Классы, объекты, элементы

Объекты - программные конструкции, включающие набор логически связанных свойств (данных) и методов. Объекты - автономные сущности, предоставляющие свою функциональность другим компонентам приложения, изолируя от них свое внутреннее устройство. Объекты создаются на основе шаблонов, которые называются ***классами*** и являются **экземплярами** этих классов. Библиотека **FCL** содержит большой набор уже созданных классов, которые можно применять для создания объектов. В приложениях можно создавать классы, требуемые для описания решаемой задачи.

Описание класса «Автомобиль»:

```
class Автомобиль
{
   // описание данных
   // описание методов
}
```

Классы - *«шаблоны» (чертежи) объектов*. Они определяют все элементы объекта: свойства (данные) и поведение (методы), а также задают начальные значения для создаваемых объектов, при необходимости. Класс - механизм, задающий структуру (данные) однотипных объектов и их функциональность (механизм, определяющий все методы, относящиеся к объектам). Среди данных и методов класса могут существовать такие, которые принадлежат не одному объекту, а всем объектам класса (метод (поле) класса может оперировать с любыми объектами класса).

Класс может представлять собой:

контейнер для методов и данных, принадлежащих всем объектам класса;
"трафарет" *(«шаблон»)*, позволяющий создавать конкретные объекты (экземпляры) класса.

Класс: *«Студент группы № курса»* может содержать:
-поля (данные) ***объекта***: ФИО, оценки, книги из библиотеки;
-методы (данные) ***объекта***: сдать экзамен, получить книги из библиотеки;
-поля (данные) ***класса***: номер курса, даты экзаменов, количество предметов;
-методы (данные) ***класса***: перевести группу на следующий курс: изменятся все данные класса (не все объекты останутся в измененном классе - не обязательно все студенты будут переведены на следующий курс).

Различие между данными и методами объектов и данными и методами их класса часто используется в C#. В определении (объявлении) класса его данные и методы, принадлежащие всем объектам класса, описывают модификатором **static** (статический).

Поля и методы, как класса, так и формируемых его помощью объектов называют ***членами класса***. При создании экземпляра класса в памяти создается копия этого класса. *Созданный таким образом экземпляр класса называют* ***объектом***

Для создания экземпляра класса используется специальная операция **new**:

```
// Объявление переменной типа Автомобиль
Автомобиль Auto; // переменная это не объект класса!
// Создание экземпляра класса Автомобиль
// и сохранение ссылки на него в переменной myAuto
Auto = new Автомобиль()
```

При создании объекта, выделяется блок оперативной памяти, в который записывается копия данных, и адрес этого блока присваивается переменной (переменная Auto хранит ссылку), Объекты класса не зависят друг от друга (являются отдельными программными конструкциями). Можно создать любое число объектов (копий) класса, которые могут существовать одновременно. Если автомобиль - ***объект***, то чертежи автомобиля - ***класс*** Автомобиль. По чертежу можно сделать много автомобилей. Если один из автомобилей будет работать не так, как все, это никак не повлияет на остальные автомобили.

У класса две различные роли: ***модуля*** и типа ***данных***.

Класс - модуль, архитектурная единица построения программной системы. Модульность - основное свойство построения систем. В ООП модульная система - состоит из классов, являющихся основным видом модуля. Размер модуля и его содержание определяется архитектурными соображениями (можно построить монолитную систему, состоящую из одного модуля - она может решать ту же задачу, что и система, состоящая из многих модулей).

Класс - тип данных, для реализации некоторой абстракции данных, характерной для задачи (для ее решения создается программная система). С этих позиций классы - не просто кирпичики, из которых строится система. Каждый кирпичик имеет важную содержательную начинку. Современный дом, построенный из кирпичей отличается от дома будущего, где каждый кирпич выполняет определенную функцию: один следит за температурой, другой - за составом воздуха в доме. ООП система - дом будущего. Состав класса определяется абстракцией, которую должен реализовать класс. Разработка ООП системы основана на стиле "***проектированием от данных***". Проектирование сводится к поиску абстракций данных, подходящих для решения задачи. Каждая абстракция реализуется в виде класса, которые и становятся модулями - архитектурными единицами построения системы.

В основе класса лежит абстрактный тип данных. В хорошо спроектированной ООП системе каждый класс играет обе роли, так что каждый модуль системы имеет вполне определенную смысловую нагрузку.

Объекты состоят из **элементов**, к которым относят ***поля***, ***свойства***, ***методы*** и ***события***, представляющие данные и функциональность объекта. ***Поля*** содержат данные объекта, ***свойства*** – предоставляют управляемый способ доступа к данным объекта, ***методы*** – определяют действия, которые объект способен выполнять, а ***события*** уведомляют, заинтересованных пользователей (другие классы, которые могут принимать (подпишутся на) эти события), если в объекте происходит что-то важное.

Свойства объектов класса «Автомобиль», например, такие как *Цвет*, *Модель*, *Расход_топлива* являются данными, описывающими состояние объекта, а ***методы*** этих объектов: *Нажать_акселератор*, *Переключить_передачу*, *Повернуть_руль* описывают функциональность автомобиля. ***Методы*** позволяют реализовать поведение объекта.

События представляют собой уведомления о важных происшествиях в объекте (количество бензина стало ниже заданной величины или объект класса «Двигатель» может уведомить объект класса «Автомобиль» о событии *Перегрев_двигателя*). В этом случае описание класса «Автомобиль» выглядит следующим образом:

```
class Автомобиль
    {  // описание свойств
       public string Модель;
       public float Расход_топлива;
       private int Число_цилиндров;
       // описание методов
       public void Повернуть_руль(){...}
       private void Регулировка_датчика(){...}
       // описание события
      event Событие Перегрев_двигателя;
    }
```

Для доступа к полям, свойствам, методам элементам объектов используется специальная операция **точка (.)**:

```
string Модель; // Объявление переменной Модель типа string
Модель = Auto.Модель; // определение модели автомобиля
Auto.Повернуть_руль(); // поворот руля автомобиля
```

Возможность использования элементов класса задается с помощью указания режима доступа. Режимы: ***private*** – элемент можно использовать только в методах того класса, где он описан (закрытые элементы); ***public*** – элемент можно использовать в других класса (открытые элементы).

