Общие понятия и ознакомление со средой программирования C#

1.1 Базовые концепции

Языки программирования предназначены для создания программ, которые могут быть исполнены ЭВМ или другими автоматическими устройствами, например, станками с числовым программным управлением.

Исходя из этого, можно сказать, что эти языки формальны, то есть они используют специальную систему команд, имеют свой алфавит и свои правила написания (синтаксис).

Существует достаточно большое количество различных языков программирования. Все они созданы так, что их команды понимает то устройство (в данном случае — ЭВМ), на которое они рассчитаны.

В данном курсе мы изучим и приобретем практические навыки программирования на языке C#.

1.2 Что такое C#

C# это объектно-ориентированный язык программирования, который позволяет создавать разнообразные приложения, запускающиеся на .NET Framework.

На нем создаются Windows приложения, веб-сервисы, мобильные приложения, клиент-серверные приложения, приложения баз данных и многое другое.

1.3 Переменные

Программы используют данные для выполнения заданий. Создание переменной резервирует место в памяти для хранения значений. Она называется переменной, потому что информация, хранящаяся в этой локации может быть изменена.

Имя переменной, также известное, как идентификатор, может содержать буквы, цифры и символ нижнего подчеркивания и должно начинаться с буквы или нижнего подчеркивания.

Хоть именем переменной может быть любой набор букв и цифр, стоит упомянуть, что наилучшим идентификатором переменной является описание информации, которую он содержит. Это очень важно для создания ясного и понятного кода!

Все переменные в C# должны быть объявлены до их применения. Это нужно для того, чтобы уведомить компилятор о типе данных, хранящихся в переменной, прежде чем он попытается правильно скомпилировать любой оператор, в котором используется переменная. Это позволяет также осуществлять строгий контроль типов в C#.

Типы данных

Тип данных определяет информацию, которая может хранится в переменной, размер необходимой памяти и операции, которые могут выполняться с переменной.

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. Следовательно, строгий

контроль типов позволяет исключить 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.

Например, для хранения целочисленного значения используется тип данных **integer**, ключевым словом которого является **int.**

```
int x; //объявление новой переменной x
```

Рисунок 1.3.1 – Пример объявления переменной.

Существует множество встроенных типов данных на C#. Наиболее часто используемыми являются:

- **Int** –целое число
- **Float** – число с плавающей точкой (нецелое число)
- **Double** – число с плавающей точкой удвоенной точности
- **Char** – один символ
- **Bool** – Булевый тип, который может иметь только два значения: True(истина) или False(ложь).
- **String** – последовательность символов.

```
int x = 20;          // целое число
float e = 2.71F;     // число с плавающей точкой
double pi = 3.14;    // число с плавающей точкой с улучшенной точностью
char y = 'X';        // один символ
bool tr = true;      // булевая функция ложь\истина
```

Рисунок 1.3.2 – Примеры типов переменных.

Стоит заметить, что значения типа char присваивается с помощью одинарных кавычек, а значения типа string требуют использование двойных кавычек.

1.4 Операторы

Оператор – символ выполняющий математические или логические действия.

Арифметические операторы

Действие арифметических операторов не требует особых пояснений, за исключением следующих особых случаев. Прежде всего, не следует забывать, что когда оператор / применяется к целому числу, то любой остаток от деления отбрасывается; например, результат целочисленного деления 13/3 будет равен 4. Остаток от этого деления можно получить с помощью оператора деления по модулю (%), который иначе называется оператором вычисления остатка. Он дает остаток от

целочисленного деления. Например, 13 % 3 равно 1. В C# оператор % можно применять как к целочисленным типам данных, так и к типам с плавающей точкой. Поэтому 13.0 % 3.0 также равно 1.

Операция	Назначение	Пример
+	Сложение	X = x + y;
+	Вычитание	X = x – y;
*	Умножение	X = x * y;
/	Деление	X = x / y;
%	Получение остатка от целочисленного деления	X = x % y

Таблица 1.4.1 – Арифметические операторы.

Оператор % возвращает остаток от деления двух целочислительных чисел.

```
static void Main(string[] args)
{
    double x = 10;
    double z = x % 4; //результат равен 2
}
```

Рисунок 1.4.1 – Пример использования оператора %.

Стоит отметить, что порядок выполнения операторов соответствует классическим правилам математики (умножение приоритетно над сложением и т.д.).

Операторы присваивания

Операторы присваивания «=» присваивает значение справа от оператора к переменной, находящейся слева. Так же существуют составные операторы присваивания, которые выполняют операцию и присваивание в одном выражении.

```
int x = 40; // присваиваем переменной x значение 40
x += 2;     // эквивалентно выражению x = x+2
x -= 6;     // эквивалентно выражению x = x-6
```

Рисунок 1.4.2 – Примеры операторов присваивания.

Оператор Инкремента

Оператор инкремента используется для увеличения значения целого числа на единицу, является часто используемым оператором в C#. Инкрементация имеет вид двойного оператора сложения (++), для C# это будет командой увеличение переменной ровно на 1.

```
int x = 5;
x++; // после выполнения кода x будет равен 6
```

Рисунок 1.4.3 – Пример оператора инкрементации.

То есть обычная запись x = x + 1 эквивалентна x = x++.

Префиксная и постфиксная формы инкремента

Оператор инкремента имеет две формы префиксную и постфиксную.

- Префикс инкрементирует значение, а затем продолжает выполнение выражения.
- Постфикс вычисляет выражение, а затем выполняет инкрементирование.

Оператор декремента **(--)** работает похожим образом, как и оператор инкремента, но вместо увеличения значения, он уменьшает его на единицу. Так же абсолютно похожим образом работает постдекрементация и предекрементация.

Операторы инкремента зачастую используют в качестве счетчика в циклах, где каждая последующая итерация увеличивает счетчик или уменьшает его на 1.

Логические операторы

Как и другие операторы является неотъемлемой частью языка программирования. Логические операторы (таблица 1.4.2) предназначены для выполнения логических операций над логическими данными, объявленными в программе при помощи ключевого слова bool.

Логические переменные могут принимать одно из двух значений — true (истина) или false (ложь). Результатом выполнения логического оператора всегда является логическое значение true или false.

Операция	Назначение
&&	Логическое И
||	Логическое ИЛИ
!	Логическое НЕ

Таблица 1.4.2 – Логические операторы.

Если оба операнда логического оператора **И** равны true, то результатом выполнения этой операции будет true. В противном случае результат будет false.

Если один из операндов логического оператора **ИЛИ** равен true, то результатом выполнения этой операции будет true. Если же оба операнда равны false, то и результат будет тоже равен false.

Оператор **НЕ** работает только с одним операндом, реверсируя его логическое состояние. Таким образом, если условие истинно, то оператор НЕ делает его ложным, и наоборот.

Оператор if

Оператор if является условным оператором, выполняющим блок кода, если условие верно. Условием может быть любое выражение, возвращающее true или false.

```
int x = 5;
int y = 8;
if (y>x)
{
    Console.WriteLine("y больше x");
}
```

Рисунок 1.4.5 Оператор if.

Код на рисунке 1.7 вычислит условие y>x. Если оно верно, будет выполнен код внутри блока if. Границами блока if являются фигурные скобки.

Операторы отношения

Используйте операторы отношения для вычисления условий. В дополнении к операторам меньше и больше, доступны следующие операторы в таблице 1.4. В целом, объекты можно сравнивать на равенство или неравенство, используя операторы отношения == и !=. А операторы сравнения <, >, <= или >= могут применяться только к тем типам данных, которые поддерживают отношение порядка. Следовательно, операторы отношения можно применять ко всем числовым типам данных. Но значения типа bool могут сравниваться только на равенство или неравенство, поскольку истинные (true) и ложные (false) значения не упорядочиваются. Например, сравнение true > false в C# не имеет смысла.

Операция	Назначение
==	Равно
!=	Не равно
>	Больше, чем
<	Меньше, чем

>=	Больше или равно
<=	Меньше или равно

Таблица 1.4.3 – Операторы отношения.

```
if (x==y)
{
    Console.WriteLine("равны");
}
// Выведет в консоль "равны" если значения x и y совпадают
```

Рисунок 1.4.6 – Пример сравнения.

Условие else

Опциональное условие else может быть определено для выполнения блока кода, когда условие в выражении if оказывается ложным.

```
int оценка = 5;
if (оценка < 3)
{
    Console.WriteLine("Вы не сдали");
}
else
{
    Console.WriteLine("Вы сдали");
}
// Выведет "Вы сдали"
```

Рисунок 1.4.7 – Условие else, в методе if.

Выражение if-else-if

Выражение if-else-if может быть использовано для выбора из трех и более действий.

```
int x = 56;
if (x == 8)
{
    Console.WriteLine("Значение x = 8");
}
else if (x == 33)
{
    Console.WriteLine("Значение x = 33");
}
else if (x == 56)
{
    Console.WriteLine("Значение x = 56");
}
else
{
    Console.WriteLine("Не наидено значений x");
}
// В данном случае выведет "Значение x = 56"
```

Рисунок 1.4.8 – Конструкция if-else-if.

