Практическое занятие №2 **ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ**

Цель практической работы: научиться составлять каркас простейшей программы в среде Visual Studio. Написать и отладить программу линейного алгоритма.

Методические указания

2.1. Структура приложения

Перед началом программирования необходимо создать проект. *Проект* содержит все исходные материалы для приложения, такие как файлы исходного кода, ресурсов, значки, ссылки на внешние файлы, на которые опирается программа, и данные конфигурации, такие как параметры компилятора.

Кроме понятия проект часто используется более глобальное понятие – *решение (solution)*. Решение содержит один или несколько проектов, один из которых может быть указан как стартовый проект. Выполнение решения начинается со стартового проекта.

Таким образом, при создании простейшей C# программы в Visual Studio. Создается папка решения, в которой для каждого проекта создается подпапка проекта, в которой будут создаваться другие подпапки с результатами компиляции приложения.

Проект это основная единица, с которой работает программист. При создании проекта можно выбрать его тип, а Visual Studio создаст каркас проекта в соответствии с выбранным типом.

Проект в Visual Studio состоит из файла проекта (файл с расширением *.csproj*), одного или нескольких файлов исходного текста (с расширением *.cs*), файлов с описанием окон формы (с расширением *.designer.cs*), файлов ресурсов (с расширением *.resx*), а также ряда служебных файлах.

В ***файле проекта*** находится информация о модулях, составляющих данный проект, входящих в него ресурсах, а также параметров построения программы. Файл проекта автоматически создается и изменяется средой Visual Studio и не предназначен для ручного редактирования.

Файл исходного текста – программный модуль предназначен для размещения текстов программ. В этом файле программист размещает текст программы, написанный на языке C#. Модуль имеет следующую структуру:

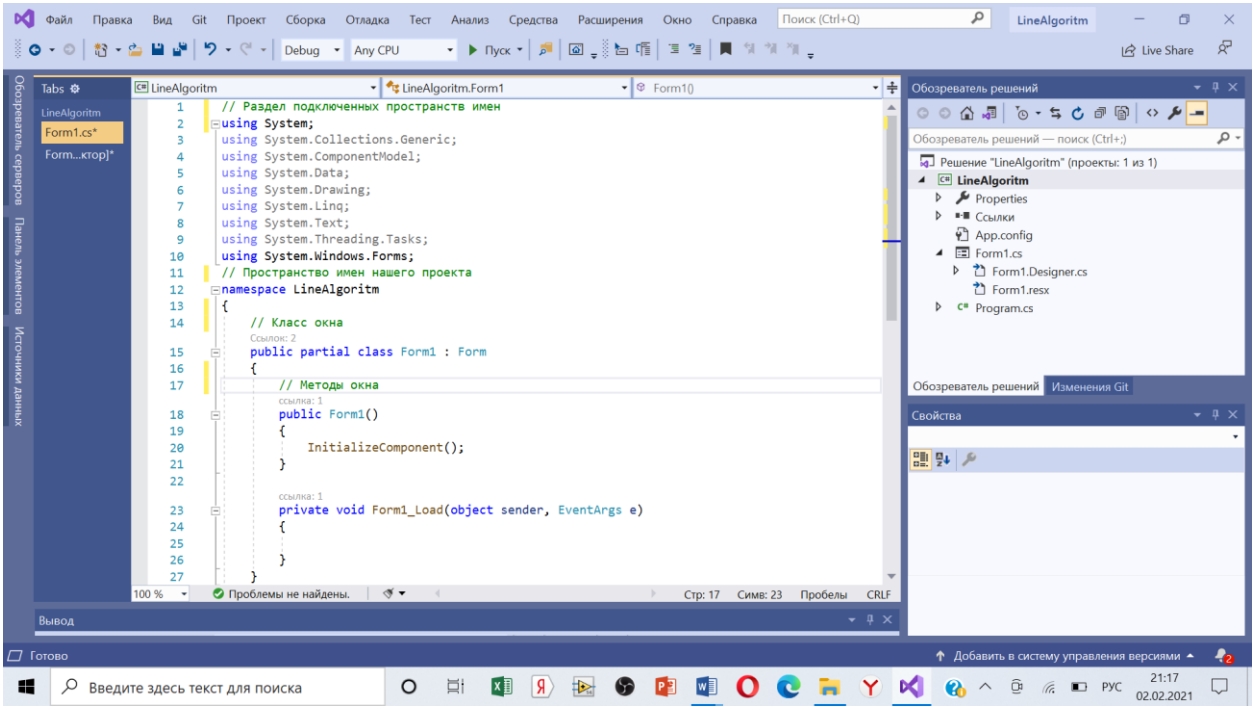

Рис 1 Вид пустого проекта

В разделе подключения пространств имен (каждая строка которого располагается в начале файла и начинается ключевым словом **`using`**) описываются используемые пространства имён. Каждое пространство имён включает в себя классы, выполняющие определённую работу, например, классы для работы с сетью располагаются в пространстве `System.Net`, а для работы с файлами – в `System.IO`. Большая часть пространств, которые используются в обычных проектах, уже подключена при создании нового проекта, но при необходимости можно дописать дополнительные пространства имён.

Для того чтобы не происходило конфликтов имён классов и переменных, классы проекта также помещаются в отдельное пространство имен. Определяется оно ключевым словом `namespace`, после которого следует имя пространства (обычно оно совпадает с именем проекта).

Внутри пространства имен помещаются классы – в новом проекте это класс окна, который содержит все методы для управления поведением окна. Обратите внимание, что в определении класса присутствует ключевое слово partial, это говорит о том, что в исходном тексте представлена только часть класса, с которой мы работаем непосредственно, а служебные методы для обслуживания окна скрыты в другом модуле (при желании их тоже можно посмотреть, но редактировать вручную не рекомендуется).

Наконец, внутри класса располагаются переменные, методы и другие элементы программы. **Фактически, основная часть программы размещается внутри класса при создании обработчиков событий.**

При компиляции программы Visual Studio создает исполняемые .exe-файлы в каталоге *bin*.

Работа с проектом

Как вы видите, проект в Visual Studio состоит из многих файлов, и создание сложной программы требует хранения каждого проекта в отдельной папке. При создании нового проекта Visual Studio по умолчанию сохраняет его в отдельной папке. Рекомендуется создать для себя свою папку со своей фамилией внутри папки своей группы, чтобы все проекты хранились в одном месте. После этого можно запускать Visual Studio и создавать новый проект (как это сделать показано в предыдущей лабораторной работе).

Сразу после создания проекта рекомендуется сохранить его в подготовленной папке: **Файл -> Сохранить всё.**

