектора
16. минимальный элемент вектора	17. максимальный элемент вектора	18. сумму элементов, имеющих индексы кратные 3
19. сумму элементов, имеющих индексы кратные 4	20. произведение элементов, имеющих индексы кратные 3	21. произведение элементов, имеющих индексы кратные 4
22. среднее значение элементов, имеющих индексы кратные 3	23. среднее значение элементов, имеющих индексы кратные 4	24. сумму минимального и максимального элементов
25. разность минимального и максимального элементов	26. произведение минимального и максимального элементов	27. минимальный среди элементов, имеющих чётные индексы
28. минимальный среди элементов, имеющих нечётные индексы	29. максимальный среди элементов, имеющих нечётные индексы	30. максимальный среди элементов, имеющих чётные индексы

Методика выполнения

1) Разместите на форме проекта следующие элементы (см. Рис. 8-1).

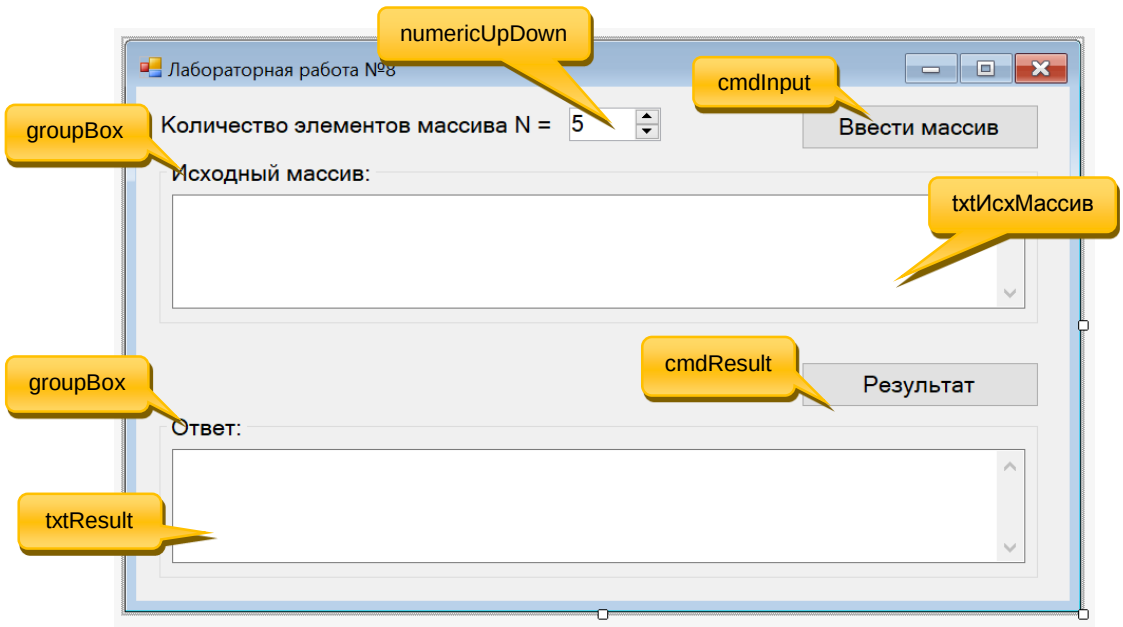

Рис. 8-1 – Конструктор формы проекта *Занятие* 8

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
numericUpDown1	Minimum	1
	Maximum	1000
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	5
txtИсхМассив, txtResult	Multiline	True
	ScrollBars	Vertical
	ReadOnly	True
	Text	<пусто>
cmdResult	Enabled	False

3) Далее объявим целочисленный массив **a[]** в классе формы (Рис. 8-2). В этом случае он **будет доступен всем фрагментам кода** нашего приложения.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    int[] a;
```

Рис. 8-2 – Объявление массива в классе формы

4) Программный код для кнопки **cmdInput**:

```
private void cmdInput_Click(object sender, EventArgs e)
{
        txtИсхМассив.Text = "";
        txtResult.Text = "";
        n = (int)numericUpDown1.Value;
        for(int i=0;i<=n-1;i++)
        {
           string s= Microsoft.VisualBasic.Interaction.InputBox("Введите элемент
                                          массива a["+i+"]=", "Ввод данных", "");
           try
           {
              a[i] = Convert.ToInt32(s);
              txtИсхМассив.Text += s + " ";
           }
           catch
           {
              MessageBox.Show("Некоректный ввод данных!", "Ввод данных",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
              txtИсхМассив.Text = ""; return;
           }
        }
```

```
        cmdResult.Enabled = true;
}
```

5) Вид проекта после нажатия на кнопку «*Ввести массив*» представлен на Рис. 8-3.

6) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
      double sum = 0;
      for (int i = 0; i <= a.Length - 1; i++) sum += a[i];
      double avg = sum / a.Length;
      txtResult.Text = "Сумма элементов массива: " + sum + "\r\n" +
      "Среднее арифметическое элементов массива: " + avg.ToString();
    }
```

7) Вид проекта после нажатия на кнопку «*Результат*» представлен на Рис. 8-4.

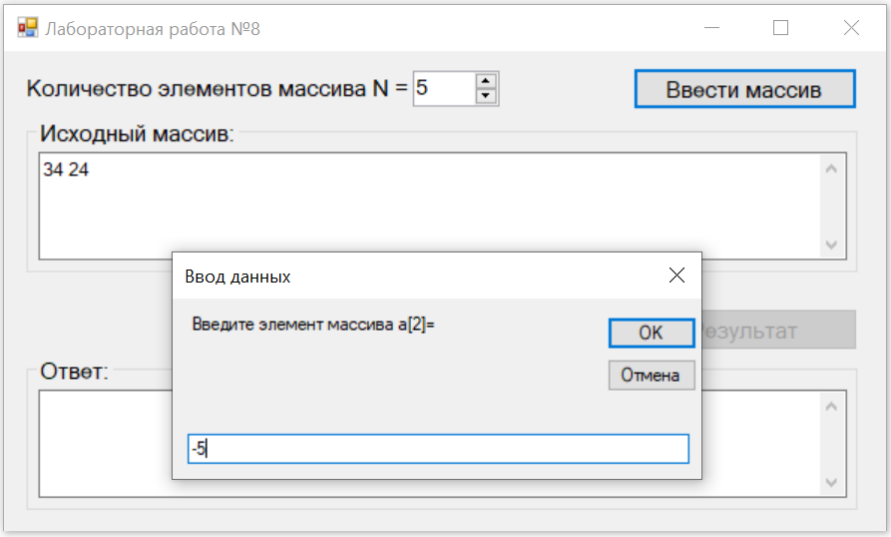

Рис. 8-3 – Старт проекта (ввод элементов массива)

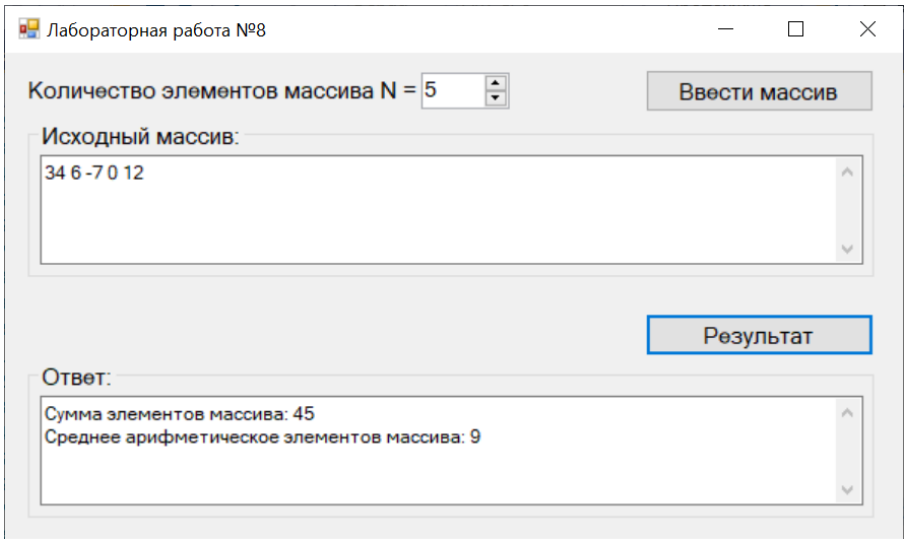

Рис. 8-4 – Результаты вычислений

8) Наконец, сделаем так, чтобы при попытке изменить размер массива все элементы окна приложения «сбрасывались»:

```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {
        txtИсхМассив.Text = "";
        txtResult.Text = "";
        cmdResult.Enabled = false;
```

Лабораторная работа 9. Одномерные массивы. Класс *Random*

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления элементов массива $B(n)$.

2. Задать размерность одномерного массива $A(n)$ с помощью элемента **NumericUpDown**.

3. Ввести элементы массива $A(n)$ с помощью генератора псевдослучайных чисел (см. Приложение 7. Массивы в C#).

4. Образовать массив $B(n)$ по правилу:

$$b_0 = a_0,$$
$$b_1 = a_0 + a_1,$$
$$\ldots\ldots$$
$$b_{n-1} = a_0 + a_1 + \ldots + a_{n-1}.$$

5. Выполнить пп.1-4 для своего варианта (см. **Варианты заданий**).

Варианты заданий

Дан вектор $A(n)$. Образовать новый массив $B(n)$ по правилу:

1) $b_0 = a_0, b_1 = a_1, b_2 = a_2,$ $b_3 = a_0 a_1 a_2,$ …… $b_{n-1} = a_{n-4} a_{n-3} a_{n-2}$	2) $b_0 = a_0,$ $b_1 = a_0 / a_1,$ …… $b_{n-1} = a_{n-2} / a_{n-1}$	3) $b_0 = a_{n-1},$ $b_1 = a_{n-2},$ …… $b_{n-1} = a_0$
4) $b_0 = a_0,$ $b_1 = 2a_1,$ …… $b_{n-1} = na_{n-1}$	5) $b_0 = (n-1)a_{n-1},$ $b_1 = (n-2)a_{n-2},$ …… $b_{n-1} = a_0$	6) $b_0 = (n-1)a_0,$ $b_1 = (n-2)a_1,$ …… $b_{n-1} = a_{n-1}$
7) $b_0 = a_0,$ $b_1 = a_0 + a_1,$ …… $b_{n-1} = a_{n-2} + a_{n-1}$	8) $b_0 = 0,$ $b_1 = a_0 + a_1,$ …… $b_{n-1} = a_{n-2} + (n-1)a_{n-1}$	9) $b_0 = 3a_0,$ $b_1 = 3a_0 / a_1,$ …… $b_{n-1} = 3a_{n-2} / a_{n-1}$

10) $b_0 = a_0,$ $b_1 = \max(a_0, a_1),$ $b_2 = \max(a_0, a_1, a_2),$ …… $b_{n-1} = \max(a_0, a_1, \ldots, a_{n-1})$	11) $b_0 = a_0,$ $b_1 = \min(a_0, a_1),$ $b_2 = \min(a_0, a_1, a_2),$ …… $b_{n-1} = \min(a_0, a_1, \ldots, a_{n-1})$	12) $b_0 = a_0,$ $b_1 = \text{avg}(a_0, a_2),$ $b_2 = \text{avg}(a_1, a_3),$ …… $b_{n-2} = \text{avg}(a_{n-3}, a_{n-1}),$ $b_{n-1} = a_{n-1}$

Методика выполнения

2) Разместите на форме проекта следующие элементы (см. Рис. 9-1).

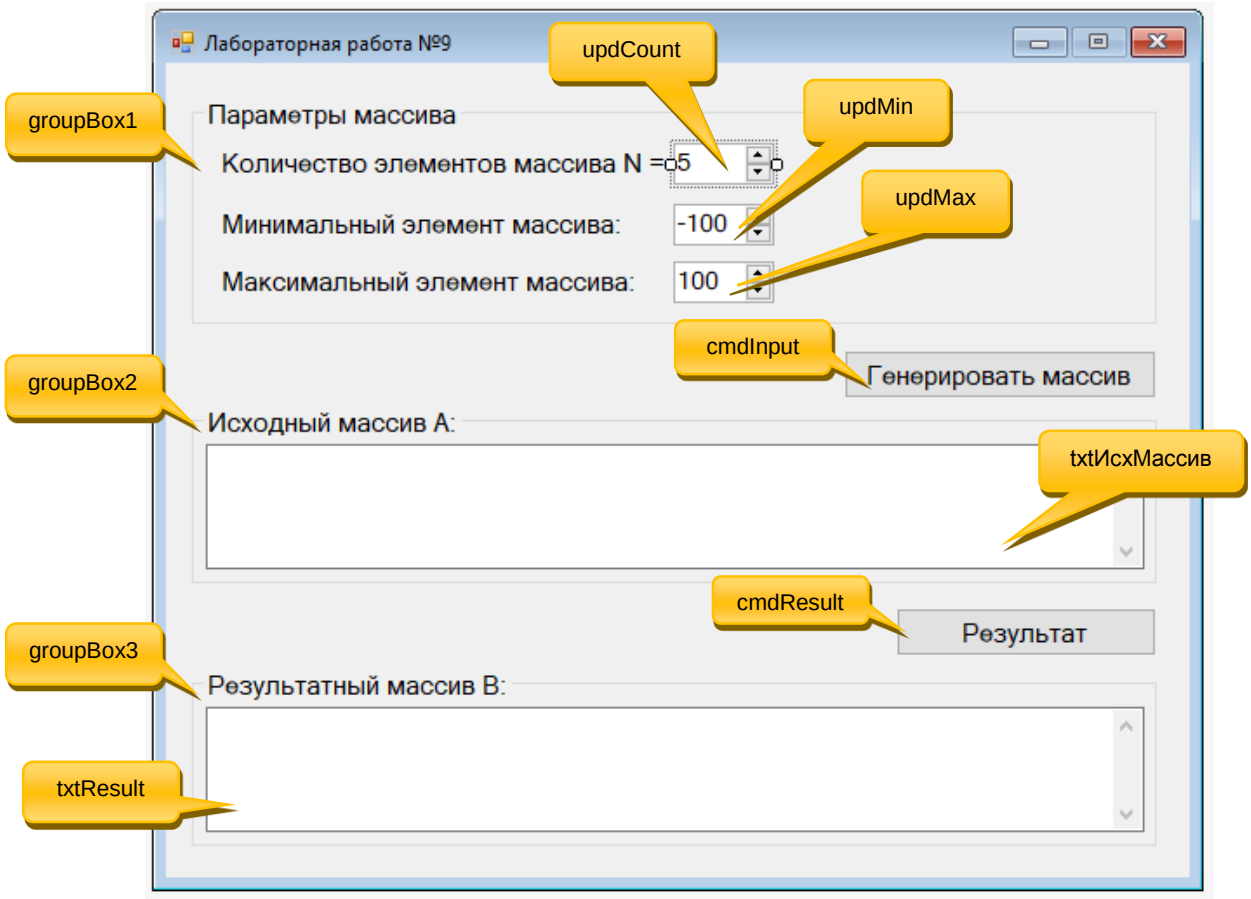

Рис. 9-1 – Конструктор формы проекта *Занятие* 9

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
updCount	Minimum	1
	Maximum	1000
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	5
updMin	Minimum	-10000
	Maximum	10000

Объект	Свойство	Значение
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	-100
updMax	Minimum	-10000
	Maximum	10000
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	100
txtИсхМассив, txtResult	Multiline	True
	ScrollBars	Vertical
	ReadOnly	True
	Text	<пусто>
cmdResult	Enabled	False

3) Далее объявим пару целочисленных массивов **a[]** и **b[]** в классе формы (Рис. 9-2). В этом случае они **будут доступны всем фрагментам кода** нашего приложения.

```
public Form1()
{
    InitializeComponent();
}

int[] a;
int[] b;
```

Рис. 9-2 – Объявление массивов

4) Программный код для кнопки **cmdInput**:

```
private void cmdInput_Click(object sender, EventArgs e)
{
        txtИсхМассив.Text = "";
        txtРезМассив.Text = "";
        int n = (int)updCount.Value;
        a = new int[n];
        b = new int[n];
        int minValue= (int)updMin.Value;
        int maxValue = (int)updMax.Value;
        if(minValue>maxValue)
        {
                MessageBox.Show("Некоректный диапазон данных!", "Ввод данных",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
        }
        Random rnd = new Random();
```

```
        for (int i=0;i<= a.Length - 1;i++)
        {
                a[i] = rnd.Next(minValue, maxValue + 1);
                txtИсхМассив.Text += a[i].ToString() + " ";
        }
        cmdResult.Enabled = true;
}
```

5) Вид проекта после нажатия на кнопку «*Генерировать массив*» представлен на Рис. 9-3.

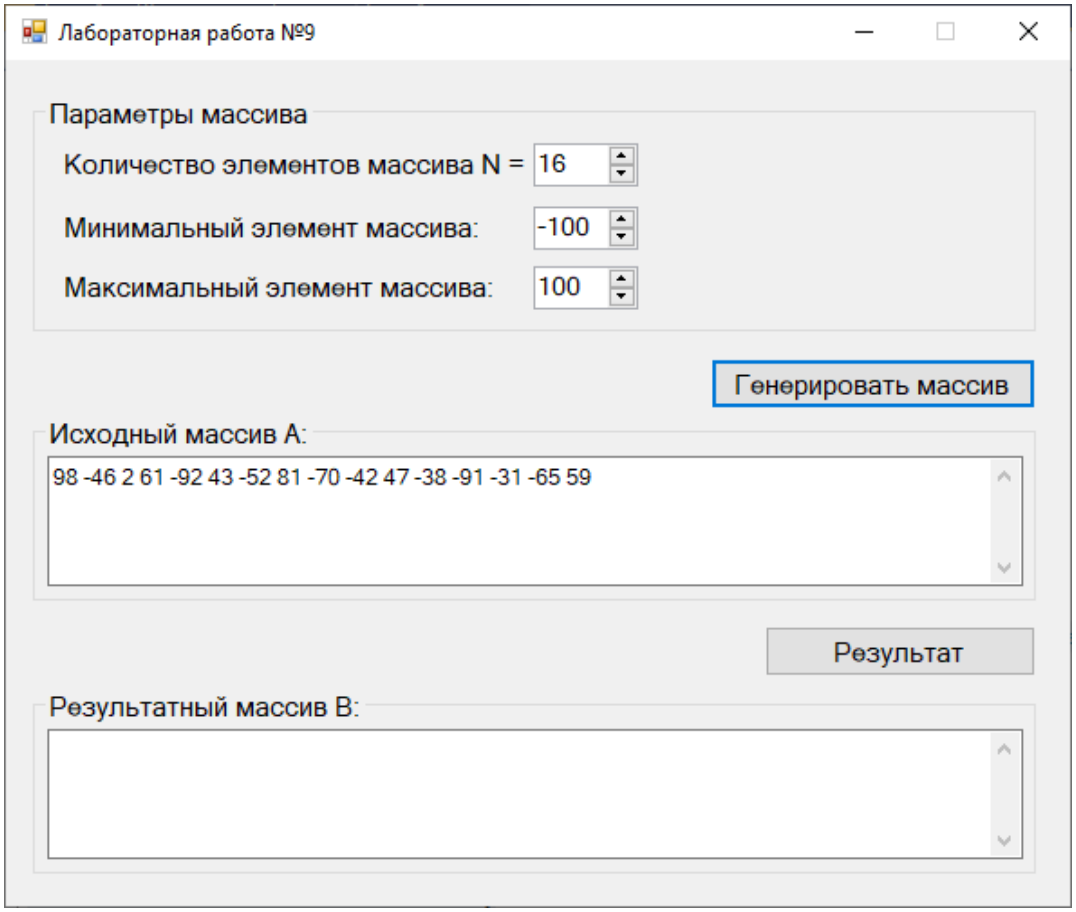

Рис. 9-3 – Массив сгенерирован!

6) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
{
        txtResult.Text = "";
        b[0]=a[0];
        txtResult.Text += b[0].ToString() + " ";
        for (int i = 1; i <= b.Length - 1; i++)
        {
                b[i] =b[i-1]+ a[i];
                txtResult.Text += b[i].ToString() + " ";
        }
}
```

7) Вид проекта после нажатия на кнопку «*Результат*» представлен на Рис. 9-4.

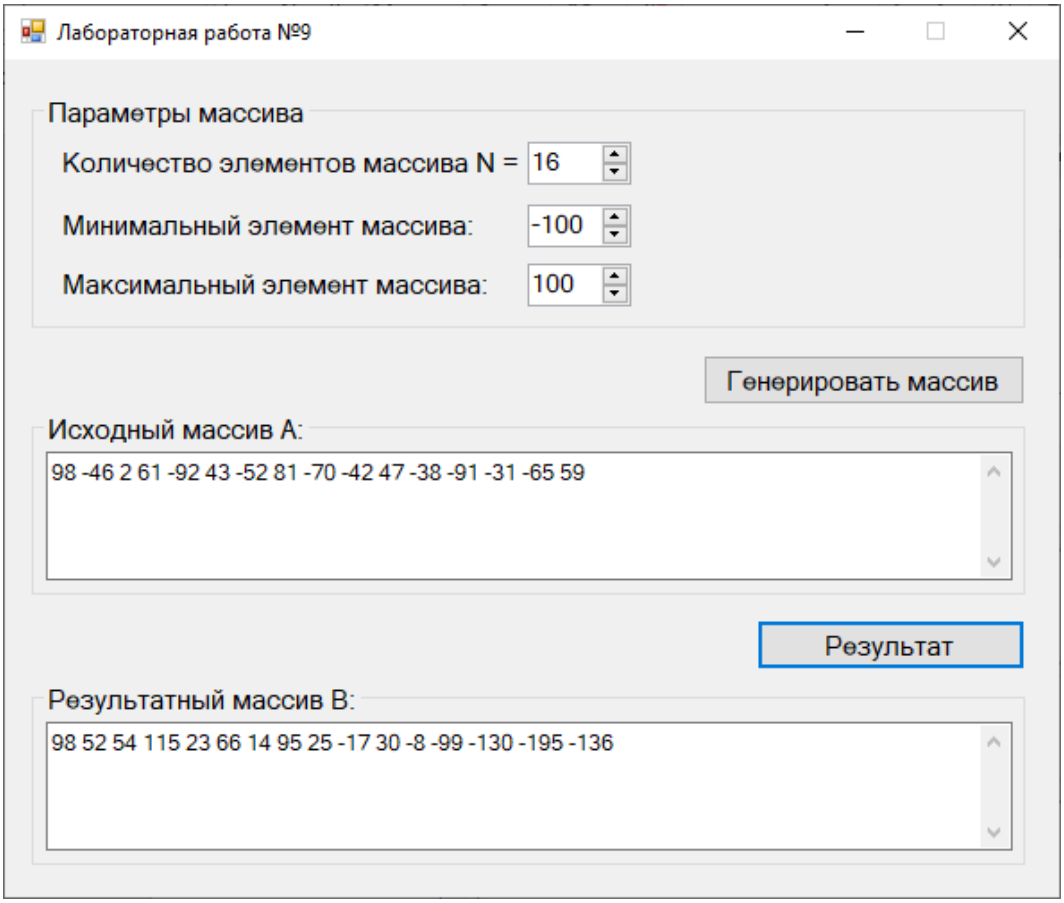

Рис. 9-4 – Результат работы приложения

8) Наконец, сделаем так, чтобы при попытке изменить размер массива все элементы окна приложения «сбрасывались»:

