

Лабораторная работа № 8

Создание простого приложения Windows Forms

Windows Forms представляет собой одну из двух технологий, используемую в **Visual C#** для создания интеллектуальных клиентских приложения на основе Windows, выполняемых в среде **.NET Framework**. Технология Windows Forms специально создана для быстрой разработки приложений, в которых обширный графический пользовательский интерфейс не является приоритетом. Для создания пользовательского интерфейса используется конструктор Windows Forms, и пользователь получает доступ к другим возможностям времени разработки и времени выполнения.

Windows Forms предоставляет для проекта такие компоненты, как диалоговые окна, меню, кнопки и многие другие элементы управления, являющиеся частью стандартного пользовательского интерфейса (UI) Windows. По существу, эти элементы управления являются просто классами из библиотеки .NET Framework. **Конструктор** в Visual Studio позволяет перетаскивать элементы управления в основную форму приложения и изменять их размеры и расположение. После этого **IDE** (среда разработки) автоматически добавит исходный код для создания и инициализации экземпляра соответствующего класса.

Создание простого приложения Windows Forms

Запускаем Visual Studio, откроется **Начальная страница**:

Для начала, надо создать проект, для этого выполним последовательно: **Файл -> Создать -> Проект...** (также можно просто нажать сочетание клавиш **Ctrl+Shift+N** или пункт «Создать проект...» на *Начальной странице*):

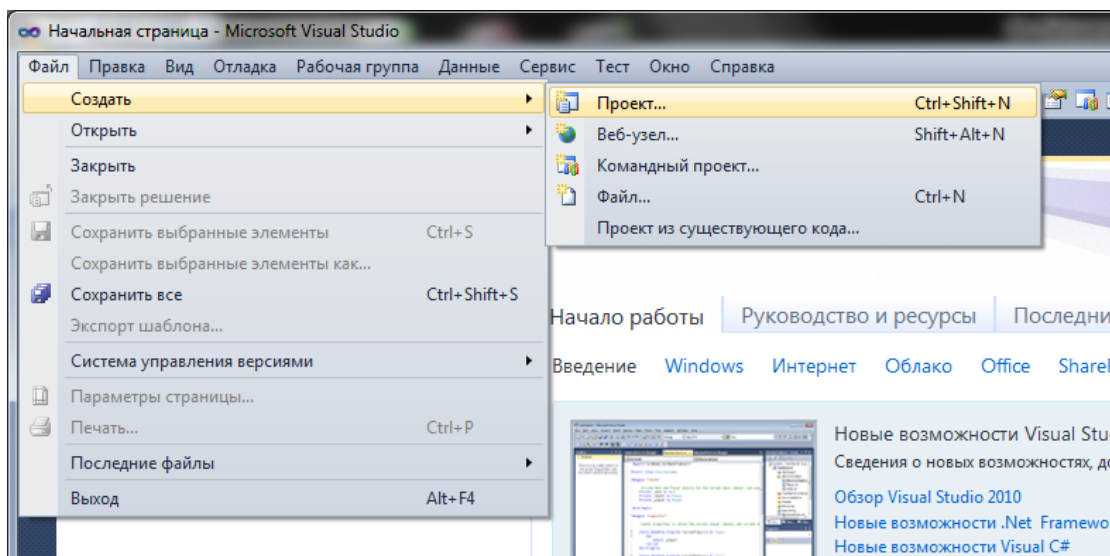


Рис. 2. 1. Создание нового проекта

Выберем слева в пункте **Установленные шаблоны** язык **Visual C#**, далее найдём в списке **Приложение Windows Forms**. Также здесь можно выбрать какой использовать «фреймворк» (набора компонентов для написания программ). В нашем случае выберем **.NET Framework 4**.

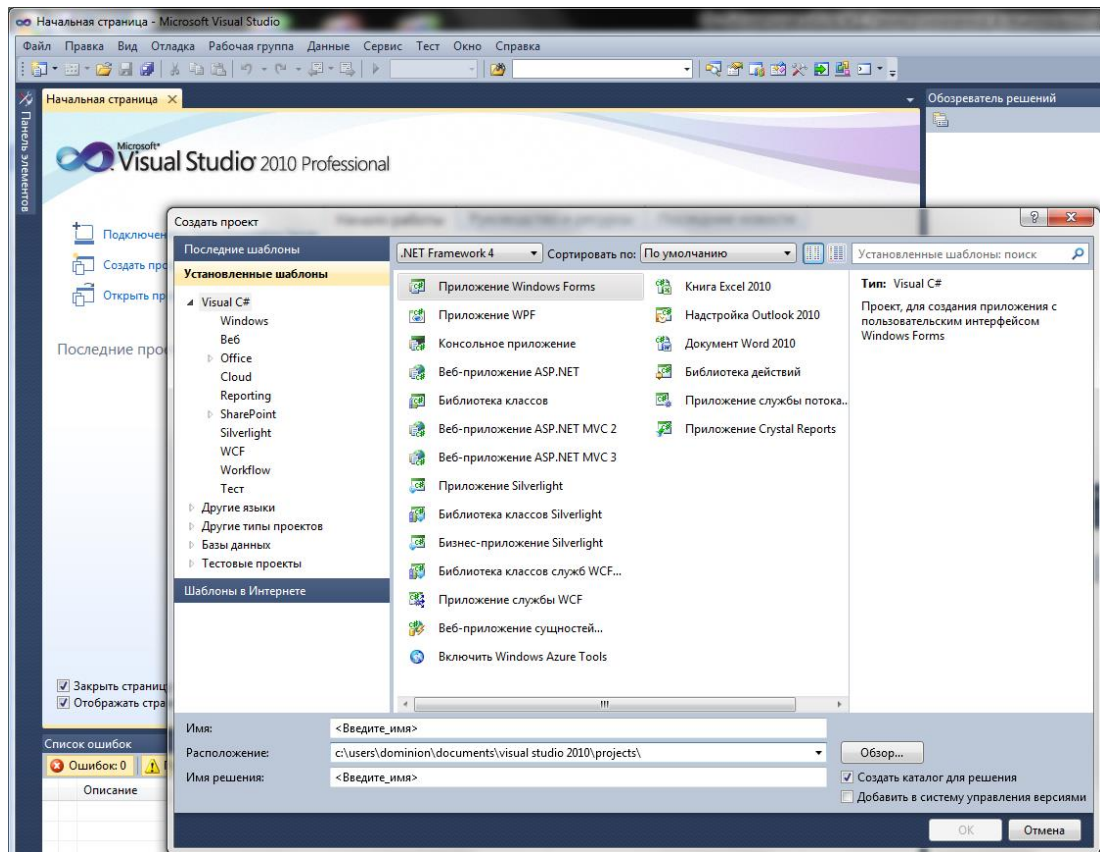


Рис. 2. 2. Окно создания нового проекта

В поле **Имя** вводим **LB02** — это название программы (выбрано по названию лабораторного практикума, номеру и названию работы). В поле **Расположение** указана конечная директория, где будет находиться весь проект (значение «по умолчанию» можно поменять, выполнив действия: **Сервис -> Параметры... -> Проекты и решения ->** меняем путь в поле **Размещение проектов**). Выберем расположение удобное для быстрого поиска. В поле **Имя решения** вводится либо название программы «по умолчанию» из поля **Имя** автоматически, либо можно ввести своё собственное. Под этим именем будет создана конечная папка проекта (если **Имя** и **Имя решения** разные).