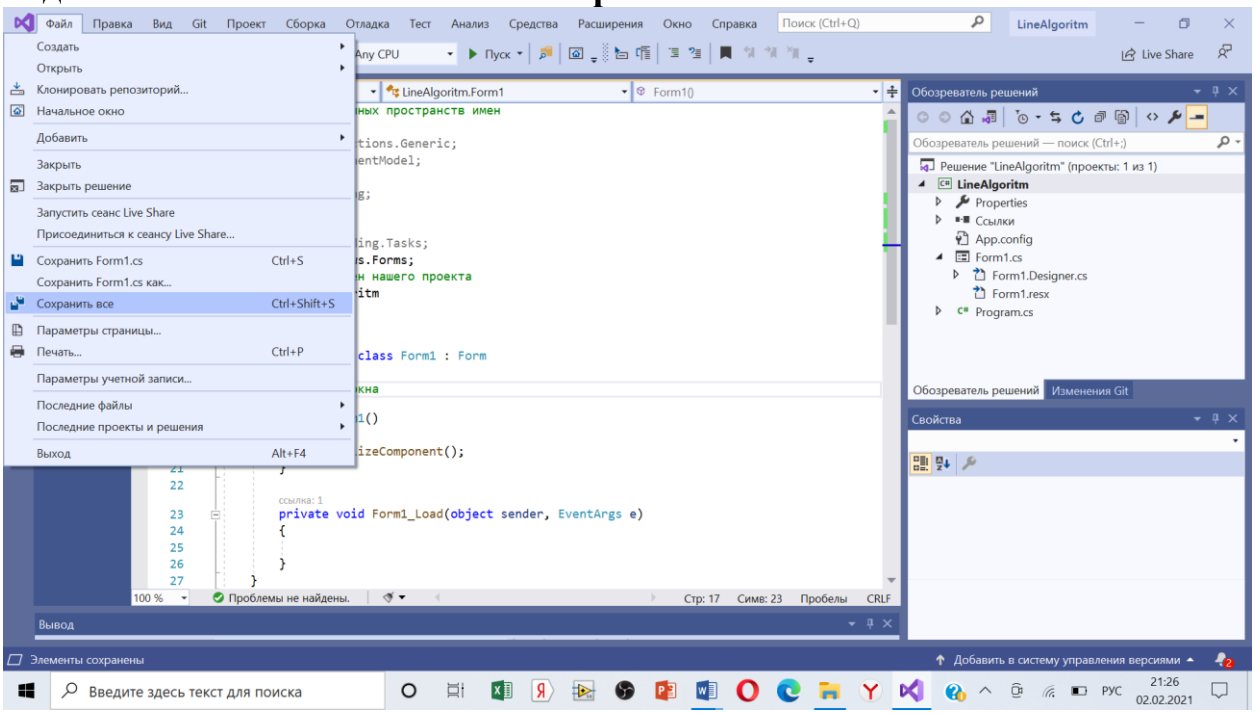

Рис. 1 Вид меню файл

При внесении значительных изменений в проект следует еще раз сохранить проект той же командой, а перед запуском программы на выполнение среда обычно сама сохраняет проект на случай какого-либо сбоя. Для открытия существующего проекта используется команда **Файл -> Открыть решение или проект**, либо можно найти в папке файл проекта с расширением *.csproj* и сделать на нём двойной щелчок.

Описание данных

Типы данных имеют особенное значение в C#, поскольку это строго типизированный язык. Это означает, что все операции подвергаются строгому контролю со стороны компилятора на соответствие типов, причем недопустимые операции не компилируются. Такая строгая проверка типов позволяет предотвратить ошибки и повысить надежность программ. Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как "бестиповая" переменная, в данном языке программирования вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять

над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

В C# имеются две общие категории встроенных типов данных: типы значений и ссылочные типы. Они отличаются по содержимому переменной. Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение.

Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как стек, а ссылочные типы — в области, называемой управляемой кучей.