Помните, что if может иметь от нуля и более выражений else if, а они должны идти перед последним else, который является опциональным. Как только выполняется else if, оставшиеся выражения не будут проверяться.

1.5 Циклы

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз.

Итерация – один шаг циклического процесса (один успешный прогон цикла).

Оператор if

Оператор if является условным оператором, который выполняет код только если условие верно. Условием может быть любое выражение, возвращающее true или false.

```
class Program
{
    static void Main(string[] args)
    {
        int x = 8;
        int y = 3;

        if (x > y)
        {
            Console.WriteLine("x болье y");// условие выполнится только в случае, если x будет больше y
        }
    }
}
```

Рисунок 1.5.1 – Пример простейшего цикла.

Условный оператор цикла while

Цикл while исполняет блок кода, до тех пор, пока условия цикла истинно.

```
class Program
{
    static void Main(string[] args)
    {
        int num = 1;
        while (num > 6)     //цикл будет исполняться до тех пор, пока переменная num будет меньше 6
        {
            Console.WriteLine(num); //каждый раз когда будет выполняться код, будет выводиться переменная num
            num++;                  //а затем к переменная num будет увеличиваться на 1
        }
    }
}
```

Рисунок 1.5.2 – Цикл while.

В тот момент, когда переменная num будет равна 5, и начнется новый цикл, условие цикла будет ложным и код остановиться, выдав ответом ряд чисел от 1 до 5. В случае если условие цикла не сможет сделать его ложным, цикл будет продолжаться бесконечно.

Если в примере пре-инкрементировать переменную прямо в условии, цикл исполниться 4 раз вместо 5. (см рисунок 1.5.3)

```
class Program
{
    static void Main(string[] args)
    {
        int num = 1;
        while (++num > 6)
        {
            Console.WriteLine(num);
        }
```

Рисунок 1.5.3 – Цикл while с пре-инкрементацией.

Цикл выдаст по окончанию вычислений вернет значения в виде ряда чисел от 2 до 6.

Цикл for

Используя цикл for можно раз за разом использовать блок операторов, пока определенное условие не будет истинным.

- **Инициализатор** - это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла). Инициализация, как правило, представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом;

- **Условие** - это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать true, чтобы была выполнена следующая итерация);

- **Итератор (изменение счетчика)** - выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла). (рис 1.5.4)

```
for ([инициализация счетчика]; [условие]; [изменение счетчика])
{
// действия
}
```

Рисунок 1.5.4 – Конструкт цикла for.

Обратите внимание на то, что эти три основные части оператора цикла for должны быть разделены точкой с запятой. Выполнение цикла for будет продолжаться до тех пор, пока проверка условия дает истинный результат. Как только эта проверка даст ложный результат, цикл завершится, а выполнение программы будет продолжено с оператора, следующего после цикла for.

```
for (int x = 10; x < 15 ; x++)
{
/*Цикл исполнит данный код 5 раз, после чего x будет равен 15 и условие будет ложным*/
}
```

Рисунок 1.5.5 – Пример использования цикла for

Код на рисунке 1.5.5 пост-инкрементирует переменную, это означает что инкрементация будет производиться после проверки на условие. В случае если мы сделаем условие с пре-инкрементацией, то цикл выполниться лишь 4 раза.

Выражение инициализации счетчика и изменения счетчика можно указывать вне конструкции for.

```
int x = 10;      // инициализатор счетчика
for (; x > 0;)  // неполная конструкция обязана иметь точки запятые
{
    Console.WriteLine(x);
    x -= 3;      // изменения счетчика
}
```

Рисунок 1.5.6 – Цикл for с инициализатором счетчика внутри метода

Цикл на рисунке 1.5.6 будет отнимать от переменной x по 3, до тех пор, пока значение переменной x не будет меньше 0.

Цикл foreach

Оператор foreach служит для перебора элементов коллекции. К коллекциям относят массивы, списки List, пользовательские классы коллекций. В отличии от for не нужно задавать переменную-счетчик, чтобы получить доступ к элементам коллекции.

```
foreach ([тип данных][переменная] in[коллекция])
{
    //тело цикла
}
```

Рисунок 1.5.7 – Конструкт цикла foreach.

```
int[] numbers = { 10, 20, 30, 40 }; // объявление коллекции
int x = 0;
foreach (int y in numbers) /* переменная y будет циклично
                              принимать значениея из коллекции numbers*/
{
    x = x+y; // складываем каждый элемент коллекции с переменной x
}

Console.WriteLine(x);
Console.ReadKey();
```

Рисунок 1.5.8 – Пример цикла foreach.

Так как на рисунке 1.5.8 переменная x равна нулю, наша программа циклично сложит все элементы из коллекции. Переменная y в каждой итерации будет последовательно принимать значения из коллекции nubers. В результате данной программы получится число 100.

Циклы do-while

Имеет ту же функционал что и while, но цикл do-while выполняется гарантированно один раз. Его объявление начинается с do и условие так же задается в методе while.

```
int a = 0;
do
{
    Console.WriteLine(a);
    a++;
} while (a < 5);
```

Рисунок 1.5.9 – Пример метода do-while.

Цикл сначала выполняет конструкцию do, а после выполнения сверяет ее со значением while.

Оператор break

Когда оператор break встречается внутри цикла, цикл немедленно прекращается, а выполнение переходит к следующему выражению после тела цикла.

```
int x = 0;
while (x < 0)
{
    if (x == 5)
        break;
    Console.WriteLine(x);
    x++;
}
```

Рисунок 1.5.10 – Оператор break

Условие цикла на рисунке 1.5.10, с учетом инкрементации x, делает его бесконечным. Но благодаря оператору break, цикл остановиться в тот момент, когда значение x будет равно 5.

Оператор continue

Оператор continue похож на оператор break, но вместо выхода из цикла, он пропускает одну итерацию цикла, продолжая со следующей итерации.

```
static void Main(string[] args)
{
    for (int x = 1; x < 5; x++) {
        if (x == 3)
            continue;
        Console.WriteLine(x);
    }
}
```

Рисунок 1.5.11 Оператор continue

Так как итерация цикла на рисунке 1.5.11 при x=3 пропускается, на выходе у нас будет ряд чисел от 1 до 5 с пропуском числа 3.

1.6 Методы

Метод это группа выражений, выполняющие специфическое задание. В C# помимо базовых методов, можно определять свои собственные.

Каждая программа в C# начинается с метода Main, которая в свою очередь относиться к классу Program. Она является точкой входа в приложение.

```
class Program
{
    static void Main(string[] args)
    {

    }
}
```

Рисунок 1.6.1 – Метод Main.

Ключевое слово static является модификатором. Далее идет тип возвращаемого значения. В данном случае ключевое слово void указывает на то, что метод ничего не возвращает.

Далее идет название метода - Main и в скобках параметры - string[] args, и в фигурные скобки заключено тело метода - все действия, которые он выполняет. В данном случае (рисунок 1.6.1) метод Main пуст, он не содержит никаких операторов и по сути ничего не выполняет.

Определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя метода, затем список аргументов в круглых скобках и далее - тело метода в фигурных скобках.

Каждый параметр состоит из имени типа параметра и имени, по которому к нему можно обратиться в теле метода. Вдобавок, если метод возвращает значение, то для указания точки выхода должен использоваться оператор возврата return вместе с возвращаемым значением.

Если метод не возвращает ничего, то в качестве типа возврата указывается void, поскольку вообще опустить тип возврата невозможно. Если же он не принимает аргументов, то все равно после имени метода должны присутствовать пустые круглые скобки. При этом включать в тело метода оператор возврата не обязательно — метод возвращает управление автоматически по достижении закрывающей фигурной скобки.

Использование параметров

При вызове метода ему можно передать одно или несколько значений. Значение, передаваемое методу, называется аргументом. А переменная, получающая аргумент, называется формальным параметром, или просто параметром. Параметры объявляются в скобках после имени метода. Синтаксис объявления параметров такой же, как и у переменных. А областью действия параметров

является тело метода. За исключением особых случаев передачи аргументов методу, параметры действуют так же, как и любые другие переменные.

В общем случае параметры могут передаваться методу либо по значению, либо по ссылке. Когда переменная передается по ссылке, вызываемый метод получает саму переменную, поэтому любые изменения, которым она подвергнется внутри метода, останутся в силе после его завершения. Но если переменная передается по значению, вызываемый метод получает копию этой переменной, а это значит, что все изменения в ней по завершении метода будут утеряны. Для сложных типов данных передача по ссылке более эффективна из-за большого объема данных, который приходится копировать при передаче по значению.

2. Ввод в консольное приложение

2.1 Создание консольного приложения

Для создания консольного приложения в Visual Studio:

1. Выберите **Файл** > **Создать** > **Проект** в меню (Или же используйте комбинацию клавиш Ctrl+Shift+N).
2. В диалоговом окне **Новый проект** выберите узел **Visual C#**, а затем выберите шаблон проекта **Консольное приложение** (.NET Core).
3. В текстовом поле **Имя** введите название своей программы.
4. Нажмите кнопку OK.