После нажатия клавиши **ОК** мы увидим сформированный проект и исходный код приложения *Windows Forms* (не пустого изначально).

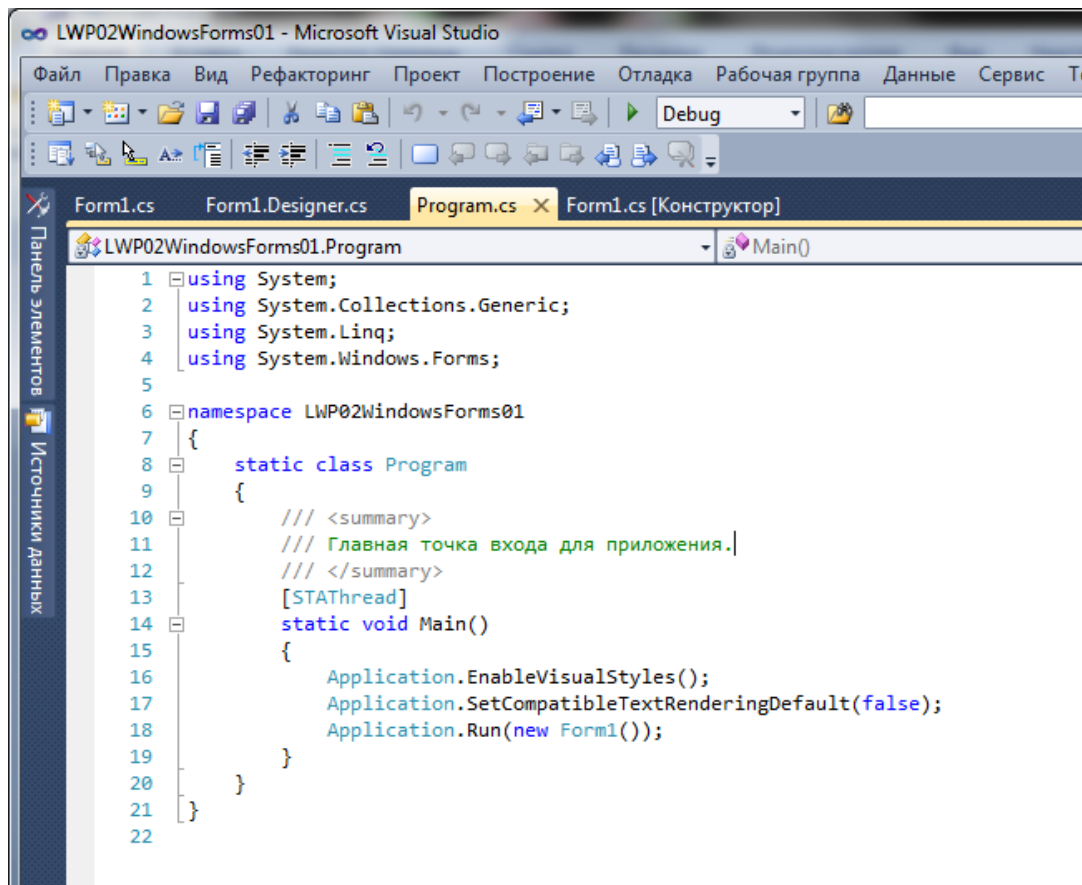


Рис. 2. 4. Исходный код приложения *Windows Forms* сформированного средой разработки (файл **Program.cs**)

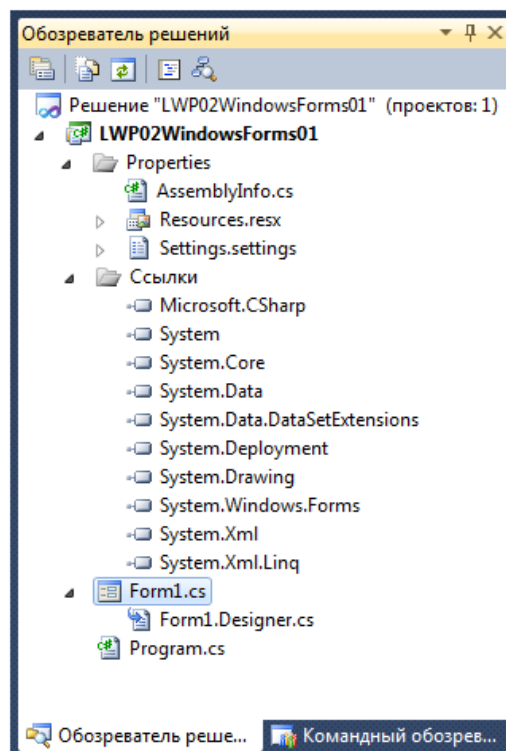
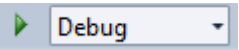


Рис. 2. 5. **Обозреватель решений**: состав проекта приложения *Windows Forms* сформированного средой разработки

Теперь, можно откомпилировать созданную программу, нажав клавишу **F5** (**Отладка** -> **Начать отладку** или нажав на иконку . Тем самым мы запускаем приложение в режиме отладки (и производим компиляцию debug-версии программы) (**Debug** выбрано изначально).