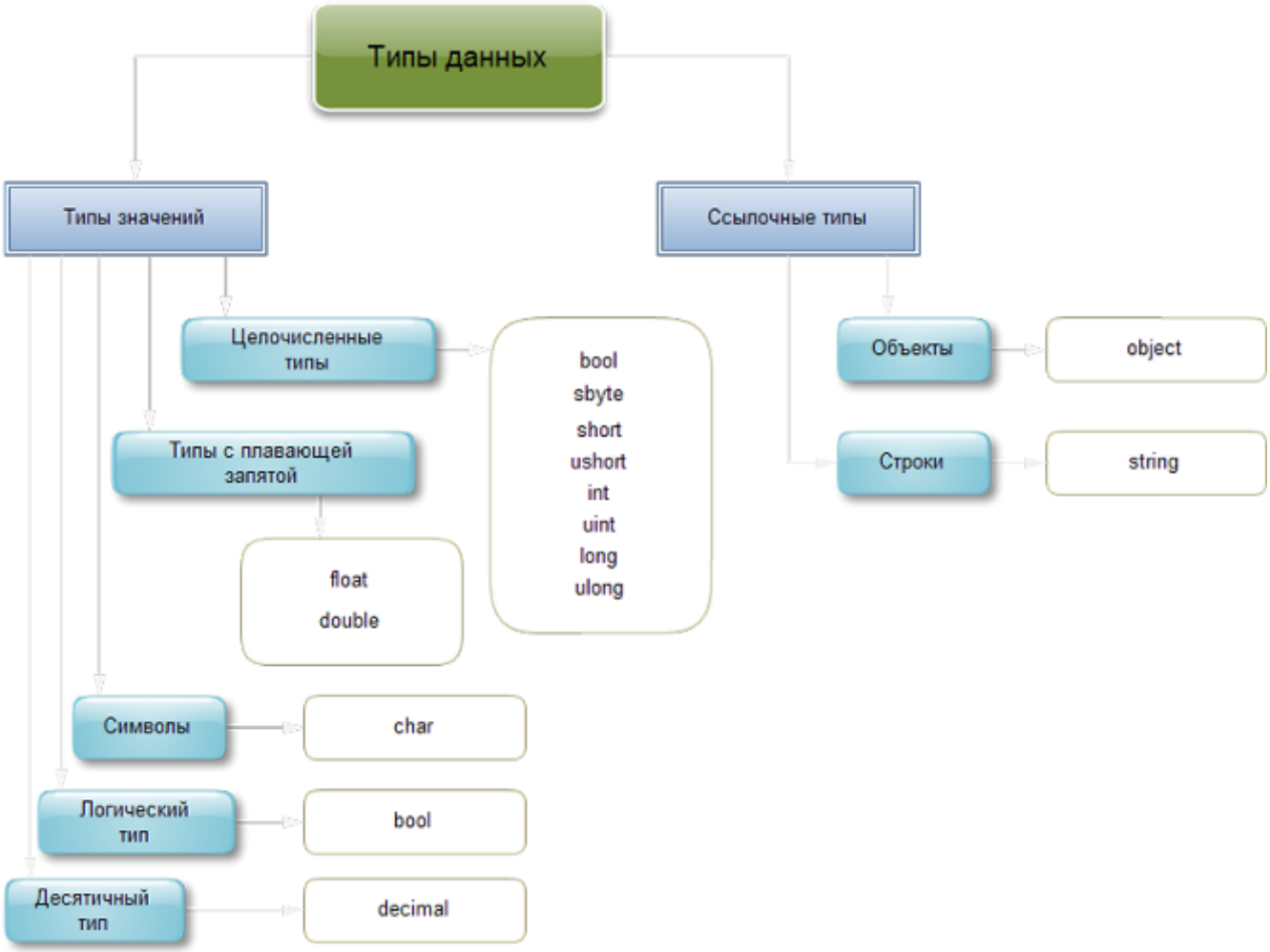

Рис.2. Типы данных

- Целочисленные типы

В C# определены девять целочисленных типов: char, byte, sbyte, short, ushort, int, uint, long и ulong. Но тип char в основном применяется для представления символов и поэтому рассматривается отдельно. Остальные восемь целочисленных типов предназначены для числовых расчетов.

- Типы с плавающей точкой

Типы с плавающей точкой позволяют представлять числа с дробной частью. В C# имеются две разновидности типов данных с плавающей точкой: float и double. Они представляют числовые значения с одинарной и двойной точностью соответственно.

- Десятичный тип данных

Для представления чисел с плавающей точкой высокой точности предусмотрен также десятичный тип decimal, который предназначен для применения в финансовых вычислениях.

- Символы

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, например C++, а 16-разрядным кодом, который называется юникодом (Unicode). В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете.

- Логический тип данных

Тип bool представляет два логических значения: "истина" и "ложь". Эти логические значения обозначаются в C# зарезервированными словами true и false соответственно. Следовательно, переменная или выражение типа bool будет принимать одно из этих логических значений.

- Строки

Основным типом при работе со строками является тип string, задающий строки переменной длины. Тип string представляет последовательность из нуля или более символов в кодировке Юникод. Класс String в языке C# относится к ссылочным типам. Над строками - объектами этого класса - определен широкий набор операций, соответствующий современному представлению о том, как должен быть устроен строковый тип. По сути, текст хранится в виде последовательной доступной только для чтения коллекции объектов Char.