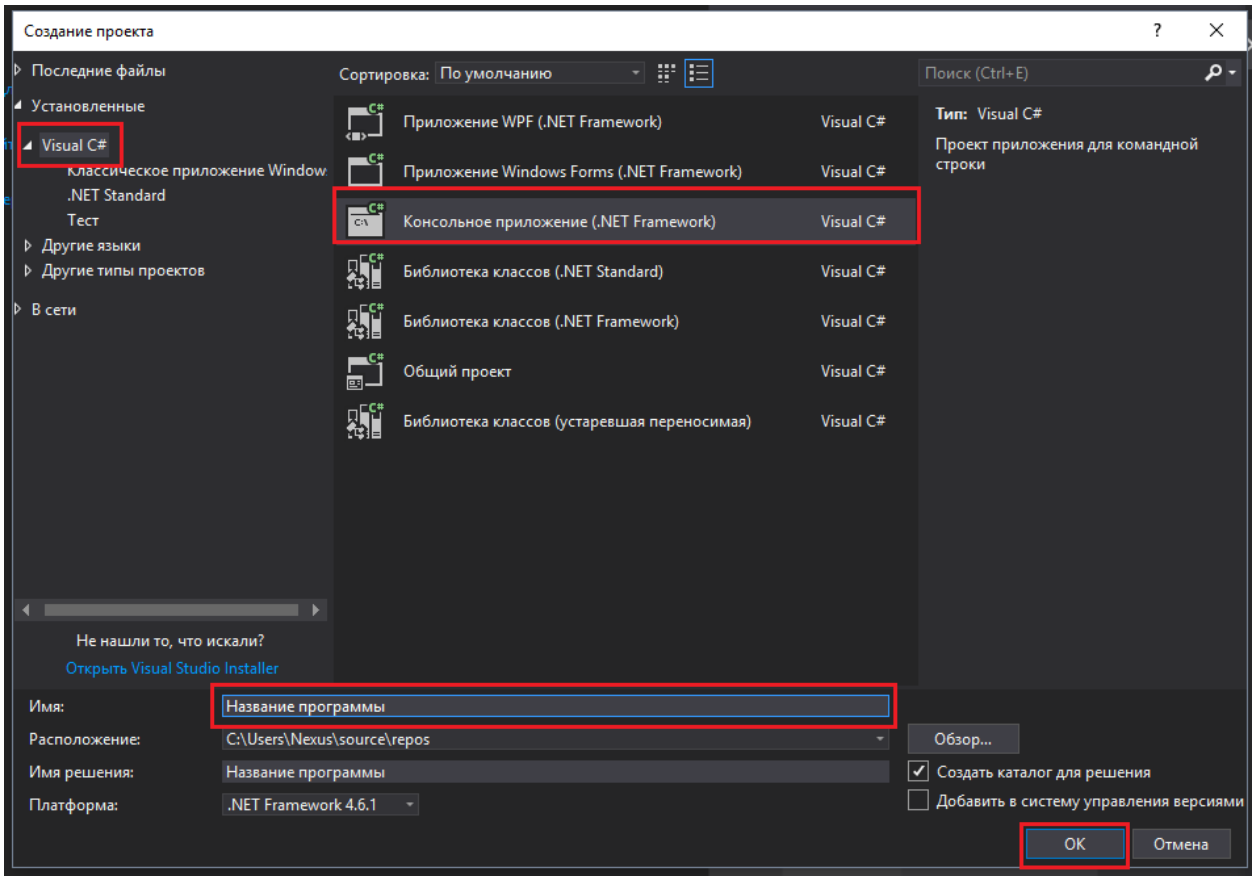

Рисунок 2.1.1 – Создание консольного приложения.

2.2 Основные команды консольного приложения

Основными консольными командами в C# являются **методы класса Console**.

Метод **WriteLine** выводит на консоль заданную строку. Похожий метод **Write** так же выводит заданное значение, но не посылает знак перехода на следующую строку.

Конструкт метода:

Вывод текста	`Console.WriteLine("любой текст");`	Текст всегда записывается в ковычках
Вывод переменной	`Console.WriteLine( x );`	Выводит значение переменной (если значение не выдаст ошибку)
Комбинированный способ	`Console.WriteLine("текст" + x)`	Выдаст текст и значение переменной

Таблица 2.2.1 – Команды для вывода текста и значений.

```
class Program
{
    static void Main(string[] args)
    {
        int ageV = 18;          // Задаем переменные, отвечающие за возраст
        int ageP = 15;
        Console.WriteLine("Возраст Вовы:" + ageV);  // Выводим текст с добавлением переменной
        Console.WriteLine("Возраст Пети:" + ageP);
        Console.ReadKey();      // Задерживаем результатат на экране
    }
}
```

Рисунок 2.2.1 – Программа Writline.

Также в значении метода можно производить простые математические действия

Так же стоит запомнить, что после вывода информации на экран программа прекращает свою работу и закрывается. Чтобы зафиксировать результат на экране можно воспользоваться методом **ReadKey**, так же относящийся к классу Console. Этот метод остановит программу и будет ожидать ввода любой клавиши, после чего программа завершит свою работу.

Метод **ReadLine** читает строку из потока ввода. Возвращает значение типа **string**.

```
 9  class Program
10  {
11      static void Main(string[] args)
12      {
13          string x;               // Объявление переменной
14          x = Console.ReadLine(); // Метод ReadLine дает возможность ввода значения переменной x
15          Console.WriteLine(x);   // Выводим результат для проверки
16          Console.ReadKey();      // Этот метод помогает "задержать" ответ на экране
17      }
18  }
19  }
20
```

Рисунок 2.2.2 – Программа ReadLine.

Программа вернет значение переменной x, сразу после ее ввода в консоль.

Методы класса Convert

Метод **Parse** используется для конвертации любого значения в значение определенного типа.

```
class Program
{
    static void Main(string[] args)
    {
        string s = "42";
        int n = int.Parse(s);
        Console.WriteLine(n);
    }
```

Рисунок 2.2.3 – Преобразование типа данных из string в int.

Является эквивалентом метода **Convert**, но с отличием где **Convert** используется для конвертации любого значения в значение определенного типа заданным способом.

Convert.ToBoolean(значение)	Конвертирует в bool
Convert.ToByte	Конвертирует в byte
Convert.ToChar	Конвертирует в char
Convert.ToInt16/32/64	Конвертирует в int

Таблица 2.2.2 – Синтаксис метода Convert.

Для примера рассмотрим ситуацию, где для ввода метода мы будем использовать метод **WriteLine**, вводимые значение которых по умолчанию принимают тип данных **string** и произведем конвертацию в тип данных **double** (смотри Рисунок 2.2.4).

```
static void Main(string[] args)
{
    string x = Console.ReadLine();
    Console.WriteLine(Convert.ToDouble(x));
    Console.ReadKey();
}
```

Рисунок 2.2.4 – Пример конвертации.

Методы класса Math

Класс **Math** содержит в себе стандартные математические функции, без которых сложно обойтись при построении математических выражений.

Abs	Возвращает абсолютное значение
Pow(double a, double b)	Возводит число **a** в степень **b**
Min(double a, double b)	Возвращает минимальное число из a и b
Max(double a, double b)	Возвращает максимальное число из a и b
Math.PI	Константа обозначающее число Пи
Truncate(число)	Возвращает целое число без дробного значения
Sqrt(double value)	возвращает квадратный корень числа value

Таблица 2.2.3 – Методы класса Math.

2.3 Лабораторные работы

Лабораторная работа № 1

Цель: Создайте программу, выводящую на экран "Hello World!", а также свою фамилию, имя, отчество и название группы.

```
class Lab1
{ // здесь должны находиться созданные пользователем классы class Program
    static void Main(string[] args){ // главная функция, здесь должны быть операторы
        Console.WriteLine("Hello World!"); /* использование встроенного класса Console, встроенного метода
                                              WriteLine для вывода на экран надпись "Hello World" */
        Console.ReadKey();                 // Задерживаем результат на экране
    }
}
```

Рисунок 2.3.1 – Первая программа на C#.

Лабораторная работа № 2

Цель работы: научиться использовать простейшие вычислительные действия, используя класс Math, определенном в пространстве имен System. Ознакомиться с операторами условия if…else в C#.

В языке C# для выбора одной из нескольких возможностей используются конструкцию с альтернативным выбором – if. Выражения if должны заключаться в круглые скобки и должны давать значения true или false. Ветвь else может отсутствовать. Условие может быть составлено из нескольких выражений, с использованием || (логическое сложение "или") и && (логическое умножение "и").

Пример: Решить квадратное уравнение

```
static void Main(string[] args)
{
    Console.Write("Введите  y=");    /*вывод    на  экран   надписи "Введите y"*/
        double y = double.Parse(Console.ReadLine()); /*присвоение переменной y вещественный тип данных double  */
    Console.Write("Введите  n=");    /*вывод    на  экран   надписи "Введите n"*/
        double n = double.Parse(Console.ReadLine()); /*присвоение переменной n вещественный тип данных double  */
    Console.Write("Введите m="); /*вывод на экран надписи "Введите n"*/
        double m = double.Parse(Console.ReadLine()); /*присвоение переменной m вещественный тип данных double  */
        double x = Math.Pow(y, 2) + 8 * n / m; /*использования класса Math и математической функции возведения в степень Pow */
    Console.WriteLine("x=" + x); /*вывод на экран результата решения"x="*/
    Console.ReadKey();
}
```

Рисунок 2.3.2 – Программа вычисляющая квадратное уравнение.