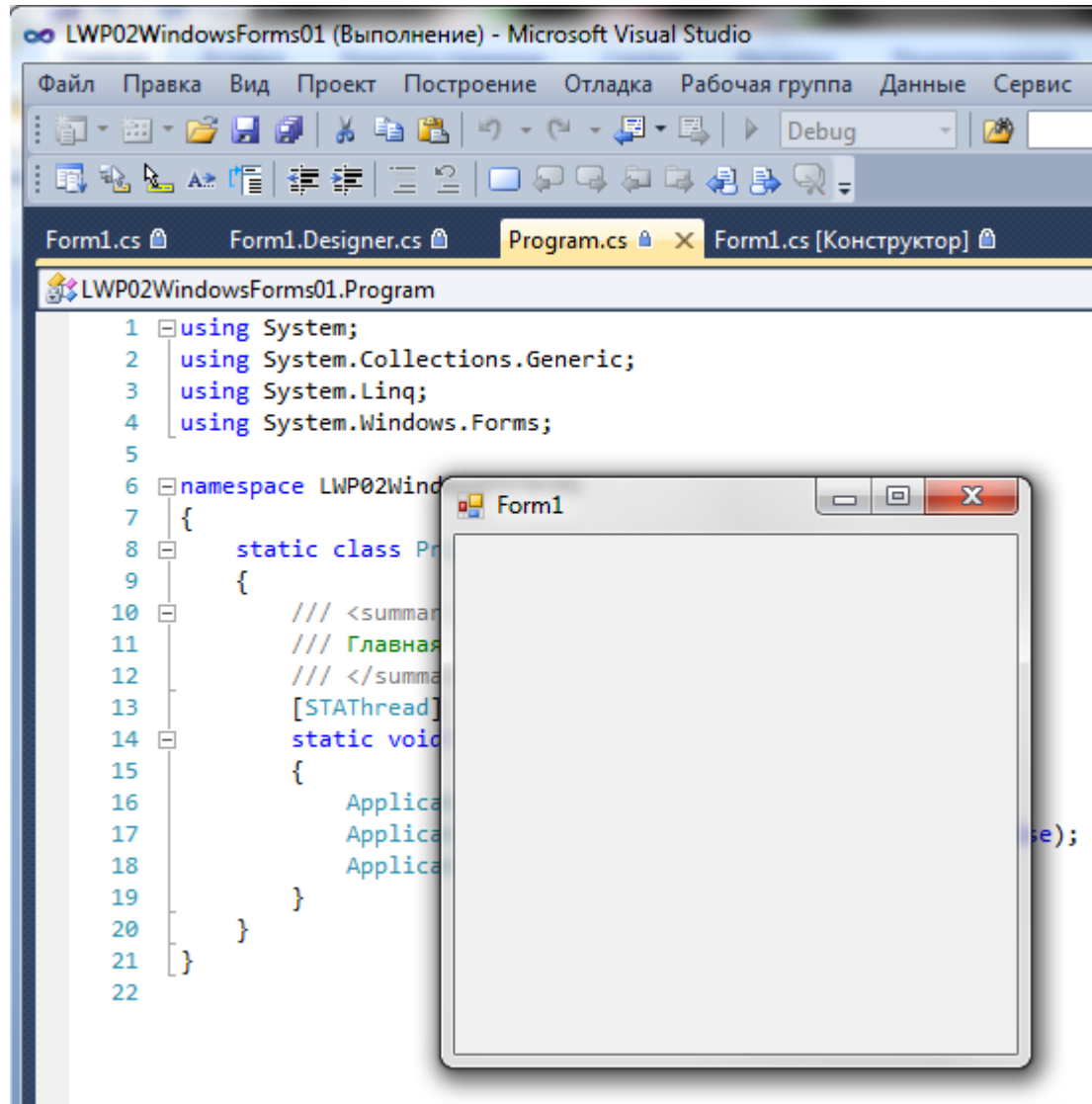


Рис. 2. 6. Запуск приложения *Windows Forms* по конфигурации *Debug*

Модификация приложения *Windows Forms*

Цель данной работы, это показать основные приёмы по работы с формами. Поэтому соберём простенькую программу калькулятора, выполняющего основные математические операции и выводящие результат в окне программы.

Что необходимо реализовать:

1. Цифры будут вводиться нажатием кнопки с обозначением соответствующей цифры (как в стандартной программе **Калькулятор** в Windows).

2. Реализуем все математические операции (сложение, вычитание, умножение и деление).

3. Все действия будут происходить в одном текстовом поле для удобства. Результат будет выводиться там же.

Стоит отметить, что реализаций «калькуляторов» за время существования C# было понаделано множество. Потому реализация приложения в данной лабораторной работе «упрощена» настолько, насколько это возможно. Иначе, код программы можно было немного уменьшить.

Первое что необходимо отметить по сравнению с консольным приложением это добавление новых инструментов в окне среды разработки. Теперь у нас есть «визуальная форма» (Рис. 3. 1), на которой можно размещать любые доступные элементы из специальной панели объектов которая называется **Панель элементов** (по умолчанию находится слева сбоку на границе среды разработки):

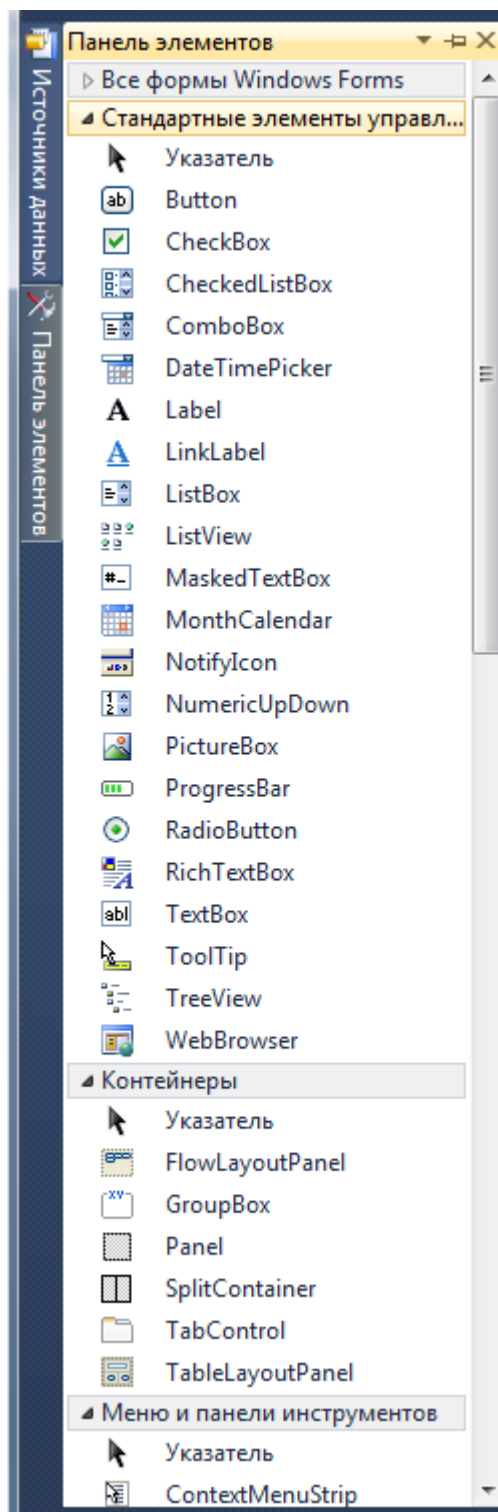


Рис. 3. 1. Панель элементов: расположение по умолчанию

Изначально, её положение весьма неудобно (она появляется и исчезает при получении фокуса мышки и её снятия, чем перекрывает поле формы), поэтому её можно закрепить нажав значок кнопки в шапке панели:

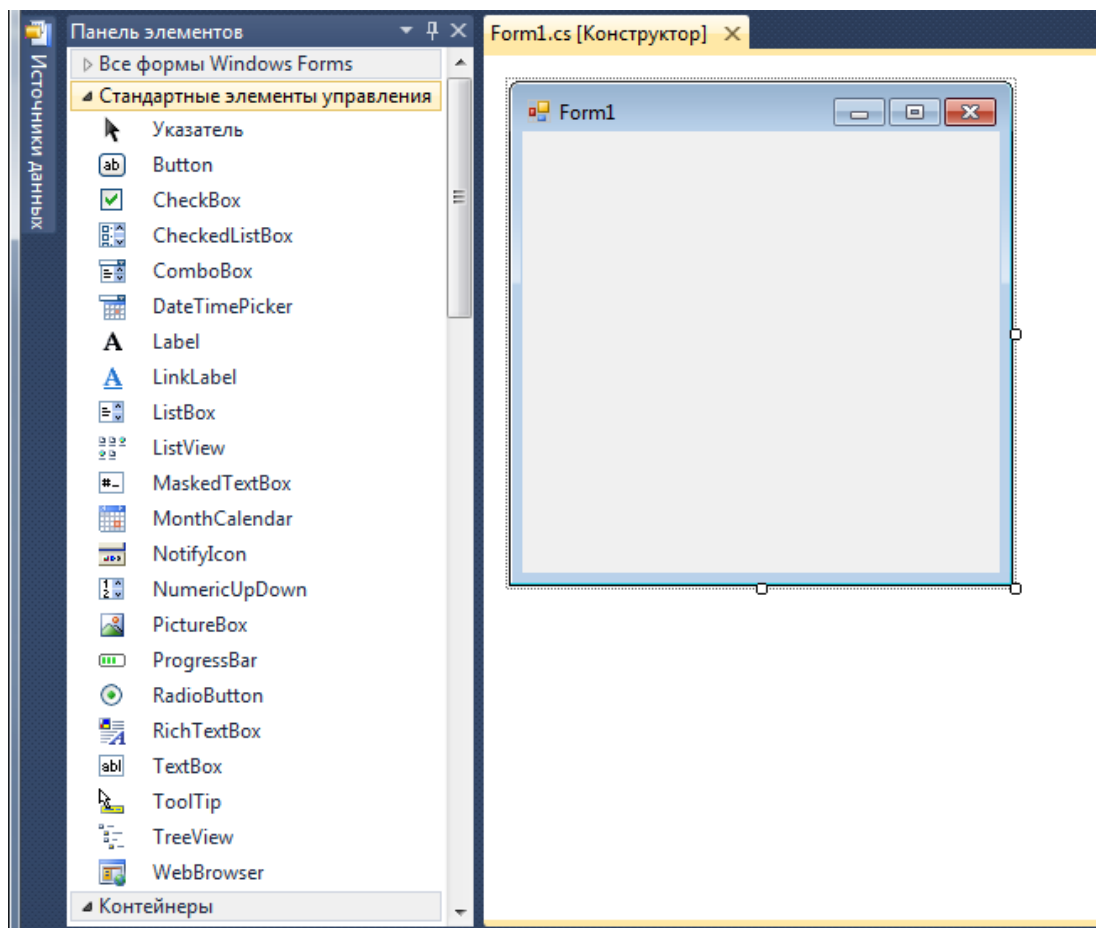


Рис. 3. 2. Закреплённая панель элементов

Добавление элементов — простой процесс. Например, нам нужно добавить **Текстовое поле редактирования**. Для этого выделяем на панели элементов элемент с названием **TextBox** и наводим мышку на любое место нашей формы. Указатель изменится на перекрестие с иконкой от *TextBox*. Далее нажимаем на точку «добавления» прямоугольника текстового поля редактирования и протягиваем мышку, тем самым расширяя добавленное поле.

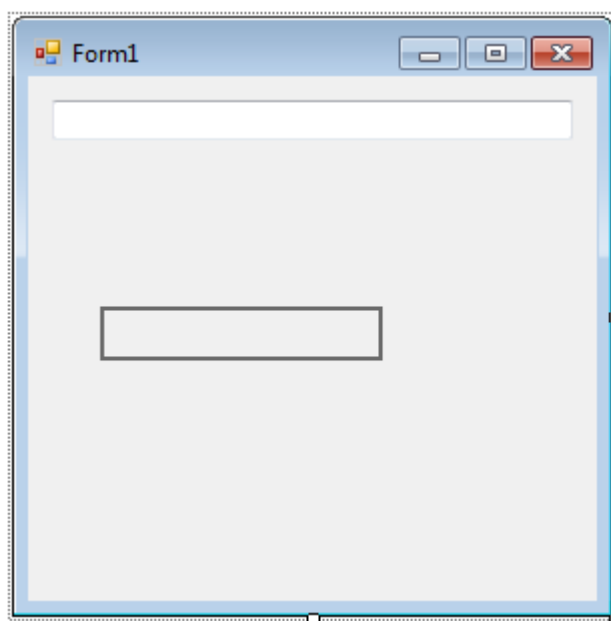


Рис. 3. 3. Добавленное текстовое поле редактирования (сверху) и процесс добавления (снизу)

Процесс расстановки элементов можно упростить. «На глаз» ставить элементы можно, а можно пользоваться различными инструментами позиционирования. Например можно потянуть за уголок установленного элемента и придвинуть его к рамке формы. Появятся направляющие линии. Также если нужно выровнять элемент относительно другого элемента, достаточно придвинуть элемент к образцовому до появления направляющих линий и дальше двигать относительно них.

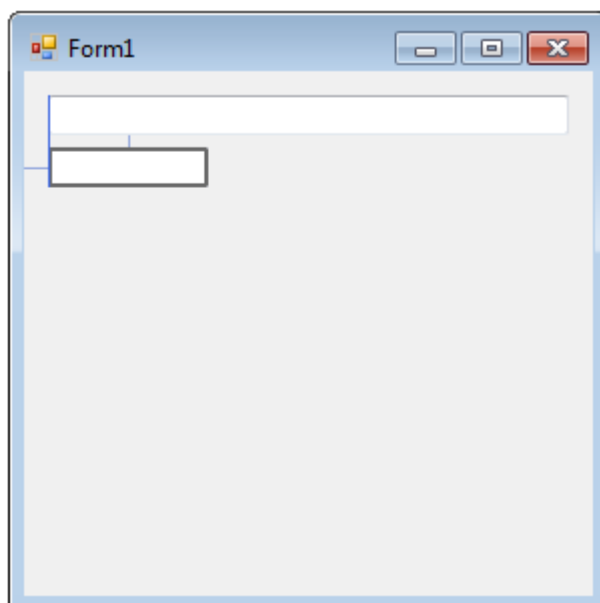


Рис. 3. 4. Направляющие линии при позиционировании элемента (синие)

Теперь немного о свойствах элементов. Естественно что **все** значимые свойства элементов настраиваемы. Свойство очень много, поэтому остановимся на тех, что будут необходимы в данной работе лабораторного практикума.

Свойства всех элементов отображаются на соответствующей панели **Свойства**. Для отображения свойства элемента достаточно выбрать необходимый установленный элемент в выпадающем списке:

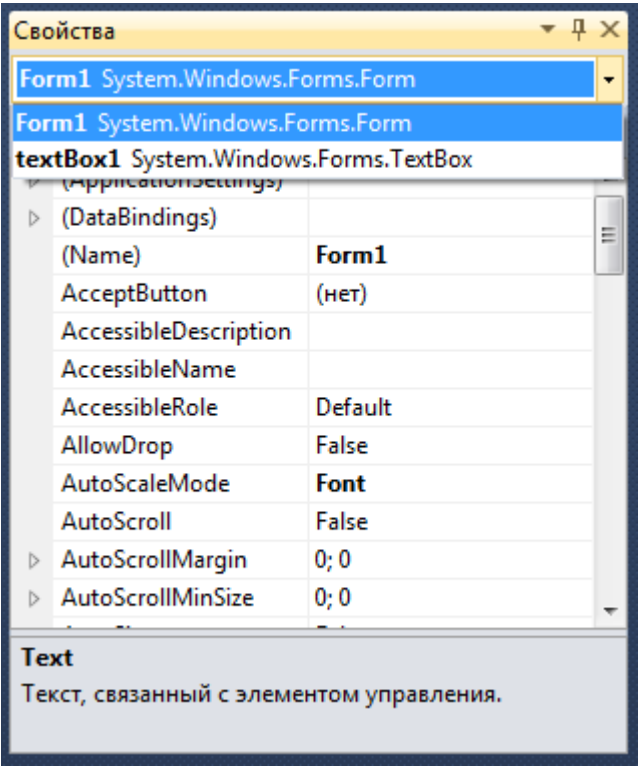


Рис. 3. 5. Выбор свойств для элементов: пока у нас всего два элемента — сама форма (с именем: **Form1**) и текстовое поле редактирования (с именем: **textBox1**)

Также можно выделить этот необходимый элемент на форме и нажатием правой кнопки мыши по нему выбрать **Свойства**.