Инкапсуляция, наследование, полиморфизм

Объект - это программная конструкция, представляющая некоторую сущность. В нашей повседневной жизни сущностями, или объектами, например, можно считать: автомобили, велосипеды, настольные компьютеры, банковский счет. Каждый объект обладает определенной функциональностью и свойствами. Объект представляет собой завершенную функциональную единицу, содержащую все данные и предоставляющую всю функциональность, необходимую для решения задачи, для которой он предназначен. Описание объектов реального мира при помощи программных объектов называют абстрагированием.

Инкапсуляция - отделение реализации объекта (его внутреннего содержания) от способа взаимодействия с ним. Другие объекты взаимодействует с рассматриваемым объектом посредством имеющихся у него открытых *public*-свойств и методов, которые составляют его интерфейс. В общем виде под интерфейсом понимается открытый способ взаимодействия между разными системами. Если интерфейс класса не будет меняться, то приложение сохраняет способность к взаимодействию с его объектами, даже если в новой версии класса его реализация значительно изменится. Объекты могут взаимодействовать друг с другом только через свои открытые методы и свойства, поэтому объект должен предоставлять доступ только к тем свойствам и методам, которые пользователям необходимы. Интерфейс не должен открывать доступ к внутренним данным объекта, поэтому поля с внутренними данными объекта обычно объявляют с модификатором *private*. У объектов класса «Автомобиль», которые могут взаимодействовать с объектами класса «Водитель» через открытый интерфейс, открытыми объявлены методы: *Ехать_вперед*, *Ехать_назад*, *Повернуть* и *Остановиться* - их достаточно для взаимодействия объектов классов «Водитель» и «Автомобиль». У объекта класса «Автомобиль» может быть вложенный объект класса «Двигатель», но он будет закрыт для объектов класса «Водитель», которому будут открыты лишь методы, для управления автомобилем. В этом случае можно заменить вложенный объект класса «Двигатель», и взаимодействующий с ним объект класса «Водитель» не заметит замены, если она не нарушит корректную работу интерфейса. Классы разных объектов, как и в реальном мире, связаны между собой.

Существует два типа взаимосвязи классов: ***вложенность*** и ***наследование***.

Вложенность - включение объектов одного класса в объекты другого класса.

Наследование - описание одного класса на основе другого класса.

Наследование позволяет создавать новые классы на основе существующих, при этом новые классы обладают всей функциональностью старых и могут модифицировать их. Класс, объявленный на основе некоторого (базового) класса, называется производным или классом-потомком. У любого класса может быть только один прямой предок – его базовый класс (***base class***).

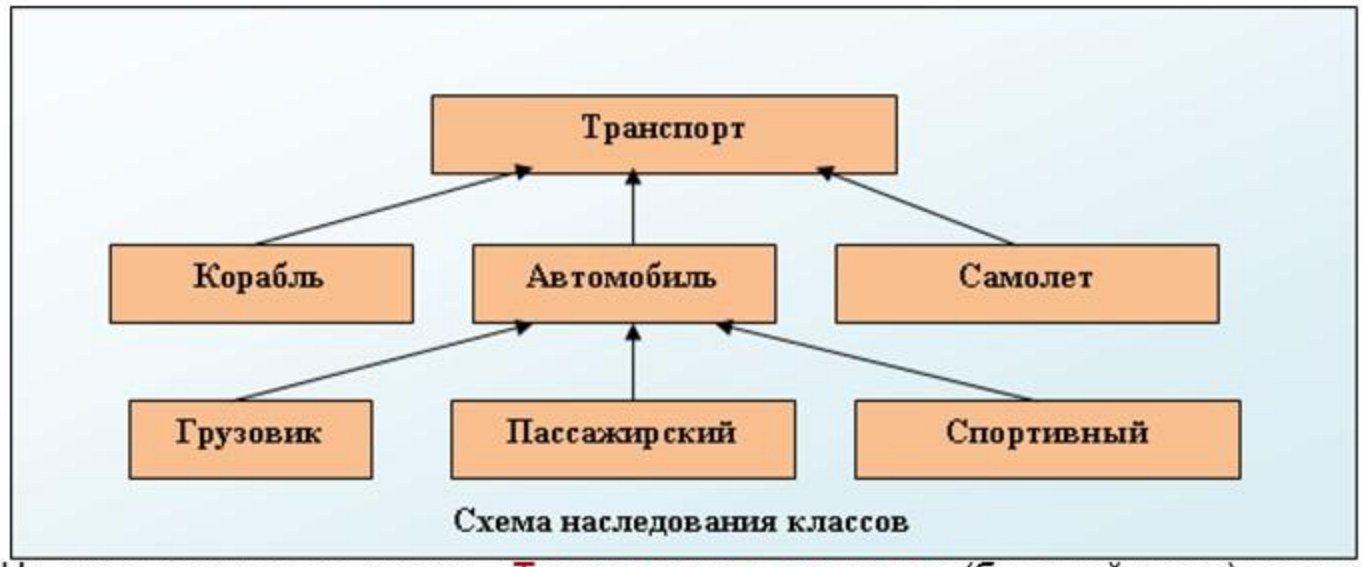


Схема наследования классов

На основе описания класса «Транспортное_средство» (базовый класс) можно описать класс «Автомобиль». Описание производного класса «Автомобиль»:

```
class Автомобиль : Транспорт
    {// описание свойств
       public string Модель;
       public float Расход_топлива;
       private int Число_цилиндров;
      // описание методов
       public void Повернуть_руль() { }
       private void Регулировка_датчика() { }
      // описание события
       event Событие Перегрев_двигателя;
    }
```

Полиморфизм состоит в том, что одни и те же открытые интерфейсы можно по-разному реализовать в разных классах. Полиморфизм позволяет вызывать методы и свойства объекта независимо от их реализации. Объект класса «Водитель» взаимодействует с объектом класса «Автомобиль» через открытый интерфейс. Если другой объект «Грузовик» или «Гоночный_автомобиль», поддерживает такой открытый интерфейс, то объект класса «Водитель» сможет взаимодействовать и с ними (управлять ими), невзирая на различия в реализации интерфейса. Основных подходов к реализации полиморфизма два: через ***интерфейсы*** и через ***наследование***.