Задание: создать программу для решения квадратного уравнения a · x2 + b · x + c = 0 (a,b,c – вводятся с клавиатуры), с помощью оператора if рассмотреть все случаи дискриминанта (в случае D<0 вывести сообщение нет корней);

Лабораторная работа №3

Цель работы: изучение приемов использования операторов выбора switch. Ознакомиться с основными операторами циклов for и foreach в C#.

В языке C# для выбора одной из нескольких возможностей используются конструкцию с разбором случаев – switch. Оператор switch содержит проверяемое выражение. Внутри есть несколько case с вариантами кода. Выполняться будет только один из case, значение которого совпадет со

значением switch. Операторов case может быть множество, но они должны быть уникальными. В конце каждого блока case должен стоять оператор перехода break. Если не выполняется ни один из случаев (case), управление переходит к оператору default. Если оператор default не предусмотрен программистом, осуществляется выход из конструкции выбора и переход к последующему фрагменту программы

Пример: калькулятор на 4 действия (сложение, вычитание, умножение, деление)

```
static void Main(string[] args)
{
    double x, y, k; /*присвоение переменным x,y,k вещественный тип данных */
    char ed;
    bool es; //присвоение переменной es логический тип данных bool char ed; /*присвоение переменной ed символьный тип данных char*/
    Console.Write("Введите первое число: ");
    x = Convert.ToDouble(Console.ReadLine());/* класс Convert преобразует значение одного базового типа данных в другой базовый
                                             тип данных, в данном случае строковый тип в вещественный тип данных */
    Console.Write("Введите знак операции: ");
    ed = (char)Console.Read(); /* присвоение переменной ed символьный тип char*/
    Console.ReadLine();
    Console.Write("Введите второе число: ");
    y = Convert.ToDouble(Console.ReadLine());
    es = true;// присвоение переменной логического значения
    switch (ed)/* разветвляет процесс вычислений на несколько на-правлений*/
    {
        case '+': k = x + y; break; /* ветвь переключателя заканчивается явным оператором перехода break */
        case '-': k = x - y; break;
        case '*': k = x * y; break;
        case '/': k = x / y; break;
        default: k = 0; es = false; break;
    }
    if (es)
        Console.WriteLine("Результат = " + k);
    else
        Console.WriteLine("Недопустимая операция");
    Console.ReadKey();
```

Рисунок 2.3.3 – Калькулятор на 4 действия.

Задание: создать калькулятор в C# на 3 действия (возводить в указанную степень, вычислять квадратный корень, вычислять проценты).

Оператор цикла for. В этом цикле – есть заданный промежуток: начало цикла, конец цикла, и число итераций. Этот цикл предпочтительнее, если вы знаете, сколько итераций вам нужно.

Оператор цикла foreach – цикл по каждому элементу. Он работает с коллекциями предметов, например массивы (например, для перебора всех элементов в массиве) или другие встроенные типы списков.

Пример: вычисление факториала f! с использованием цикла for

```
class Program
{
    static void Main(string[] args)
    {
        int f = 1;
        int x = int.Parse(Console.ReadLine());  // создаём запрос ввода значения x, факториал которого необходимо найти
        for (int i = 2; i <= x; i++)            //создаём цикл увеличивающий значение i на единицу начиная с 2 до x
        {
            f = f * i;                          // Переопределяем переменную f, которая будет равна:f=1*2*3*…*x
        }
        Console.WriteLine(f);                   //Выводим на экран значение f
    }
}
```

Рисунок 2.3.4 – Нахождения факториала.

Задание: вычислите значение факториала: $f =$, где x – вводиться с клавиатуры.

Лабораторная работа №4

Цель работы: Работа с циклами while и do while

Пример: Найти минимум функции $y=x^2$

```
static void Main(string[] args)
{
    double a, b, c, q, y1, y2;                  /*присвоение переменным вещественный тип данных */
    Console.WriteLine("Введите начальную точку");
    a = Convert.ToDouble(Console.ReadLine());    /* преобразует строковый тип в вещественный тип данных */
    Console.WriteLine("Введите конечную точку");
    b = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("Введите допустимую погрешность q");
    q = Convert.ToDouble(Console.ReadLine());
    while (Math.Abs(b + a) > q) /* оператор цикла while с преду-
*/
    {
        Console.WriteLine("a={0}", a);  // первое значение аргумента
        Console.WriteLine("b={0}", b);  // второе значение аргумента
        c = (a + b) / 2;                // середина отрезка [a,b]
        y1 = Math.Pow((c - q), 2);      // возведение в квадрат с учетом допустимой погрешности q
        y2 = Math.Pow((c + q), 2);
        if (y2 < y1) a = c; if (y2 > y1) b = c;
        if (y2 == y1)                   // если выполняются условие сравнения y2=y1, то происходит вычисление a и b
        {
            a = c - q; b = c + q;
        }
    }
    Console.WriteLine("Минимум функции y=x^2 находится в точке x=" + (a + b) / 2);
    Console.ReadKey();
```

Рисунок 2.3.5 – Нахождение минимума функции.

Задание:

1) преобразовать данную программу таким образом, чтобы использовался цикл do…while;

2) Найти минимум функции $y=2x^3$ с помощью циклов while и do…while.

Лабораторная работа №5

Цель работы: научиться использовать операторы обнаружения исключительных ситуаций.

В языке C# есть операторы, позволяющие обнаруживать и обрабатывать ошибки (исключительные ситуации), возникающие в процессе выполнения программы (деление на нуль, выход индекса за границы).

Но их изобилие удлиняет программу и затрудняет восприятие ее логики.

Исключения обнаруживаются и обрабатываются в операторе **try**, который включает в себя:

- контролируемый блок — составной оператор try. В контролируемый блок включаются потенциально опасные операторы программы.
- один или несколько обработчиков исключений — блоков catch, в которых описывается, как обрабатываются ошибки различных типов.
- блок завершения finally выполняется независимо от того, возникла ошибка в контролируемом блоке или нет.

В блоке после try находятся операторы, проверяемые на наличие исключительной ситуации. Если ни одна исключительная ситуация не возникла, то все блоки catch будут пропущены и выполнение программы продолжается с блока finally. При возникновении исключительной ситуации выполнение блока try прерывается и начинается поочередное выполнение блоков catch. Завершается выполнение блоком finally. Отсутствовать могут либо блоки catch, либо блок finally, но не оба одновременно.

Пример: Посчитать силу тока

```
static void Main()
{
    string buf;     //переменная ввода
    double u, i, r;
    try
    {
        Console.Write("Введите напряжение: " );
        buf = Console.ReadLine();
        u = double.Parse(buf);  // перевод из переменной ввода(buf) в переменную вычисления(u) с конвертацией
        Console.Write("Введите сопротивление: ");
        buf = Console.ReadLine();
        r = double.Parse(buf);
        i = u / r;
        Console.WriteLine("Сила тока: " + i + " Ампер");
    }
    catch (FormatException) // ошибка неверного формата ввода
    {
        Console.WriteLine("Неверный формат ввода!");
    }
    catch                   // общий случай
    {
        Console.WriteLine("Неопознанное исключение");
    }
    Console.ReadKey();
}
```

Рисунок 2.3.6 – Расчет силы тока.

Задание: посчитать плотность вещества введя его массу и объем с клавиатуры, используя исключения для проверки ввода и деления на ноль (DivideByZeroException - название ошибки).

Лабораторная работа № 6

Цель работы: работа с массивами.

В языке C# массив представляет собой указатель на непрерывный участок памяти (динамические массивы). Перед использованием массива он должен быть инициализирован, т.е. под него должна быть выделена память.

В C# минимальное значение индекса всегда равно нулю, поэтому максимальное равно количеству элементов минус 1.

В C# массивы рассматривают как классы. Это дает возможность использовать при их обработке свойства. Для работы с одномерными массивами полезными окажется свойство mas.Length – возвращает количество элементов массива mas.