```
private void updCount_ValueChanged(object sender, EventArgs e)
    {
        txtИсхМассив.Text = "";
        txtResult.Text = "";
        cmdResult.Enabled = false;
    }
```

9) Аналогичным образом обработайте элементы **updMin** и **updMax**.

Лабораторная работа 10. Класс *Array* и массивы

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для вычисления наибольший элемента массива $A(n)$. Если их несколько, то удалить из массива все вхождения наибольшего элемента кроме первого.

Использовать при этом (обязательно!) методы класса **Array** для управления массивом (см. Приложение 7. Массивы в C#).

2. Задать размерность одномерного массива $A(n)$ с помощью элемента **NumericUpDown**.

3. Ввести элементы массива $A(n)$ с помощью генератора псевдослучайных чисел (см. Приложение 7. Массивы в C#).

4. Вывести элементы изменённого массива $A(n)$ в объект **TextBox**.

5. Вывести сообщение о значении наибольшего элемента и их количестве.

6. Выполнить пп.1-5 для своего варианта (см. **Варианты заданий**).

Варианты заданий

1. Объединить два массива в один неубывающий массив.	2. Дан вектор $A(n)$, содержащий нулевые элементы. Уплотнить его, выбросив нулевые элементы.	3. Дан вектор $A(n)$. Если рядом стоят равные элементы, то вставить между ними нулевой элемент.
4. Из вектора $A(n)$ удалить все элементы $A(n) < B$, где B – произвольное положительное число/	5. В массиве $A(n)$ вычеркнуть элементы, имеющие значения 6, 7, 11, 15, 25, сдвигая оставшиеся.	6. После каждой пары элементов массива $A(n)$, расположенных в порядке возрастания, вставить их сумму.
7. После каждой пары элементов вектора $A(n)$ вставить наибольшее из них.	8. Между каждой парой элементов вектора $A(n)$ вставить нуль, сдвигая элементы вектора вправо.	9. После каждой пары элементов вектора $A(n)$ вставить их сумму.
10. После каждой пары элементов вектора $A(n)$ вставить наименьшее из них.	11. После каждой пары элементов вектора $A(n)$ вставить их среднее арифметическое.	12. Из вектора $A(n)$ удалить все отрицательные числа.

Методика выполнения

1) Разместите на форме проекта следующие элементы (см. Рис. 10-1).

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
updCount	Minimum	1
	Maximum	1000
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	5
updMin	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	-10
updMax	Minimum	-100
	Maximum	100

Объект	Свойство	Значение
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	10
txtИсхМассив, txtResult	Multiline	True
	ScrollBars	Vertical
	ReadOnly	True
	Text	<пусто>
cmdResult	Enabled	False

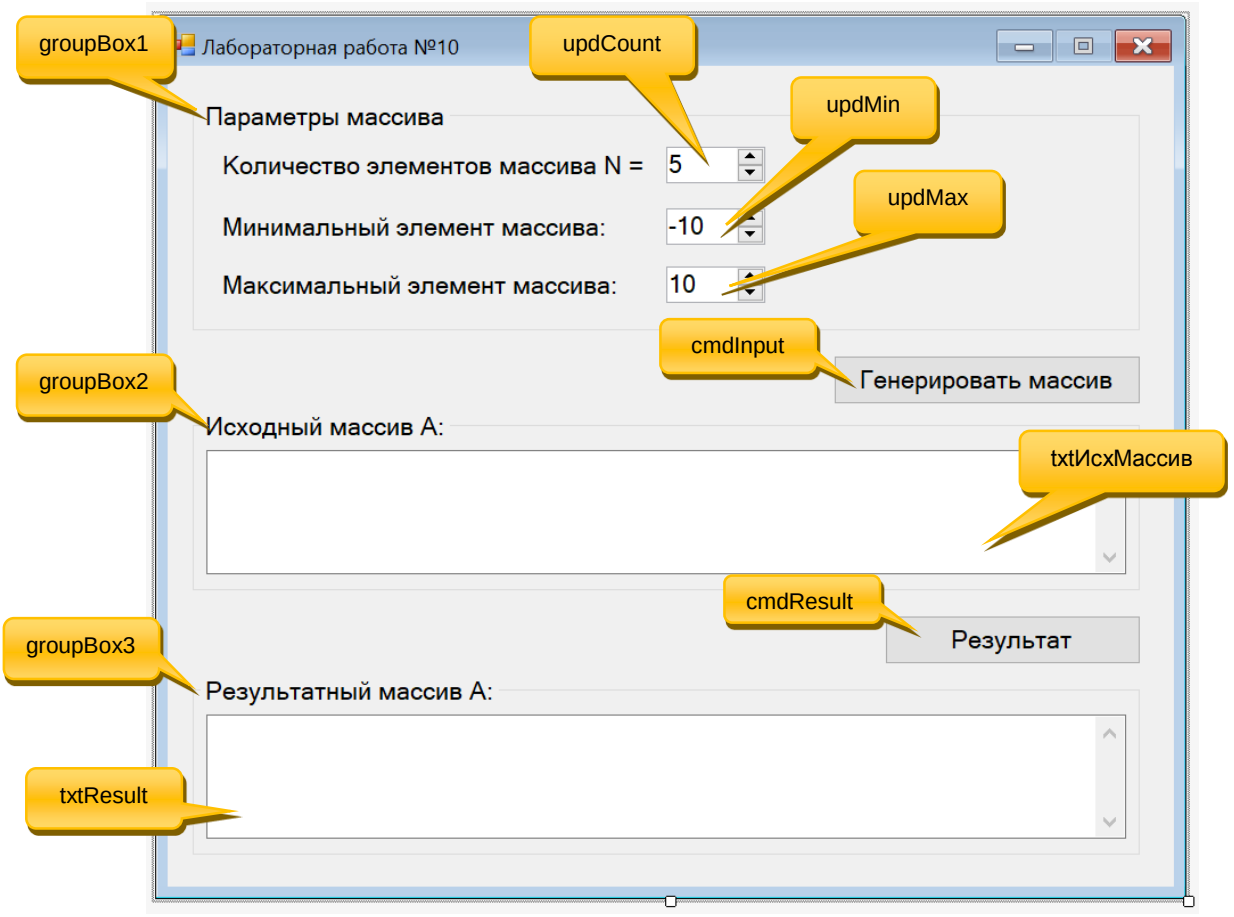

Рис. 10-1 – Конструктор формы проекта *Занятие* 10

3) Далее объявим целочисленный массив **a[]** в классе формы (Рис. 10-2). В этом случае он **будет доступен всем фрагментам кода** нашего приложения.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    int[] a;
```

Рис. 10-2 – Объявление исходного массива

4) Программный код для кнопки **cmdInput**:

```
private void cmdInput_Click(object sender, EventArgs e)
{
        txtИсхМассив.Text = "";
        txtРезМассив.Text = "";
        int n = (int)updCount.Value;
        a = new int[n];
        int minValue= (int)updMin.Value;
        int maxValue = (int)updMax.Value;
        if(minValue>maxValue)
        {
                MessageBox.Show("Некоректный диапазон данных!", "Ввод данных",
                                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
        }
        Random rnd = new Random();
        for (int i=0;i<= a.Length - 1;i++)
        {
                a[i] = rnd.Next(minValue, maxValue + 1);
                txtИсхМассив.Text += a[i].ToString() + " ";
        }
        cmdResult.Enabled = true;
}
```

5) Вид проекта после нажатия на кнопку «*Генерировать массив*» представлен на Рис. 10-3.

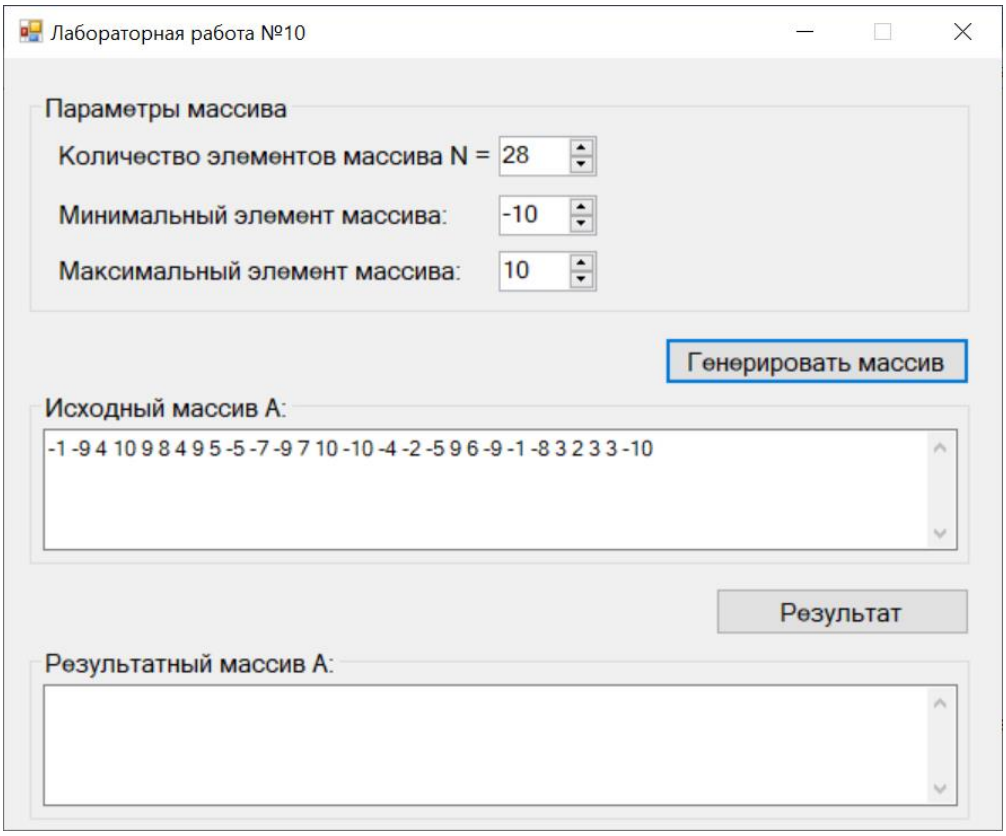

Рис. 10-3 – Исходные данные

6) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
{
        int indMax = 0;
        for (int i = 1; i <= a.Length - 1; i++)
                        if (a[i] > a[indMax]) indMax = i;
        int k = 0;
        while(true)
        {
                int y=Array.FindIndex(a, indMax + 1, x=>(x==a[indMax]));
                if (y == -1) break;
                k++;
                Array.ConstrainedCopy(a, y + 1, a, y,a.Length-y-1);
                Array.Resize(ref a, a.Length - 1);
        }
        txtResult.Text = "";
        for (int i = 0; i <= a.Length - 1; i++)
                        txtResult.Text += a[i].ToString() + " ";
        MessageBox.Show("Max="+a[indMax].ToString()+
                ". Пришлось удалить "+k.ToString()+" элем. массива", "На заметку",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        cmdResult.Enabled = false;
}
```

7) Вид проекта после нажатия на кнопку «*Результат*» представлен на Рис. 10-4.

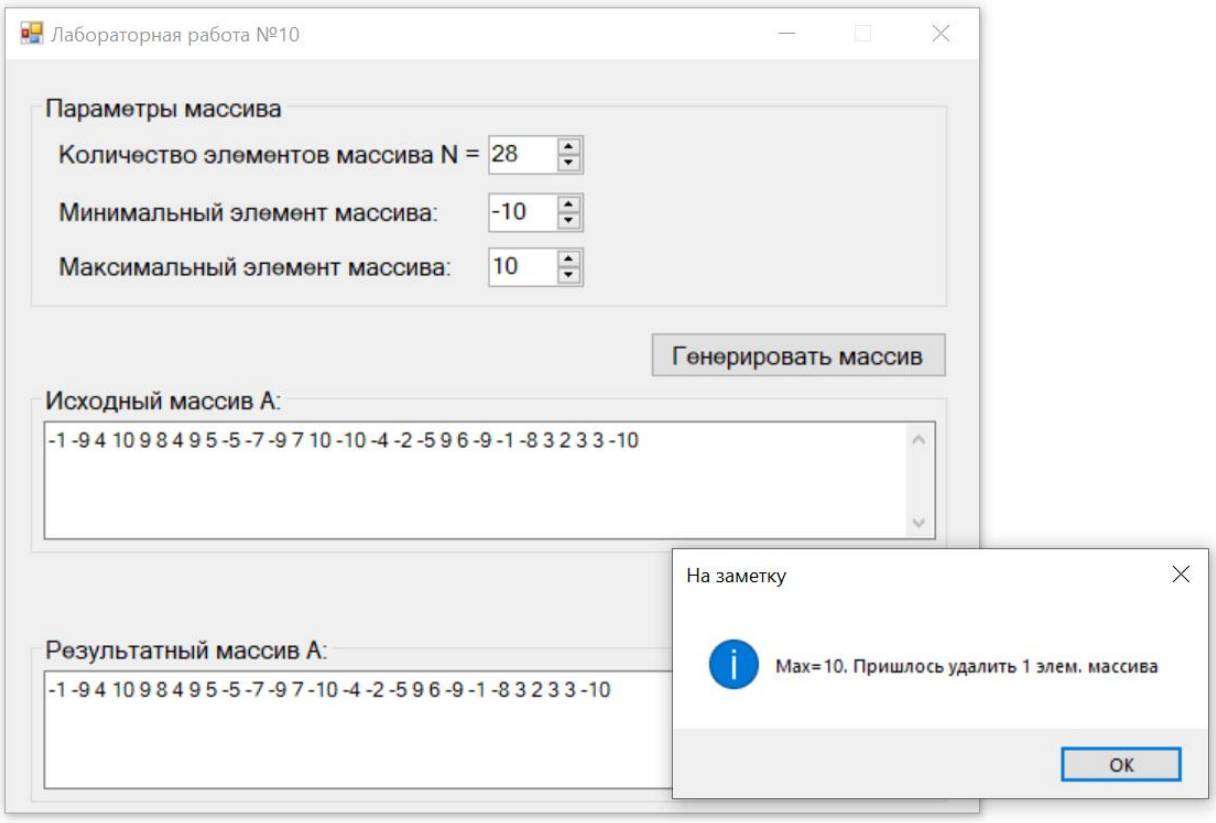

Рис. 10-4 – Результат обработки массива

8) Наконец, сделаем так, чтобы при попытке изменить размер массива все элементы окна приложения «сбрасывались»:

```
private void updCount_ValueChanged(object sender, EventArgs e)
    {
        txtИсхМассив.Text = "";
        txtResult.Text = "";
        cmdResult.Enabled = false;
    }
```

9) Аналогичным образом обработайте элементы **updMin** и **updMax**.

Лабораторная работа 11. Многомерные массивы. Объект DataGridView

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Разработать графический интерфейс и программный код приложения для генерации элементов двумерного массива $A(n,m)$ и вывода в объект **DataGridView** (см. Приложение 2. Элементы управления).

2. Предусмотреть возможность выбора режима «*Уникальные значения*» (значения в матрице не повторяются) с помощью элемента **CheckBox** (см. Приложение 2. Элементы управления).

3. Задать размерность (количество строк n и столбцов m) и диапазон элементов будущей матрицы с помощью элементов **NumericUpDown**.

4. Выполнить пп. 1-3 для своего варианта (см. **Варианты заданий**).

Варианты заданий

1. $n=3;\ m=4;$ $\min=-22;\ \max=8$	2. $n=13;\ m=14;$ $\min=22;\ \max=88$	3. $n=8;\ m=6;$ $\min=-2;\ \max=8$
4. $n=6;\ m=9;$ $\min=-6;\ \max=9$	5. $n=5;\ m=4;$ $\min=-7;\ \max=4$	6. $n=8;\ m=4;$ $\min=-2;\ \max=18$
7. $n=4;\ m=7;$ $\min=-12;\ \max=7$	8. $n=4;\ m=2;$ $\min=21;\ \max=38$	9. $n=8;\ m=3;$ $\min=6;\ \max=9$
10. $n=2;\ m=14;$ $\min=-2;\ \max=17$	11. $n=3;\ m=6;$ $\min=4;\ \max=10$	12. $n=3;\ m=4;$ $\min=5;\ \max=10$
13. $n=6;\ m=2;$ $\min=-7;\ \max=4$	14. $n=3;\ m=4;$ $\min=-12;\ \max=3$	15. $n=3;\ m=5;$ $\min=-6;\ \max=2$
16. $n=4;\ m=7;$ $\min=3;\ \max=9$	17. $n=4;\ m=4;$ $\min=-6;\ \max=5$	18. $n=3;\ m=6;$ $\min=-4;\ \max=7$
19. $n=7;\ m=4;$ $\min=-9;\ \max=4$	20. $n=3;\ m=3;$ $\min=-3;\ \max=2$	21. $n=6;\ m=6;$ $\min=21;\ \max=38$
22. $n=12;\ m=3;$ $\min=-1;\ \max=4$	23. $n=13;\ m=5;$ $\min=14;\ \max=28$	24. $n=7;\ m=7;$ $\min=19;\ \max=44$
25. $n=8;\ m=5;$ $\min=-22;\ \max=-8$	26. $n=8;\ m=3;$ $\min=-33;\ \max=-2$	27. $n=4;\ m=7;$ $\min=44;\ \max=77$

28. $n=6;\ m=8;$ $\min=-32;\ \max=-12$	29. $n=5;\ m=9;$ $\min=56;\ \max=98$	30. $n=7;\ m=5;$ $\min=18;\ \max=67$

Методика выполнения

1) Разместите на форме проекта следующие элементы (см. Рис. 11-1).

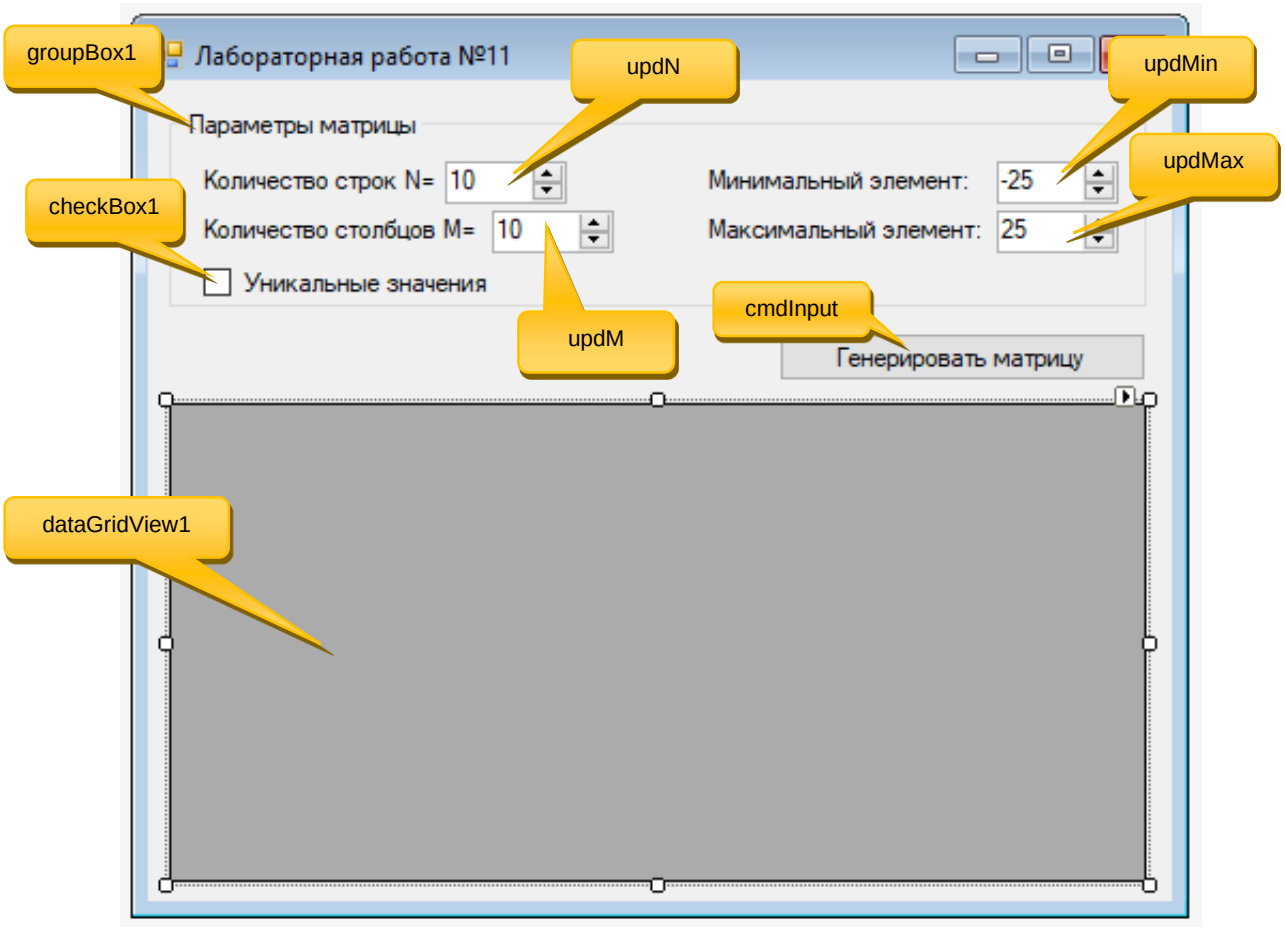

Рис. 11-1 – Конструктор формы проекта *Занятие* 11

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
updN, updM	Minimum	1
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	10
updMin	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	-25
updMax	Minimum	-100

Объект	Свойство	Значение
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	25
dataGridView1	AutoSizeColumnsMode	AllCels
	AutoSizeRowsMode	AllCels
	ColumnHeadersVisible	False
	RowHeadersVisible	False
	ReadOnly	True

3) Далее объявим целочисленный двумерный массив **a[,]** в классе формы (Рис. 11-2). В этом случае он **будет доступен всем фрагментам кода** нашего приложения. Кроме этого, здесь же объявим пару целочисленных переменных *n* (количество строк) и *m* (количество столбцов).

```
public Form1()
{
    InitializeComponent();
}
int[,] a;
int n, m;
```

Рис. 11-2 – Объявление необходимых переменных

4) Программный код для кнопки **cmdInput**:

```
private void cmdInput_Click(object sender, EventArgs e)
    {
        n = (int)updN.Value;
        m = (int)updM.Value;
        a = new int[n, m];
        int minVal = (int)updMin.Value;
        int maxVal = (int)updMax.Value;
        if (checkBox1.Checked && (n * m > maxVal - minVal + 1))
        {
            MessageBox.Show("Невозможно сгенерировать уникальные значения!",
                "Ошибка ввода данных", MessageBoxButtons.OK,MessageBoxIcon.Error);
            checkBox1.Checked = false;
        }
        dataGridView1.RowCount = n;
        dataGridView1.ColumnCount = m;
        Random rnd = new Random();
        List<int> v = new List<int>();
        for (int i=0;i<n;i++)
            for (int j = 0; j < m; j++)
            {
                while (true)
                {
                    a[i, j] = rnd.Next(minVal, maxVal + 1);
```

```
                if (!checkBox1.Checked) break;
                if (!v.Contains(a[i, j])) { v.Add(a[i, j]); break; }
            }
            dataGridView1.Rows[i].Cells[j].Value = a[i, j];
        }
    v.Clear();
}
```

5) Вид проекта после нажатия на кнопку «*Генерировать матрицу*» представлен на Рис. 11-3 – Рис. 11-5.

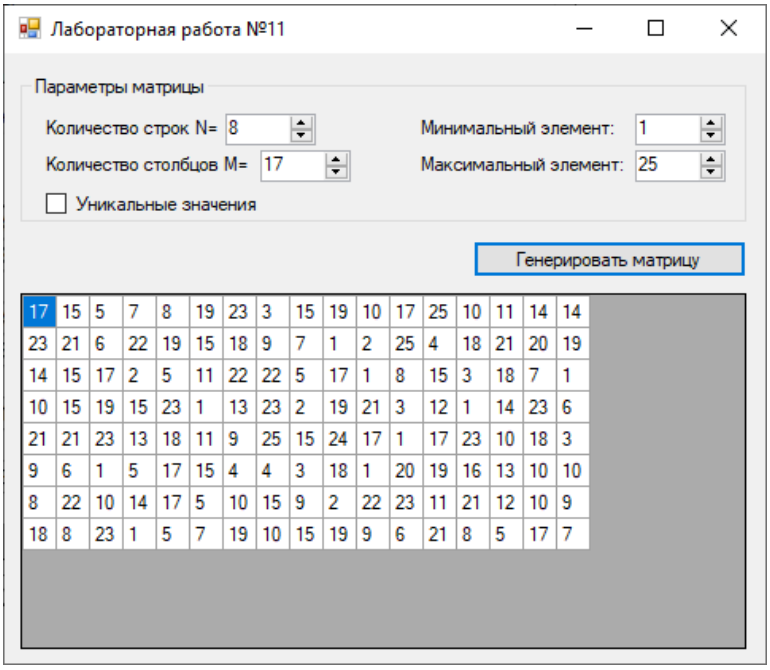

Рис. 11-3 – Значения в матрице могут повторяться

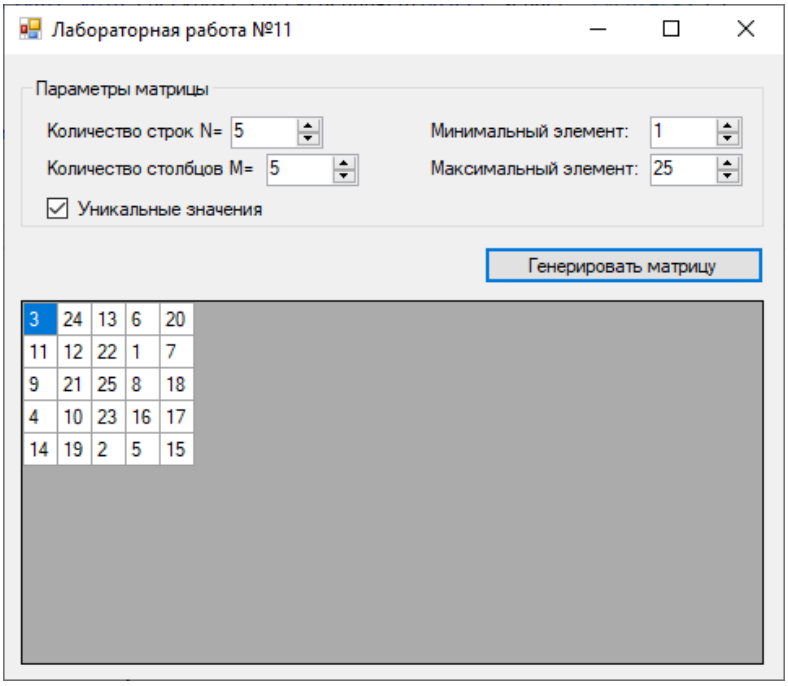

Рис. 11-4 – Значения в матрице уникальные

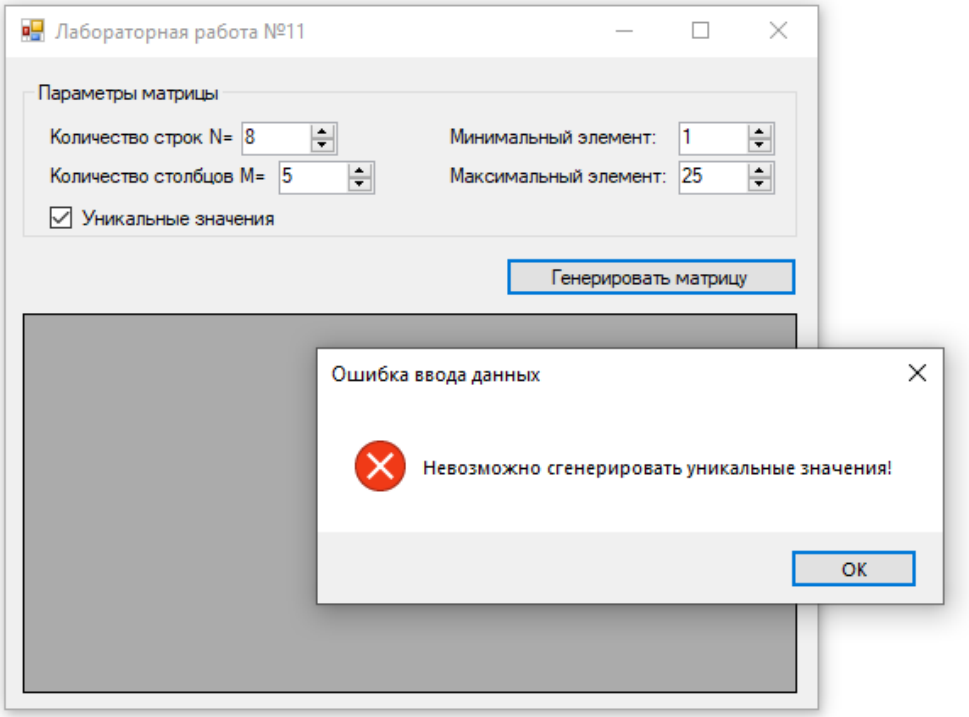

Рис. 11-5 – Уникальность значений невозможна

6) Наконец, сделаем так, чтобы при попытке изменить количество строк в матрице содержимое элемента **DataGridView** очищалось:

```
private void updN_ValueChanged(object sender, EventArgs e)
    {
      dataGridView1.Columns.Clear();
    }
```

7) Аналогичным образом обработайте элементы **updM**, **updMin**, **updMax** и **checkBox1**.

Лабораторная работа 12. Обработка многомерных массивов. Элементы NumericUpDown, CheckBox и DataGridView

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Задать размерность n и диапазон $[\min, \max]$ элементов будущей квадратной матрицы с помощью элементов **NumericUpDown**.

2. Ввести элементы матрицы $A(n,n)$ с помощью генератора псевдослучайных чисел.

3. Предусмотреть возможность выбора режима «*Уникальные значения*» (значения в матрице не повторяются) с помощью элемента **CheckBox**.

4. Вывести элементы матрицы $A(n,n)$ в объект **DataGridView** (см. Приложение 2. Элементы управления).

5. Выделить главную диагональ матрицы цветом.

6. Найти сумму элементов матрицы, расположенных на главной диагонали. Результат вывести в объект **Label** на текущей форме.

7. Выполнить аналогичные задания для своего варианта (см. **Варианты заданий**).

Варианты заданий

Для матрицы $A(n,n)$ вычислить:

1. наименьший элемент побочной диагонали при $n=4,\ \min=-22,\ \max=8$	2. наибольший элемент побочной диагонали при $n=14,\ \min=22,\ \max=88$	3. наибольший элемент главной диагонали при $n=6,\ \min=-2,\ \max=8$
4. наибольший элемент побочной диагонали при $n=9,\ \min=-6,\ \max=11$	5. наименьший элемент побочной диагонали при $n=4,\ \min=-7,\ \max=4$	6. отношение суммы элементов главной и побочной диагоналей при $n=4,\ \min=-2,\ \max=18$
7. наибольший элемент главной диагонали при $n=7,\ \min=-12,\ \max=3$	8. отношение суммы элементов побочной и главной диагоналей при $n=2,\ \min=21,\ \max=38$	9. наименьший элемент побочной диагонали при $n=3,\ \min=6,\ \max=9$
10. отношение суммы элементов главной и побочной диагоналей при $n=14,\ \min=-2,\ \max=17$	11. наименьший элемент побочной диагонали при $n=6,\ \min=4,\ \max=10$	12. наибольший элемент побочной диагонали при $n=4,\ \min=5,\ \max=10$
13. наименьший элемент побочной диагонали при $n=2,\ \min=-7,\ \max=4$	14. наибольший элемент главной диагонали при $n=4,\ \min=-12,\ \max=3$	15. отношение суммы элементов главной и побочной диагоналей при $n=5,\ \min=-6,\ \max=2$
16. наибольший элемент побочной диагонали при $n=7,\ \min=3,\ \max=9$	17. наименьший элемент побочной диагонали при $n=4,\ \min=-6,\ \max=5$	18. наибольший элемент главной диагонали при $n=6,\ \min=-4,\ \max=7$
19. отношение суммы элементов главной и побочной диагоналей при $n=4,\ \min=-9,\ \max=4$	20. отношение суммы элементов побочной и главной диагоналей при $n=3,\ \min=-3,\ \max=2$	21. наименьший элемент побочной диагонали при $n=6,\ \min=21,\ \max=38$
22. отношение суммы элементов побочной и главной диагоналей при $n=3,\ \min=-1,\ \max=4$	23. наименьший элемент побочной диагонали при $n=5,\ \min=14,\ \max=28$	24. наибольший элемент побочной диагонали при $n=7,\ \min=19,\ \max=44$
25. наименьший элемент побочной диагонали при $n=5,\ \min=-22,\ \max=-8$	26. наибольший элемент побочной диагонали при $n=3,\ \min=-33,\ \max=-2$	27. наибольший элемент главной диагонали при $n=7,\ \min=44,\ \max=77$
28. отношение суммы элементов главной и побочной диагоналей при $n=8,\ \min=-32,\ \max=-12$	29. наименьший элемент побочной диагонали при $n=9,\ \min=56,\ \max=98$	30. наибольший элемент побочной диагонали при $n=5,\ \min=18,\ \max=67$

Методика выполнения

1) Разместите на форме проекта следующие элементы управления (см. Рис. 12-1).

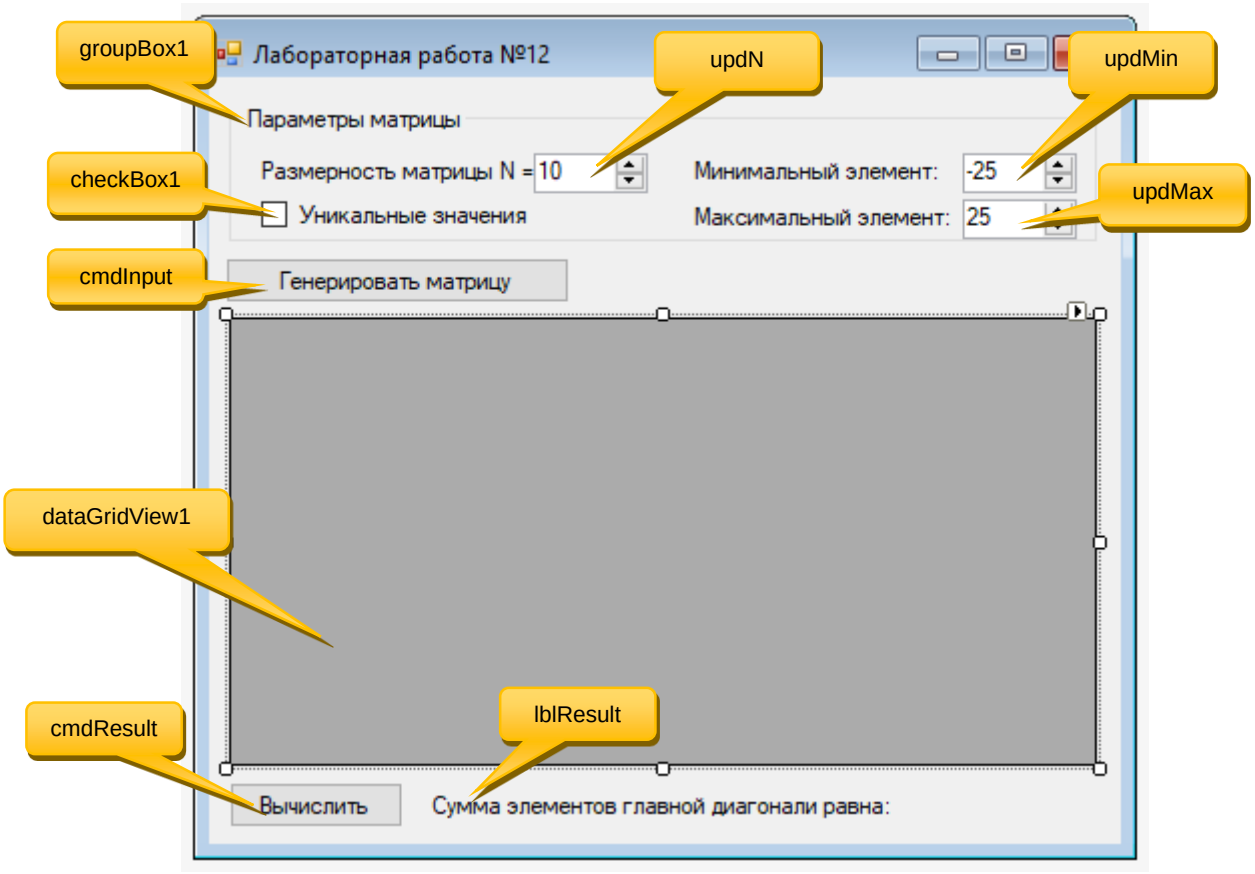

Рис. 12-1 – Конструктор формы проекта *Занятие* 12

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
updN	Minimum	1
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	10
updMin	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	-25
updMax	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	25
dataGridView1	AutoSizeColumnsMode	AllCels
	AutoSizeRowsMode	AllCels
	ColumnHeadersVisible	False
	RowHeadersVisible	False

Объект	Свойство	Значение
	ReadOnly	True
cmdResult	Enabled	False
lblResult	Text	Сумма элементов главной диагонали равна:

3) Далее объявим целочисленный двумерный массив **a[,]** в классе формы (Рис. 12-2). В этом случае он **будет доступен всем фрагментам кода** нашего приложения. Кроме этого, здесь же объявим целочисленную переменную n (размерность матрицы).

```
public Form1()
{
    InitializeComponent();
}
int[,] a;
int n;
```

Рис. 12-2 – Объявление необходимых переменных

4) Программный код для кнопки **cmdInput**:

```
private void cmdInput_Click(object sender, EventArgs e)
    {
        n = (int)updN.Value;
        a = new int[n, n];
        int minVal = (int)updMin.Value;
        int maxVal = (int)updMax.Value;
        if (checkBox1.Checked && (n * n > maxVal - minVal + 1))
        {
            MessageBox.Show("Невозможно сгенерировать уникальные значения!",
                "Ошибка ввода данных", MessageBoxButtons.OK,MessageBoxIcon.Error);
            checkBox1.Checked = false;
        }
        dataGridView1.RowCount = n;
        dataGridView1.ColumnCount = n;
        Random rnd = new Random();
        List<int> v = new List<int>();
        for (int i=0;i<n;i++)
            for (int j = 0; j < n; j++)
            {
                while (true)
                {
                    a[i, j] = rnd.Next(minVal, maxVal + 1);
                    if (!checkBox1.Checked) break;
                    if (!v.Contains(a[i, j]))
                    {
                        v.Add(a[i, j]);
                        break;
                    }
                }
                dataGridView1.Rows[i].Cells[j].Value = a[i, j];
```

```
            if(i==j)dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.Yellow;
        }
    v.Clear();
    cmdResult.Enabled = true;
}
```

5) Вид проекта после нажатия на кнопку «*Генерировать матрицу*» представлен на Рис. 12-3 – Рис. 12-5.

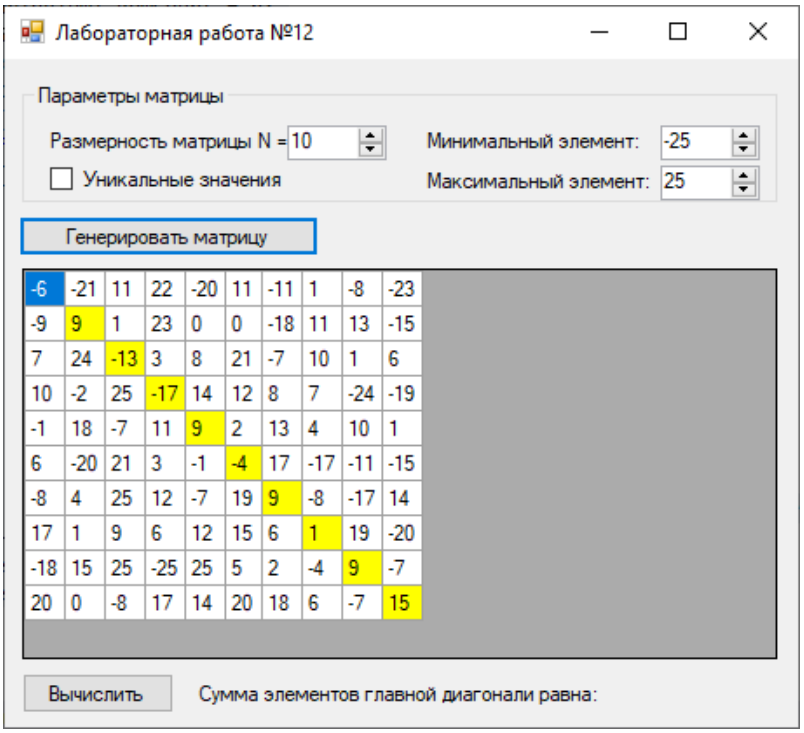

Рис. 12-3 – Значения в матрице могут повторяться

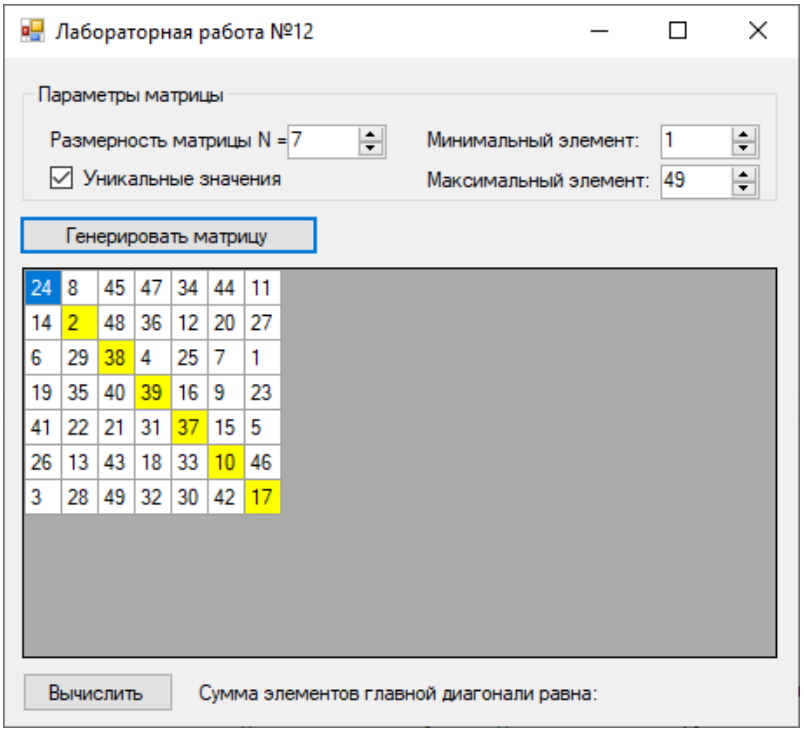

Рис. 12-4 – Значения в матрице уникальные

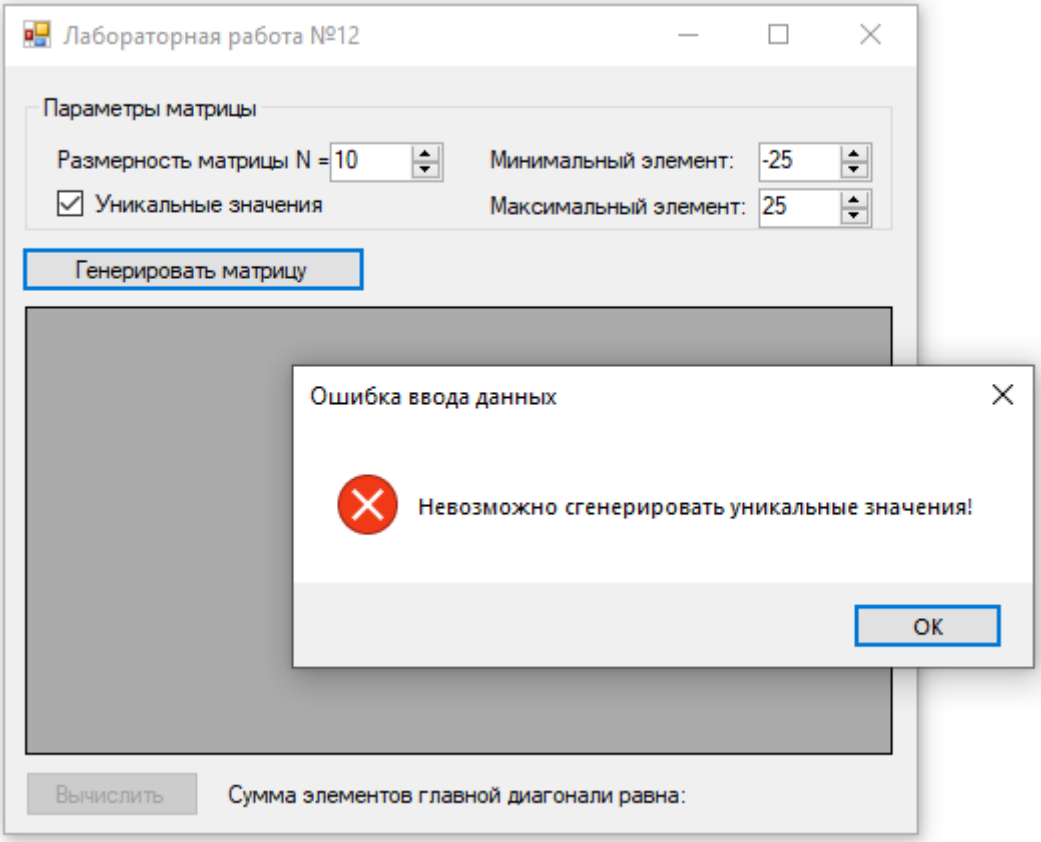

Рис. 12-5 – Уникальность значений невозможна

6) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
    {
        int sum = 0;
        for (int i = 0; i < n; i++) sum += a[i, i];
        lblResult.Text = "Сумма элементов главной диагонали равна: "+sum;
    }
```

7) Вид проекта после нажатия на кнопку «*Вычислить*» представлен на Рис. 12-6.

8) Наконец, сделаем так, чтобы при попытке изменить размерность матрицы содержимое элементов **DataGridView** и **lblResult** очищалось, а кнопка **cmdResult** становилась недоступной:

```
private void updN_ValueChanged(object sender, EventArgs e)
    {
        dataGridView1.Columns.Clear();
        lblResult.Text = "Сумма элементов главной диагонали равна: ";
        cmdResult.Enabled = false;
    }
```

9) Аналогичным образом обработайте элементы **updMin**, **updMax** и **checkBox1**.

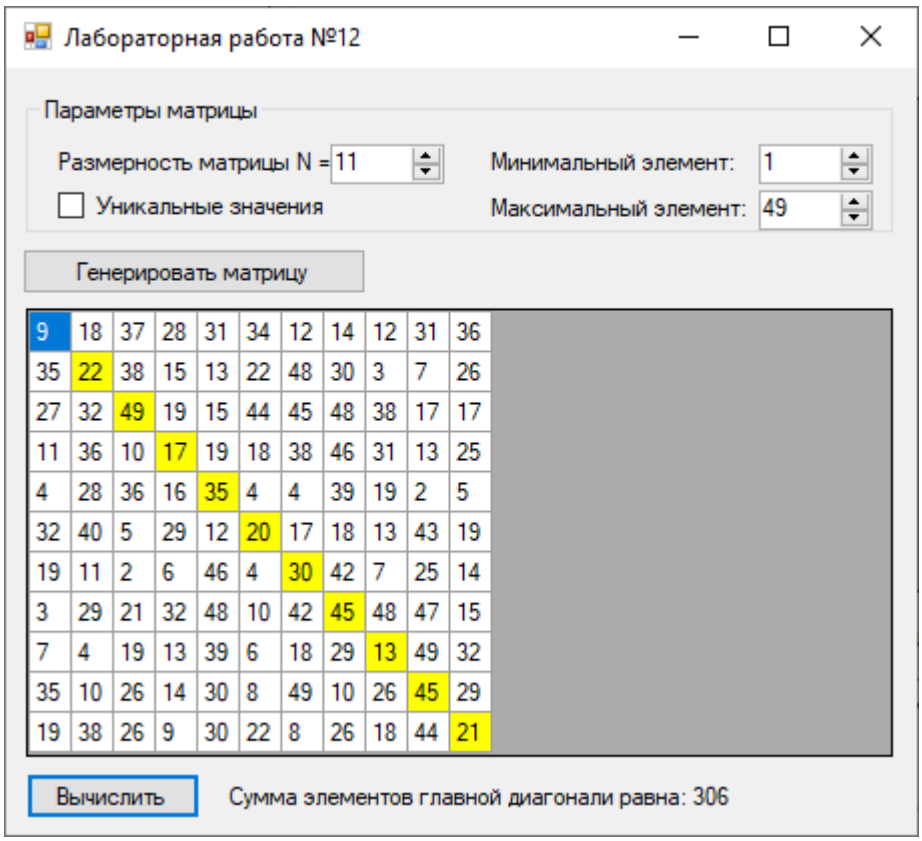

Рис. 12-6 – Результат готов!

Лабораторная работа 13. Построение графика функции с помощью компонента Chart

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Переменная x изменяется на отрезке от –14 до 16 с шагом 2. Оформив вычисление факториала и пользовательской функции $y(x)$ в виде методов-функций, выдать на экран максимальное значение функции $y(x)$, которая вычисляется по правилу:

$$y = \begin{cases} (x-3)! & \textit{при } 3 \le x \le 6, \\ (x-7)! & \textit{при } 8 < x \le 13, \\ 4x & \textit{в остальных случаях.} \end{cases}$$

2. Организовать ввод исходных данных (начального, конечного значения x и шага изменения) с помощью элементов **NumericUpDown**.

3. Вывести результаты вычислений (точки функции) в объект **DataGridView**.

4. Построить график функции $y(x)$ с помощью компонента **Chart** (см. Приложение 2. Элементы управления).

5. Наибольшее значение функции на указанном отрезке вывести в объект **Label** на текущей форме.

6. Выполнить задание 1-5 для своего варианта (см. **Варианты заданий**).

Варианты заданий

1. Переменная *x* изменяется в интервале от –10 до 10 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \textit{при } 3\le x\le 6,\\ (x-4)! & \textit{при } 6<x\le 10,\\ x^4 & \textit{в остальных случаях.}\end{cases}$$

2. Переменная *x* изменяется в интервале от –5 до 8 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+5)! & \textit{при } -2\le x\le 4,\\ (x-3)! & \textit{при } 5<x\le 8,\\ x^2 & \textit{в остальных случаях.}\end{cases}$$

3. Переменная *x* изменяется в интервале от –15 до 12 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран минимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+9)! & \textit{при } -8\le x\le 3,\\ (x-6)! & \textit{при } 8<x\le 12,\\ \operatorname{tg} x & \textit{в остальных случаях.}\end{cases}$$

4. Переменная *x* изменяется в интервале от –5 до 20 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-7)! & \textit{при } 8\le x\le 10,\\ (x-11)! & \textit{при } 12<x\le 20,\\ x+k & \textit{в остальных случаях.}\end{cases}$$

5. Переменная *x* изменяется в интервале от –10 до 10 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \textit{при } 3\le x\le 6,\\ (x-4)! & \textit{при } 6<x\le 10,\\ \sin x & \textit{в остальных случаях.}\end{cases}$$

6. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление боковой стороны в виде функции, определить треугольник с наименьшей боковой стороной и выдать сообщение на экран (длину боковой стороны и номер треугольника).

7. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление

площади треугольника в виде функции, выдать на экран наибольшую площадь и номер этого треугольника.

8. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление периметра треугольника в виде функции, выдать на экран наибольший периметр и номер этого треугольника.

9. Даны радиусы R_i десяти окружностей (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление площади окружности в виде функции, выдать на экран сообщение о наибольшей площади окружности и номер этой окружности.

10. Переменная ***x*** изменяется в интервале от –14 до 11 с шагом 1. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции ***y***, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \textit{при } 3\le x\le 6,\\ (x-4)! & \textit{при } 7<x\le 10,\\ x^4 & \textit{в остальных случаях.}\end{cases}$$

11. Переменная *x* изменяется в интервале от –5 до 10 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+5)! & \textit{при } -2\le x\le 4,\\ (x-3)! & \textit{при } 6<x\le 9,\\ x^2 & \textit{в остальных случаях.}\end{cases}$$

12. Переменная *x* изменяется в интервале от –15 до 12 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран минимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+9)! & \textit{при } -8\le x\le 4,\\ (x-6)! & \textit{при } 8<x\le 13,\\ \operatorname{tg} x & \textit{в остальных случаях.}\end{cases}$$

13. Переменная *x* изменяется в интервале от –5 до 20 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран минимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-7)! & \textit{при } 8\le x\le 11,\\ (x-11)! & \textit{при } 12<x\le 19,\\ x+k & \textit{в остальных случаях.}\end{cases}$$

14. Переменная *x* изменяется в интервале от –10 до 12 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \textit{при } 4\le x\le 8,\\ (x-4)! & \textit{при } 8<x\le 12,\\ x^3 & \textit{в остальных случаях.}\end{cases}$$

15. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление боковой стороны в виде функции, определить треугольник с наибольшей боковой стороной и выдать сообщение на экран (длину боковой стороны и номер треугольника).

16. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление площади треугольника в виде функции, выдать на экран наименьшую площадь и номер этого треугольника.

17. Даны основания a_i и высоты h_i десяти равнобедренных треугольников (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление периметра треугольника в виде функции, выдать на экран наименьший периметр и номер этого треугольника.

18. Даны радиусы R_i десяти окружностей (значения ввести в диалоговом режиме с клавиатуры). Оформив вычисление площади окружности в виде функции, выдать на экран сообщение о наименьшей площади окружности и номер этой окружности.