Реализация полиморфизма с помощью интерфейсов

Интерфейс (interface) - это соглашение, определяющее набор открытых методов, реализованных классом. Интерфейс определяет список методов класса, но ничего не говорит об их реализации. В объекте допустимо реализовать несколько интерфейсов, а один и тот же интерфейс можно реализовать в разных классах. Описание интерфейса возможности управления объектами (имена интерфейсов обычно начинаться с буквы **I**):

```
interface IУправлять
{
   int Ехать(...);
   float Повернуть(...);
   bool Остановиться(...);
}
```

При реализации в классе интерфейса в классе должны быть описаны все методы этого интерфейса:

```
class Автомобиль : Транспорт, IУправлять
{
 int Ехать(...) {<реализация метода>};
 float Повернуть(...){<реализация метода>};
 bool Остановиться(...){<реализация метода>};
 // описание других элементов ...
}
```

Объекты, в которых реализован интерфейс, способны взаимодействовать друг с другом с его помощью. Интерфейс *IУправлять* можно реализовать и в других классах, таких, как «Грузовик», «Автопогрузчик» или «Катер». Тогда эти объекты смогут взаимодействовать с объектом класса «Водитель». Объект класса «Водитель» в полном неведении относительно реализации интерфейса, с которым он взаимодействует, он знает лишь сам интерфейс.

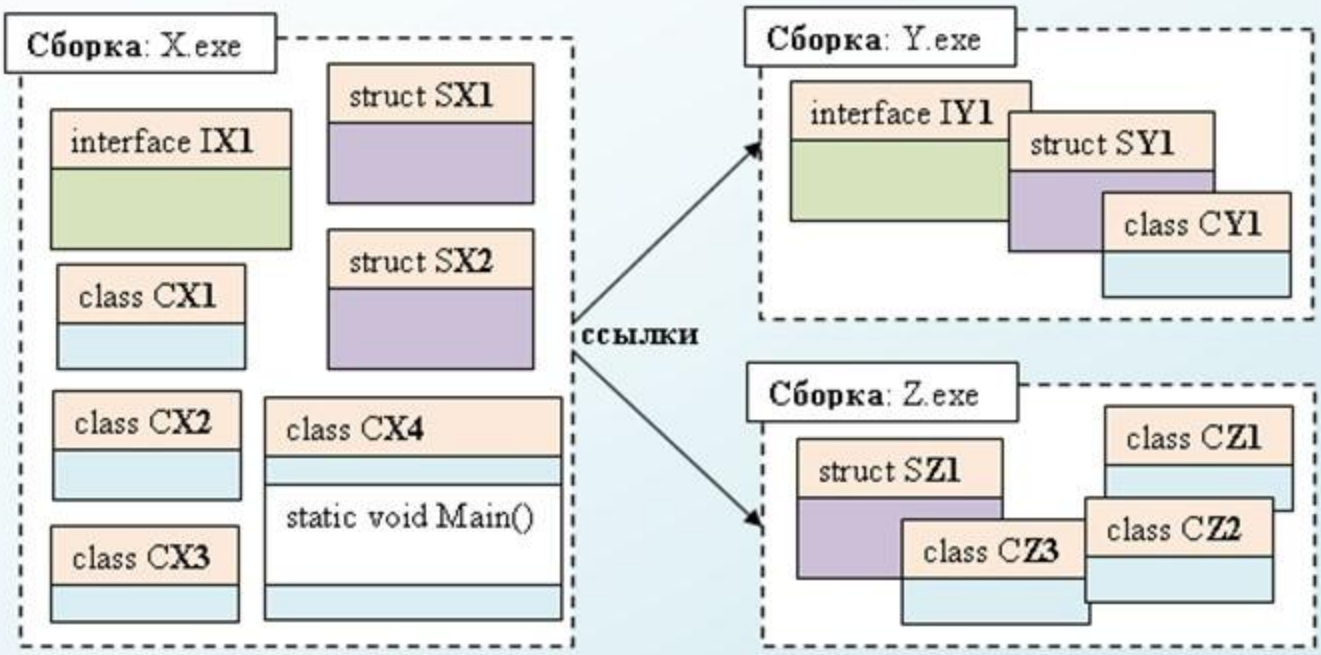
Реализация полиморфизма через наследование

Производные классы сохраняют все характеристики своих базовых классов и способны взаимодействовать с другими объектами, под видом *экземпляров базового класса* (переменным базового типа можно присваивать ссылки на объекты производных классов).

```
Автомобиль Auto; // переменная это не объект класса!
Спортивный_автомобиль sportAuto = new Спортивный_автомобиль();
 // можно присвоить, так как есть наследование
 Auto = sportAuto;
```

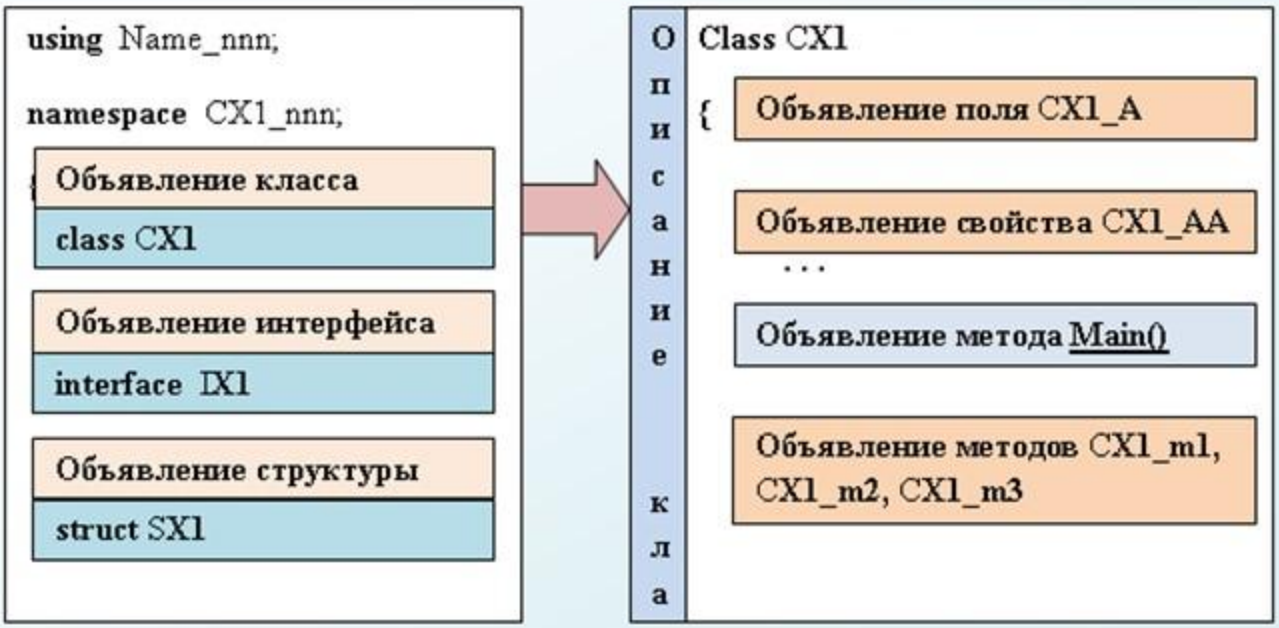
В этом случае можно работать с объектом производного класса, как если бы он был объектом базового класса.