Пример: Создать на диске d текстовый файл с расширением .txt, записав в него построчно цифры. Затем создать программу, которая прочитает этот файл и создаст одномерный массив из цифр указанных в файле и определит сумму, количество отрицательных элементов, а также максимальный элемент массива.

```
using System;
using System.IO; /* подключение пространства имен System.IO в котором описываются стандартные классы для работы с файлами */
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Lab6
{
    Ссылок: 0
    class Classl
    {
        Ссылок: 0
        static void Main()
        {
            StreamReader f = new StreamReader("d:\\Lab6.txt");/* StreamReader объявляет файловую переменную и связывает ее с файлом на диске */
            string s = f.ReadLine(); int nm = Convert.ToInt32(s);/* преобразование символьного типа string в целочисленный тип int и присвоение этого значения переменной nm*/
            string q = f.ReadLine(); int nj = Convert.ToInt32(q);
            string w = f.ReadLine(); int nk = Convert.ToInt32(w);
            string e = f.ReadLine(); int nh = Convert.ToInt32(e);
            string r = f.ReadLine(); int nl = Convert.ToInt32(r);
            string t = f.ReadLine(); int np = Convert.ToInt32(t);
            const int n = 6; //размер массива
            int[] a = new int[n] { nm, nj, nk, nh, nl, np };/* объявление и инициализация массива, т.е. выделяется память и элементам массива сразу будут присвоены значения */
            Console.WriteLine ( "Исходный массив:" );
            for ( int i = 0; i < n; ++i ) /* цикл, с помощью которого выводим исходный массив на экран */
                Console.Write ( " \t " + a [ i ] );
                Console.WriteLine();
                long sum = 0; // сумма отрицательных элементов
                int num = 0 ; // количество отрицательных элементов
            for ( int i = 0; i < n; ++i )
                if ( a [ i ] < 0 ) /* цикл, с помощью которого определяют сумму и количество отрицательных элементов */
                {       sum += a [i] ;
                        ++num;
                }
            Console.WriteLine( "Сумма отрицательных = " + sum );
            Console.WriteLine( "Кол-во отрицательных = " + num );
            int max = a [ 0 ] ; // начальный максимальный элемент
            for ( int i = 1 ; i < n; ++i ) /* цикл, с помощью которого определяется максимальный элемент в массиве */
                if ( a [ i ] > max ) max = a [ i ] ;
            Console.WriteLine( "Максимальный элемент = " + max );
        }
    }
```

Рисунок 2.3.7 – Работа с массивом текстового документа Lab6.

Задание: Создать на диске d текстовый файл с расширением .txt, записав в него построчно цифры. Затем создать программу, которая прочитает этот файл и создаст одномерный массив из цифр указанных в файле и определит сумму и количество положительных элементов, а также произведение максимального отрицательного и положительного элемент массива.

Лабораторная работа №7

Цель работы: Работа с классами, приобретение практических навыков работы классами и методами C#.

Пример: создать класс Demo, добавить в него методы установки и получения значения поля y.

```
{
    public class Demo  // создание класса
    {
    // блок объявления полей данных
    public int a = 1;
    public const double c = 1.66;
    static string s = "Demo";
    double y;

    // блок задания методов заботы с полями данных
    public double Gety() // метод получения поля y
  {
   return y; // возвращает поле y
  }
    public void Sety(double y_) // метод установки поля y
    {
        y = y_; // полю y присваеваем значение y_
    }
    public static string Gets() // метод получения поля s
    {
        return s; // возвращает поле s
    }
}
class classl
{
    static void Main()
    {
        Demo x = new Demo(); /* создание объекта типа Demo с помощью базового конструктора главного класса Object*/
        x.Sety(12.3);// вызов метода установки поля y
        Console.WriteLine(x.Gety()); //вызов метода получения поля y
        Console.WriteLine(Demo.Gets()); //вызов метода получения поля
        Console.WriteLine( Gets() ); // не работает, так как метод должен быть вызван через имя класса
    }
}
```

Рисунок 2.3.8 – Создание класса.

Данная программа сообщит об ошибке, так как в конце мы вызываем метод без объявления имени класса, которому он принадлежит.

Задание: с помощью класса создать два массива (размер и данные массива ввести с клавиатуры), найти максимальный и минимальный элементы каждого массива, а также вывести эти массивы на экран.

Windows Forms

3.1 Ввод в Windows Forms

Windows.Forms используется в Microsoft .NET для создания приложений, снабженных графическим интерфейсом. Основывается он на .NET Framework class library. Windows.Forms устраняют многие ошибки Windows API.

По сути, Windows.Forms — это набор различных управляемых библиотек, с помощью которых вы можете выполнить все необходимые для оконного приложения действия, начиная от обмена сообщениями с операционной системой для отслеживания любых событий клиентского окна, заканчивая диалоговыми системами, связью с другими компьютерами по сети и многими другими возможностями. В данном случае под формой понимается видимая поверхность окна, включающая информацию для конечного пользователя, а также содержащую в себе набор инструментов (элементов управления) для работы с представленными данными или взаимодействия с пользователем.

Так как Windows.Forms, по сути, должна включать сотни организованных классов, чтобы обеспечивать все необходимые возможности разработчику, .NET Framework разбита на ряд иерархических разделов, имеющих свои имена. System является корневым разделом и предназначен для описания фундаментальных типов данных.

3.2 Создание проекта

Для создания приложения Windows Forms в меню выберем пункт File (Файл) и в подменю выберем **New - > Project** (Создать - > Проект). После этого перед нами откроется диалоговое окно создания нового проекта. Выбираем **Приложение Windows Forms**, задаем **имя** нашему проекту и нажимаем **ок.**

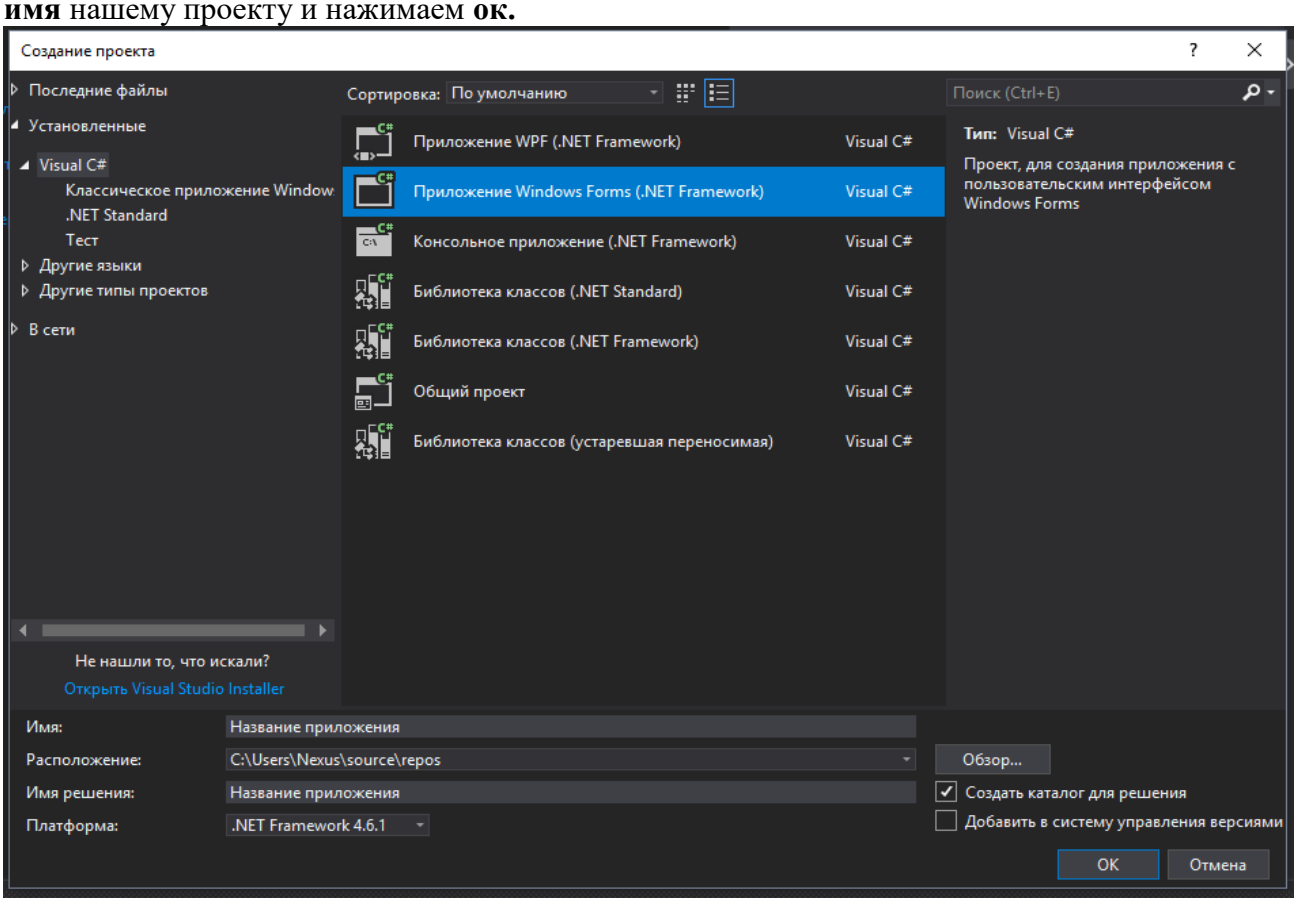

Рисунок 3.1.1 – Создание Windows Forms.

Среда разработки содержит три окна: главное окно, **Обозреватель решений** и окно **Свойства**.