И так, для начала изменим свойства самой формы. Для этого перейдём в свойства **Form1.cs**. Нам нужны следующие поля (информация о значении поля можно получить на панели свойств ниже на тёмно сером поле):

(Name)	изменим с Form1.cs на LWP02Main
^ Поменяем внутреннее имя формы.	
Text	изменим с Form1 на Простой калькулятор (C#)


^ Поменяем заголовок формы (то что отображается в шапке приложения слева).

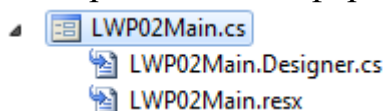
MaximizeBox	изменим с True на False
--------------------	---------------------------------------

^ Уберём кнопку **Развернуть**.

FormBorderStyle	изменим с Sizable на FixedDialog
------------------------	--

^ Сделаем окно «неизменяем» по размерам.

Для того, чтобы поменять имя файла нашей формы, необходимо выполнить следующее: выделить в обозревателе решений значок формы () и нажать правую кнопку мыши, затем выбрать **Переименовать**. Ввести необходимое новое имя **СОХРАНЯЯ** расширение *.cs. После смены имени, автоматически меняются имена проассоциированных непосредственно с формой файлов:



Теперь отредактируем свойства текстового поля редактирования:

TextAlign⁵:	изменим с Right на Center
-------------------------------	---

^ Изменим положение курсора и вывода текста в поле.

Если необходимо восстановить значение по умолчанию, нужно дважды нажать на пункт, который нужно вернуть к первоначальному значению (например двойным нажатием на *TextAlign* возвращаем значение *Right*). Если значений в поле больше двух (или значение не выбирается, а меняется вводом с клавиатуры), то первоначальным считается то, что не выделено **жирным** текстом. Также работает следующее (для вводимых значений): выбираем поле которое мы изменили, нажимаем на нём правую кнопку мыши, далее жмём на **Сброс**.

ReadOnly:	изменим с False на True
------------------	---------------------------------------

^ Делаем поле неизменяем «извне» программы. Значение из поля можно только копировать.

(Name):	изменим с textBox1 на ResultBox
----------------	---

Осталось только расставить оставшиеся элементы калькулятора. Ими станут кнопки (**Button**) с панели элементов и ещё элемент **NumericUpDown** (числовой «ползунок»).

NumericUpDown: элемент позволяет двигаться по числовому ряду (зависит от выбранного шага движения) при помощи нажатия стрелки вверх или вниз, либо ввода числа из этого диапазона. Определяющие свойства у элемента следующие:

Hexadecimal (True): определяет вывод числа в шестнадцатеричной форме (иначе в десятичной по умолчанию).

Maximum и **Minimum** задают диапазон числового ряда. **Increment** задаёт шаг по этому ряду.

ThousandSeparator (True): разделяет группы цифр (тысячи).

DecimalPlaces: число отображаемых знаков после запятой.

Value: текущее предустановленное число.

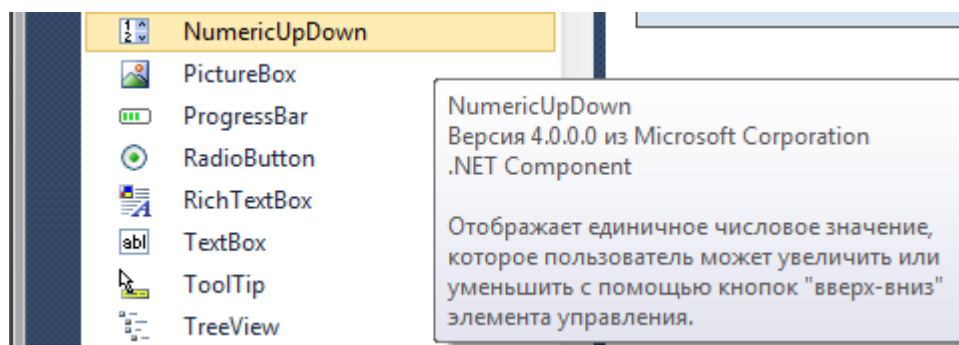


Рис. 3. 11. Панель элементов: элемент управления *NumericUpDown*

Расставляем четыре кнопки математических действий, 10 кнопок цифр от 0 до 9, одну кнопку «запятой», одну кнопку «очистить», одну кнопку «=» («вычисление»), одну кнопку «Округлить», и элемент *NumericUpDown* справа от «Округлить»:

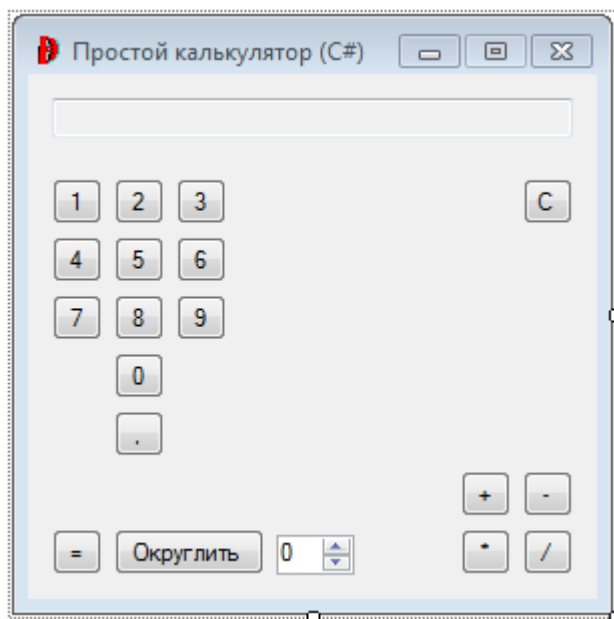


Рис. 3. 12. Готовый шаблон приложения *Windows Forms*

Теперь осталось добавить события по нажатию кнопок и весь необходимый код.

Кнопки цифр:

(Name):	B0...B9	Text:	0...9
----------------	---------	--------------	-------

Кнопка «запятая»:

(Name):	BD.	Text:	,
----------------	-----	--------------	---

Кнопка «очистить»:

(Name):	BC	Text:	C
----------------	----	--------------	---

Кнопка «=»:

(Name):	BResult	Text:	=
----------------	---------	--------------	---

Кнопки действий:

(Name):	BOperation1	Text:	+
(Name):	BOperation2	Text:	-
(Name):	BOperation3	Text:	*
(Name):	BOperation4	Text:	/

Кнопка «Округлить»:

(Name):	BSpecial	Text:	Округлить
----------------	----------	--------------	-----------

Элемент *NumericUpDown*:

(Name):	NumericSpecial	Maximum:	5	Minimum:	0	Increment:	1
---------	----------------	----------	---	----------	---	------------	---

Для добавления события нажатия для кнопки, необходимо дважды кликнуть на соответствующую кнопку, либо перейти в свойства кнопки и нажать на значок «молнии» (**События**):