Платформа .Net и C# полностью соответствуют принципам и методологии ООП. Программа состоит из описанных разработчиком типов данных, а также использует типы библиотеки FCL и сборок (используемых в приложении).



Распределение пользовательских типов программы по сборкам

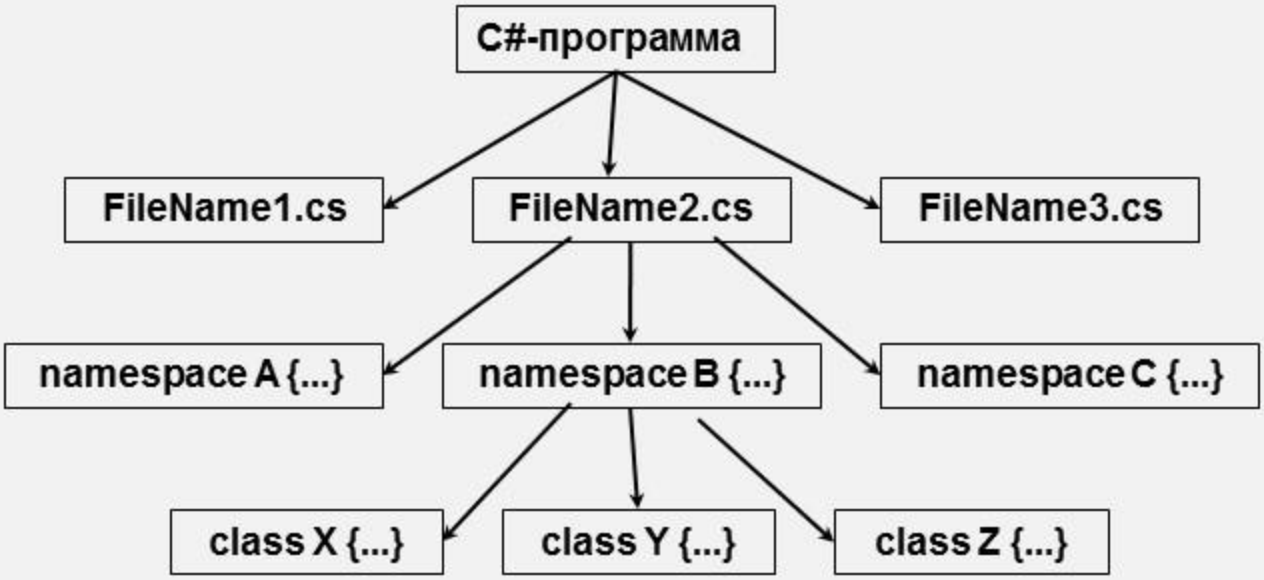
Программа на C# состоит из классов (в простом варианте - включает один класс). *Один из классов* программы на C# *должен* ***обязательно*** *включать метод* - **static Main()**, *с которого начинается выполнения приложения C#.*



Структура приложения и описания класса

Пространства имен представляют собой способ организации различных типов, присутствующих в программах C#. Их можно сравнить с папкой в компьютерной файловой системе. Подобно папкам, пространства имен определяют для классов уникальные полные имена.

Структура программы на языке C#



Программа на C# может состоять как из одного, так и из нескольких файлов, содержащих исходный текст на языке программирования C#. Каждый такой файл имеет расширение **.CS** (в примере файлы названы FileName1.cs, FileName2.cs и FileName1.cs).

Любой файл с исходным текстом на языке программирования C# может как содержать пространства имен, так и не содержать их (в примере файл названы FileName2.cs содержит три пространства имен (A, B и C), а FileName1.cs и FileName3.cs не содержат пространств имен).

Каждое пространство имен может как содержать описание (одного или нескольких) классов, так и не содержать их (в нашем примере пространство имен B содержит три описания трех классов (X, Y и Z), а пространство имен A и C не содержат ни одного описания классов).

Файл: Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Имена;
using Имена1;
 namespace Пространство__имен
{
   class Program
   {
      static void Main(string[] args)
      {
         Class1.wr();
         Class2.wr();
//Class0.wr(); Нельзя использовать классы с одинаковыми именами,
находящиеся в  разных используемых пространствах имен, без
указания их полного имени
//Ошибка "Class0"-неоднозначная ссылка между "Имена.Class0" и
"Имена1.Class0"
         Имена.Class0.wr();
         Имена1.Class0.wr();
         Console.ReadLine();
      }
   }
}
```

Файл: Class1.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Пространство__имен
{
   public class Class1
   {
      public static void wr()
      {  Console.WriteLine("Пространство__имен.Class1");  }
    }
}
```