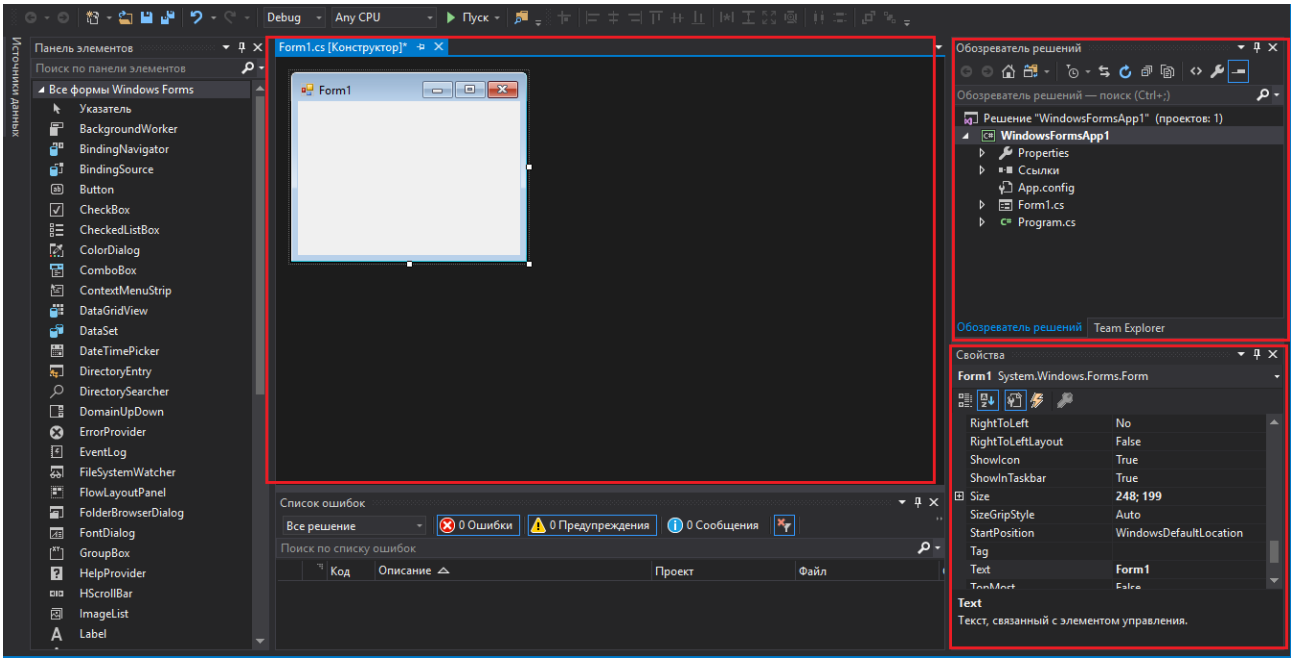

Рисунок 3.1.2 – Среда разработки.

Если какое-либо из этих окон отсутствует, восстановите макет окон по умолчанию, выбрав в строке меню **Окно** > **Сбросить макет окна**. Можно также отобразить окна с помощью команд меню. В строке меню выберите **Вид** > **Окно "Свойства"** или **Обозреватель решений**. Если открыты какие-либо другие окна, закройте их с помощью кнопки **Закрыть**(x) в верхнем правом углу.

- **Главное окно**. В этом окне выполняется основная часть работы, например работа с формами и редактирование кода. В окне показана форма в **редакторе форм**.
- **Окно "Обозреватель решений"** . В этом окне можно просматривать все элементы, входящие в решение, и переходить к ним. Если выбрать файл, содержимое в окне **Свойства** изменится. Если открыть файл кода (с расширением *.cs* в Visual C# и *.vb* в Visual Basic), откроется файл кода или конструктор для файла кода. Конструктор — это визуальная поверхность, на которую можно добавлять элементы управления, такие как кнопки и списки. При работе с формами Visual Studio такая поверхность называется **конструктор Windows Forms**.
- **Окно "Свойства"** . В этом окне производится изменение свойств элементов, выбранных в других окнах. Например, выбрав форму Form1, можно изменить ее название путем задания свойства **Text**, а также изменить цвет фона путем задания свойства **Backcolor**.

В верхней строке в **обозревателе решений** отображается текст **Решение "PictureViewer" (1 проект)** . Это означает, что Visual Studio автоматически создала для вас решение. Решение может содержать несколько проектов, но пока что вы будете работать с решениями, которые содержат только один проект.

Для предварительного запуска программы, так же как и в консольном приложении, мы можем запустить отладчик на клавишу **F5.** Конструктор непосредственно запустить спроектированное вами приложения для проверки а так же даст возможность использовать любые элементы задействованные в вашей программе.

3.2 Настройка свойств формы

В окне свойств можно задавать различные параметры для объектов формы или же для самой формы.

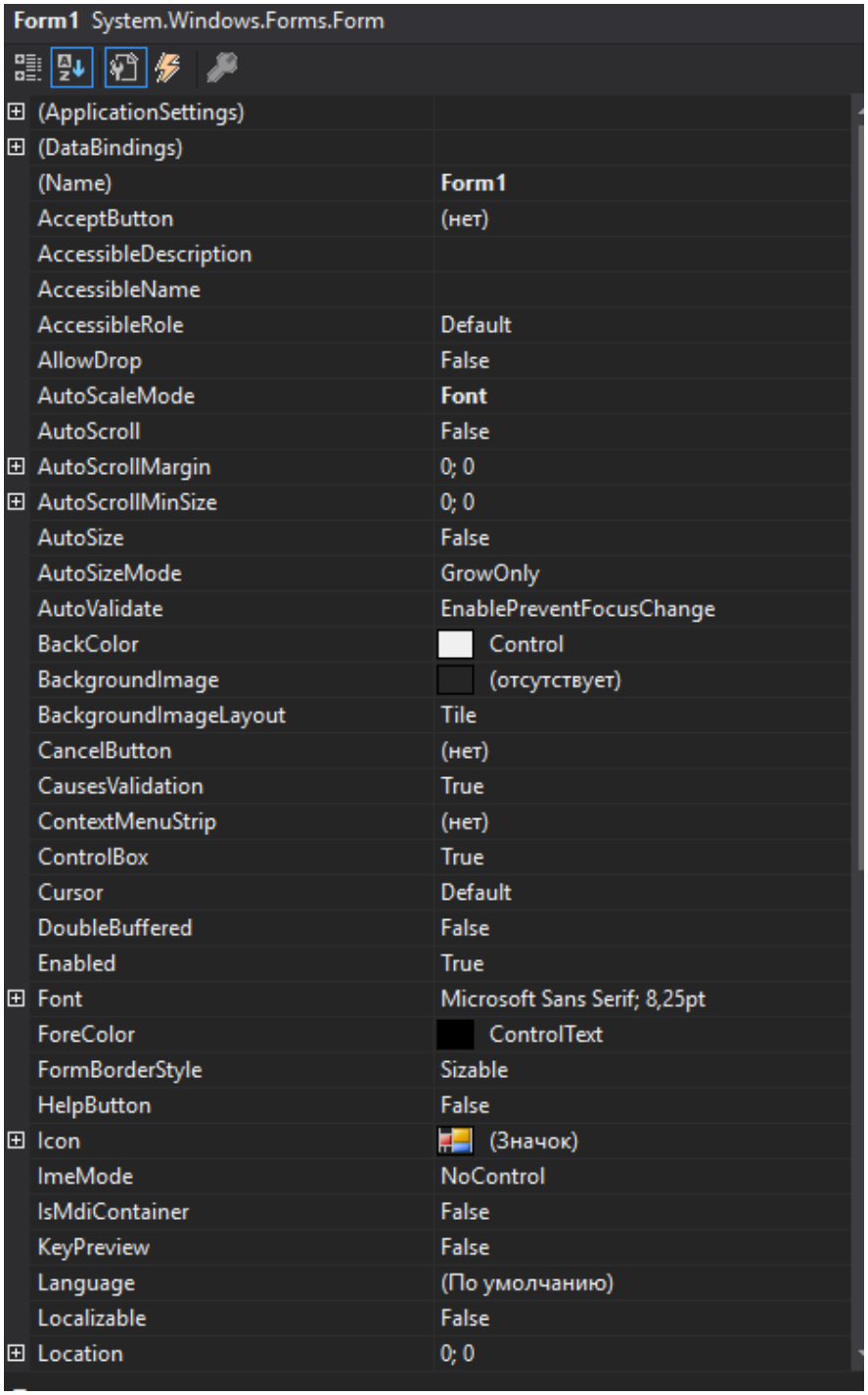

Рисунок 3.2.1 – Свойства формы Form1

Для того чтобы в окне свойств появились свойства формы необходимо кликнуть непосредственно на саму форму.