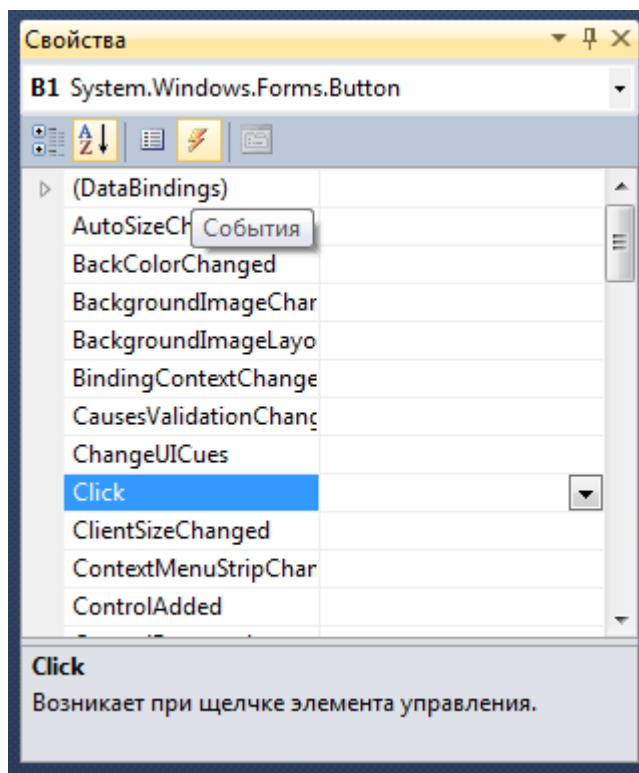


Рис. 3. 13. Переключение со страницы свойств кнопки на страницу событий

Нас интересует событие **Click**. Дважды нажимаем мышкой на слово *Click* (Рис. 3. 14):

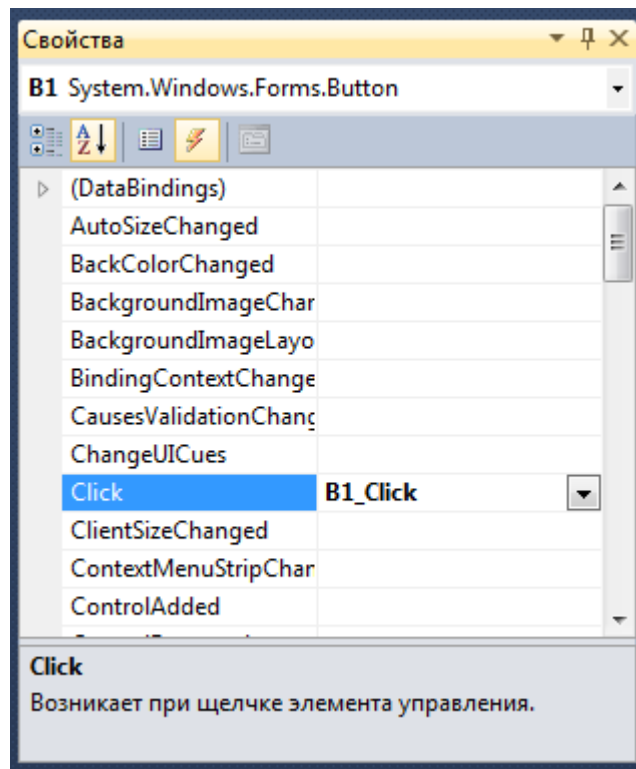


Рис. 3. 14. Событие *Click* для кнопки **B1**.

Код добавленного события такой (пока что он пуст):

```
private void B1_Click(object sender, EventArgs e)
{
}
}
```

Добавим сюда наш код для кнопки «1»:

```
private void B1_Click(object sender, EventArgs e)
{
    if (Clear == true)
    {
        ResultBox.Clear();
        Clear = false;
        Dot = false;
    }
}
```

```
    if (Operation1 == false && Operation2 == false && Operation3 == false
    && Operation4 == false)
    {
        ResultBox.AppendText("1");
        A = Convert.ToDouble(ResultBox.Text.Replace(".", ","));
    }
}
```

```

    }
    else
    {
        ResultBox.AppendText("1");
        B = Convert.ToDouble(ResultBox.Text.Replace(".", ","));
    }
}

```

Теперь необходимо объявить переменные, которые есть в коде выше. При добавлении этого кода видно, что среда уже заметила ошибки (а именно нет этих самых переменных).

Найдём строчку кода в этом же файле (**LWP02Main.cs**):

```

public partial class LWP02Main : Form
{

```

Добавим после (6 переменных типа **bool**, и три типа **double**):

```

Boolean Operation1 = false;
Boolean Operation2 = false;
Boolean Operation3 = false;
Boolean Operation4 = false;
Boolean Clear = false;
Double A;
Double B;
Double Result;
Boolean Dot = false;

```

Добавим по аналогии (с кнопкой «1» код для всех цифровых кнопок). Затем добавим код для кнопки «=»:

```

private void BResult_Click(object sender, EventArgs e)
{
    if (Operation1 == true)
        Result = A + B;
    if (Operation2 == true)
        Result = A - B;
    if (Operation3 == true)
        Result = A * B;
    if (Operation4 == true)
        Result = A / B;
}

```

```
ResultBox.Text = Result.ToString();
Operation1 = false;
Operation2 = false;
Operation3 = false;
Operation4 = false;
Clear = true;
}
```

Кнопка «ОЧИСТИТЬ»:

```
private void BC_Click(object sender, EventArgs e)
{
    ResultBox.Clear();
    Operation1 = false;
    Operation2 = false;
    Operation3 = false;
    Operation4 = false;
    Clear = false;
    A = 0;
    B = 0;
    Result = 0;
    Dot = false;
}
```

Кнопка операции «сложение»:

```
private void BOperation1_Click(object sender, EventArgs e)
{
    Operation1 = true;
    Operation2 = false;
    Operation3 = false;
    Operation4 = false;
    Dot = true;
    ResultBox.Clear();
}
```

Кнопка «ВЫЧИТАНИЕ»:

```
private void BOperation2_Click(object sender, EventArgs e)
{
    Operation1 = false;
    Operation2 = true;
    Operation3 = false;
```



```
Operation4 = false;
Dot = true;
ResultBox.Clear();
}
```

«Умножение» и «деление»:

```
private void BOperation3_Click(object sender, EventArgs e)
{
    Operation1 = false;
    Operation2 = false;
    Operation3 = true;
    Operation4 = false;
    Dot = true;
    ResultBox.Clear();
}
```

```
private void BOperation4_Click(object sender, EventArgs e)
{
    Operation1 = false;
    Operation2 = false;
    Operation3 = false;
    Operation4 = true;
    Dot = true;
    ResultBox.Clear();
}
```

Кнопка для ввода дробных чисел (с точкой). Код этой кнопки такой:

```
private void BD_Click(object sender, EventArgs e)
{
    if (Clear == true)
    {
        ResultBox.Clear();
        Clear = false;
        Dot = false;
    }

    if (Operation1 == false && Operation2 == false && Operation3 == false
    && Operation4 == false)
    {
        if (Dot == false)
```

```

        {
            ResultBox.AppendText(",");
            Dot = true;
        }
    }
else
{
    if (Dot == true)
    {
        ResultBox.AppendText(",");
        Dot = false;
    }
}
}

```

Кнопка *Округлить*:

```

private void BSpecial_Click(object sender, EventArgs e)
{
    SByte d = Convert.ToSByte(NumericSpecial.Value); // 8-битное целое
    число со знаком d = конвертируем число из NumericSpecial
    /* Выводим в главное текстовое поле округлённый результат
    * Округление выполняет метод Round() из класса Math, принимая
    округляемый Result и число, до которого выполняется округление
    (количество дробных разрядов) */
    ResultBox.Text = Convert.ToString(Math.Round(Result, d));
}

```

Компилируем приложение (**Release**) и запускаем. Результат работы показан ниже (Рис. 4. 1):