Файл: Class2.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Пространство__имен
{   public class Class2
    {  static public void wr()
       {   Console.WriteLine("Пространство__имен.Class2");  }
}  }
```

Файл: CodeFile1.cs

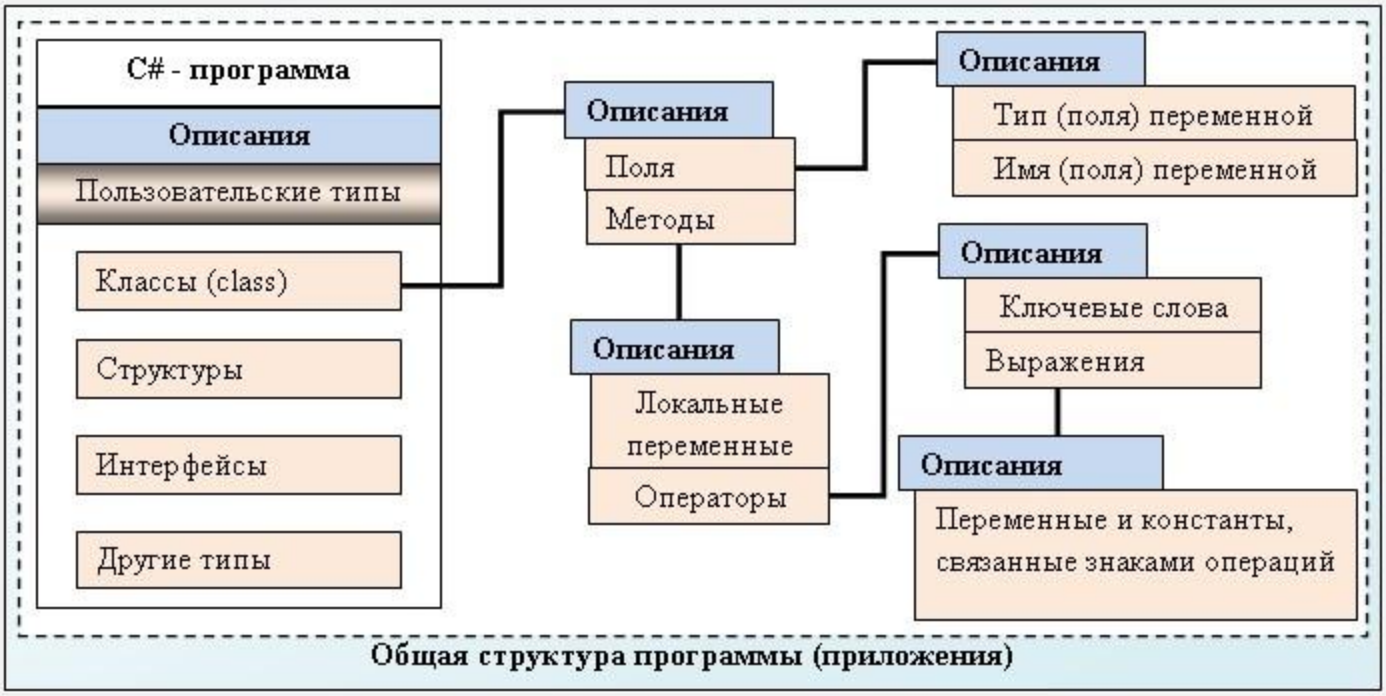
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Имена
{  public class Class0
    {  static public void wr()
       {   Console.WriteLine("Имена.Class0"); }
    }
    public class Class2
    {   static public void wr()
       {    Console.WriteLine("Имена.Class2");  }
    }
 }
```

Файл: CodeFile2.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Имена1
{   public class Class0
    {  static public void wr()
       {    Console.WriteLine("Имена1.Class0");   }
    }
    public class Class2
    {  static public void wr()
       {Console.WriteLine("Имена1.Class2");}
    }
}
```

Результат выполнения:

```
Имена.Class0
Пространство__имен.Class1
Пространство__имен.Class2
Имена.Class2
```



Общая структура программы (приложения)

Программа состоит из описаний пользовательских типов (классов).

- Описания классов состоят из описания полей (переменных) и методов.
- Описание переменных состоит из указания типа и имени переменной.
- Описание методов состоит из описания переменных и набора операторов.
- Оператор состоит из набора ключевых слов и выражений.

```
 1 - using System;
 2 - namespace ConsoleApplication;
 3 - {
 4 -    class Program
 5 -    {
 6 -       static void Main()
 7 -       {
 8 -           Console.Write("Введите радиус круга:");
 9 -           string  s = Console.ReadLine();
10 -           double r = Convert.ToDouble(s);
11 -           double p = Math.PI * r * r;
12 -           Console.WriteLine("Площадь круга = {0}", p);
13 -           Console.ReadLine();
14 -           return;
15 -       }
16 -    }
17 - }
```

Результат выполнения:
Введите радиус круга: 10
Площадь круга =314,159265358979

Разработка приложений. ***Проект*** - множество файлов с описаниями классов, ссылками на используемые сборки. Файлы проекта хранятся в одной специально создаваемой папке. ***Решение*** содержит один (несколько) ***проект***, ресурсы, необходимые этим у***проекту***, дополнительные файлы, не входящие в *проект*. Один из ***проектов решения*** должен быть указан, как ***стартовый проект***. Выполнение ***решения*** начинается со ***стартового проекта***. ***Стартовый проект*** должен иметь точку входа – класс, в котором содержится статический метод **Main()** - с него начинается выполнение .

Класс Console

Свойства	Описание
CursorVisible	Возвращает или задает значение, указывающее, видимость курсора.
CursorLeft	Возвращает или задает позицию столбца курсора.
CursorSize	Возвращает или задает высоту курсора в символьной ячейке.
CursorTop	Возвращает или задает позицию строки курсора.
Title	Возвращает или задает заголовок для отображения в строке заголовка консоли.
TreatControlCAsInput	Возвращает или задает значение, указывающее, как интерпретируется ли сочетание клавиш Control и C console key (CTRL+C): как обычный ввод или как сигнал прерывания, обрабатываемый операционной системой.
BackgroundColor	Возвращает или задает цвет фона консоли.
ForegroundColor	Возвращает или задает цвет текста консоли.
WindowHeight	Возвращает или задает высоту области окна консоли.
WindowLeft	Возвращает или задает позицию левого края области окна консоли.
WindowTop	Возвращает или задает позицию верхнего края области окна консоли.
WindowWidth	Возвращает или задает ширину окна консоли.

Методы	Описание
ReadLine()	Читает строку символов, введенную с клавиатуры (ввод завершается нажатием клавиши Enter). Возвращает строку типа string.
Read()	Ждет нажатия клавиш (ввод завершаются нажатием клавиши Enter) и возвращает код первого введенного символа (если в буфере есть символы, то они читаются).

Методы	Описание
ReadKey()	Получает следующий нажатый пользователем символ или функциональную клавишу. Нажатая клавиша отображается в окне консоли.
Write()	Вывод значений различных типов в консольное окно без перехода на новую строку.
WriteLine()	Вывод значений различных типов в консольное окно с переходом на новую строку.
Clear()	Очистка содержания консольного окна.
Beep()	Издает звуковой сигнал через динамик ПК.
SetWindowSize (Width, Height)	Устанавливает заданные значения высоты (Height) и ширины (Width) окна консоли.
SetCursorPosition (Left, Top)	Устанавливает положение курсора (*Lef* -позиция колонки курсора, *Top* -позиция строки курсора). Метод определяет позицию курсора для следующей операция записи в окно консоли. Если заданная позиция курсора находится вне области, которая в данный момент видима в окне консоли, начало координат окна консоли автоматически изменяется, чтобы курсор стал видимым.
ResetColor	Устанавливает для цветов фона и текста консоли их значения по умолчанию.

Применение свойств и методов класса Console

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Окно_консоли
{
   class Program
   {
      static int origWidth = Console.WindowWidth, width = origWidth;
      static int origHeight = Console.WindowHeight, height = origHeight;
      static string s1 = "Размеры окна консоли. Ширина: {0}, высота: {1}.";
      static ConsoleKeyInfo cki;
      static void WriteConsole()
      {
         Console.SetCursorPosition(0, 0);
     Console.WriteLine(s1, Console.WindowWidth, Console.WindowHeight);
         Console.SetCursorPosition(1, Console.WindowHeight - 2);
         Console.Write("Нажмите любую клавишу...\n");
         Console.Write("Нажмите клавишу Escape (Esc) для выхода");
Console.SetCursorPosition((Console.WindowWidth-22)/2,
    Console.WindowHeight/2);
```

```
            Console.Write("Вы нажали клавиши: ");
            Console.TreatControlCAsInput = true;
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.ForegroundColor = ConsoleColor.White;
        }
        static void Main(string[] args)
        {
            Console.CursorVisible=false;
            WriteConsole();
            do
            {
                width = width - 2;
                height = height - 2;
                if ((width <= 25) | (height <= 10))
                {
                    width = origWidth;              height = origHeight;
                }
                Console.SetWindowSize(width, height);
                Console.Clear();
                WriteConsole();
if ((cki.Modifiers & ConsoleModifiers.Alt) != 0)    Console.Write("ALT+");
if ((cki.Modifiers & ConsoleModifiers.Shift) != 0) Console.Write("SHIFT+");
 if ((cki.Modifiers & ConsoleModifiers.Control) != 0) Console.Write("CTL+");
                Console.WriteLine(cki.Key.ToString());
                cki = Console.ReadKey();
            } while (cki.Key != ConsoleKey.Escape);
        }
    }
}
```