Наименование поля Form	Свойство поля Form
Name	Устанавливает имя формы - точнее имя класса, который наследуется от класса Form
BackColor	Указывает на фоновый цвет формы
BackgroundImage	Указывает на фоновое изображение формы
BackgroundImageLayout	Определяет, как изображение, заданное в свойстве BackgroundImage, будет располагаться на форме
ControlBox	Указывает, отображается ли меню формы (свернуть, в окне, выход)
Cursor	Определяет тип курсора, который используется на форме
Enabled	Возможность для пользователя вносить изменения или взаимодействовать с элементами (текстовые поля, кнопки и.д.)
Font	Задает шрифт для всей формы и всех помещенных на нее элементов управления (не запрещает менять шрифты для отдельных элементов)
ForeColor	Цвет шрифта на форме
FormBorderStyle	Указывает, как будет отображаться граница формы и строка заголовка
Location	Определяет положение по отношению к верхнему левому углу экрана, если для свойства StartPosition установлено значение Manual
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
Opacity	задает прозрачность формы
Size	определяет начальный размер формы
StartPosition	указывает на начальную позицию, с которой форма появляется на экране
Text	определяет заголовок формы
TopMost	если данное свойство имеет значение true, то форма всегда будет находиться поверх других окон
Visible	видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение false

Таблица 3.2.1 – Основные свойства Form

Помимо графического редактора можно использовать код формы для заданий ей свойств. Для того чтобы перейти к коду формы необходимо кликнуть по ней правой кнопкой мыши и в выпадающем меню выбрать опцию **Перейти к коду** или воспользоваться горячей клавишей **F7**

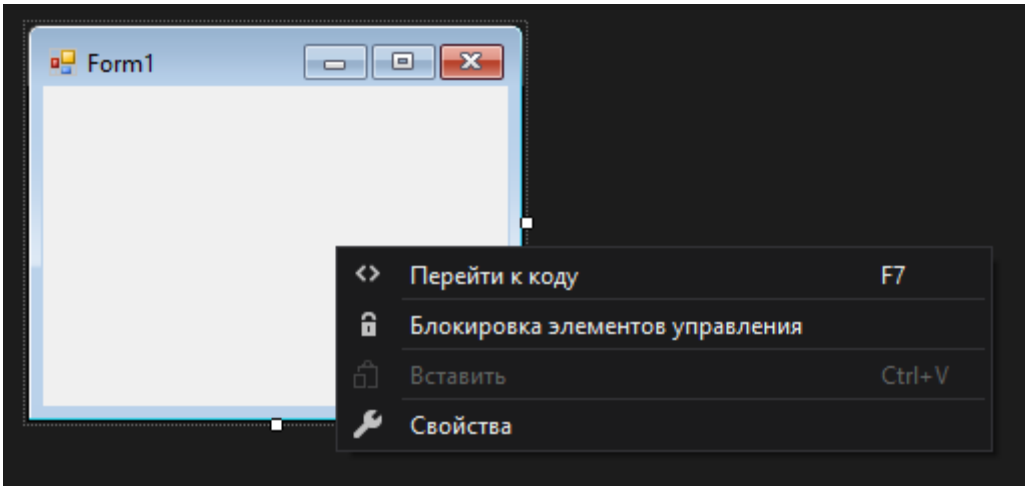

Рисунок 3.2.2 – Переход к коду формы.

Для того чтобы объявить изменения параметра в коде необходимо использовать метод **this**. Свойство задается конструктом: *this.ИмяСвойства = Значение свойства* (прим. Рисунок 3.2.3).

```
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            Text = "Hello World!";
            this.BackColor = Color.Aquamarine;
            this.Width = 250;
            this.Height = 250;
        }
    }
}
```

Рисунок 3.2.3 – Пример задания свойств.

Результатом кода изображенного на рисунке 3.2.3 будет форма аквамаринового цвета, с высотой и шириной 250 пикселей.

3.3 События

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.

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.

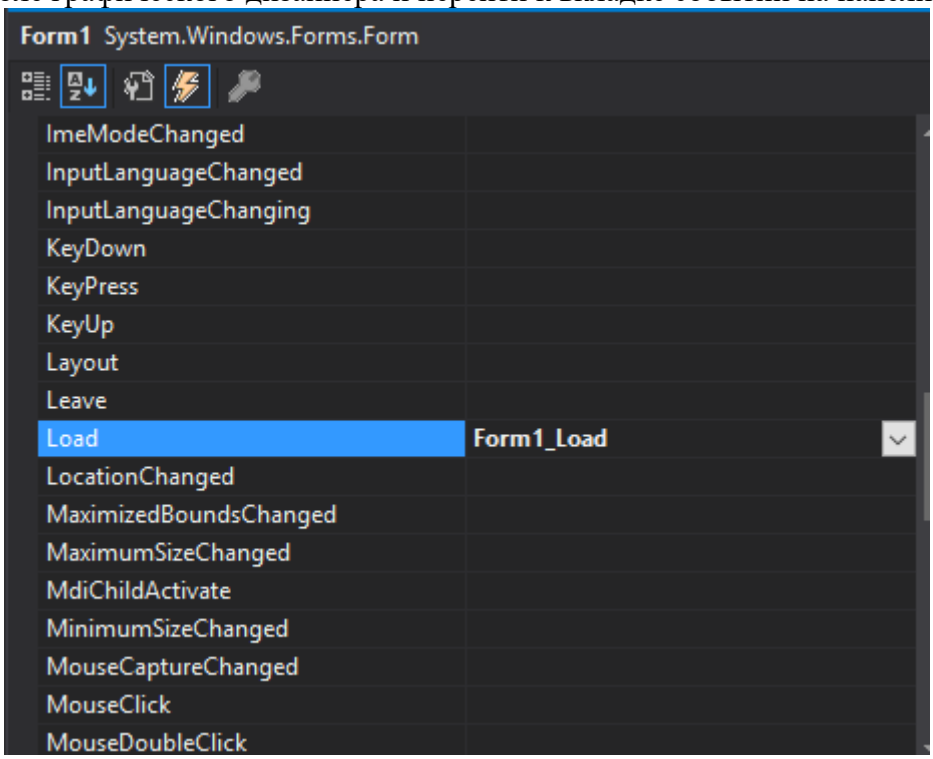

Рисунок 3.3.1 – события формы при загрузке.

Если мы перейдем в файл кода формы Form1.cs, то увидим автосгенерированный метод Form1_Load (рисунок 3.3.2).

```
    private void Form1_Load(object sender, EventArgs e)
    {

    }
}
```

Рисунок 3.3.2 – Автоматически созданный код.

И при каждой загрузке формы будет срабатывать код в обработчике Form1_Load. Как правило, большинство обработчиков различных визуальных компонентов имеют два параметра: sender - объект, инициировавший событие, и аргумент, хранящий информацию о событии (в данном случае EventArgs e).

3.4 Элементы GroupBox, Panel и FlowLayoutPanel

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.

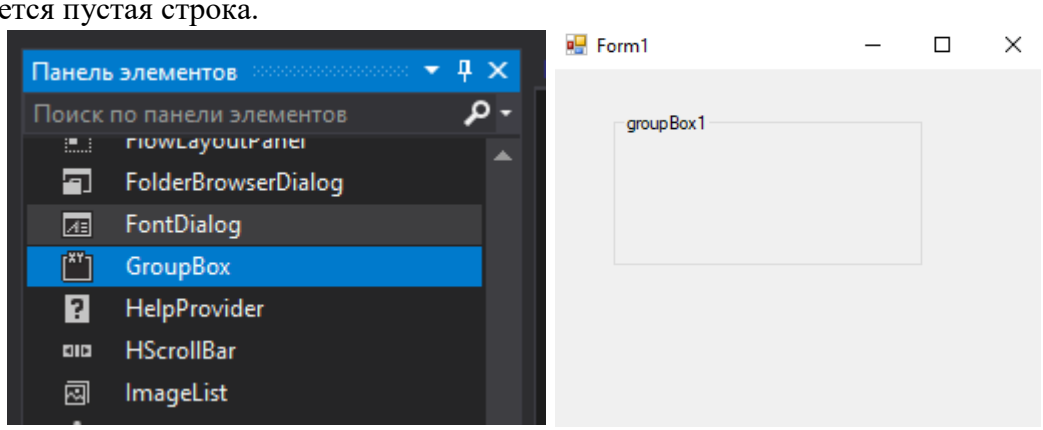

Рисунок 3.4.1 – Элемент GroupBox.

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

Элемент **Panel** представляет панель и также, как и GroupBox, объединяет элементы в группы. Она может визуально сливаться с остальной формой, если она имеет то же значение цвета фона в свойстве BackColor, что и форма. Чтобы ее выделить можно кроме цвета указать для элемента границы с помощью свойства BorderStyle, которое по умолчанию имеет значение None, то есть отсутствие границ.

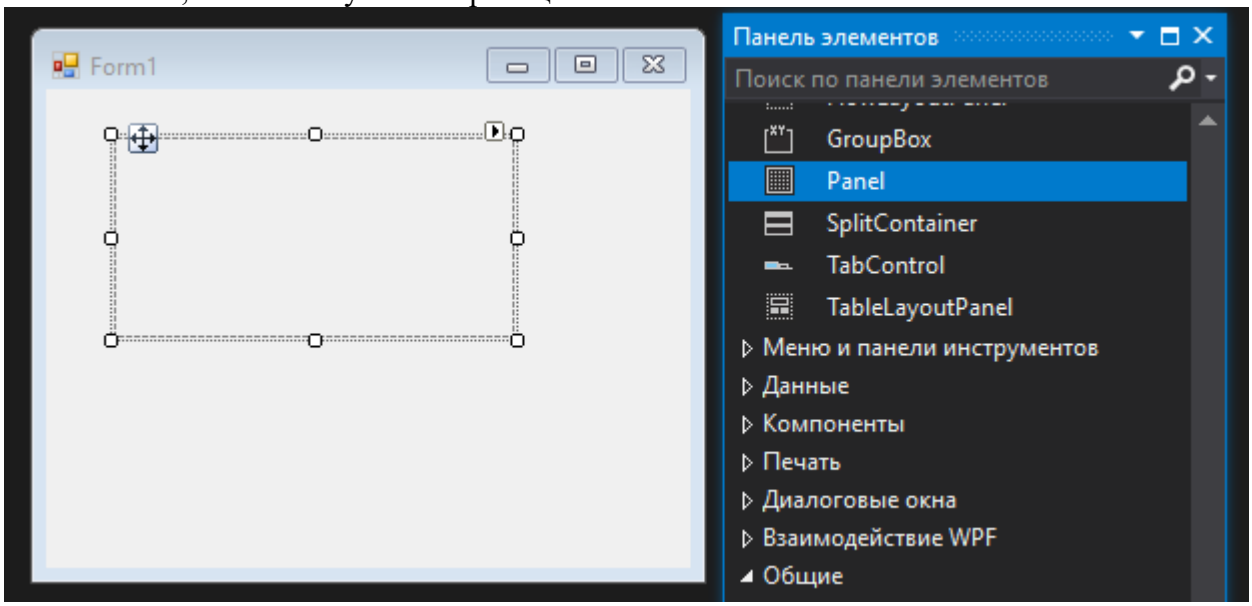

Рисунок 3.4.2 Элемент Panel.