Рассмотрим самые популярные данные – переменные и константы. Переменная - это ячейка памяти, которой присвоено некоторое имя и это имя используется для доступа к данным, расположенным в данной ячейке. Для каждой переменной задаётся тип данных – диапазон всех возможных значений для данной переменной.. Объявляются переменные непосредственно в тексте программы. Лучше всего сразу присвоить им начальное значение с помощью знака присвоения "=" (*переменная = значение*):

```
int x;      // Только объявление
int y = 9; // Объявление и инициализация значением
```

Для того чтобы присвоить значение символьной переменной, достаточно заключить это значение (т.е. символ) в одинарные кавычки:

```
char ch;      // Только объявление
char symbol = 'X'; // Объявление и инициализация значением
```

Несмотря на то что тип char определен в C# как целочисленный, его не следует путать со всеми остальными целочисленными типами.

Частным случаем переменных являются константы. Константы - это переменные, значения которых не меняются в процессе выполнения программы. Константы описываются как обычная переменная, только с ключевым словом `const` впереди:

```
const int x = 2;
```

2.4. Ввод/вывод данных в программу

Рассмотрим один из способов ввода данных через элементы, размещенные на форме. Для ввода данных чаще всего используют элемент управления TextBox, через обращение к его свойству **Text**. Свойство Text хранит в себе строку введенных символов. Поэтому данные можно считать таким образом:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
}
```

Однако со строкой символов трудно производить арифметические операции, поэтому лучше всего при вводе числовых данных перевести строку в целое или вещественное число. Для этого у типов, или `int` и `double` существуют методы `Parse` для преобразования строк в числа. С этими числами можно производить различные арифметические действия. Таким образом, предыдущий пример можно переделать следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
    int x = int.Parse(s);
    int y = x * x;
}
```

Перед выводом числовые данные следует преобразовать назад в строку. Для этого у каждой переменной существует метод `ToString`, который возвращает в результате строку с символьным представлением

значения. Вывод данных можно осуществлять в элементы **TextBox** или **Label**, использую свойство **Text**. Например:

```
    private void button1_Click(object sender, EventArgs
e)
    {
        string s = textBox1.Text;
        int x = int.Parse(s);
        int y = x * x;
        label1.Text = y.ToString();
    }
```

2.5. Арифметические действия и стандартные функции

При вычислении выражения стоящего в правой части оператора присвоения могут использоваться арифметические операции: ***** умножение, **+** сложение, **-** вычитание, **/** деление, **%** взятие остатка при делении. Для задания приоритетов операций могут использоваться круглые скобки **( )**. Также могут использоваться стандартные математические функции, представленные методами класса Math:

- Math.Sin(x) – синус (аргумент задается в радианах);
- Math.Cos(x) – косинус (аргумент задается в радианах);
- Math.Atan(x) – арктангенс (аргумент задается в радианах);
- Math.Log(x) – натуральный логарифм;
- Math.Exp(x) – экспонента;
- Math.Pow(x,y) – возводит переменную x в степень y;
- Math.Sqrt(x) – квадратный корень;
- Math.Abs(x) – модуль числа;
- Math.Truncate(x) – целая часть числа;
- Math.Round(x) – округление числа;

Для вывода результатов работы программы в программе используется текстовое окно, которое представлено обычным элементом управления. После установки свойства **Multiline** в **True** появляется возможность растягивать элемент управления не только по горизонтали, но и по вертикали. А после установки свойства **ScrollBars** в значение **Both** в окне появится вертикальная, а при необходимости и горизонтальная полосы прокрутки.