Результат выполнения:

```
Размеры окна консоли. Ширина: 74, высота: 19.
                    Вы нажали клавиши: SHIFT+CTL+K
Нажмите любую клавишу...
Нажмите Escape (Esc) для выхода
```

В примере используются 3 перечисления и 1 структура:

ConsoleKey - перечисление, задающее значения стандартных клавиш консоли (**ESC**: значение Escape **END**: значение End; **СТРЕЛКА ВЛЕВО:** значение LeftArrow; **F3**: значение F3). Перечисление **ConsoleKey** используется в структуре System.ConsoleKeyInfo (возвращаемой методом Console.ReadKey) для получения значения нажатой клавиши.

ConsoleModifiers - перечисление, задающее значения клавиш SHIFT, ALT и CTRL.

ConsoleColor - перечисление, задающее значения констант цветов (от Black до White) для фона и текста консоли.

ConsoleKeyInfo - структура, описывающая клавишу, нажатую на консоли (включая символ, представленный этой клавишей, и состояние управляющих клавиш CTRL, SHIFT и ALT). Свойство KeyChar - возвращает символ нажатой клавиши. Свойство Modifiers - возвращает сочетание значений из перечисления System.ConsoleModifiers, указывающее, были ли одновременно с клавишей консоли нажаты управляющие клавиши SHIFT, ALT или CTRL.

Класс Convert, методы Parse, TryParse

Для преобразования строковых данных введенных с клавиатуры используется статический класс System.Convert, выполняет преобразование строковых типов в заданный встроенный тип.

Convert.To <тип> (string s), где <тип> - это название системного типа CLR, в который выполняется преобразование.

string ss; ss = Console.ReadLine(); int a = Convert.ToInt32(ss);

Для преобразования строкового представления любого числа используется методы **Parse()** и **TryParse()**, которые реализованы для всех числовых типов данных. Если соответствующее преобразование выполнить невозможно, то метод **Parse** генерирует исключение *System.FormatExeption* (входная строка имеет недопустимый формат и выполнение программы прекращается), а метод **TryParse** не генерирует исключение, а возвращает значение "false".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Пример_Convert_Parse
{  class Program
   {
      static void Main(string[] args)
      {
         string s = Console.ReadLine(); // ввод строки
```

```
   int i1 = Convert.ToInt32(Console.ReadLine()); // преобразование в целое
   Int64 i2 = Int64.Parse(Console.ReadLine()); // преобразование в целое
double x1=Convert.ToDouble(Console.ReadLine());//преобразование в вещ.
   double Res;
   double x2 = (double.TryParse(Console.ReadLine(), out Res) ==
true)?Res:double.NaN;      // преобразование в вещественное
    Console.WriteLine("Строка s: {0}\n" +  "Число целое i1 (Convert): {1}\n" +
            "Число целое i2 (Parse): {2}\n" +
            "Число вещественное x1 (Convert): {3}\n" +
            "Число вещественное x2 (TryParse): {4}\n",s,i1,i2,x1,x2);
         Console.ReadLine();
      }
   }
}
```

Результат выполнения:<Пример строки>