19. Переменная *x* изменяется в интервале от –13 до 12 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y = \begin{cases} (x-2)! & \textit{при } 3 \le x \le 6, \\ (x-4)! & \textit{при } 7 < x \le 10, \\ x^4 & \textit{в остальных случаях.} \end{cases}$$

20. Переменная *x* изменяется в интервале от –7 до 15 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y = \begin{cases} (x+3)! & \textit{при } -2 \le x \le 4, \\ (x-5)! & \textit{при } 6 < x \le 9, \\ x^2 & \textit{в остальных случаях.} \end{cases}$$

21. Переменная *x* изменяется в интервале от –15 до 14 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран минимальное значение функции *y*, которая вычисляется по правилу:

$$y = \begin{cases} (x+9)! & \textit{при } -8 \le x \le 4, \\ (x-6)! & \textit{при } 8 < x \le 12, \\ \operatorname{tg} x & \textit{в остальных случаях.} \end{cases}$$

22. Переменная *x* изменяется в интервале от –5 до 22 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-6)! & \text{при } 8\le x\le 11,\\ (x-10)! & \text{при } 12<x\le 19,\\ x+k & \text{в остальных случаях.}\end{cases}$$

23. Переменная *x* изменяется в интервале от –11 до 14 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \text{при } 3\le x\le 6,\\ (x-4)! & \text{при } 6<x\le 10,\\ x^3 & \text{в остальных случаях.}\end{cases}$$

24. Переменная *x* изменяется в интервале от 1 до 11 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & \text{при } 3\le x\le 6,\\ (x-4)! & \text{при } 7<x\le 10,\\ x^2 & \text{в остальных случаях.}\end{cases}$$

25. Переменная *x* изменяется в интервале от –5 до 15 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+3)! & \text{при } -2\le x\le 4,\\ (x-5)! & \text{при } 6<x\le 9,\\ 5x & \text{в остальных случаях.}\end{cases}$$

26. Переменная *x* изменяется в интервале от –9 до 14 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран минимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+5)! & \text{при } -4\le x\le 4,\\ (x-6)! & \text{при } 8<x\le 12,\\ \operatorname{tg} x & \text{в остальных случаях.}\end{cases}$$

27. Переменная *x* изменяется в интервале от –3 до 23 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-6)! & \text{при } 8\le x\le 11,\\ (x-10)! & \text{при } 12<x\le 19,\\ x+k & \text{в остальных случаях.}\end{cases}$$

28. Переменная *x* изменяется в интервале от –11 до 14 с шагом 3. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-2)! & при\ 2\le x\le 6,\\ (x-4)! & при\ 7<x\le 10,\\ 4x & в\ остальных\ случаях.\end{cases}$$

29. Переменная *x* изменяется в интервале от –7 до 12 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран максимальное значение функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x-3)! & при\ 3\le x\le 6,\\ (x-7)! & при\ 7<x\le 10,\\ x^2 & в\ остальных\ случаях.\end{cases}$$

30. Переменная *x* изменяется в интервале от –2 до 16 с шагом 2. Оформив вычисление факториала в виде функции, выдать на экран значения функции *y*, которая вычисляется по правилу:

$$y=\begin{cases}(x+3)! & при\ -2\le x\le 4,\\ (x-5)! & при\ 6<x\le 9,\\ 3x^2 & в\ остальных\ случаях.\end{cases}$$

Методика выполнения

1) Разместите на форме проекта следующие элементы (см. Рис. 13-1).

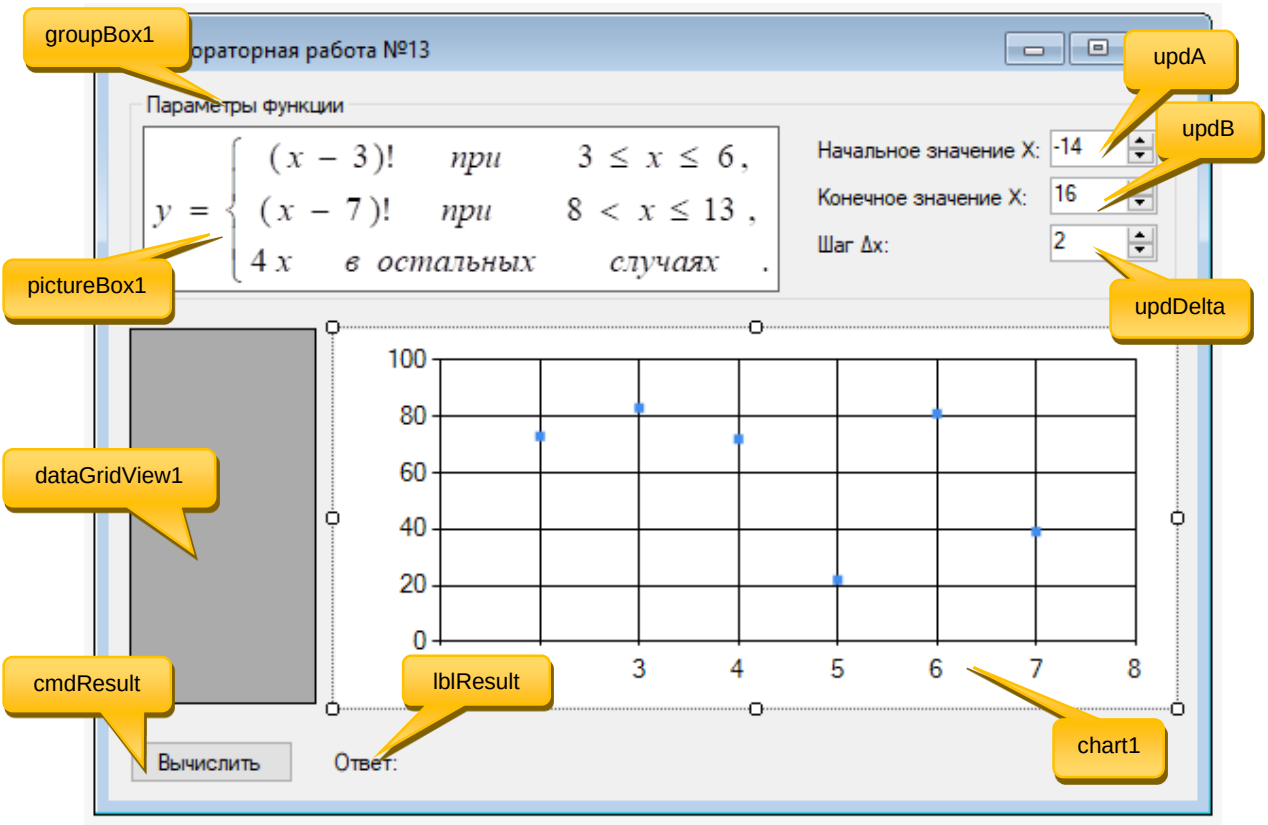

Рис. 13-1 – Конструктор формы проекта *Занятие* 13

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
updA	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	-14
updB	Minimum	-100
	Maximum	100
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	16
updDelta	Minimum	1
	Maximum	10
	BackColor	White
	Increment	1
	ReadOnly	True
	Value	2
dataGridView1	AutoSizeColumnsMode	AllCels
	AutoSizeRowsMode	AllCels
	ColumnHeadersVisible	True
	RowHeadersVisible	False
	ReadOnly	True
lblResult	Text	Ответ:
chart1	Legends	<пусто> //Очистить коллекцию от элементов

3) Обратившись к свойству **Columns** элемента **dataGridView1**, добавить (см. Рис. 13-2) пару столбцов **Column1** и **Column2**.

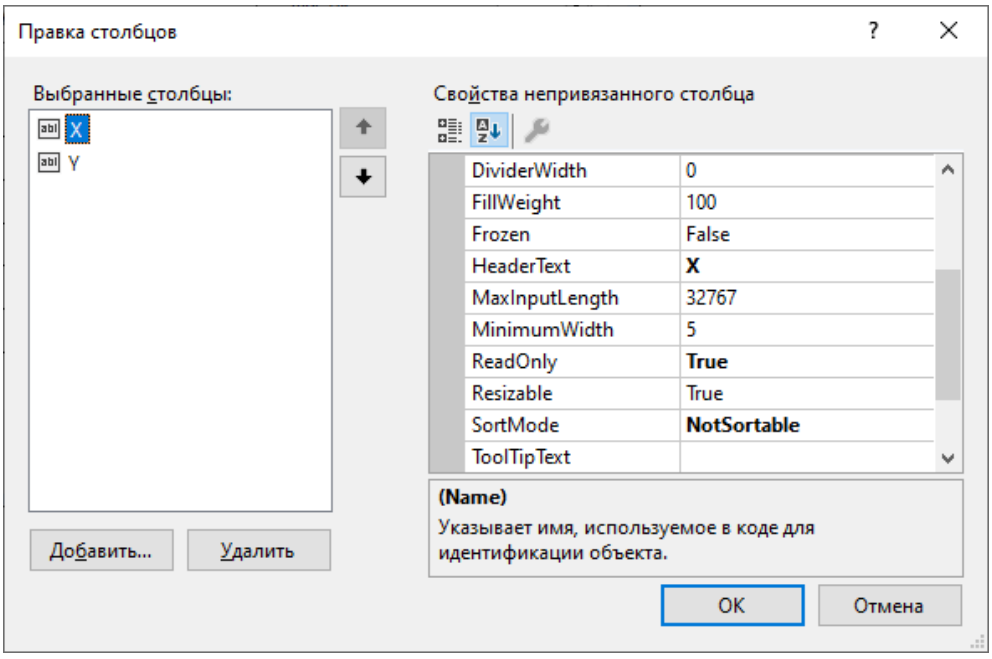

Рис. 13-2 – Правка столбцов элемента **dataGridView1**

Задайте свойства этих столбцов (см. таблицу ниже):

Объект	Свойство	Значение
Column1	HeaderText	X
	SortMode	NotSortable
Column2	HeaderText	Y
	SortMode	NotSortable

4) Обратившись к свойству **Series** элемента **chart1**, изменить (см. Рис. 13-3) у ряда **Series1** значение свойства **ChartType** на **Point** (точечный график).

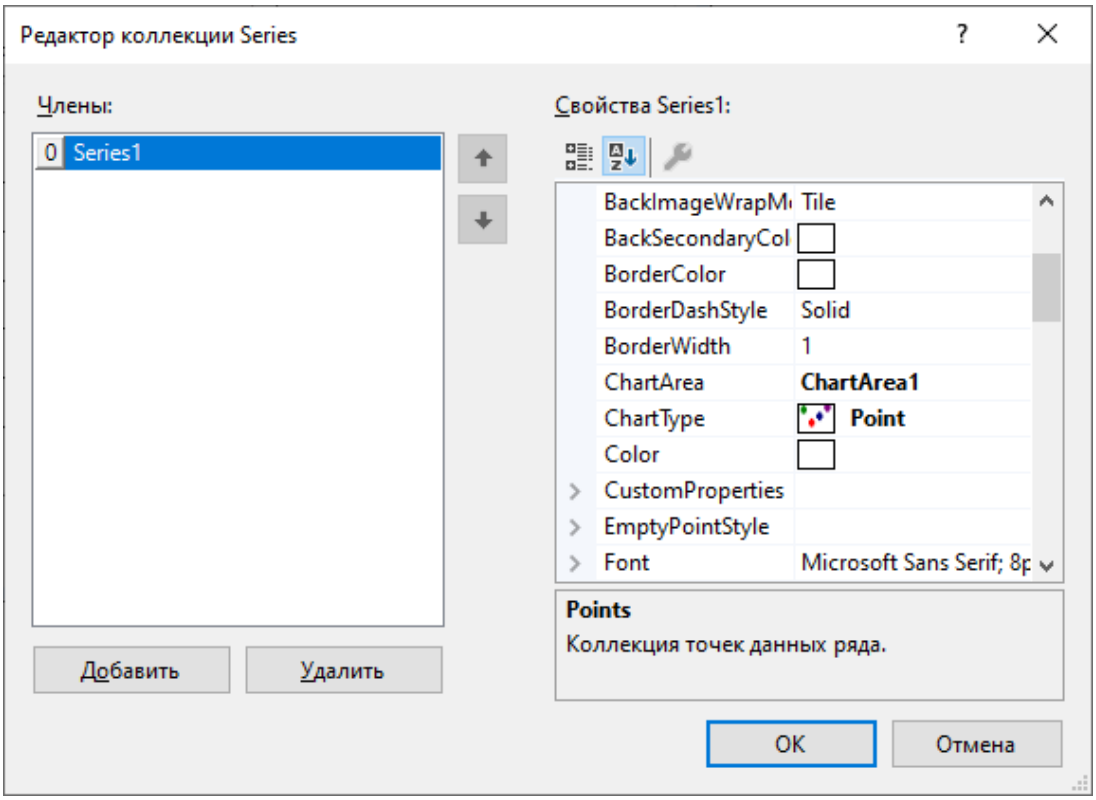

Рис. 13-3 – Редактор коллекции **Series**

5) Далее опишем метод, для вычисления факториала числа (используем рекурсию):

```
private static double Factorial(int x)
{
   if (x == 0) return 1;
   else return x * Factorial(x-1);
}
```

6) Добавим метод, для вычисления пользовательской функции $y(x)$:

```
private static double Y(int x)
{
   if (x >= 3 && x <= 6) return Factorial(x - 3);
   else if (x > 8 && x <= 13) return Factorial(x - 7);
   else return 4 * x;
}
```

7) Добавим метод **Reset**(), для очистки окна приложения от результатов предыдущих вычислений:

```
private void Reset()
{
   dataGridView1.Rows.Clear();
   chart1.Series[0].Points.Clear();
   lblResult.Text = "Ответ:";
}
```

8) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
{
   int a = (int)updA.Value;
   int b = (int)updB.Value;
   int dx = (int)updDelta.Value;
   Reset();
   int xmax = a;
   for (int x = a; x <= b; x += dx)
   {
      dataGridView1.Rows.Add(x, Y(x));
      chart1.Series[0].Points.AddXY(x, Y(x));
      if (Y(x) > Y(xmax)) xmax = x;
   }
   lblResult.Text = "Ответ: максимум функции равен "+Y(xmax).ToString() +
                                                    " при x=" + xmax.ToString();
}
```

9) Вид проекта после нажатия на кнопку «*Вычислить*» представлен на Рис. 13-4.

10) Наконец, сделаем так, чтобы при попытке изменить начальное значение аргумента *x* содержимое элементов **dataGridView1**, **chart1** и **lblResult** очищалось:

```
private void updA_ValueChanged(object sender, EventArgs e)
{
   Reset();
}
```

11) Аналогичным образом обработайте элементы **updB** и **updDelta**.

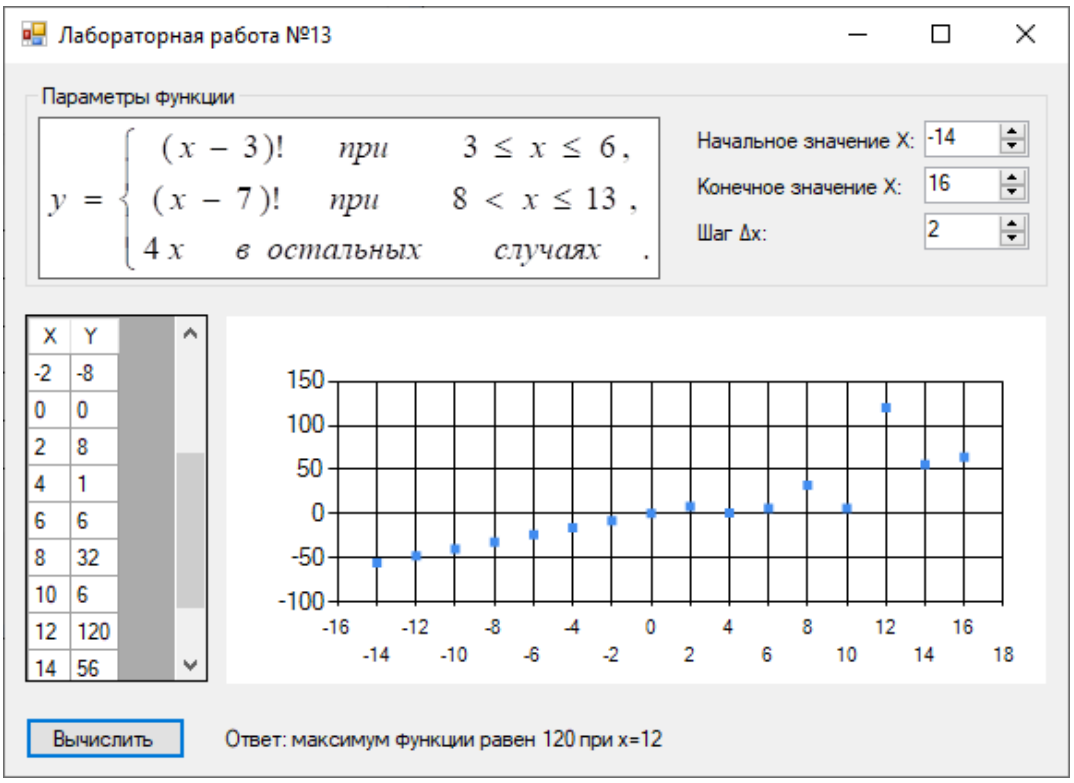

Рис. 13-4 – Результат готов!

Лабораторная работа 14. Библиотеки классов

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Организовать работу проекта из лабораторной работы №13, вынеся метод расчёта факториала в отдельную библиотеку.

2. Выполнить задание 1 для своего варианта (см. **Варианты заданий** из лабораторной работы №13).

Методика выполнения

1) Рано или поздно, вы придёте к мысли об объединении своих собственных наработок в единый пакет (ну или в несколько пакетов), чтобы созданные один раз классы, методы, элементы можно было использовать в разных программах. В C# такие пакеты называют сборками. Сборки могут быть оформлены в виде DLL, но не так называемых нативных DLL, а специализированных, .NET DLL (часто говорят, «нэтовских» DLL).

DLL – это библиотека, содержащая код и данные, которые могут использоваться несколько программами одновременно. Например, в операционных системах Windows, библиотека **Comdlg32.dll** выполняет общие функции, связанные с диалоговыми окнами. Таким образом каждая программа может использовать функцию, которая содержится в этой библиотеке для реализации диалогового окна **Открыть**. Это позволяет повысить уровень повторного использования кода и эффективного использования памяти.

С помощью библиотек можно реализовать модульность для программы, в виде отдельных компонентов. Например, бухгалтерскую программу можно продать по модулям. Каждый модуль может быть загружен в основной программе во время выполнения установки. Отдельные модули загружается только при запросе функций, заложенных в них, поэтому загрузка программы выполняется быстрее.

Кроме того, обновления легче применить для каждого модуля, не влияя на другие части программы. Например, имеется программа по зарплате и надо изменить налоговые ставки за каждый год. Когда эти изменения изолированы в библиотеке, можно применить обновления без необходимости построения или установки программы целиком.

Использование динамически подключаемой библиотеки состоит из двух этапов:

– создание библиотеки классов;

– подключение созданной библиотеки к проекту.

2) Итак, начнем. Для создания библиотеки классов в C# необходимо создать новый проект, как показано на Рис. 14-1 и Рис. 14-2.

3) После указания всех параметров нужно нажать на кнопку «*Создать*» и дождаться создания проекта.

Мы видим, что теперь у нас есть класс с именем **Class1** (это видно в *Обозревателе решений*). Давайте переименуем его, присвоим ему осмысленное название. А называться он будет **MyFunctions**. Чтобы переименовать класс, нам нужно навести на его имя в *Обозревателе решений* мышь и вызвать контекстное меню, в котором выбрать пункт «Переименовать». Результат виден на Рис. 14-3.

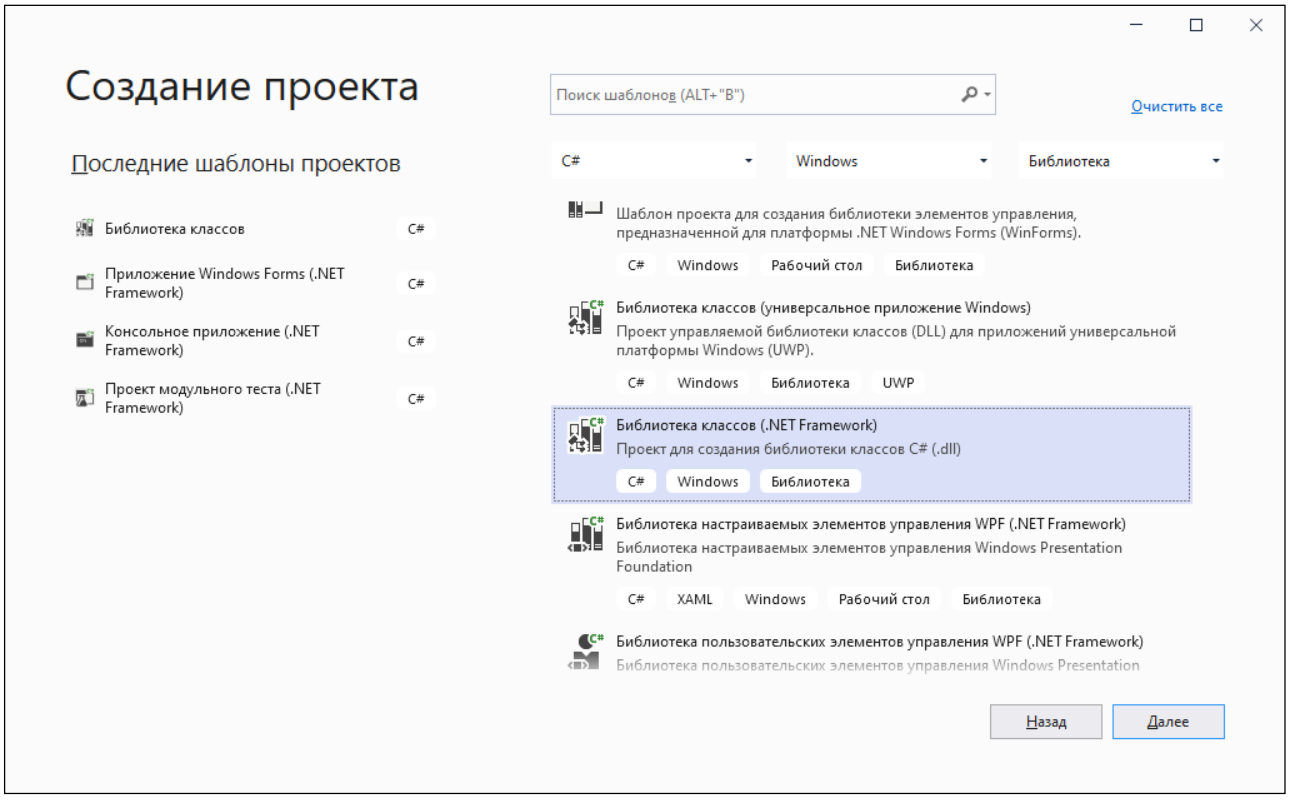

Рис. 14-1 – Создание библиотеки классов (шаг 1)

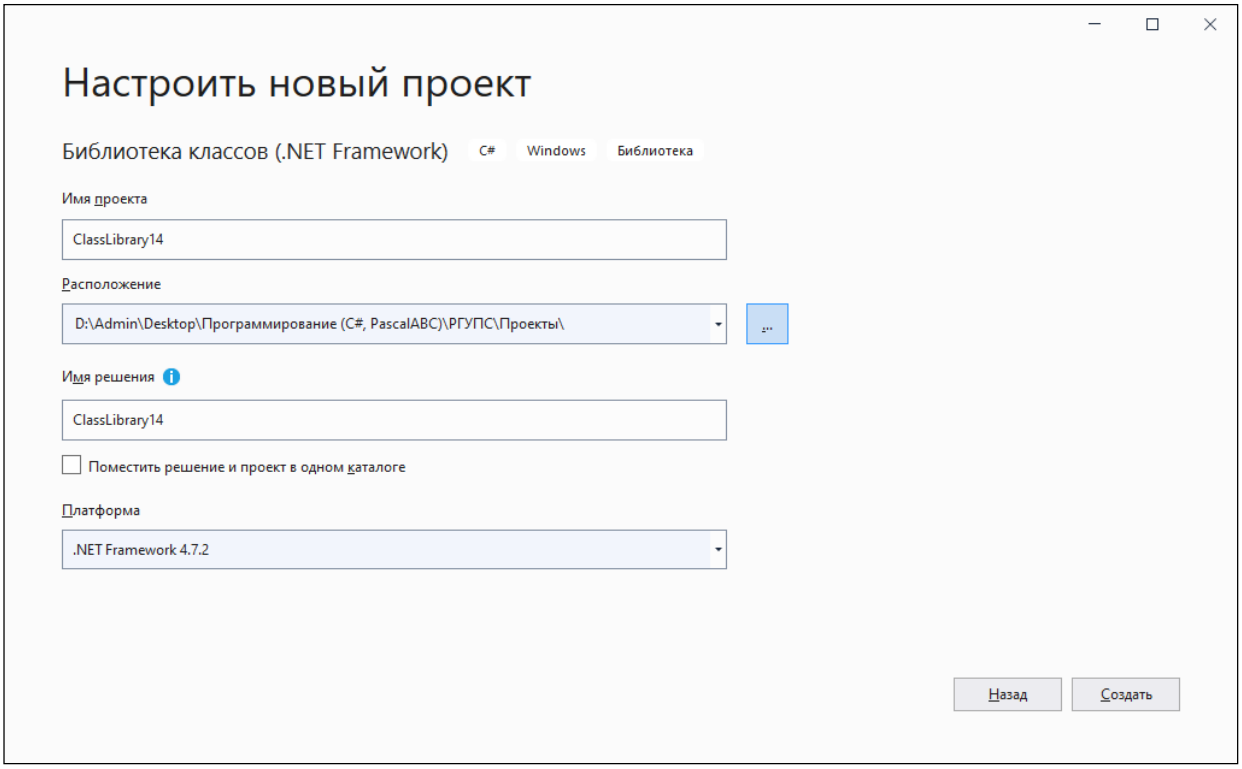

Рис. 14-2 – Создание библиотеки классов (шаг 2)

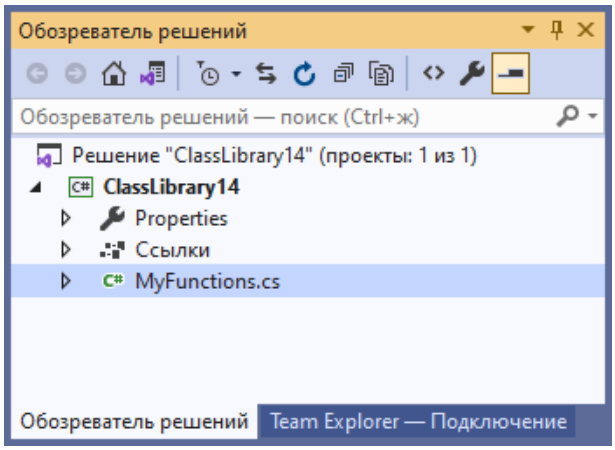

Рис. 14-3 – Обозреватель решений

4) Затем в созданной библиотеке следует открыть файл программного кода, представленного на Рис. 14-4.

Наконец, внести в класс **MyFunctons** программный код для расчёта факториала числа, как показано на Рис. 14-4.

5) Всё, теперь **нужно собрать проект** («Сборка ▸ Собрать решение» в главном меню русской версии Visual Studio). Только **запускать проект не нужно**, всё равно это бессмысленно, DLL так просто не запустить… Вместо этого, давайте найдем созданную DLL на диске вашего ПК. Для этого нужно перейти в папку, в которой мы создавали проект (путь мы указывали при создании проекта). А уже в этой папке перейти по пути «ClassLibrary14\bin\Debug», и в папке «Debug» будет файл «ClassLibrary14.dll». Запомните местонахождение библиотеки.

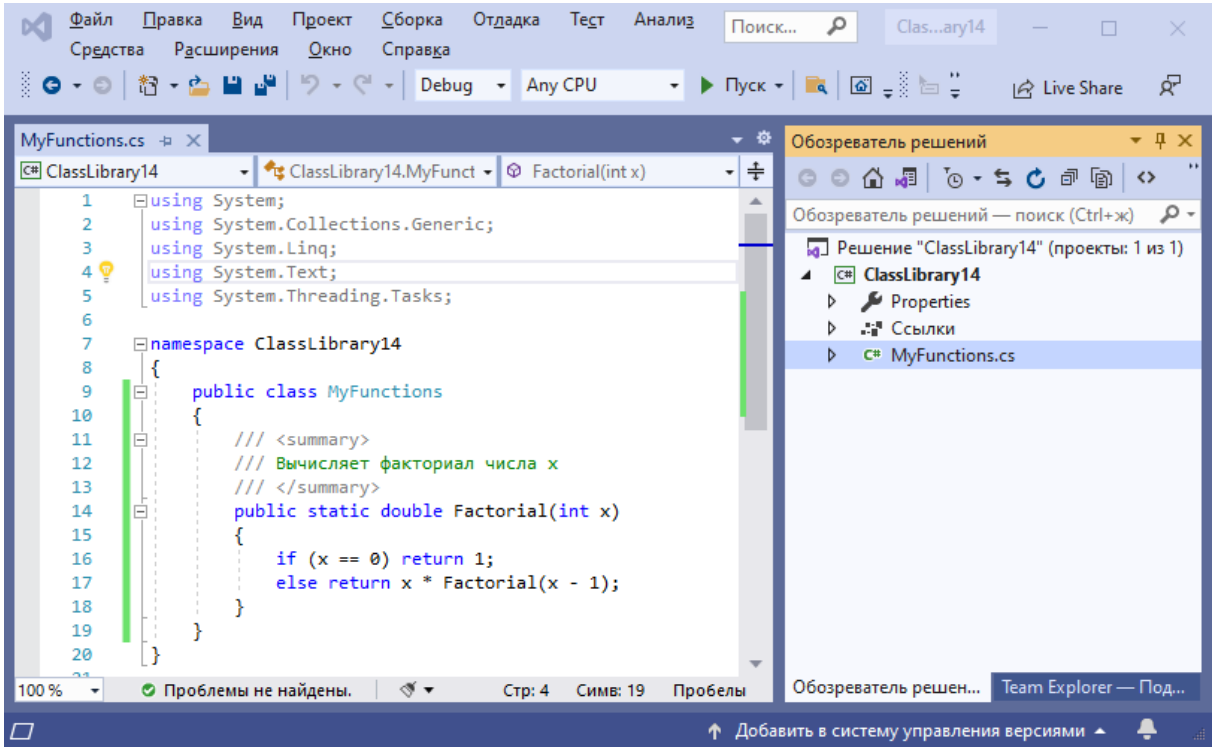

Рис. 14-4 – Библиотека классов

6) Для выполнения лабораторной работы №14 проще всего создать копию проекта лабораторной работы №13. Сделайте это.

7) Теперь подключим созданную библиотеку к проекту лабораторной работы №14.

В окне Visual Studio, в обозревателе решений, ищем группу «Ссылки» и раскрываем её, как показано на Рис. 14-5.

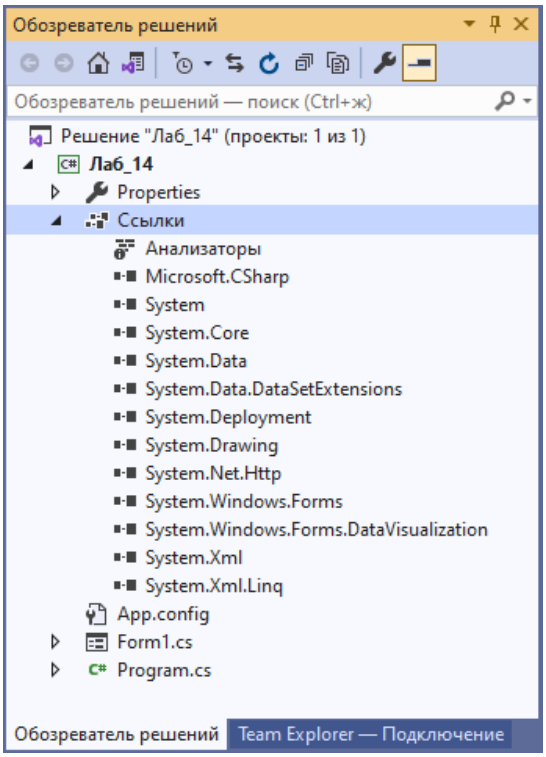

Рис. 14-5

Вызываем контекстное меню на пункте «Ссылки», и выбираем в нем команду «Добавить ссылку». В появившемся окне выбираем в левой области пункт «Обзор» и внизу окна, нажимаем на кнопку «Обзор», как показано на Рис. 14-6.

В появившемся окне, перейти в папку, в которой лежит библиотека (DLL) созданная нами ранее (см. пункт 5), выбрать эту библиотеку и нажать на кнопку «Добавить».

После чего, нажать на кнопку «OK» в предыдущем окне. В результате, список ссылок проекта, будет пополнен еще одной (см. Рис. 14-7).

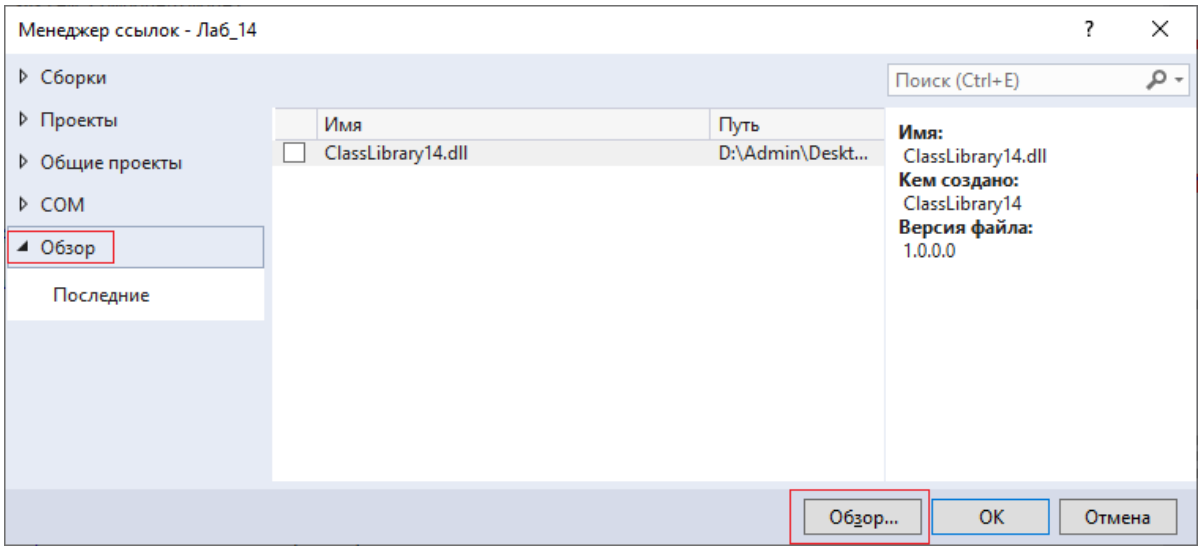

Рис. 14-6

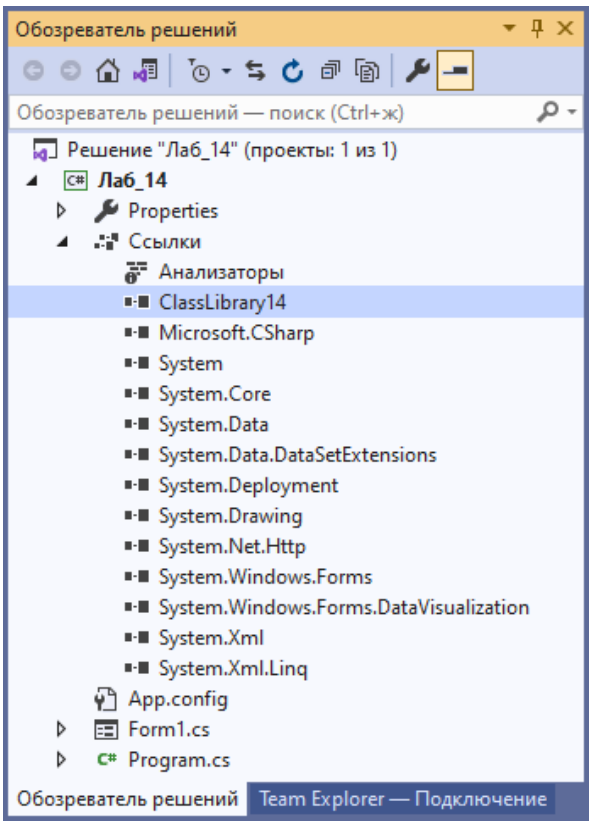

Рис. 14-7

8) Теперь, мы можем использовать в нашей программе класс **MyFunctons** из подключенной библиотеки. Но перед этим, мы должны выполнить еще одну операцию.

Обратите внимание на выделенную строку (см. Рис. 14-8), это объявление пространства имен (**namespace**), как бы некого контейнера, в котором находится класс *MyFunctons*. И мы не сможем воспользоваться классом, пока не укажем системе что нужно взять во внимание это пространство имен, т.е. нужно подключить пространство имен в нашем проекте. Для этого, добавим строку «*using ClassLibrary14*;» в конец блока директив *using*, который расположен в самом начале основного файла проекта (см. Рис. 14-9):

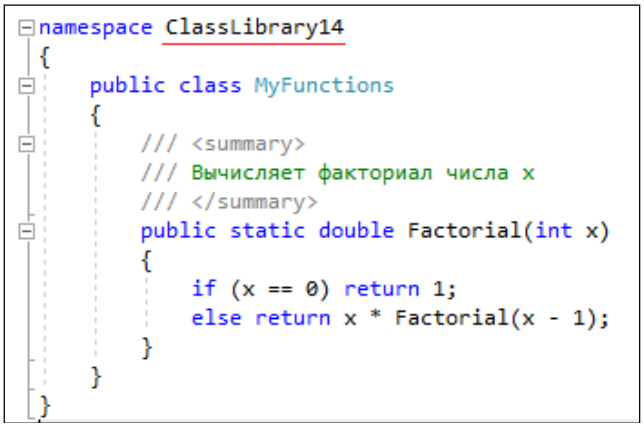

Рис. 14-8

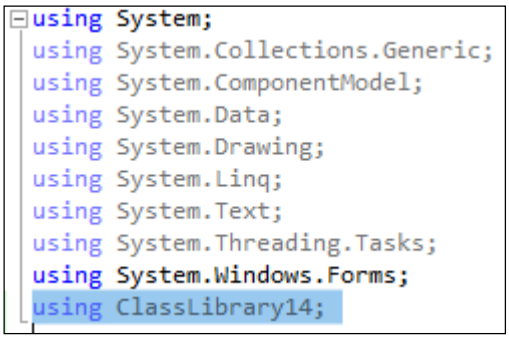

Рис. 14-9

9) Вот теперь точно можно использовать класс из подключенной библиотеки. Осталось лишь внести изменения в проект лабораторной работы 14:

- удалить описание метода-функции *Factorial*();
- внести изменения в программный код метода-функции *Y*:

```
private static double Y(int x)
{
   if (x >= 3 && x <= 6) return MyFunctions.Factorial(x - 3);
   else if (x > 8 && x <= 13) return MyFunctions.Factorial(x - 7);
   else return 4 * x;
}
```

10) Приложение обновлено!

Лабораторная работа 15. Обработка строк. Использование компонента TextBox

Цель работы. На основе лекционного материала выполнить следующие ниже задания.

1. Выполнить обработку заданного текста: удалить пробелы в начале и конце строки; между словами оставить по одному пробелу.

2. Вывести список слов заданного текста в алфавитном порядке, предварительно переведя все символы в них к нижнему регистру и удалив знаки пунктуации (.,!?:;).

3. Выполнить задание для своего варианта (см. **Варианты заданий**).

Варианты заданий

1. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Найти количество слов, которые начинаются и заканчиваются одной и той же буквой.

2. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Найти количество слов, которые содержат хотя бы одну букву «А».

3. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Найти количество слов, которые содержат ровно три буквы «А».

4. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Найти длину самого короткого слова.

5. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Найти длину самого длинного слова.

6. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним символом «.» (точка). В конце строки точку не ставить.

7. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Преобразовать каждое слово в строке, заменив в нем все последующие вхождения его первой буквы на символ «.» (точка). Например, слово «МИНИМУМ» надо преобразовать в «МИНИ.У.».

8. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Преобразовать каждое слово в строке, заменив в нем все предыдущие вхождения его последней буквы на символ «.» (точка). Например, слово «МИНИМУМ» на-до преобразовать в «.ИНИ.УМ».

9. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним пробелом и расположенные в обратном порядке.

10. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним пробелом и расположенные в алфавитном порядке.

11. Дана строка-предложение. Преобразовать строку так, чтобы каждое слово начиналось с заглавной буквы. Словом считать набор символов, не

содержащий пробелов и ограниченный пробелами или началом/концом строки. Слова, не начинающиеся с буквы, не изменять.

12. Дана строка-предложение. Подсчитать количество содержащихся в строке знаков препинания.

13. Дана строка-предложение на русском языке. Подсчитать количество содержащихся в строке гласных букв.

14. Дана строка-предложение. Вывести самое длинное слово в предложении. Если таких слов несколько, то вывести первое из них. Словом считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки.

15. Дана строка-предложение на русском языке. Вывести самое короткое слово в предложении. Если таких слов несколько, то вывести последнее из них. Словом считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки.

16. Даны целые положительные числа N_1 и N_2 и строки S_1 и S_2. Получить из этих строк новую строку, содержащую первые N_1 символов строки S_1 и последние N_2 символов строки S_2 (в указанном порядке).

17. Дан символ C и строка S. Удвоить каждое вхождение символа C в строку S.

18. Дан символ C и строки S, S_0. Перед каждым вхождением символа C в строку S вставить строку S_0.

19. Дан символ C и строки S, S_0. После каждого вхождения символа C в строку S вставить строку S_0.

20. Даны строки S и S_0. Удалить из строки S последнюю подстроку, совпадающую с S_0. Если совпадающих подстрок нет, то вывести строку S без изменений.

21. Даны строки S и S_0. Удалить из строки S все подстроки, совпадающие с S_0. Если совпадающих подстрок нет, то вывести строку S без изменений.

22. Даны строки S, S_1 и S_2. Заменить в строке S последнее вхождение строки S_1 на строку S_2.

23. Даны строки S, S_1 и S_2. Заменить в строке S все вхождения строки S_1 на строку S_2.

24. Дана строка, содержащая по крайней мере один символ пробела. Вывести подстроку, расположенную между первым и вторым пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку.

25. Дана строка, содержащая по крайней мере один символ пробела. Вывести подстроку, расположенную между первым и последним пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку.

26. Дана строка, содержащая *полное имя файла*, то есть имя диска, список каталогов (путь), собственно имя и расширение. Выделить из этой строки имя файла (без расширения).

27. Дана строка-предложение. Зашифровать ее, поместив вначале все символы, расположенные на четных позициях строки, а затем, в обратном

порядке, все символы, расположенные на нечетных позициях (например, строка «Программа» превратится в «ргамамроП»).

28. Дана строка, содержащая полное имя файла. Выделить из этой строки название первого каталога (без символов «\»). Если файл содержится в корневом каталоге, то вывести символ «\».

29. Дана строка, содержащая полное имя файла. Выделить из этой строки название последнего каталога (без символов «\»). Если файл содержится в корневом каталоге, то вывести символ «\».

30. Дана строка, содержащая цифры и строчные латинские буквы. Если буквы в строке упорядочены по алфавиту, то вывести 0; в противном случае вывести номер первого символа строки, нарушающего алфавитный порядок.

Методика выполнения

1) Разместите на форме проекта следующие элементы управления (см. Рис. 15-1).

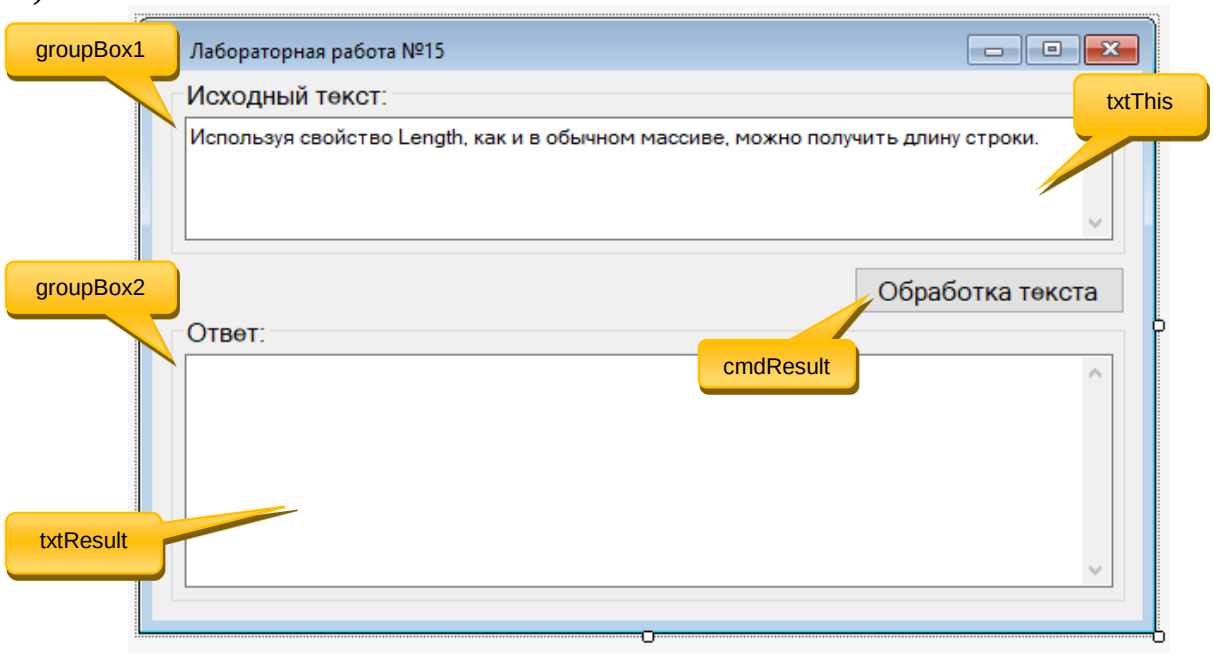

Рис. 15-1 – Конструктор формы проекта *Занятие* 15

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
txtThis	Multiline	True
	Text	Используя свойство Length, как и в обычном массиве, можно получить длину строки.
txtResult	Multiline	True
	ScrollBars	Vertical
	ReadOnly	True
	Text	<пусто>

3) Далее опишем метод-процедуру, для удаления лишних пробелов из заданной строки:

```
void SpaceDel(ref string s)
{
   s = s.Trim();
   while (s.Contains("␣␣")) s=s.Remove(s.IndexOf("␣␣"),1);
}
```

4) Добавим метод-процедуру, для удаления знаков пунктуации (.,!?:;) в конце данного слова:

```
void PunctuationDel(ref string s)
{
   string x = ":;!?.,";
   while(x.Contains(s[s.Length-1])) s = s.Remove(s.Length - 1, 1);
}
```

5) Программный код для кнопки **cmdResult**:

```
private void cmdResult_Click(object sender, EventArgs e)
{
   string s = txtThis.Text;
   txtResult.Text = "Все слова из текста в алфавитном порядке:";
   SpaceDel(ref s);
   s = s.ToLower();
   string[] words=s.Split('␣');
   Array.Sort(words);
   for (int i = 0; i < words.Length; i++)
   {
      PunctuationDel(ref words[i]);
      txtResult.Text = txtResult.Text+ "\r\n"+words[i];
   }
}
```

6) Вид проекта после нажатия на кнопку «*Вычислить*» представлен на Рис. 15-2.

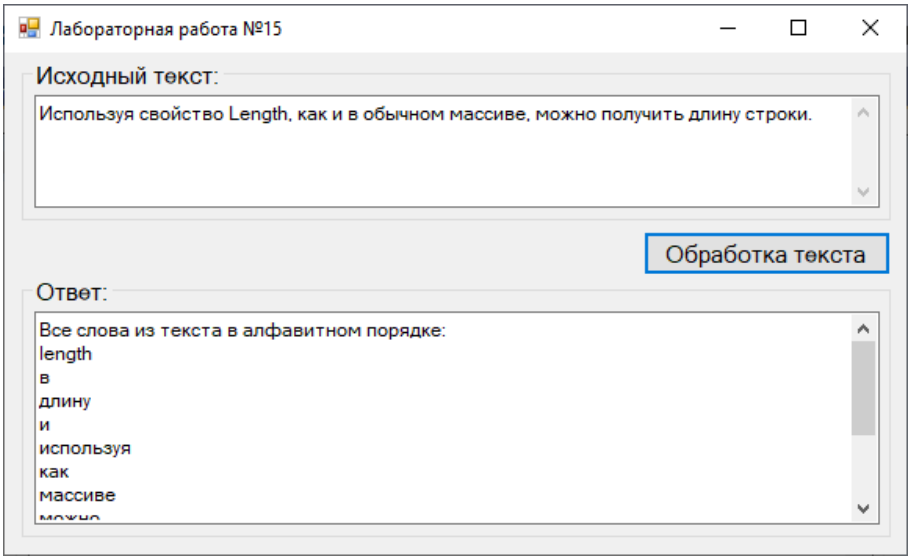

Рис. 15-2 – Текст обработан!

7) Наконец, сделаем так, чтобы при попытке изменить исходный текст содержимое элемента **txtThis** очищалось:

```
private void txtThis_TextChanged(object sender, EventArgs e)
{
   txtResult.Text = "";
}
```

Лабораторная работа 16. Элементы управления MenuStrip, OpenFileDialog и SaveFileDialog

Цель работы. На основе лекционного материала выполнить следующее ниже задание.

Создать простейший текстовый редактор (аналогичный **NotePad** для Windows). Для создания использовать приёмы и правила работы с файлами последовательного доступа (см. **Методику выполнения**).

Методика выполнения

1) Разместите на форме проекта следующие элементы (см. Рис. 16-1).

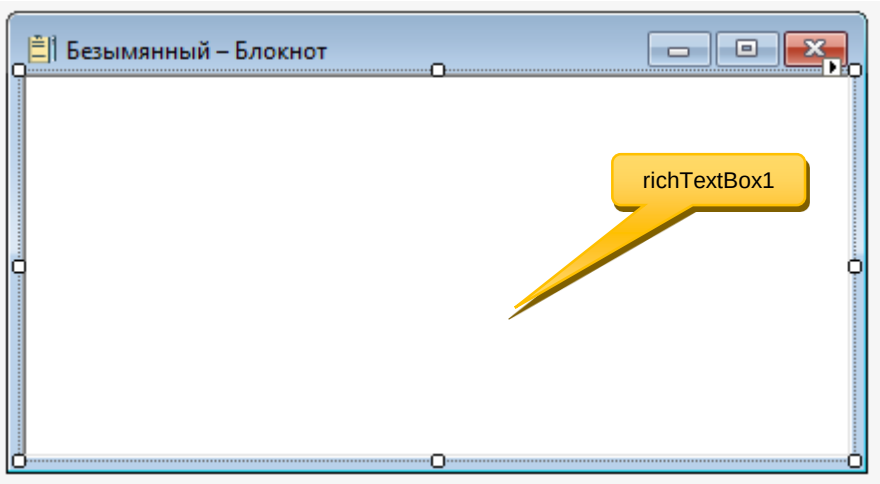

Рис. 16-1 – Конструктор формы проекта *Занятие* 16

2) Задайте свойства этих объектов (см. таблицу ниже):

Объект	Свойство	Значение
richTextBox1	Dock	Fill
Form1	Text	Безымянный – Блокнот

3) Создайте на форме приложения один элемент управления **MenuStrip** (стандартное меню верхнего уровня) (см. Рис. 16-2). Дайте ему имя *mnuGeneral*.

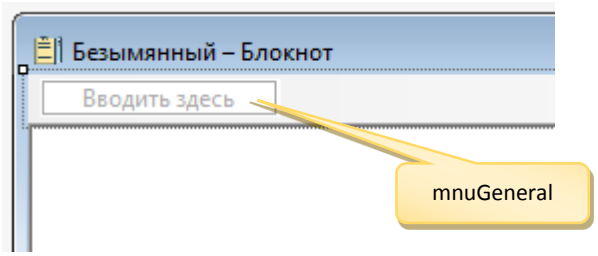

Рис. 16-2 – Создание меню формы (1-й шаг)

4) Далее добавьте один пункт в это меню – *mnuFile* (Рис. 16-3). Свойство **Text** для него задайте *Файл*.

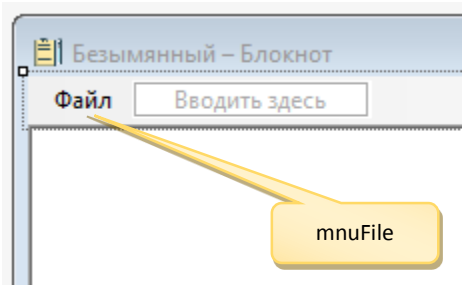

Рис. 16-3 – Создание меню формы (2-й шаг)

5) Поместите в меню *mnuFile* приведенные ниже элементы **MenuItem** (пункты):

Text	Name
Создать	mnuNew
Открыть…	mnuOpen
Сохранить	mnuSave
Сохранить как…	mnuSaveAs
Выход	mnuExit

6) Это можно сделать в режиме конструктора (Рис. 16-4) или обратиться к свойству **DropDownItems** компонента *mnuFile* и в открывшемся окне добавить и настроить все элементы меню (Рис. 16-5).

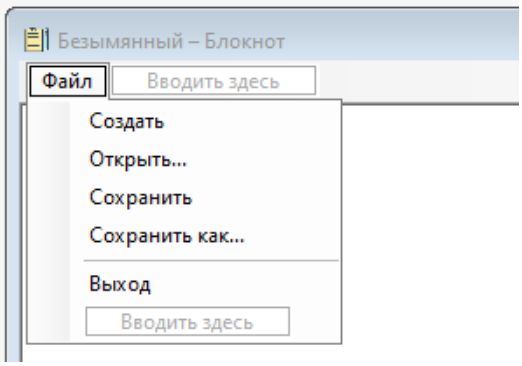

Рис. 16-4 – Создание меню формы (3-й шаг)

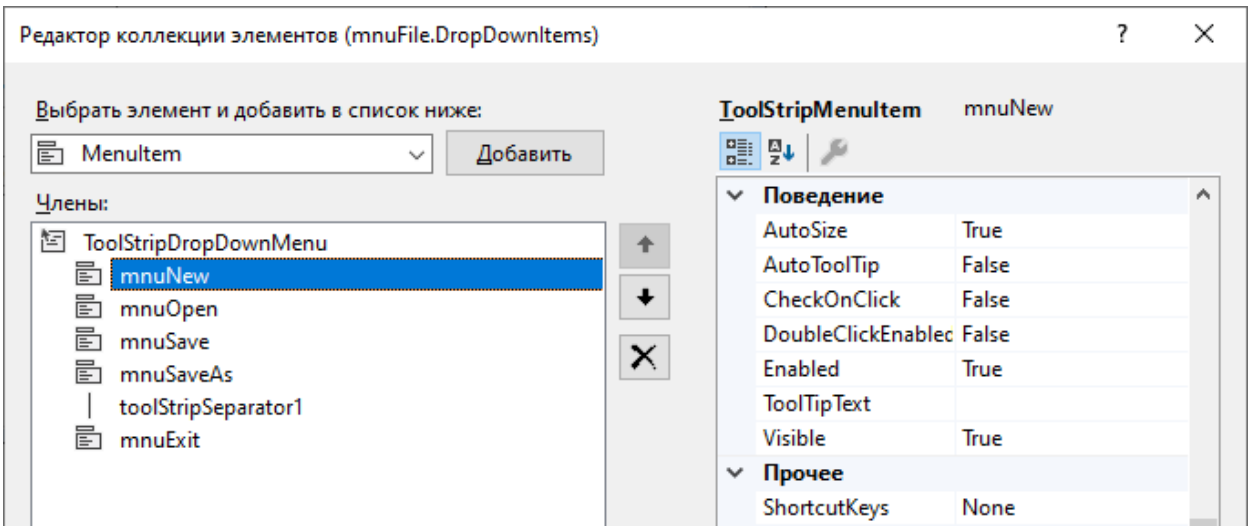

Рис. 16-5 – Создание меню формы (4-й шаг)

7) Черту, которая разделяет пункты *Сохранить* и *Выход*, можно добавить кликнув правой кнопкой на *mnuExit* и выбрав команду **Separator** (Рис. 16-6).

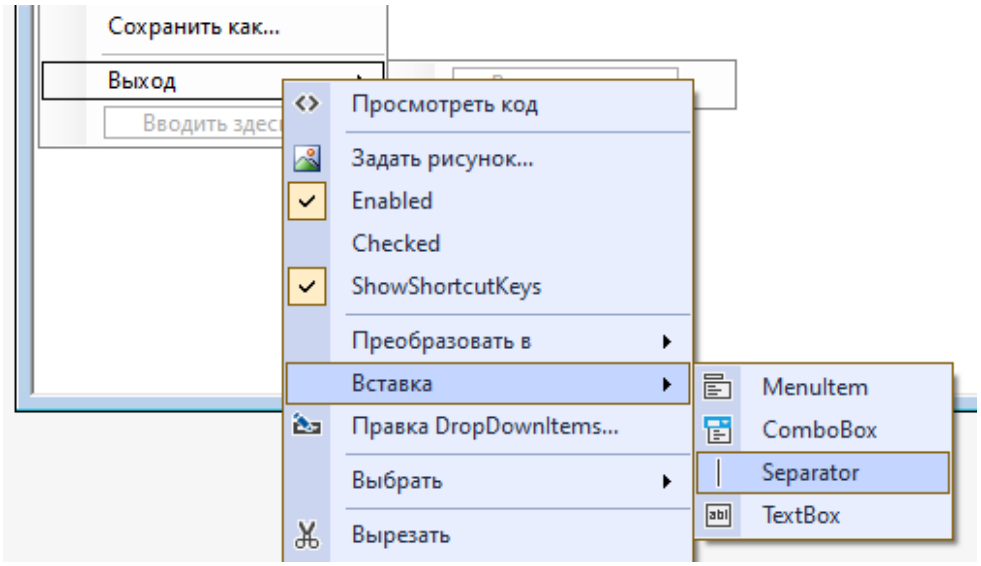

Рис. 16-6 – Создание меню формы (5-й шаг)

8) Для работы с каталогами и файлами нам потребуется пространство имен **System.IO**. Добавьте его в свой проект (см. Рис. 16-7):

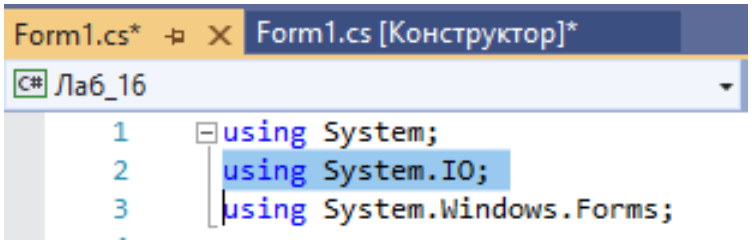

Рис. 16-7

9) Далее объявим строковую переменную **filename** (будет хранить полное имя текущего открытого текстового файла) в классе формы (Рис. 16-8). В этом случае она **будет доступна всем фрагментам кода** нашего приложения.

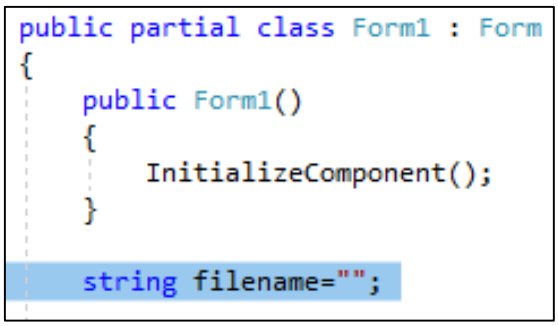

Рис. 16-8

10) Опишем метод-процедуру, которая сохраняет содержимое **richTextBox1** в текстовый файл:

```
/// <summary>
/// Сохраняет содержимое richTextBox1 в текстовый файл
/// </summary>
void SaveText(string filename)
{
   StreamWriter f = new StreamWriter(filename,false);
   foreach (string s in richTextBox1.Lines) f.WriteLine(s);
   f.Close();
}
```

11) Опишем метод-процедуру, которая выводит короткое имя файла без расширения в заголовок текущего окна:

```
/// <summary>
/// Выводит короткое имя файла без расширения в заголовок текущего окна
/// </summary>
void Rename(string filename)
{
   string s = new FileInfo(filename).Name.ToString();
   s = s.Remove(s.Length - 4);
   this.Text = s + " – Блокнот";
}
```

12) Обработайте событие **Click** для первого пункта нашего меню **mnuNew** следующим образом:

```
private void mnuNew_Click(object sender, EventArgs e)
{
   richTextBox1.Clear();
   filename = "";
   this.Text = "Безымянный – Блокнот";
}
```

13) Обработайте событие **Click** для пункта нашего меню **mnuOpen** следующим образом:

```
private void mnuOpen_Click(object sender, EventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog
    {
        Filter = "Текстовые файлы (*.txt)|*.txt",
    };
    DialogResult res = ofd.ShowDialog();
    if (res != DialogResult.OK) return;
    filename = ofd.FileName;
    Rename(filename);
    StreamReader f = new StreamReader(filename);
    richTextBox1.Text = f.ReadToEnd();
    f.Close();
}
```

14) Обработайте событие **Click** для пункта нашего меню **mnuSave** следующим образом:

```
private void mnuSave_Click(object sender, EventArgs e)
{
    if (filename == "")
    {
        SaveFileDialog sfd = new SaveFileDialog
        {
            Filter = "Текстовые файлы (*.txt)|*.txt",
        };
        DialogResult res = sfd.ShowDialog();
        if (res != DialogResult.OK) return;
        filename = sfd.FileName;
        Rename(filename);
    }
    SaveText(filename);
}
```

15) Обработайте событие **Click** для пункта нашего меню **mnuSaveAs** следующим образом:

```
private void mnuSaveAs_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog
    {
        Filter = "Текстовые файлы (*.txt)|*.txt",
    };
    DialogResult res = sfd.ShowDialog();
    if (res != DialogResult.OK) return;
    filename = sfd.FileName;
    SaveText(filename);
    Rename(filename);
}
```

16) Обработайте событие **Click** для пункта нашего меню **mnuExit** следующим образом:

```
private void mnuExit_Click(object sender, EventArgs e)
{
    Close();
}
```

17) Вид проекта после запуска и открытия файла представлен на Рис. 16-9 и Рис. 16-10.

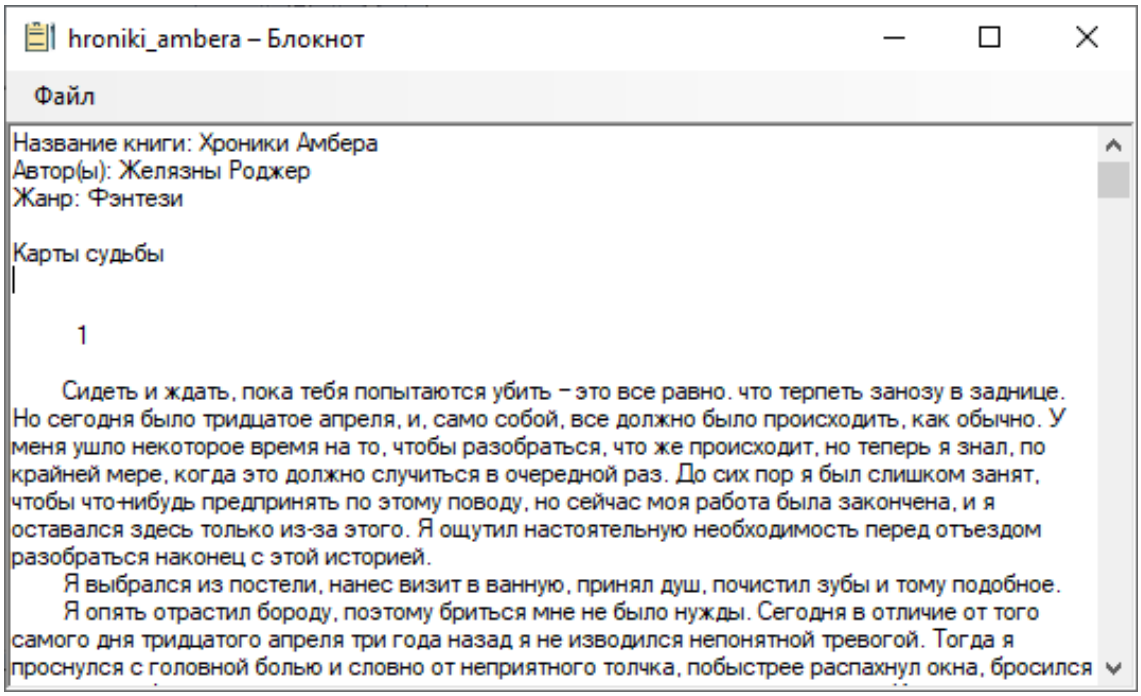

Рис. 16-9

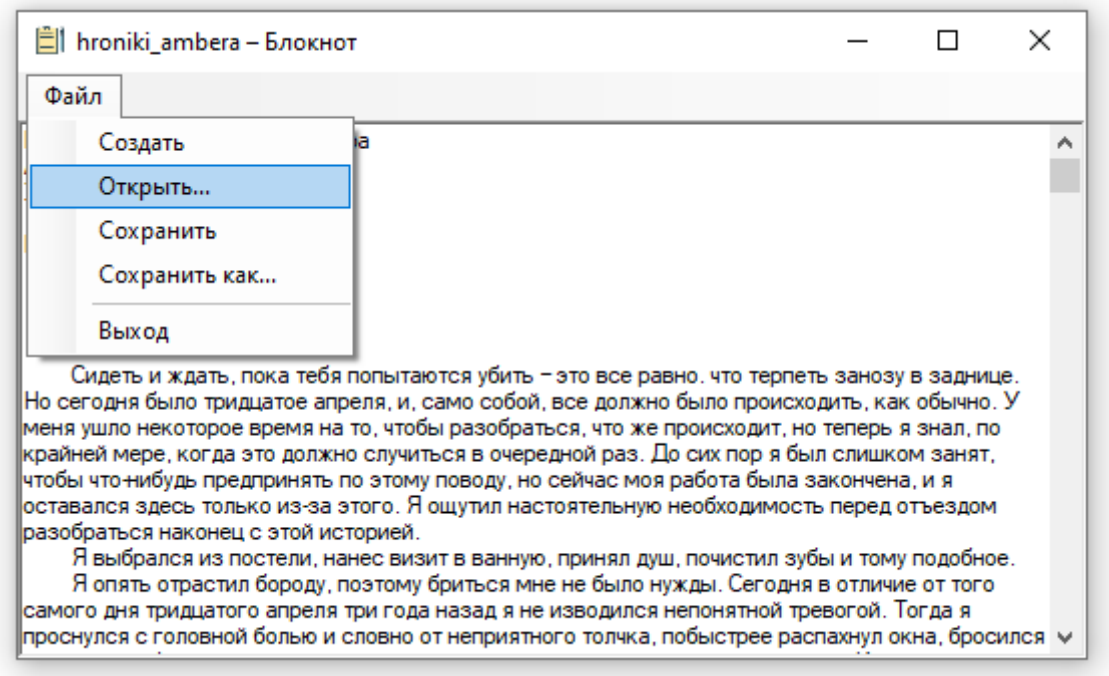

Рис. 16-10

Лабораторная работа 17. Разработка приложений с графическим интерфейсом: обработка событий «мыши»

Цель работы. На основе лекционного материала выполнить следующее ниже задание*:* разработайте приложение «*Графический редактор*», в котором инструменты *не будут похожи* на инструменты стандартного графического редактора *Paint*.

Справочный материал

Использование событий мыши

В большинстве программ *Windows Forms* для обработки ввода с помощью мыши используются события мыши.

События мыши

Основной способ реагирования на ввод с помощью мыши заключается в обработке событий мыши. В следующей таблице показаны основные события мыши и описание их возникновения.

Событие мыши	Описание
Click	Это событие возникает при отпускании кнопки мыши, как правило, перед событием ***MouseUp***. Обработчик этого события принимает аргумент типа ***EventArgs***. Обрабатывать это событие следует в случае, если нужно только определить, когда происходит щелчок.
MouseClick	Это событие возникает, когда пользователь щелкает элемент управления. Обработчик этого события принимает аргумент типа ***MouseEventArgs***. Обрабатывать это событие следует в случае, когда необходимо получить сведения о мыши при щелчке.
MouseDoubleClick	Это событие возникает, когда пользователь дважды щелкает мышью. Обработчик этого события принимает аргумент типа ***MouseEventArgs***. Обрабатывать это событие следует в случае, когда необходимо получить сведения о мыши при двойном щелчке.
MouseDown	Это событие происходит при нажатии пользователем кнопки мыши, когда указатель мыши находится на элементе управления. Обработчик этого события принимает аргумент типа ***MouseEventArgs***.
MouseEnter	Это событие возникает, когда указатель мыши перемещается на границу или клиентскую область элемента управления в зависимости от типа элемента управления. Обработчик этого события принимает аргумент типа ***EventArgs***.
MouseLeave	Это событие возникает, когда указатель мыши покидает границу или клиентскую область элемента управления в зависимости от типа элемента управления. Обработчик этого события принимает аргумент типа ***EventArgs***.
MouseMove	Это событие возникает при перемещении указателя мыши на элемент управления. Обработчик этого события принимает аргумент типа ***MouseEventArgs***.
MouseUp	Это событие возникает, когда указатель мыши находится на элементе управления и пользователь отпускает кнопку мыши. Обработчик этого события принимает аргумент типа ***MouseEventArgs***.

Событие мыши	Описание
MouseWheel	Это событие возникает, когда пользователь вращает колесико мыши, когда фокус находится на элементе управления. Обработчик этого события принимает аргумент типа ***MouseEventArgs***. Для определения того, насколько прокручено колесико мыши, можно использовать свойство ***Delta*** элемента ***MouseEventArgs***.

Сведения о мыши

Объект *MouseEventArgs* отправляется обработчикам событий мыши, связанных с нажатием кнопки мыши и отслеживанием ее движений. Объект *MouseEventArgs* предоставляет сведения о текущем состоянии мыши, включая положение указателя мыши в клиентских координатах, какие кнопки мыши нажаты и произошла ли прокрутка колесика мыши. Некоторые события мыши, например, те, которые возникают, когда указатель мыши пересек границы элемента управления, отправляют обработчику событий объект *EventArgs* без подробных сведений.

Если нужно знать текущее состояние кнопок мыши или положение ее указателя, но при этом избежать обработки события мыши, можно также использовать свойства *MouseButtons* и *MousePosition* класса *Control*. Свойство *MouseButtons* возвращает сведения о том, какие кнопки мыши в настоящее время нажаты. Свойство *MousePosition* возвращает экранные координаты указателя мыши, которые эквивалентны значению, возвращаемому методом *Position*.

Введение в Graphics

Основным классом работы с графикой является класс *Graphics*, который находится в пространстве имен *System.Drawing*. Если вы захотите воспользоваться графическими методами этого класса, то не забудьте подключить к модулю его пространство имен, чтобы упростить себе доступ. В этом же пространстве имен можно найти и другие классы, которые помогут вам при работе с графикой.

Класс *Graphics* реализует поверхность рисования и все методы рисования GDI+. GDI расшифровывается как Graphic Device Interface (интерфейс графического устройства), и это понятие хорошо отражает его суть, потому что GDI реализует методы для рисования не только на дисплее, но и на других графических устройствах, например, на принтерах. GDI+ стал продолжением первой версии графического интерфейса.

Давайте пройдемся по ***основным методам и свойствам класса Graphics***. Для начала посмотрим на свойства:

Clip – регион (прямоугольная область), который определяет область рисования;

ClipBounds – область рисования в виде четырехугольника (класс *RectangleF*);

CompositingMode – способ рисования композитных картинок;

CompositingQuality – позволяет задать качество отображения композитных изображений;

DpiX – горизонтальное разрешение поверхности;

DpiY – вертикальное разрешение поверхности;

PageScale – масштабирование;

PageUnit – единицы измерения для поверхности.

Теперь посмотрим на основные методы класса Graphics. Именно они представляют наибольшую ценность и интерес:

Clear() – очистить поверхность рисования и залить ее цветом, указанным в качестве параметра;

DrawArc() – нарисовать дугу;

DrawBezier() – нарисовать кривую Безье;

DrawBeziers() – нарисовать серию (несколько) кривых Безье;

DrawCurve() – нарисовать кривую;

DrawClosedCurve() – нарисовать замкнутую кривую, конец которой будет соединен с началом кривой;

DrawEllipse() – нарисовать эллипс;

DrawIcon() – нарисовать значок;

DrawImage() – нарисовать картинку;

DrawLine() – нарисовать линию;

DrawLines() – нарисовать серию линий;

DrawPolygon() – нарисовать многоугольник по массиву точек;

DrawRectangle() – нарисовать прямоугольник;

DrawString() – отобразить строку текста;

FillEllipse(), *FillPolygon*(), *FillRectangle*(), *FillRegion*() – залить цветом эллипс, многоугольник, прямоугольник или область;

FromHwnd() – статичный метод для создания объекта Graphics на основе Hwnd значения компонента;

FromImage() – статичный метод для создания объекта Graphics на основе картинки;

MeasureString() – рассчитывает размеры строки текста на поверхности при использовании определенного шрифта.

Ограничимся пока этим списком методов.

Методика выполнения

1 Создайте новый проект *Windows Form* приложения и на новой форме *frmРедактор* элемент управления *PictureBox* для вывода графического изображения (Рис. 17-1) и дайте ему имя *picКартинка*. Первый инструмент, который мы создадим, будет *Линии в центр*, т. е. при перемещении указателя мыши по *PictureBox* будут рисоваться линии от указателя к центру *PictureBox*.

2 Объявите новые объекты в классе текущей формы (перо для рисования, буфер для Bitmap-изображения и «холст»):

```
Pen p = new Pen(Color.Black, 1);
Bitmap buf;
Graphics gr;
```

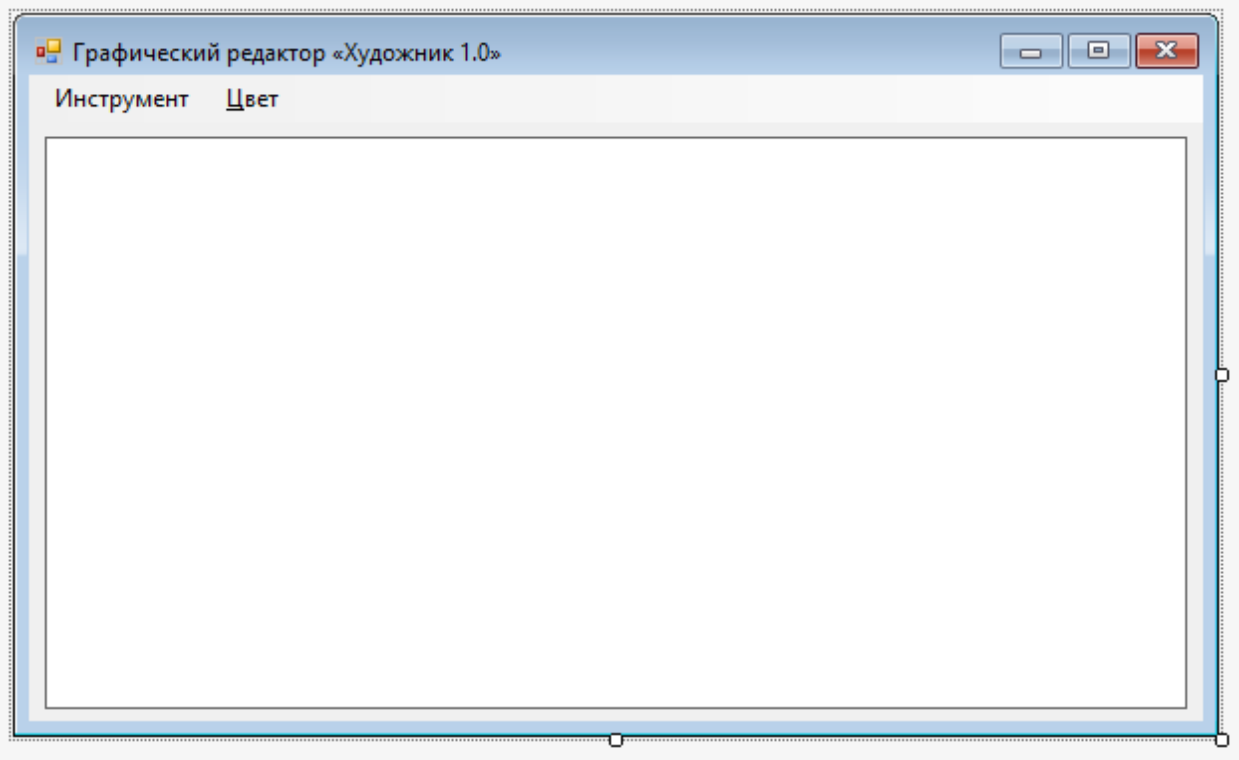

Рис. 17-1 – Конструктор формы

3 Внесите изменения в конструктор формы *frmРедактор*():

```
public frmРедактор()
{
        InitializeComponent();
        buf = new Bitmap(picКартинка.Width, picКартинка.Height);
        gr = Graphics.FromImage(buf);
}
```

4 Запрограммируйте событие *MouseMove* (перемещение мыши) для *PictureBox* следующим образом:

```
private void picКартинка_MouseMove(object sender, MouseEventArgs e)
{
        gr.DrawLine(p, e.X, e.Y, picКартинка.Width / 2, picКартинка.Height / 2);
        picКартинка.Image = buf;
}
```

5 Эта процедура рисует линию чёрного цвета (Рис. 17-2) из центра *picКартинка* к координатам указателя мыши *X* и *Y*, которые передаются в эту процедуру через параметр MouseEventArgs e.

Но в данном случае мы не управляем процессом вывода линий. Необходимо, чтобы мы могли подавать команды для начала и окончания рисования линий. Самый удобный вариант – это подавать данные команды мышью, к примеру, чтобы линии рисовались при ее нажатой левой клавише.

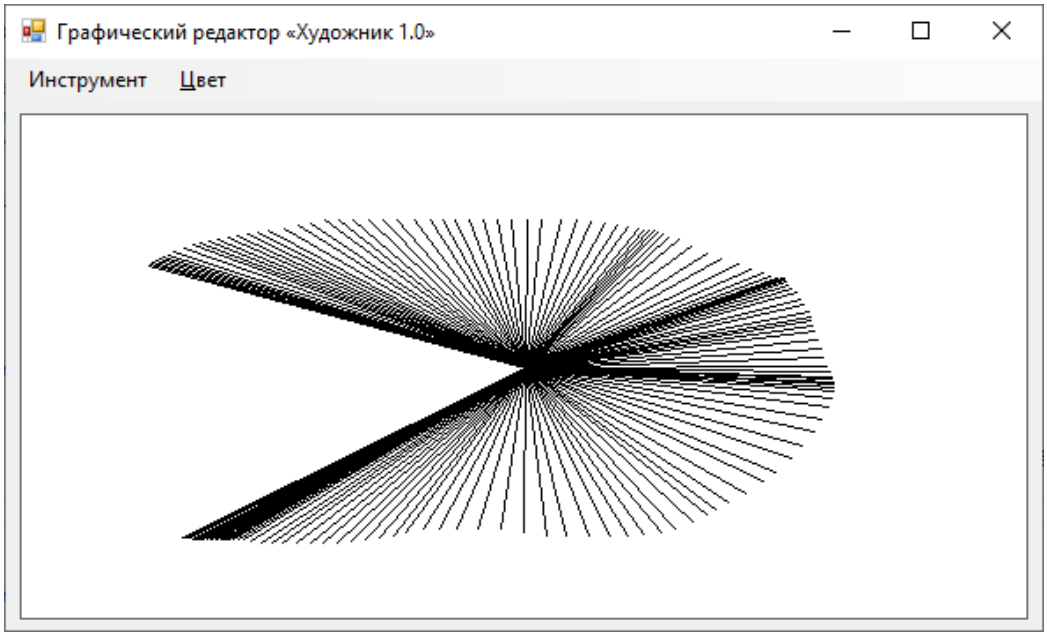

Рис. 17-2 – Линии в центр

6 Для этого обработаем значение параметра MouseEventArgs e события *MouseMove* для *picКартинка* следующим образом:

```
private void picКартинка_MouseMove(object sender, MouseEventArgs e)
{
        if (e.Button != MouseButtons.Left) return;
        gr.DrawLine(p, e.X, e.Y, picКартинка.Width / 2, picКартинка.Height / 2);
        picКартинка.Image = buf;
}
```

7 Для выполнения дополнительного и контрольных заданий добавим в нашу форму элемент *MenuStrip* (стандартное меню) (Рис. 17-3), в котором будут перечисляться созданные графические инструменты. Дайте ему имя *mnuИнструмент*, а свойство *Text* сделайте равным *Инструмент*.

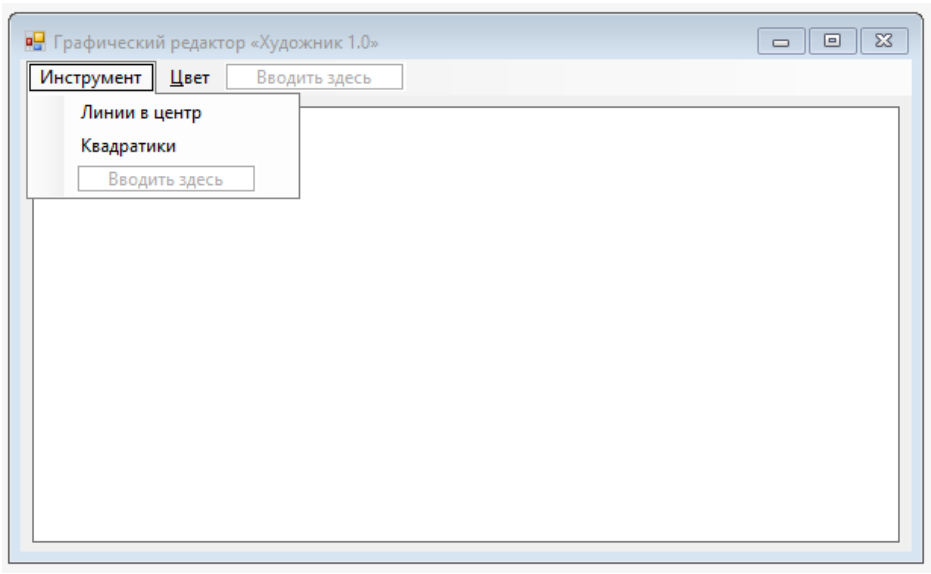

Рис. 17-3 – Конструктор меню *Инструменты*

8 Поместите в меню *mnuИнструмент* приведенные ниже элементы *ToolStripMenuItem* (пункты) (табл. 1).

Таблица 1

Text	Name
Линии в центр	toolStripMenuItem1
Квадратики	toolStripMenuItem2

9 Заставим работать пункты меню *Инструмент*.

10 Объявите новую переменную в классе текущей формы:

```
int Инструмент = 1;
```

11 Обработайте событие *Click* (перемещение мыши) для первого пункта нашего меню *toolStripMenuItem1* следующим образом:

```
private void toolStripMenuItem1_Click(object sender, EventArgs e)
{
        Инструмент = 1;
}
```

12 Измените событие *MouseMove* (перемещение мыши) для *PictureBox* следующим образом:

```
private void picКартинка_MouseMove(object sender, MouseEventArgs e)
{
        if (e.Button != MouseButtons.Left) return;
        if (Инструмент==1)gr.DrawLine(p, e.X, e.Y, picКартинка.Width / 2,
                picКартинка.Height / 2);
        picКартинка.Image = buf;
}
```

13 Запустите и отладьте приложение. Сохраните его.

Дополнительное задание

1 Добавьте инструмент *Квадратики*(☐), который будет оставлять шлейф из квадратиков за перемещающимся указателем мыши.

2 Добавьте новое меню *Цвет* и возможность менять цвет пера. Используйте при этом элемент *ColorDialog*.

3 Добавьте возможность очистки окна рисования по щелчку правой кнопкой мыши на нем.

Контрольные задания

Добавьте в наш графический редактор следующие новые функции:

1) инструмент *Диагональный отрезок* (/);
2) инструмент *Диагональный отрезок* (\);
3) инструмент *Перекрестье* (+);
4) инструмент *Перекрестье* (×);
5) инструмент *Окружность*(○);

6) инструмент *Круги* (●);
7) инструмент *Треугольники* (Δ);
8) инструмент *Распылитель*;
9) пункт меню *Толщина* для изменения толщины рисования.

Лабораторная работа 18. Разработка приложения с многодокументным интерфейсом (*MDI-приложения*)

Цель работы. Преобразовать приложение из занятия 17 в приложение с многодокументным интерфейсом.

Справочный материал

Приложения с интерфейсом MDI
Справку можно получить по ссылке:
https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/advanced/multiple-document-interface-mdi-applications?view=netframeworkdesktop-4.8

Bitmap Конструкторы
Справку можно получить по ссылке:
https://docs.microsoft.com/ru-ru/dotnet/api/system.drawing.bitmap.-ctor?view=net-5.0

Image Класс
Справку можно получить по ссылке:
https://docs.microsoft.com/ru-ru/dotnet/api/system.drawing.image?view=net-5.0

Методика выполнения

1 Многодокументный интерфейс *Windows* (*MDI*) позволяет программам работать с множеством форм (Рис. 18-1), расположенных в одной родительской форме. Применение *MDI* сделает интерфейс вашей программы более аккуратным, поскольку находящиеся в нем формы не будут разбросаны по всему экрану.

Стандарт MDI позволяет совершенствовать программы в двух направлениях. Во-первых, вы сможете ограничиться одной формой контейнером, которая станет рабочим фоном вашего приложения. Если пользователь переместит контейнер, вместе с ним переместятся и все дочерние формы, что сделает интерфейс вашей программы организованным и самодостаточным. Во-вторых, что еще важнее, пользователь может одновременно работать с несколькими документами. MDI-приложения позволяют использовать несколько экземпляров одной и той же формы, что увеличивает производительность и гибкость ваших программ.

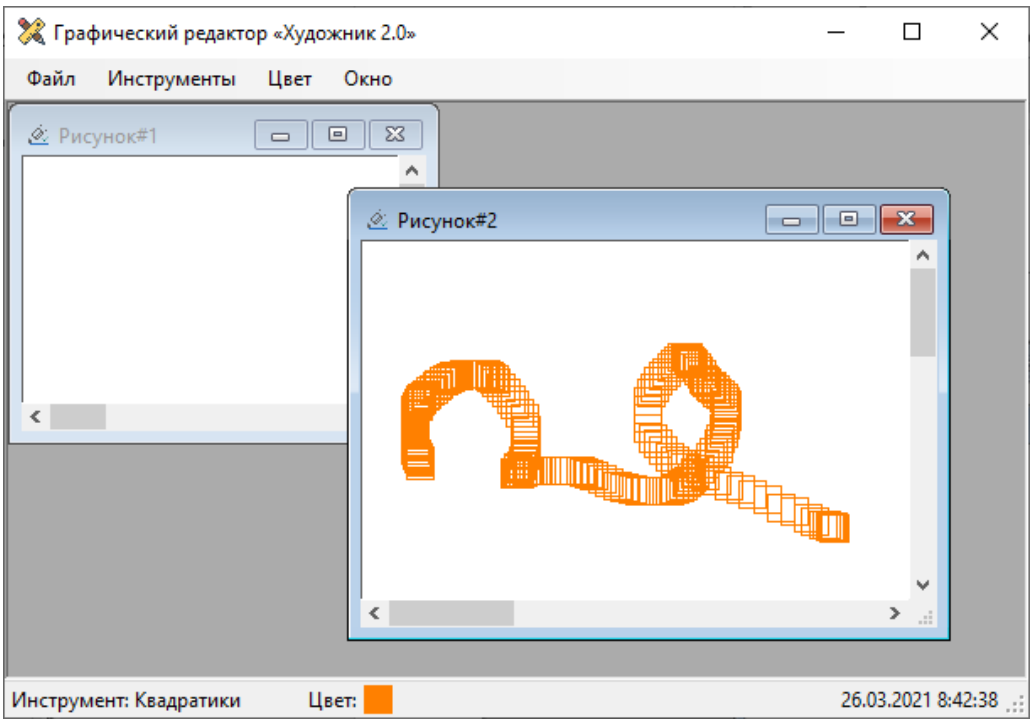

Рис. 18-1 – Редактор стал «взрослее»!

2 Рассмотрим на примере, как работает *MDI*-интерфейс, усовершенствовав наш графический редактор. В данном случае из проекта 6 будет взят лишь код, так как модернизация приложения будет неэффективной.

3 Итак, создайте новый проект приложения Windows в Visual Studio.

4 Дайте основной форме проекта имя *mdiGeneral*. Установите значение свойства *Text* родительской формы равным *Графический редактор Художник* 2.0. В окне *Свойства* присвойте *IsMdiContainer* свойству значение *true*. При этом форма назначается в качестве MDI-контейнера для дочерних окон. Кроме этого для свойства *WindowState* можно задать значение *Maximized*, так как управлять дочерними MDI-окнами проще, когда родительская форма развернута.

5 Каждое загруженное дочернее окно занимает некоторый объем оперативной памяти. Поэтому, если предполагается интенсивно использовать в программе несколько экземпляров дочерних окон, необходимо свести в них к минимуму объем программного кода и количество элементов управления. В противном случае ваше приложение может резко замедлить работу компьютера из-за нехватки оперативной памяти. Родительская MDI-форма является контейнером, в котором располагаются все дочерние окна. Поэтому именно она ответственна за создание дочерних окон, и, как правило, в ней отслеживается количество созданных дочерних окон. Кроме того, в родительской форме обычно хранятся совместно используемые элементы интерфейса, например, такие как панель инструментов, строка состояния и т. д. Поэтому выполните следующие операции.

6 Объявите новые объекты в классе родительской формы *mdiGeneral* (перо для рисования, счетчик дочерних окон, идентификатор для инструмента рисования):

```
public static Pen p = new Pen(Color.Black, 1);
int countPicture = 0;
public static int Инструмент = 1;
```

Две переменные объявлены глобальными!

7 Создайте на форме *mdiGeneral* один элемент управления *MenuStrip* (стандартное меню верхнего уровня) (Рис. 18-2). Дайте ему имя *mnuGeneral*.

Далее добавьте четыре пункта в это меню (Рис. 18-2). Свойство *Text* для них задайте *Файл*, *Инструменты*, *Цвет* и *Окно* соответственно.

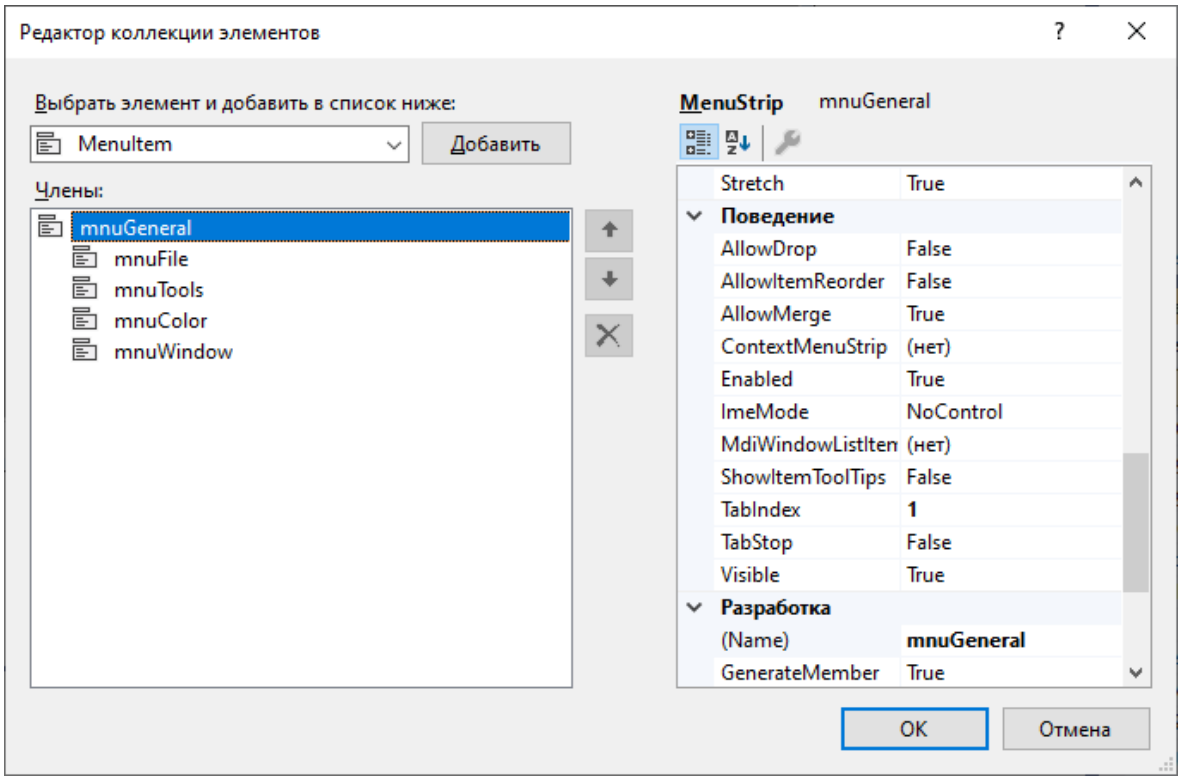

Рис. 18-2 – Основное меню Редактора

8 В меню *mnuTools* добавьте соответствующие подпункты и обработайте процедуру события *Click* для них (см. Занятие 17).

9 Заставьте работать меню *mnuColor*.

10 Добавьте в проект ещё одну форму с помощью команды стандартного меню *Проект* → *Добавить форму (Windows Forms)*. Дайте ей имя *frmPicture*. Эта форма будет шаблоном для дочерних форм MDI.

11 Добавьте на форму *frmPicture* один элемент управления (контейнер) *Panel* (рис. 18-3). В него поместите элемент управления *PictureBox* для вывода графического изображения (см. рис. 18-3) и дайте ему имя *picКартинка*.

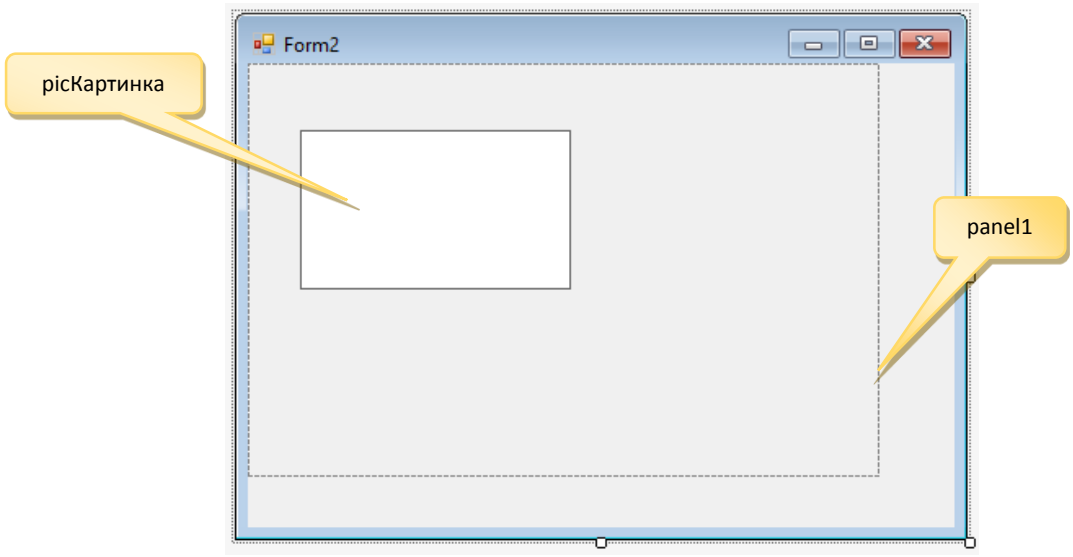

Рис. 18-3 – Основное меню Редактора

12 Задайте свойства этих элементов (табл. 2).

Таблица 2

Элемент	Свойство	Значение
panel1	AutoScroll	True
	Dock	Fill
	Location	0; 0
picКартинка	BackColor	White
	BorderStyle	FixedSingle
	Location	0; 0
	SizeMode	AutoSize

13 Объявите новые объекты в классе дочерней формы *frmPicture* (буфер для *Bitmap*-изображения и «холст»):

```
public Bitmap buf;
Graphics gr;
```

Одна из них объявлена глобальной!

14 Скопируйте из *Занятия 17* обработчики событий *MouseMove* и *MouseClick* для элемента *picКартинка*. Внесите небольшие изменения в них:

```
private void picКартинка_MouseMove(object sender, MouseEventArgs e)
{
        if (e.Button != MouseButtons.Left) return;
        if (mdiGeneral.Инструмент == 1) gr.DrawLine(mdiGeneral.p, e.X, e.Y,
                                        picКартинка.Width / 2, picКартинка.Height / 2);
        if (mdiGeneral.Инструмент == 2) gr.DrawRectangle(mdiGeneral.p, e.X - 8,
                                        e.Y - 8, 16, 16);
        picКартинка.Image = buf;
}

private void picКартинка_MouseClick(object sender, MouseEventArgs e)
{
```

```
        if (e.Button == MouseButtons.Right)
            {
                gr.Clear(Color.White);
                picКартинка.Image = buf;
            }
}
```

15 Обработайте процедуру загрузки дочерней формы:

```
private void frmPicture_Load(object sender, EventArgs e)
{
    gr = Graphics.FromImage(buf);
    picКартинка.Image = buf;
}
```

16 Все подготовительные операции проведены. Когда базовая дочерняя форма создана, нам потребуется инструмент для создания экземпляров этой формы во время выполнения программы и вывод их на экран *MDI*-приложения.

17 Добавьте три пункта в меню *Файл* (рис. 18-4). Свойство *Text* для них задайте *Новое полотно*, *Открыть изображение* и *Сохранить изображение* соответственно.

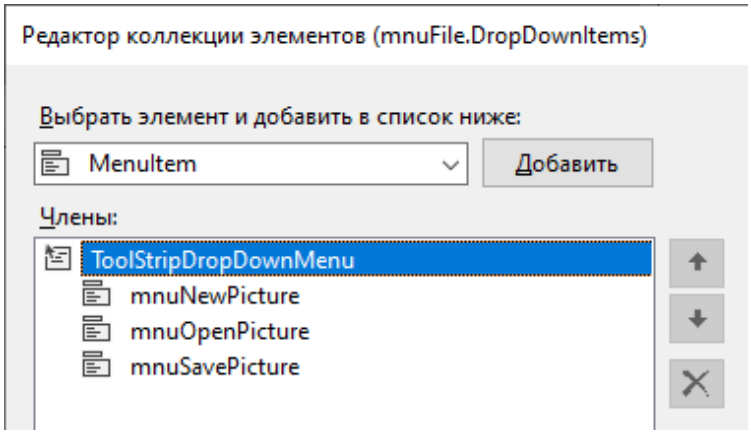

Рис. 18-4 – Меню *Файл*

18 Обработайте процедуру *mnuNewPicture_Click* следующим образом:

```
private void mnuNewPicture_Click(object sender, EventArgs e)
{
    countPicture++;
    frmPicture childForm = new frmPicture
    {
        MdiParent = this,
        Text = "Рисунок#" + countPicture.ToString(),
        buf = new Bitmap(1280, 720)
    };
    childForm.Show();
}
```

Давайте разберемся в приведенном листинге. Сначала актуализируем счетчик открытых за время сеанса дочерних форм.

Затем создаем новый экземпляр дочерней формы, используя форму *frmPicture* как шаблон.

В момент создания инициализируем свойства новой формы: сообщаем имя родительской формы, изменяем заголовок окна и инициализируем буфер для изображения)

После того, как создана объектная переменная для вывода формы на экран, используется метод *Show*.

19 Сразу же запрограммируем команду меню *Файл* → *Открыть изображение*:

```
private void mnuOpenPicture_Click(object sender, EventArgs e)
{
        OpenFileDialog openFileDialog1 = new OpenFileDialog();
        openFileDialog1.ShowDialog();
        Image pic= Image.FromFile(openFileDialog1.FileName);
        float k=1;
        if (pic.Width >= pic.Height && pic.Width > 1280) k = 1280f / pic.Width;
        else if (pic.Width < pic.Height && pic.Height > 720) k = 720f / pic.Height;
        frmPicture childForm = new frmPicture
        {
                MdiParent = this,
                Text = openFileDialog1.FileName,
                buf = new Bitmap(pic, (int)(pic.Width * k), (int)(pic.Height * k))
        };
        childForm.Show();
}
```

Давайте разберемся в приведенном листинге. Сначала мы динамически создаем новый экземпляр элемента *OpenFileDialog* и вызываем его.

Затем формируем новую переменную pic типа *Image* на базе содержимого выбранного графического файла. Если рисунок слишком большой, то предварительно масштабируем его до размеров 1280×720. Для этих целей вычисляем коэффициент k.

Наконец, создаем новый экземпляр дочерней формы, используя форму *frmPicture* как шаблон.

В момент создания инициализируем свойства новой формы: сообщаем имя родительской формы, изменяем заголовок окна и инициализируем буфер для изображения)

После того как создана объектная переменная для вывода формы на экран, используется метод *Show*.

20 Сохраните проект и запустите программу. Создайте пару новых дочерних окон и посмотрите, как ведут себя дочерние формы внутри родительской. Чтобы лучше понять особенности их поведения, выполните следующие действия:

- Минимизируйте дочернюю форму и обратите внимание на расположение ее пиктограммы.
- Переместите дочернюю форму. Она не выйдет за границы окна родительской формы.

- Максимизируйте дочернюю форму.
- Минимизируйте и максимизируйте родительскую форму.

21 Один из способов доступа к дочерним формам приложения – создание списка открытых дочерних форм. Реализовать эту возможность довольно просто. Мы уже создали меню *mnuWindow*. Теперь добавьте в конструктор родительской формы одну строчку кода (рис. 18-5).