Информация, которая отображается построчно в окне, находится в массиве строк **Lines**, каждая строка которого имеет тип **string**. Однако нельзя напрямую обратиться к этому свойству для добавления новых строк, поскольку размер массивов в C# определяется в момент их инициализации. Для добавления нового элемента используется свойство Text, к текущему содержимому которого можно добавить новую строку:

```
textBox2.Text += Environment.NewLine + " Рассчитать значение выражения y=x^2";
```

В этом примере к текущему содержимому окна добавляется символ перевода курсора на новую строку (который может отличаться в разных операционных системах и потому представлен свойством класса `Environment`) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно привести в символьный вид методом `ToString()`.

Работа с программой рис.1 происходит следующим образом. Нажмите (щелкните мышью) кнопку “Выполнить”. В окне textBox2 появляется результат. Измените исходное значение x в окнах textBox1 и снова нажмите кнопку ”Выполнить” - появится новые результаты.

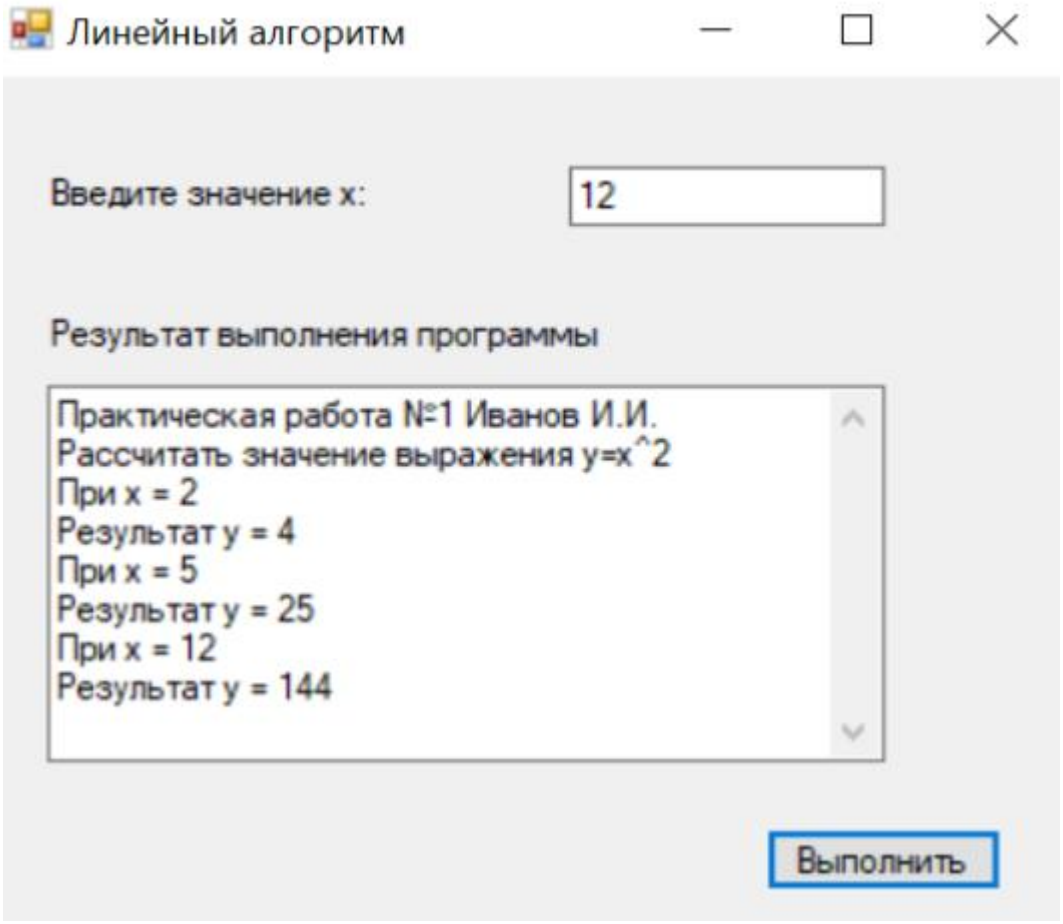

Рис. 3 Внешний вид программы

Полный текст программы имеет следующий вид:

```
// Раздел подключенных пространств имен
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
        using System.Threading.Tasks;
        using System.Windows.Forms;
        // Пространство имен проекта
        namespace LineAlgoritm
        {
            // Класс окна
            public partial class Form1 : Form
            {
                // Методы окна
                public Form1()
                {
                    InitializeComponent();
                }

                private void Form1_Load(object sender, EventArgs e)
                {
                    textBox1.Text = "2";
                    // Вывод строки в многострочный редактор
                    textBox2.Text = "Практическая работа №1 Иванов И.И.";
                    textBox2.Text += Environment.NewLine + "Рассчитать значение выражения
y=x^2";

                }

                private void label1_Click(object sender, EventArgs e)