```
12345
123456789012
1234567,1234
1234567890,12e+298
Строка s: <Пример строки>
Число целое i1: 12345
Число целое i2: 123456789012
Число вещественное x1: 1234567,1234
Число вещественное x2: 1,23456789012E+307
```

Форматирование строк

Методы Write(), WriteLine() с форматированием. *В классе* Console *содержатся разные перегруженные варианты методов* Write() *и* WriteLine(). *Наиболее используемыми являются методы, выполняющие форматирование вывода строки с заголовком:*

public static string WriteLine (String, Object №0, Object №1, ... , Object №N);

WriteLine (["***строка форматирования***" ,] **список вывода {*arg0, ... , argN*}**);

При выводе нескольких элементов использование строки форматирования обязательно. *В качестве элементов списка вывода могут фигурировать имена переменных*, константы или выражения (значения которых перед выводом должны быть вычислены).

Console.WriteLine("s1={0}, s2={1}", s1, s2);); При вызове метода в качестве первого параметра типа **string** передается строка, которая задает формат вывода . Форматирующая строка помимо текста (который выводится без изменений), также может содержать *спецификации*, заключенные в фигурные скобки ("x={0}"). Число *спецификаций* форматирующей строки должно соответствовать числу элементов списка вывода.

Спецификация задает форматирование вывода значения переменной, на которую она ссылается (значение после преобразования его в строку, будет выведено на экран вместо *спецификации)*. Формат спецификации имеют следующий общий вид:

{N [, Width [:<коды_форматирования>]]}, где

1-й (обязательный) параметр **N** – индекс объекта, который обрабатывается спецификацией (индексация объектов начинается с 0).

2-й (необязательный) параметр **Width** (целое число со знаком) - определяет минимальную ширину поля, которое отводится строке, вставляемой вместо *спецификации.* Если значение *Width* меньше длины форматируемой строки, то *Width* игнорируется, и в качестве значения ширины поля используется длина форматируемой строки.

Форматируемые данные выравниваются в поле по правому краю, если *Width* имеет положительное значение, или по левому краю, если *Width* имеет отрицательное значение.

При необходимости отформатированная строка дополняется пробелами.

3-й (необязательный) параметр **коды_форматирования** - задает шаблон вывода (***спецификатор*** форматирования) значения переменной .

Для примеров используются числа:

Int32 d4 = 123456789;

Double d1=1.2345678, d2=5.6789E10, d3=9.87654E-6;

Спецификаторы стандартных числовых форматов

Спецификатор	Описание	Пример использования
C (или c) Currency (валюта) Денежный Поддерживается: всеми числовыми типами данных	Значение валюты (форматирование денежных значений). Число после буквы **"C"** (**c**) означает число знаков дробной части.	("{0:C}",d1)===>1,23р. ("{0:C0}",d1)===>1р. ("{0:C5}",d1)===>1,23457р. ("{0:C10}",d1)=>1,2345678000р. ("{0:C}",d2)===>56 789 000 000,00р. ("{0:C0}",d2)===>56 789 000 000р. ("{0:C5}",d2)===>56 789 000 000,00000р. ("{0:C10}",d2)=>56 789 000 000,0000000000р ("{0:C}",d3)===>0,00р. ("{0:C0}",d3)===>0р. ("{0:C5}",d3)===>0,00001р. ("{0:C10}",d3)=>0,0000098765р. ("{0:C}",d4)===>123 456 789,00р. ("{0:C0}",d4)===>123 456 789р. ("{0:C5}",d4)===>123 456 789,00000р. ("{0:C10}",d4)=>123 456 789,0000000000р.

Спецификатор	Описание	Пример использования
D (или d) Decimal (десятичное число) Поддерживаетсяцелочисленными типами данных	Цифры с необязательным отрицательным знаком. Число преобразуется в строку, состоящую из цифр (0-9); если число отрицательное, перед ним ставится знак "-". Число после буквы **"D"** (**d**) задает мин. Кол-во знаков в выходной строке. Недостающие знаки заменяются нулями.	("{0:D}",d4)===>123456789 ("{0:D0}",d4)==>123456789 ("{0:D5}",d4)==>123456789 ("{0:D10}",d4)=>0123456789
E (или e) Exponential (scientific) экспоненциальный (научный) Экспоненциальное представление чисел с использованием букв **E/e** Поддерживается всеми числовыми типами данных	Экспоненциальная нотация. Число преобразуется в строку вида "-d.ddd..E+ddd" или "-d.ddd..e+ddd", где "d" - цифра (0-9). Если число отрицательное, в начале строки появляется знак "-". Перед разделителем целой и дробной части всегда стоит один знак. Регистр спецификатора формата задает регистр буквы, стоящей перед экспонентой ("E" или "e"). Экспонента состоит из знака "+" или "-" и 3 цифр. Недостающие до минимума цифры заменяются 0, если это необходимо. Число после буквы **"E"** (**e**) означает число знаков дробной части. По умолчанию число знаков дробной части равно 6.	("{0:E}",d1)===>1,234568E+000 ("{0:E0}",d1)==>1E+000 ("{0:E5}",d1)==>1,23457E+000 ("{0:E10}",d1)=>1,2345678000E+000 ("{0:E}",d2)===>5,678900E+010 ("{0:E0}",d2)==>6E+010 ("{0:E5}",d2)==>5,67890E+010 ("{0:E10}",d2)=>5,6789000000E+010 ("{0:E}",d3)===>9,876540E-006 ("{0:E0}",d3)==>1E-005 ("{0:E5}",d3)==>9,87654E-006 ("{0:E10}",d3)=>9,8765400000E-006 ("{0:E}",d4)===>1,234568E+008 ("{0:E0}",d4)==>1E+008 ("{0:E5}",d4)==>1,23457E+008 ("{0:E10}",d4)=>1,2345678900E+008
F (или f) Fixed-point (фиксированная запятая) Поддерживается: всеми числовыми типами данных	Цифры целой и дробной частей с необязательным отрицательным знаком. Число преобразуется в строку вида "-ddd.ddd…", где знак 'd' - цифра (0-9). Если число отрицательное, в начале строки появляется знак "минус". Число после буквы **"F"** (**f**) означает число знаков дробной части.	("{0:F}",d1)===>1,23 ("{0:F0}",d1)==>1 ("{0:F5}",d1)==>1,23457 ("{0:F10}",d1)=>1,2345678000 ("{0:F}",d2)===>56789000000,00 ("{0:F0}",d2)==>56789000000 ("{0:F5}",d2)==>56789000000,00000 ("{0:F10}",d2)=>56789000000,0000000000 ("{0:F}",d3)===>0,00 ("{0:F0}",d3)==>0 ("{0:F5}",d3)==>0,00001 ("{0:F10}",d3)=>0,0000098765 ("{0:F}",d4)===>123456789,00 ("{0:F0}",d4)==>123456789 ("{0:F5}",d4)==>123456789,00000 ("{0:F10}",d4)=>123456789,0000000000