```
public mdiGeneral()
{
    InitializeComponent();
    mnuGeneral.MdiWindowListItem=mnuWindow;
}
```

Рис. 18-5 – Изменение в конструкторе родительской формы

Теперь по мере добавления дочерних форм, список окон в данном меню будет пополняться. Название пункта меню берется из строки заголовка созданной вами формы.

22 Сохраните проект и запустите программу. Протестируйте новые пункты меню.

23 Наконец, немного полезной информации! При работе с *MDI*-приложениями широко используются два особенных ключевых слова – *this* и *ActiveMdiChild*. Ключевое слово *this* используется в любой форме для ссылки на нее саму. В свойстве родительской *MDI*-формы *ActiveMdiChild* находится указатель на активную в текущий момент дочернюю форму.

Дополнительное задание

1 Самостоятельно усовершенствуйте меню *Окно*. Добавьте в него команды для упорядочивания дочерних окон (*Каскад, Вертикально, Горизонтально, Иконки*). Воспользуйтесь методом *MDI*-формы *LayoutMdi*.

2 Запрограммируем команду меню *Файл* → *Сохранить изображение*:

```
private void mnuSavePicture_Click(object sender, EventArgs e)
{
  SaveFileDialog saveFileDialog1 = new SaveFileDialog();
  saveFileDialog1.ShowDialog();
  frmPicture X = (frmPicture)ActiveMdiChild;
  X.buf.Save(saveFileDialog1.FileName,
System.Drawing.Imaging.ImageFormat.Png);
  ActiveMdiChild.Text = saveFileDialog1.FileName;
}
```

Самостоятельно разберитесь в приведенном листинге.

Контрольные задания

Создайте меню *Вид* и добавьте в него команды *Панель инструментов* и *Строка состояния* для отображения соответствующих элементов окна родительской формы.

В *строке состояния* отобразить:

1 инструмент для рисования

2 цвет для рисования

3 размер изображения

4 системную дату и время

На *панели инструментов* отобразить:

5 кнопки для инструментов рисования

Усовершенствуйте приложение так, чтобы в каждом дочернем окне в промежутках между активизациями сохранялась следующая информация:

6 инструмент для рисования

7 цвет для рисования

8 Усовершенствуйте пункт меню *Новое полотно* так, чтобы можно было задать размер рисунка.

9 Усовершенствуйте пункт меню *Открыть изображение* так, чтобы можно было задать размер рисунка.

Приложение 1. Работа с формами

Внешний вид приложения является нам преимущественно через *формы*. Формы являются основными строительными блоками. Они предоставляют контейнер для различных элементов управления. А механизм *событий* позволяет элементам формы отзываться на ввод пользователя, и, таким образом, взаимодействовать с пользователем.

При открытии проекта в Visual Studio в графическом редакторе мы можем увидеть визуальную часть формы – ту часть, которую мы видим после запуска приложения и куда мы переносим элементы с панели управления. Но на самом деле форма скрывает мощный функционал, состоящий из методов, свойств, событий и прочее.

Рассмотрим основные *свойства* форм. Большинство этих свойств оказывает влияние на визуальное отображение формы.

Свойство	Описание характеристики
Name	устанавливает имя формы, точнее имя класса, который наследуется от класса *Form*
BackColor	указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры
BackgroundImage	указывает на фоновое изображение формы
ControlBox	указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение *false*, то не видны ни иконка, ни крестик, с помощью которого обычно закрывается форма
Cursor	определяет тип курсора, который используется на форме
Font	задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, возможно тем самым переопределить его
ForeColor	цвет шрифта на форме
FormBorderStyle	указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в *None* можно создавать внешний вид приложения произвольной формы
Icon	задает иконку формы
MaximizeBox	указывает, будет ли доступна кнопка максимизации окна в заголовке формы
MinimizeBox	указывает, будет ли доступна кнопка минимизации окна
MaximumSize	задает максимальный размер формы
MinimumSize	задает минимальный размер формы
Size	определяет начальный размер формы
StartPosition	указывает на начальную позицию, с которой форма появляется на экране
Text	определяет заголовок формы
Visible	видима ли форма, если нужно скрыть форму от пользователя, то следует задать данному свойству значение *false*
WindowState	указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном

Для взаимодействия с пользователем в Windows Forms используется механизм *событий*. События в Windows Forms представляют стандартные события на C#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в C#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего в Windows Forms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм. Например, события формы:

Событие	Описание события
Click	происходит при щелчке основной кнопкой мыши
Load	происходит до первоначального отображения формы

Более подробно с данным компонентом можно познакомиться по **ссылке**: https://metanit.com/sharp/windowsforms/2.1.php

Приложение 2. Элементы управления

Элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события. Все элементы управления наследуются от класса **Control** и поэтому имеют ряд общих свойств:

Свойство	Описание характеристики
BackColor	Определяет фоновый цвет элемента
BackgroundImage	Определяет фоновое изображение элемента
ContextMenu	Контекстное меню, которое открывается при нажатии на элемент правой кнопкой мыши. Задается с помощью элемента **ContextMenu**
Cursor	Представляет, как будет отображаться курсор мыши при наведении на элемент
Dock	Задает расположение элемента на форме
Enabled	Определяет, будет ли доступен элемент для использования. Если это свойство имеет значение *False*, то элемент блокируется.
Font	Устанавливает шрифт текста для элемента
ForeColor	Определяет цвет шрифта
Location	Определяет координаты верхнего левого угла элемента управления
Name	Имя элемента управления
Size	Определяет размер элемента
Width	ширина элемента
Height	высота элемента
TabIndex	Определяет порядок обхода элемента по нажатию на клавишу **Tab**

КНОПКА BUTTON

Наиболее часто используемым элементом управления является кнопка. Обрабатывая событие нажатия кнопки, мы может производить те или иные действия.

Более подробно с компонентом **Button** можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.1.php

ЭЛЕМЕНТ GROUPBOX (КОНТЕЙНЕР)

GroupBox представляет собой специальный контейнер, который ограничен от остальной формы границей. Он имеет заголовок, который устанавливается через свойство **Text**. Чтобы сделать **GroupBox** без заголовка, в качестве значения свойства **Text** просто устанавливается пустая строка.

Нередко этот элемент используется для группирования переключателей – элементов **RadioButton**, так как позволяет разграничить их группы.

Более подробно с компонентом **GroupBox** можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/3.2.php

МЕТКА LABEL

Для отображения простого текста на форме, доступного только для чтения, служит элемент **Label**. Чтобы задать отображаемый текст метки, надо установить свойство **Text** элемента.

Свойство	Описание характеристики
Text	получает или задает текст, сопоставленный с этим элементом управления
TextAlign	получает или задает выравнивание текста в элементе управления *Label*
Image	возвращает или задаёт изображение, отображаемое на элементе управления *Label*
Visible	получает или задает значение, указывающее, отображается ли элемент управления

Событие	Описание события
Click	происходит при щелчке элемента управления
DoubleClick	происходит при двойном щелчке мышью элемента управления *Label*

Более подробно с компонентом **Label** можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.2.php

ТЕКСТОВОЕ ПОЛЕ TEXTBOX

Для ввода и редактирования текста предназначены текстовые поля – элемент **TextBox**. Как и у элемента **Label** текст элемента **TextBox** можно установить или получить с помощью свойства **Text.**

По умолчанию при переносе элемента с панели инструментов создается однострочное текстовое поле. Для отображения больших объемов информации в текстовом поле нужно использовать его свойства **Multiline** и **ScrollBars.**

При установке для свойства **Multiline** значения *true*, все избыточные символы, которые выходят за границы поля, будут переноситься на новую строку.

Свойство	Описание характеристики
Text	получает или задает текст, сопоставленный с этим элементом управления.
TextAlign	получает или задает выравнивание текста в элементе управления *TextBox*
MaxLength	получает или задает максимальное разрешенное число знаков в текстовом окне (по умолчанию 32767 символов)
Multiline	определяет поддерживает ли элемент многострочный текст
Visible	получает или задает значение, указывающее, отображается ли элемент управления
ReadOnly	возвращает или задает значение, определяющее возможность изменения содержимого элемента управления *TextBox*
Rows	возвращает или задает число строк, отображаемых в многострочном текстовом окне
Columns	возвращает или задает ширину отображаемого текстового окна в знаках

Событие	Описание события
Click	происходит при щелчке элемента управления
DoubleClick	происходит при двойном щелчке мышью элемента управления *TextBox*
KeyDown	происходит при нажатии клавиши на клавиатуре, когда фокус находится на элементе *TextBox*
KeyPress	происходит при вжатом положении клавиши на клавиатуре, когда фокус находится на элементе *TextBox*
KeyUp	происходит при отпускании клавиши на клавиатуре, когда фокус находится на элементе *TextBox*
TextChanged	происходит при изменении содержимого текстового окна *TextBox*

Более подробно с компонентом **TextBox** можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.3.php

ЭЛЕМЕНТ RICHTEXTBOX

RichTextBox – расширенное поле ввода – дает возможность пользователю вводить и обрабатывать большие объемы информации (более 64 кБт). Кроме того, **RichTextBox** позволяет редактировать цвет текста, шрифт, добавлять изображения. **RichTextBox** включает все возможности текстового редактора Microsoft Word.

Свойство	Описание характеристики
Text	получает или задает текст, сопоставленный с этим элементом управления
Lines	массив строк текстового поля
MaxLength	получает или задает максимальное разрешенное число знаков в текстовом окне (по умолчанию 2147483647 символов)
Multiline	определяет поддерживает ли элемент многострочный текст
Name	возвращает или задает имя элемента управления. Стандарты те же что и для переменных C++
Visible	получает или задает значение, указывающее, отображается ли элемент управления

Свойство	Описание характеристики
ReadOnly	возвращает или задает значение, определяющее возможность изменения содержимого элемента управления *RichTextBox*
Rows	возвращает или задает число строк, отображаемых в многострочном текстовом окне
Columns	возвращает или задает ширину отображаемого текстового окна в знаках
ScrollBars	получает или задает тип полос прокрутки, отображающихся в элементе управления *RichTextBox*

Событие	Описание события
Click	происходит при щелчке элемента управления
DoubleClick	происходит при двойном щелчке мышью элемента управления *RichTextBox*
KeyDown	происходит при нажатии клавиши на клавиатуре, когда фокус находится на элементе *RichTextBox*
KeyPress	происходит при вжатом положении клавиши на клавиатуре, когда фокус находится на элементе *RichTextBox*
KeyUp	происходит при отпускании клавиши на клавиатуре, когда фокус находится на элементе *RichTextBox*
TextChanged	происходит при изменении содержимого текстового окна *RichTextBox*

Более подробно с компонентом **RichTextBox** можно познакомиться по ссылке: https://ci-sharp.ru/obuchenie/gui-vizualnye-komponenty/komponent-richtextbox-c-podrobnyj-razbor-i-primery/

ОКНО СООБЩЕНИЯ MESSAGEBOX

Как правило, для вывода сообщений применяется элемент **MessageBox**. Однако кроме собственно вывода строки сообщения данный элемент может устанавливать ряд настроек, которые определяют его поведение.

Для вывода сообщения в классе **MessageBox** предусмотрен метод **Show()**, который имеет различные версии и может принимать ряд параметров. Рассмотрим одну из наиболее используемых версий:

```
public static DialogResult Show(
    string text,
    string caption,
    MessageBoxButtons buttons,
    MessageBoxIcon icon,
    MessageBoxDefaultButton defaultButton,
    MessageBoxOptions options
)
```

Здесь применяются следующие параметры:

- **text**: текст сообщения;
- **caption**: текст заголовка окна сообщения;
- **buttons**: кнопки, используемые в окне сообщения. Принимает одно из значений перечисления **MessageBoxButtons**;
- **icon**: значок окна сообщения. Может принимать одно из значений перечисления **MessageBoxIcon**;

▪ **defaultButton**: кнопка, на которую по умолчанию устанавливается фокус. Принимает одно из значений перечисления **MessageBoxDefaultButton**.

Нередко используется один параметр – текст сообщения.

Однако нам не просто дается возможность установки кнопок в окне сообщения. Метод **MessageBox.Show()** возвращает объект **DialogResult**, с помощью которого мы можем узнать, какую кнопку в окне сообщения нажал пользователь. **DialogResult** представляет перечисление, в котором определены следующие значения:

- **Abort**: нажата кнопка **Прервать**;
- **Retry**: нажата кнопка **Повторить**;
- **Ignore**: нажата кнопка **Пропустить**;
- **OK**: нажата кнопка **OK**:
- **Cancel**: нажата кнопка **Отмена**;
- **None**: отсутствие результата;
- **Yes**: нажата кнопка **Да**;
- **No**: нажата кнопка **Нет**.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.19.php

INPUTBOX В C#

C# предлагает разработчику удобный класс **MessageBox** в пространстве имён **System.Windows.Forms**. Однако ориентирован он исключительно на диалоговые окна вывода различного рода сообщений. Диалоговых окон ввода данных (InputBox) не предусмотрено. Странная асимметрия. А ведь получать данные от пользователя приходится почти в каждом проекте! Конечно, можно создать собственный класс диалогового окна и реализовать весь необходимый функционал. Но гораздо быстрее и проще в условиях ограниченности времени использовать **InputBox** из сборки **Microsoft.VisualBasic.dll**. Подключив сборку к проекту, можно написать в тексте программы

```
var s = Interaction.InputBox("Text", "Caption", "Value", posX, posY);
```

Первые три параметра имеют формат строки. Последние два в формате целого числа управляют расположением окошка на экране монитора. Это очень важные параметры, так как **InputBox** не предлагает никаких средств для привязки его к родительскому окну. Поэтому, чтобы окошко не появлялось на экране где попало используется его позиционирование. Значение, введённое пользователем, возвращается функцией **InputBox** в виде строки.

Более подробно с функцией можно познакомиться по ссылке: https://learn.microsoft.com/ru-ru/office/vba/language/reference/user-interface-help/inputbox-function

ЭЛЕМЕНТ PICTUREBOX

В C# – это элемент управления Windows Forms, предназначенный для отображения изображений из различных источников, таких как файлы (*bmp, jpg, gif*), метафайлы и иконки. Его можно использовать для загрузки изображений во

время разработки или выполнения, а также для масштабирования, рисования и обработки событий, например, щелчков мыши.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.16.php

ЭЛЕМЕНТ CHECKBOX

Элемент CheckBox (чекбокс) в C# – это элемент управления для создания графического пользовательского интерфейса (GUI), позволяющий пользователю выбрать один из двух вариантов: «да/нет» или «true/false».

Для управления его состоянием используется свойство **Checked** (возвращает *bool* или *ThreeState*), а для реагирования на изменения – событие **CheckedChanged**.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.5.php

ЭЛЕМЕНТ NUMERICUPDOWN

NumericUpDown – это элемент управления Windows Forms в C#, который позволяет пользователю выбирать числовое значение с помощью кнопок со стрелками вверх/вниз или путем ввода значения непосредственно в текстовое поле. Он полезен для задания чисел в определенном диапазоне, например, для установки громкости, количества предметов или значения настройки. Ключевые свойства включают **Value**, **Minimum**, **Maximum** и **DecimalPlaces** для управления текущим значением, диапазоном и точностью.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.10.php

ЭЛЕМЕНТ DATAGRIDVIEW

DataGridView в C# – это элемент управления, который используется для отображения данных в виде таблицы и позволяет пользователям работать с ними. С его помощью можно выводить данные из различных источников (например, из базы данных), редактировать их непосредственно в ячейках, сортировать, выбирать строки, а также настраивать внешний вид таблицы и отдельных столбцов и ячеек.

Более подробно с данным компонентом можно познакомиться по ссылке: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/controls/customizing-the-windows-forms-datagridview-control

ЭЛЕМЕНТ CHART

Элемент управления **Chart** (диаграмма) – это объект, используемый в C# для создания и отображения диаграмм, который предоставляет события для программирования, например, для добавления данных или настройки внешнего вида. Он доступен в проектах Visual Studio, в том числе в проектах, связанных с

MS Office, где вы можете добавлять диаграммы непосредственно на лист и работать с ними программно.

Построение графика (диаграммы) производится после вычисления таблицы значений функции $y = f(x)$ на интервале $[x_{min}, x_{max}]$ с заданным шагом. Полученная таблица передается в специальный массив **Points** объекта **Series** элемента управления **Chart** с помощью метода **DataBindXY**. Элемент управления **Chart** осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. В элементе управления **Chart** можно настроить толщину, стиль и цвет линий, параметры шрифта подписей, шаги разметки координатной сетки и многое другое. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам элемента управления **Chart.** Так, например, свойство **AxisX** содержит значение максимального предела нижней оси графика, и при его изменении во время работы программы автоматически изменяется изображение графика.

Более подробно с данным компонентом можно познакомиться по ссылке: https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1

ЭЛЕМЕНТ MENUSTRIP

MenuStrip – это элемент управления в C# (Windows Forms), который предоставляет контейнер для создания структурированной строки меню приложения. Он используется для добавления команд, подменю и раскрывающихся списков с помощью **ToolStripMenuItem**. **MenuStrip** заменяет собой устаревший элемент **MainMenu** и предназначен для замены основного меню приложения на форме.

Более подробно с данным компонентом можно познакомиться по ссылке: https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1

ЭЛЕМЕНТ OPENFILEDIALOG

OpenFileDialog – это компонент C# для Windows Forms, который отображает стандартное диалоговое окно «Открыть файл», позволяя пользователю выбрать один или несколько файлов на диске. Чтобы использовать его, необходимо добавить компонент на форму, настроить его свойства, такие как **Title** и **Filter**, и вызвать метод **ShowDialog()**, который открывает диалоговое окно. Затем, если пользователь нажал OK, можно получить полный путь к выбранному файлу из свойства **FileName** или **FileNames**.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.20.php

ЭЛЕМЕНТ SAVEFILEDIALOG

SaveFileDialog – это компонент Windows Forms C# для создания диалогового окна, которое позволяет пользователю выбрать путь и имя файла для сохранения. Это не сам компонент для записи данных на диск, а лишь инструмент для выбора места и имени файла; фактическая запись данных требует дополнительного кода, например, с использованием **System.IO.File** или **StreamWriter**.

Более подробно с данным компонентом можно познакомиться по ссылке: https://metanit.com/sharp/windowsforms/4.20.php

Приложение 3. Описание некоторых операций C#

В C# используется большинство операций, которые применяются и в других языках программирования. Операции представляют определенные действия над операндами – участниками операции. В качестве операнда может выступать переменной или какое-либо значение (например, число). Операции бывают унарными (выполняются над одним операндом), бинарными – над двумя операндами и тернарными – выполняются над тремя операндами. Рассмотрим некоторые виды операций.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ ЯЗЫКА C#

Операция	Описание операции
+x	возвращает значение *x*
-x	числовое отрицание
!x	логическое отрицание
x*y	умножение
x/y	деление. При делении двух целых чисел результатом также будет целое число. Например, при делении 9/5 результатом будет число 1. **Чтобы получить точный результат с десятичной точкой, нужно чтобы делимое и/или делитель были типа float или double. Например, при делении 9 / 5f (суффикс f указывает, что данная константа типа float) результатом будет 1,8.**
x%y	остаток от деления. Если операнды имеют целые числа, это возвращает остаток от деления *x* на *y*. Если $q = x / y$ и $r = x \% y$, то $x = q * y + r$.
x||y	логическое ИЛИ. Если первый операнд имеет значение *true*, то C# не вычисляет второй операнд
x&&y	логическое И. Если первый операнд имеет значение *false*, то C# не вычисляет второй операнд

Более подробно с арифметическими операциями языка C# можно познакомиться по ссылке: https://metanit.com/sharp/tutorial/2.3.php

ОПЕРАЦИИ ПРИСВАИВАНИЯ

Инструкция	«Обычная» инструкция присваивания
x++	x = x + 1
x--	x = x – 1

x+=y	x = x + y
x-=y	x = x – y
x*=y	x = x * y
x%=y	x = x % y

Более подробно с операциями присваивания языка C# можно познакомиться по ссылке: https://metanit.com/sharp/tutorial/2.23.php

ОПЕРАТОРЫ СРАВНЕНИЯ

Оператор	Описание	Результат сравнения
>	Больше	*True*, если первый операнд больше второго, иначе *False*
<	Меньше	*True*, если первый операнд меньше второго, иначе *False*
==	Равно	*True*, если первый операнд равен второму, иначе *False*
!=	Не равно	*True*, если первый операнд не равен второму, иначе *False*
>=	Больше или равно	*True*, если первый операнд больше или равен второму, иначе *False*
<=	Меньше или равно	*True*, если первый операнд меньше или равен второму, иначе *False*

Приложение 4. Математические вычисления и класс Math

Для выполнения различных математических операций в библиотеке классов .NET предназначен класс **Math**. Он является статическим, поэтому все его методы также являются статическими.

Запись на C#	Возвращаемый результат
Math.Abs(X);	Модуль числа X
Math.Ceiling (X);	Округление числа X до большего целого
Math.Floor(X);	Округление числа X до меньшего целого
Math.Cos (X);	Косинус аргумента X
Math.E	Число e = 2,718282
Math.Exp (X);	Экспонента, число e в степени X
Math.Log(X);	Логарифм натуральный числа X
Math.Log10(X);	Логарифм десятичный числа X
Math.Max(X,Y);	Максимум из двух чисел X и Y.
Math.Min (X,Y);	Минимум из двух чисел X и Y
Math.Pi	Число π
Math.Pow(X,Y);	Число X в степени Y
Math.Round(X);	Простое округление числа X
Math.Sing(X);	Знак числа X
Math.Sin(X);	Синус аргумента X
Math.Sqrt(X);	Квадратный корень числа X
Math.Tan(X);	Тангенс аргумента X

Более подробно с содержимым класса **Math** можно познакомиться по ссылке: https://metanit.com/sharp/tutorial/20.2.php

Приложение 5. Встроенные типы данных в C#

Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных. Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.

В языке C# есть следующие базовые типы данных:

Тип	Область значений	Размер
sbyte	-128 до 127	Знаковое 8-бит целое
byte	0 до 255	Беззнаковое 8-бит целое
char	U+0000 до U+ffff	16-битовый символ Unicode
bool	true или false	Логическое. 1 байт
short	-32768 до 32767	Знаковое 16-бит целое
ushort	0 до 65535	Беззнаковое 16-бит целое
int	-2147483648 до 2147483647	Знаковое 32-бит целое
uint	0 до 4294967295	Беззнаковое 32-бит целое
long	-9223372036854775808 до 9223372036854775807	Знаковое 64-бит целое
ulong	0 до 18446744073709551615	Беззнаковое 64-бит целое
float	$\pm 1{,}5*10^{-45}$ до $\pm 3{,}4*10^{33}$	4 байта, точность – 7 разрядов
double	$\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$	8 байт, точность – 16 разрядов
decimal	$\pm 1.0*10^{-28}$ до $\pm 7.9228*10^{28}$	16 байт, точность – 28 разрядов

Более подробно со встроенными в C# типами данных можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.1.php

Приложение 6. Управляющие конструкции языка C#

УСЛОВНЫЙ ОПЕРАТОР IF..ELSE

Условные конструкции – один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий. Одной из таких конструкций в языке программирования C# является конструкция **if..else**

Конструкция **if..else** проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код.

Ее простейшая форма состоит из блока **if**:

```
if(условие)
{
    выполняемые инструкции
}
```

Более подробно с данной конструкцией можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.5.php

ЦИКЛ FOR

Цикл **for** в C# – это оператор итерации, который выполняет блок кода заданное количество раз, пока выполняется определённое условие. Он имеет три основные части: инициализатор, условие и итератор, разделённые точкой с запятой:

```
for ([действия_до_выполнения_цикла]; [условие]; [действия_после_выполнения])
{
    // действия
}
```

Инициализатор выполняется один раз в начале, условие проверяется перед каждой итерацией, а итератор выполняется после каждого выполнения тела цикла.

Более подробно с данной конструкцией можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.6.php

ЦИКЛ DO..WHILE

В цикле **do** сначала выполняется код цикла, а потом происходит проверка условия в инструкции **while**. И пока это условие истинно, цикл повторяется:

```
do
{
    действия цикла
}
while (условие)
```

Более подробно с данной конструкцией можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.6.php

ЦИКЛ WHILE

В отличие от цикла **do** цикл **while** сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

```
while (условие)
{
    действия цикла
}
```

Более подробно с данной конструкцией можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.6.php

Приложение 7. Массивы в C#

В C# массив – это структура данных, представляющая собой упорядоченный набор переменных (элементов) одного и того же типа, к которым

можно обращаться по вычисляемому индексу. Массивы являются ссылочными типами и реализованы в виде объектов в управляемом коде. Они используются для хранения связанных данных и могут быть одномерными, многомерными или «зубчатыми» (массивами массивов).

Более подробно с массивами можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/2.4.php

КЛАСС SYSTEM.RANDOM

Класс **System.Random** в C# используется для генерации псевдослучайных чисел – последовательностей чисел, которые отвечают определенным статистическим требованиям случайности.

Чтобы использовать его, нужно сначала создать экземпляр класса **Random**, а затем вызывать его методы, например, *Next*() для получения случайного целого числа или *NextDouble*() для получения числа с плавающей запятой.

Более подробно с классом можно познакомиться по **ссылке**: https://learn.microsoft.com/ru-ru/dotnet/fundamentals/runtime-libraries/system-random

КЛАСС SYSTEM.ARRAY

System.Array – это базовый класс для всех массивов в C#, который предоставляет общие методы и свойства для работы с ними, такие как **Length** (длина массива), **Rank** (размерность) и методы для поиска, сортировки и копирования элементов. Пользователи обычно не работают напрямую с классом **Array**, а используют синтаксис массивов, предоставляемый языком, но функциональность **System.Array** доступна для всех массивов.

Более подробно с классом можно познакомиться по **ссылке**: https://metanit.com/sharp/tutorial/20.5.php

Библиографический список

1. **Агуров, Павел** C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.

2. **Албахари, Джозеф** C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2012. - 944 с.

3. **Албахари, Джозеф** C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2013. - 944 с.

4. **Альфред, В. Ахо** Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.

5. **Бишоп, Дж.** C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.

6. **Вагнер, Билл** C# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 с.

7. **Зиборов, В.В.** Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 с.

8. **Зиборов, Виктор** Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.

9. **Ишкова, Э.А.** Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.

10. **Касаткин, А.И.** Профессиональное программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2012. - 432 с.

11. **Лотка, Рокфорд** C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

12. **Мак-Дональд, Мэтью** Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

13. **Марченко, А.Л.** Основы программирования на C# 2.0 / А.Л. Марченко. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. - 552 с.

14. **Подбельский, В.В.** Язык C#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

15. **Прайс, Джейсон** Visual C# 2.0. Полное руководство / Джейсон Прайс , Майк Гандэрлой. - М.: Век +, Корона-Век, Энтроп, 2010. - 736 с.

16. **Рихтер, Джеффри** CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 с.

17. **Смоленцев, Н.К.** MATLAB. Программирование на Visual C#, Borland JBuilder, VBA (+ CD-ROM) / Н.К. Смоленцев. - М.: ДМК Пресс, 2011. - 456 с.

18. **Троелсен, Эндрю** Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 с.

19. **Троелсен, Эндрю** Язык программирования C# 2008 и платформа .NET 3.5 / Эндрю Троелсен. - М.: Вильямс, 2010. - 370 с.

20. **Фримен, Адам** ASP.NET MVC 3 Framework с примерами на C# для профессионалов / Адам Фримен , Стивен Сандерсон. - М.: Вильямс, 2011. - 672 с.