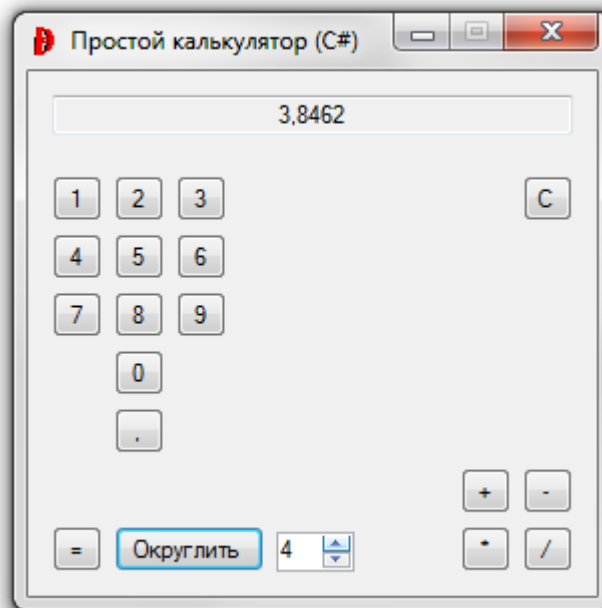


Рис. 4. 1. Модифицированное приложение Windows Forms

Основной рабочий файл, который был отредактирован нами это файл **LPW02Main.cs** (перейти к исходному коду можно нажав на него правой кнопкой мышки и выбрав в раскрывающемся меню пункт **Перейти к коду** либо нажав клавишу **F7** после выделения мышкой). Файл начинается стандартно, с подключения ссылок и задания пространства имён. Далее идёт объявление класса (**LWP02Main**) и типа формы (**Form**) (класс по ключевому слову **partial**):

```
namespace LWP02WindowsForms01
{
    public partial class LWP02Main : Form
    {
        Boolean Operation1 = false;
        Boolean Operation2 = false;
        Boolean Operation3 = false;
        Boolean Operation4 = false;
        Boolean Clear = false;
        Double A;
        Double B;
        Double Result;
        Boolean Dot = false;

        public LWP02Main()
        {
            InitializeComponent();
        }
    }
}
```

```
}
```

Основной метод нашей формы (**LWP02Main**) и вложенный в него метод **InitializeComponent()**. Это автоматически подставляемый обязательный метод для поддержки конструктора класса. Фактически конструктор всех компонентов формы. Можно посмотреть содержимое метода в другом файле (об этом ниже).

Далее идёт объявление переменных. Всего их 9, шесть из которых имеют логический тип («правда» или «ложь»), оставшиеся три — тип *double*⁷ (приблизительный диапазон чисел от $\pm 5,0 \times 10^{-324}$ до $\pm 1,7 \times 10^{308}$). Переменные *A* и *B* служат для хранения временного слагаемого. *Result* для хранения результата вычисления.

Все логические переменные служат для «переключателей» во время работы программы. И регулируют работы всех кнопок.

К примеру, при нажатии на кнопку операции «сложения» флаг *Operation1* переходит из *false* в *true*, и далее программа выполняет именно сложение (все остальные флаги остаются *false*).

```
private void B0_Click(object sender, EventArgs e)
{
    if (Clear == true)
    {
        ResultBox.Clear();
        Clear = false;
        Dot = false;
    }

    if (Operation1 == false && Operation2 == false && Operation3 == false
    && Operation4 == false)
    {
        ResultBox.AppendText("0");
        A = Convert.ToDouble(ResultBox.Text.Replace(".", ","));
    }
    else
    {
        ResultBox.AppendText("0");
        B = Convert.ToDouble(ResultBox.Text.Replace(".", ","));
    }
}
```

Это метод нажатия кнопки «0». В самом начале во время события нажатия, идёт проверка на переменную *Clear*. Если она *true*, происходит очистка текстового поля, после чего состояние «очистки» отключается вместе с состоянием нажатия кнопки «точка».

Дальше идёт проверка на нажатие кнопок математических операций. Если кнопки не нажаты, то вводимое в поле число будет отправлено в переменную *A*. Точнее при нажатии кнопки «0» в этом случае к тексту, хранимому в данный момент в *ResultBox.Text* добавляется ещё один нуль. После чего идёт проверка на символы «.» (точка) и замена его символом «,» (запятая). В завершении вся строка конвертируется из *String* в *Double* и полученное значение отправляется в переменную *A*. Если же нажата кнопка какой-либо операции, все введённые цифры присваивается переменной *B*.

```
private void BOperation1_Click(object sender, EventArgs e)
{
    Operation1 = true;
    Operation2 = false;
    Operation3 = false;
    Operation4 = false;
    ResultBox.Clear();
}
```

Кнопка операции «сложения». Здесь реализована защита от ввода неверных данных, так как используется одно и тоже поле для ввода двух разных чисел. После нажатия кнопки операции «сложения», поле очищается, а флаги переходят в нужное состояние.

```
private void BD_Click(object sender, EventArgs e)
{
    if (Clear == true)
    {
        ResultBox.Clear();
        Clear = false;
        Dot = false;
    }

    if (Operation1 == false && Operation2 == false && Operation3 == false
    && Operation4 == false)
    {
        if (Dot == false)
        {
            ResultBox.AppendText(",");
            Dot = true;
        }
    }
    else
    {
```

```

        if (Dot == true)
        {
            ResultBox.AppendText(",");
            Dot = false;
        }
    }
}

```

Кнопка «точка». Идентична кнопкам цифр, за исключением поведения при нажатии. Изначально переменная *Dot* в состоянии *false*. Если мы вводим в переменную *A*, нажатие кнопки «точка» меняет *Dot* на *true*. После этого в текстовое поле добавляется собственно знак «точки». И здесь срабатывает механизм замены. После этого дальнейшие нажатия не дадут эффекта и лишняя точка не будет введена (и не будет выдана ошибка).