                {

                }

                private void label2_Click(object sender, EventArgs e)
                {

                }

                private void button1_Click(object sender, EventArgs e)
                {
                    // Считывание значения X
                    double x = double.Parse(textBox1.Text);
                    // Вывод значения X в окно
                    textBox2.Text += Environment.NewLine +
                        "При x = " + x.ToString();
                    // Вычисляем арифметическое выражение
                    double y = Math.Pow(x,2);

                    // Выводим результат в окно
                    textBox2.Text += Environment.NewLine +
                        "Результат y = " + y.ToString();

                }
            }
        }
```

Индивидуальные задания

1. $y=\dfrac{\sqrt{1+e^{\sqrt{x}}+\cos x^2}}{\left|1-\sin^3 x\right|}+\ln|2x|;$

2. $y=\dfrac{0{,}5x^2-1}{\sqrt{x}}+\dfrac{\left|10-e^{0{,}5x^2+1}\right|}{\ln 2x-1};$

9. $y=\dfrac{\sin 3x+e^{1=x^2}}{2^{x+1}x+1-e^{5x^2-1}};$

10. $y=\dfrac{\left|1+\cos\sqrt{x}-\ln^2 2x\right|}{\sqrt{3x+4}-2^{0{,}5x-1}};$

3.$y=\dfrac{\sqrt{2+3\cos^2 2x+2}-1}{2^{1+\ln x}-\left|1-\sin\sqrt{2+x}\right|}$;

4.$y=\dfrac{\cos\sqrt{x+1}}{2\left|1-\sqrt{x}\right|+3\dfrac{1+e^x}{2^{x+1}-1}}$;

5.$y=\dfrac{x+\sin 0{,}5x+\sqrt{\ln x}}{2+3\cos x}+\dfrac{e^{2x-0,5}}{2+x}$;

6.$y=\dfrac{\sqrt{x^3}-\ln x}{\cos 2x^2+|x-3|}+\dfrac{3(x-2)^2}{\ln x+2}$;

7.$y=\dfrac{2\ln x\cos 2x-3\dfrac{(x+1)^2}{x-1}}{2+\sqrt{x}}$;

8.$y=\dfrac{3(x^2-|3x|)+\left|1-\cos x^2\right|}{2e^{3x-2}-3\dfrac{2+x}{x}}$;

11.$y=\dfrac{\sqrt{3+\ln x+15-x}}{1+\sin\dfrac{2+x^2}{1+x}}$;

12.$y=\dfrac{e^{x-\sqrt{x}}}{\ln\dfrac{x+5}{x+1}-2\dfrac{x^3+x}{1+|\sin x|}}$;

13.$y=\dfrac{2\ln\sqrt{x}-3\dfrac{x+1}{2x+3}}{\sin^2 2x+\left|e^{2x}-3x^2\right|}$;

14.$y=\dfrac{\sqrt{2+0{,}5x^2+e^{x+2}}}{x^3-3(2x-1)^2}+\sqrt{x}$;

15.$y=\dfrac{5+3\ln\sqrt{1+2x}}{2e^{1+0,5x^2}-|1-2x|+3}$;

16.$y=\dfrac{x^2-3\sqrt{x+1}+\ln 2x^2}{3+\sin\dfrac{2x^2+1}{2+\sqrt{x}}}$.