Также если панель имеет много элементов, которые выходят за ее границы, мы можем сделать прокручиваемую панель, установив ее свойство AutoScroll в true

Элемент **FlowLayoutPanel** является унаследован от класса Panel, и поэтому наследует все его свойства. Однако при этом добавляя дополнительную функциональность. Так, этот элемент позволяет изменять позиционирование и компоновку дочерних элементов при изменении размеров формы во время выполнения программы.

Свойство элемента FlowDirection позволяет задать направление, в котором направлены дочерние элементы. По умолчанию имеет значение LeftToRight - то есть элементы будут располагаться начиная от левого верхнего края. Следующие элементы будут идти вправо. Это свойство также может принимать следующие значения:

- RightToLeft - элементы располагаются от правого верхнего угла в левую сторону
- TopDown - элементы располагаются от левого верхнего угла и идут вниз
- BottomUp - элементы располагаются от левого нижнего угла и идут вверх

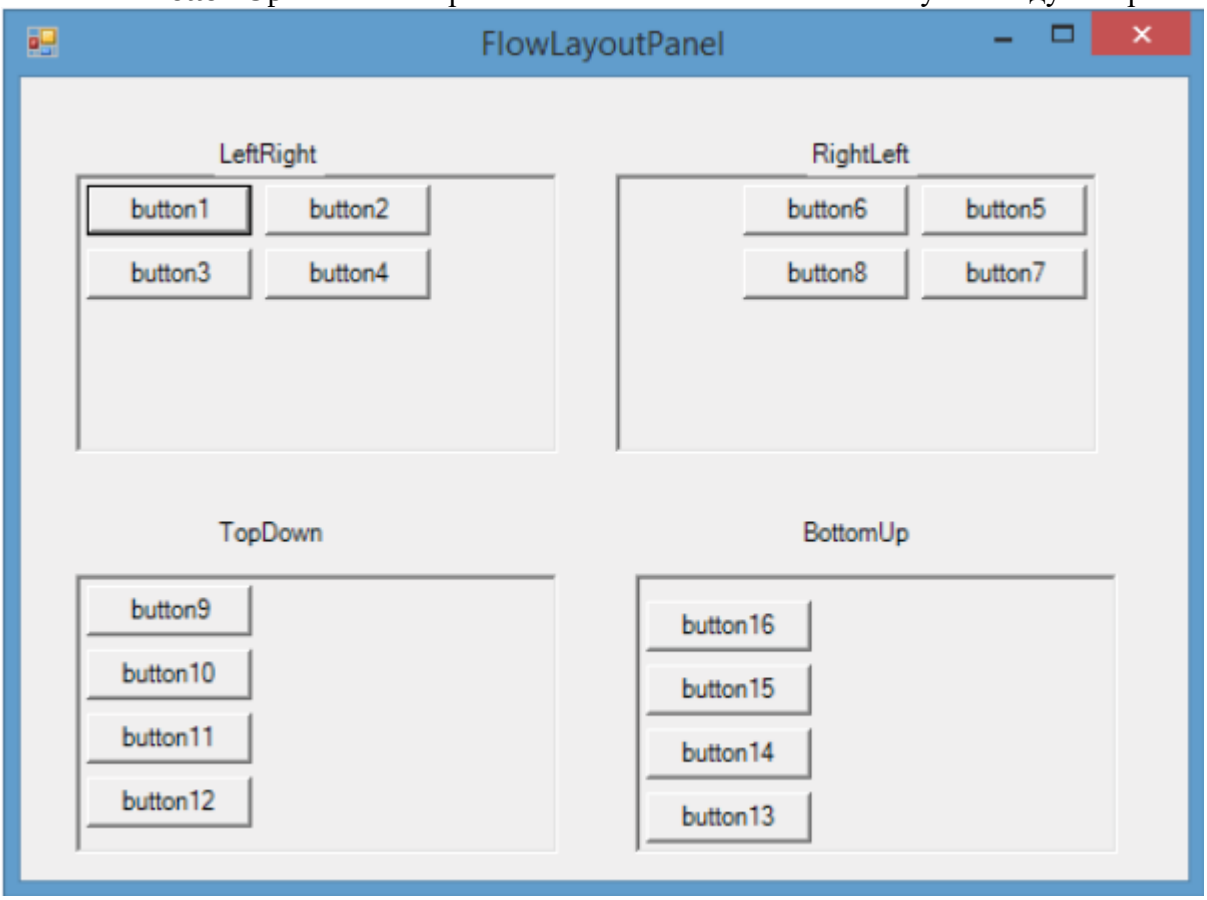

Рисунок 3.4.3 – Пример использования FlowLayoutPanel.

3.5 Лабораторная работа

Цель работы: познакомиться с принципами визуального программирования, научиться создавать интерфейсы.

Для создания интерфейса необходимо создать:

- Выбрать по очереди во вкладке Файл – Создать – Проект;
- Выбрать тип проекта Visual C#, Windows – приложение Windows Forms;
- Определить местонахождение нового проекта (Расположение) и дать ему имя.

Появиться поле с заголовком Form1. Нажав на нем два раза левой кнопкой мышки, мы сможем увидеть листинг программы, относящийся к нашему интерфейсу. Чтобы добавлять элементы на наше поле необходимо во вкладке «Вид» выбрать пункт «Панель элементов». Из данного списка добавим несколько элементов: Label (4 шт.) метка – предназначена для нанесения на форму пояснительных текстов и для вывода результатов; TextBox (2 шт.) строка редактирования – предназначена для ввода/вывода, тип данных (всегда String); Button (2 шт.) командная кнопка – можно ставить в соответствие функцию, которая будет выполнена при нажатии на кнопку (в нашем примере это кнопка «Вычисление» и «Выход»); RadioButton (4 шт.) радиокнопка – радиокнопки обычно объединяют в радиогруппы и из каждой группы может быть выбрана одна и только одна радиокнопка. Для создания радиогруппы необходимо занести на форму компонент рамка GroupBox и лишь после этого на него требуемое количество (в нашем случае 4) радиокнопок.

Чтобы связать с программный код с добавляемым компонентом необходимо также щелкнуть два раза левой кнопкой мышки на данном компоненте, в результате появиться поле в которое необходимо будет заполнить соответствующими для данного элемента командами.

Чтобы изменять свойства элементов необходимо во вкладке «Вид» выбрать «Окно свойств», которое дает информацию об элементе при нажатии на него.

Скопируйте код и добавьте все описанные элементы в нем на форму, перетаскивая их из панели элементов и давая им имена следуя коду и функционалу.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Lab8
{
    public partial class Form1 : Form
    {
        public Form1() { InitializeComponent(); }
        private void button1_Click(object sender, EventArgs e)
        {
            double x, y, a = 0; bool ok; ok = false; x = Convert.ToInt32(textBox1.Text); /*
присвоение переменной "x" значение первого числа */
            y = Convert.ToInt32(textBox2.Text); if (radioButton1.Checked) // описание
радиокнопки (сложение)
            {
                a = x + y;
                ok = true;
            }
            if (radioButton2.Checked) /* описание радиокнопки (вычитание)*/
            {
                a = x - y;
                ok = true;
            }
            if (radioButton3.Checked) /* описание радиокнопки (умножение) */
            {
                a = x * y;
                ok = true;
            }
```

```
            if (radioButton4.Checked) // описание радиокнопки (деление)
            {
                a = x/y;
                ok = true;
            }
            if (ok) // вывод результата вычисления
                label4.Text = Convert.ToString(a);
            else // проверка нажатия радиокнопки
                label4.Text = "ВЫБЕРЕТЕ ОПЕРАЦИЮ!";
        }
        private void button2_Click(object sender, EventArgs e)
        {
            // завершение работы приложения
            Close();
        }
    }
}
```

Задание:

1) добавить в интерфейс калькулятора ещё 3 действия;
возводить в указанную степень, вычислять квадратный корень, вычислять проценты.

2) создать интерфейс для решения квадратного уравнения $a \cdot x^2 + b \cdot x + c = 0$.