Если же математическая операция уже выбрана, то *Dot* переведена в состояние *true* и сработает последняя часть метода. Опять же нажать «точку» можно лишь один раз.

```

private void BResult_Click(object sender, EventArgs e)
{
    if (Operation1 == true)
        Result = A + B;
    if (Operation2 == true)
        Result = A - B;
    if (Operation3 == true)
        Result = A * B;
    if (Operation4 == true)
        Result = A / B;
    ResultBox.Text = Result.ToString();
    Operation1 = false;
    Operation2 = false;
    Operation3 = false;
    Operation4 = false;
    Clear = true;
}

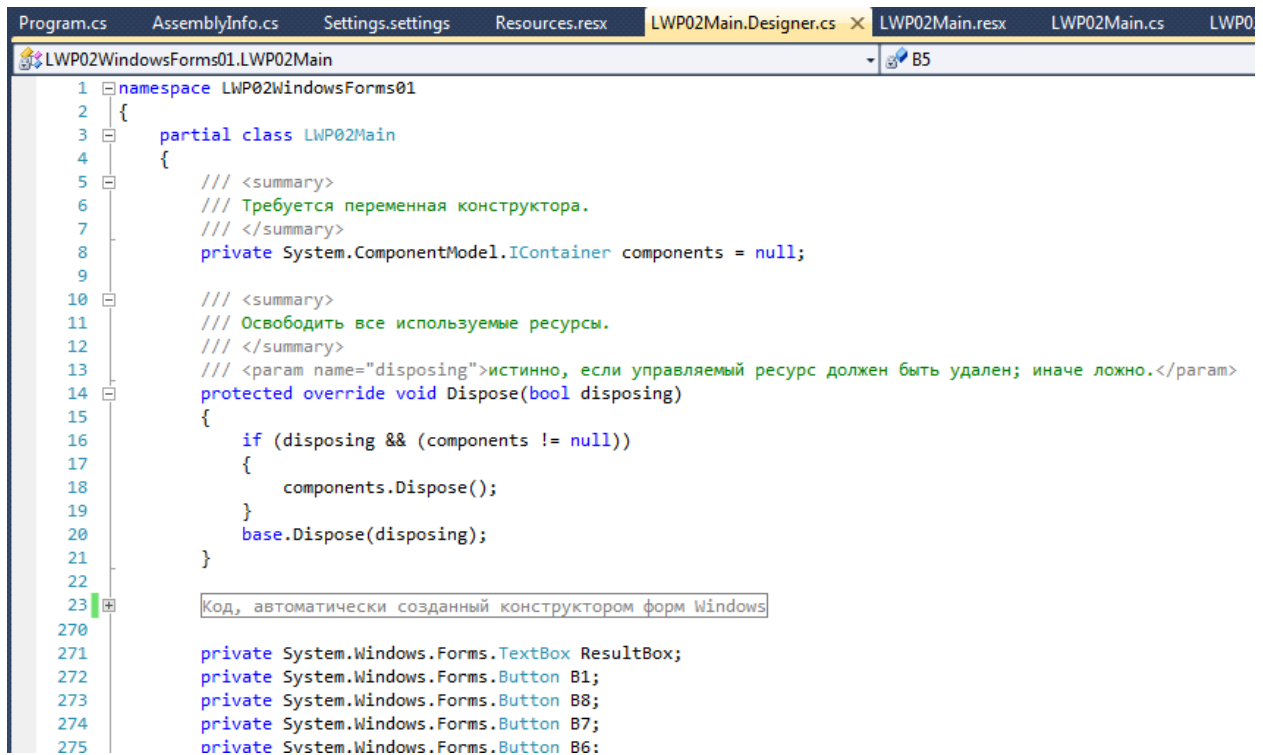
```

Кнопка «вычисление» производит основное действия по вычислению в зависимости от выбранной операции, а также от содержания переменных *A* и *B*. Если просто нажать кнопку математической операции, переменная *A* будет содержать ноль. Но на ноль поделить нельзя. Результат будет забавным (ошибка выдана не будет, что примечательно). Основная операция по выводу результата это конвертирование переменной *Result* обратно в *String*.

Кнопка «Округлить» пояснена в комментариях к кода события *Click* для кнопки.

Теперь немного остановимся на файле (**LWP02Main.Designer.cs**). Это и есть пресловутый конструктор формы. Точнее файл с настройками формы.

Большая часть кода генерируется средой разработки автоматически:



```
1 namespace LWP02WindowsForms01
2 {
3     partial class LWP02Main
4     {
5         /// <summary>
6         /// Требуется переменная конструктора.
7         /// </summary>
8         private System.ComponentModel.IContainer components = null;
9
10        /// <summary>
11        /// Освободить все используемые ресурсы.
12        /// </summary>
13        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>
14        protected override void Dispose(bool disposing)
15        {
16            if (disposing && (components != null))
17            {
18                components.Dispose();
19            }
20            base.Dispose(disposing);
21        }
22
23        Код, автоматически созданный конструктором форм Windows
24
25        private System.Windows.Forms.TextBox ResultBox;
26        private System.Windows.Forms.Button B1;
27        private System.Windows.Forms.Button B8;
28        private System.Windows.Forms.Button B7;
29        private System.Windows.Forms.Button B6;
```

Рис. 4. 2. Содержимое файла **LWP02Main.Designer.cs**

```
/// <summary>
/// Требуется переменная конструктора.
/// </summary>
private System.ComponentModel.IContainer components = null;
```

Здесь (строчка выше) держатся все компоненты формы. Генерируется автоматически. Если компонент не используется, и никто на него не ссылается, включается в работу метод **Dispose()** и удаляет, сбрасывает или высвобождается неуправляемый ресурс.

```
/// <summary>
/// Освободить все используемые ресурсы.
/// </summary>
/// <param name="disposing">истинно, если управляемый ресурс должен
быть удален; иначе ложно.</param>
protected override void Dispose(bool disposing)
{
```

```

    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

```

Собственным метод *Dispose()*. Вызывается в случае необходимости уничтожения управляемого ресурса.

#region Код, автоматически созданный конструктором форм Windows

```

/// <summary>
/// Обязательный метод для поддержки конструктора - не изменяйте
/// содержимое данного метода при помощи редактора кода.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(LWP02Main));
    this.ResultBox = new System.Windows.Forms.TextBox();
    this.B1 = new System.Windows.Forms.Button();
    ...
#endregion

```

Как уже было сказано, этот код генерируется автоматически. По большей части это создание экземпляров определённых ресурсов и изменение их свойств, такие как позиция, имена, размеры и прочее. Это «конструктор» всех компонентов формы, которые пользователь установил при помощи визуального редактора. Разумеется, добавлять новые компоненты (кнопки, текстовые поля, **DataGridView**’ы и другое) можно добавлять «руками», прописывая всё то что делает конструктор автоматически. Зачастую, ручное добавление элементов управления и настройку их можно увидеть в «одно файловых» проектах (где код помещён в один файл *.cs). Единственное что можно сказать о преимуществе визуального конструктора формы — это его наглядность и несомненное удобство.

И наконец, обратимся к файлу **Program.cs**:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

```



```

namespace LWP02WindowsForms01
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new LWP02Main());
        }
    }
}

```

Можно сказать, здесь использован стандартный шаблон. В этом файле содержится точка входа в программу (`static void Main()`). Две строчки ниже относятся к инициализации визуального стиля для приложения и установки значений по умолчанию для определенных элементов управления программы. Сам запуск и инициализация формы выполняется здесь:

```
Application.Run(new LWP02Main());
```

из кусков кода приведённых в данной лабораторной работе, можно загрузить по ссылке в конце этого материала.

Задание на лабораторную работу

1. Разработать приложение, описанное в методических указаниях.
2. Добавить функцию в калькулятор, согласно варианту:

№ вар	Функция	Описание
1	Abs(x)	Вычисляет модуль (абсолютное значение) числа <i>x</i> . Перегружен для всех числовых типов (int , double и т.д.)
2	Acos(x)	Функция арккосинуса. Значение аргумента должно находиться в диапазоне от -1 до +1
3	Asin(x)	Функция арксинуса. Значение аргумента должно находиться в диапазоне от -1 до +1
4	Atan(x)	Функция арктангенса

5	Cos(x)	Функция косинуса. Аргумент задается в радианах
6	Exp(x)	Вычисляет значение e^x (экспоненциальная функция)
7	Log(x)	Возвращает значение натурального логарифма ($\ln x$)
8	Log10(x)	Возвращает значение десятичного логарифма ($\log_{10} x$)
9	Max(a, b)	Возвращает максимум из двух чисел a и b
10	Min(a, b)	Возвращает минимум из двух чисел a и b
11	Pow(x, a)	Возвращает значение x^a , то есть возводит число x в степень a
12	Sin(x)	Функция синуса. Угол задается в радианах
13	Sqrt(x)	Возвращает положительное значение квадратного корня \sqrt{x}
14	Tan(x)	Функция тангенса. Угол задается в радианах