Спецификатор	Описание	Пример использования
G (или g) General (default) (Общий) Поддерживается: всеми числовыми типами данных	Число преобразуется в наиболее короткую запись из записи с фиксированной запятой или экспоненциальной записи, в зависимости от типа числа и наличия спецификатора точности. Число после буквы **"G"** (**g**) означает число знаков дробной части. Если спецификатор точности не задан, точность задается типом переменной: Byte или SByte: 3 Int16 или UInt16: 5 Int32 или UInt32: 10 Int64 или UInt64: 19 Single: 7 Double: 15 Decimal: 29.	("{0:G}",d1)===>1,2345678 ("{0:G0}",d1)==>1,2345678 ("{0:G5}",d1)==>1,2346 ("{0:G10}",d1)=>1,2345678 ("{0:G}",d2)===>56789000000 ("{0:G0}",d2)==>56789000000 ("{0:G5}",d2)==>5,6789E+10 ("{0:G10}",d2)=>5,6789E+10 ("{0:G}",d3)===>9,87654E-06 ("{0:G0}",d3)==>9,87654E-06 ("{0:G5}",d3)==>9,8765E-06 ("{0:G10}",d3)=>9,87654E-06 ("{0:G}",d4)===>123456789 ("{0:G0}",d4)==>123456789 ("{0:G5}",d4)==>1,2346E+08 ("{0:G10}",d4)=>123456789
N (или n) Number (число) Поддерживается: всеми числовыми типами данных	Цифры целой и дробной частей, разделители групп и разделитель целой и дробной частей с необязательным отрицательным знаком. Число преобразуется в строку вида "-d,ddd,ddd.ddd…", где знак '-' при необходимости представляет знак 'минус', знак 'd' - цифра (0-9), знак ',' - разделитель тысяч, а знак '.' - разделитель целой и дробной части. Число после буквы **"N"** (**n**) означает число знаков дробной части.	("{0:N}",d1)===>1,23 ("{0:N0}",d1)==>1 ("{0:N5}",d1)==>1,23457 ("{0:N10}",d1)=>1,2345678000 ("{0:N}",d2)===>56 789 000 000,00 ("{0:N0}",d2)==>56 789 000 000 ("{0:N5}",d2)==>56 789 000 000,00000 ("{0:N10}",d2)=>56 789 000 000,0000000000 ("{0:N}",d3)===>0,00 ("{0:N0}",d3)==>0 ("{0:N5}",d3)==>0,00001 ("{0:N10}",d3)=>0,0000098765 ("{0:N}",d4)===>123 456 789,00 ("{0:N0}",d4)==>123 456 789 ("{0:N5}",d4)==>123 456 789,00000 ("{0:N10}",d4)=>123 456 789,0000000000
P (или p) Percent (Процент) Поддерживается: всеми числовыми типами данных	Число, умноженное на 100 отображается с символом процента. Число после буквы **"P"** (**p**) означает число знаков дробной части.	("{0:P}",d1)===>123,46% ("{0:P0}",d1)==>123% ("{0:P5}",d1)==>123,45678% ("{0:P10}",d1)=>123,4567800000% ("{0:P}",d2)===>5 678 900 000 000,00% ("{0:P0}",d2)==>5 678 900 000 000% ("{0:P5}",d2)==>5 678 900 000 000,00000%

Спецификатор	Описание	Пример использования
		("{0:P10}",d2)=>5 678 900 000 000,0000000000% ("{0:P}",d3)===>0,00% ("{0:P0}",d3)==>0% ("{0:P5}",d3)==>0,00099% ("{0:P10}",d3)=>0,0009876540% ("{0:P}", d4)===>12 345 678 900,00% ("{0:P0}",d4)==>12 345 678 900% ("{0:P5}",d4)==>12 345 678 900,00000% ("{0:P10}",d4)=>12 345 678 900,0000000000%
X (или x) Fixed-point (Шестнадцатеричный) Поддерживается: только целочисленными типами данных	Число преобразуется в строку шестнадцатеричных чисел. Регистр шестнадцатеричных чисел, превосходящих 9, совпадает с регистром указателя формата.	("{0:X}",d4)===>X ("{0:X0}",d4)==>X123456789 ("{0:X5}",d4)==>X5 ("{0:X10}",d4)=>X1123456789 Для получения вида "ABCDEF" используется буква "X" ("x" для получения "abcdef"). Число после буквы "X" (x) означает минимальное количество знаков в выходной строке. Недостающие знаки заменяются 0/

Для определения пользовательского способа форматирования числовых данных, необходимо создать строку настраиваемого числового формата, состоящую из одного или нескольких спецификаторов (описателей) настраиваемого формата. Строка настраиваемого числового формата - любая строка формата, не являющаяся строкой стандартного числового формата. Строки настраиваемых числовых форматов используются в методе **ToString** всех числовых типов .

Спецификаторы пользовательских числовых форматов

Спецификатор	Описание	Пример использования
"0" Знак-заместитель нуля	Заменяет ноль соответствующей цифрой, если цифра есть в выходном наборе (в противном случае в результирующей строке будет стоять 0). Позиция крайнего левого "0" до десятичной точки и позиция крайнего правого "0" после десятичной точки определяют диапазон цифр, которые всегда включаются в выходную строку. Спецификатор "0…" приводит к округлению значения до ближайшего значения цифры, предшествующей десятичной точке-разделителю, если назначено использование округления от 0.	("{0:0}",d1)==========>1 ("{0:00}",d1)==========>01 ("{0:00000}",d1) ======>00001 ("{0:0000000000}",d1)=>0000000001 ("{0:0}",d2)==========>56789000000 ("{0:00}",d2)==========>56789000000 ("{0:00000}",d2) ======>56789000000 ("{0:0000000000}",d2)=>56789000000 ("{0:0}",d3)==========>0 ("{0:00}",d3)==========>00 ("{0:00000}",d3) ======>00000 ("{0:0000000000}",d3)=>0000000000 ("{0:0}",d4)==========>123456789 ("{0:00}",d4)==========>123456789 ("{0:00000}",d4) ======>123456789 ("{0:0000000000}",d4)=>0123456789

Спецификатор	Описание	Пример использования
"#" Заместитель цифры	Заменяет знак "#" соответствующей цифрой, если цифра есть в выходном наборе (в противном случае в результирующей строке цифра стоять не будет). Ноль не будет отображен, если он не является значащей цифрой, даже если это единственный знак строки. Ноль отображается, только если он является значащей цифрой форматируемого значения. Формат "#..." приводит к округлению значения до ближайшего значения цифры, предшествующей десятичному разделителю, если назначено использование округления от 0.	("{0:#}",d1)=========>1 ("{0:##}",d1)=========>1 ("{0:#####}",d1) =====>1 ("{0:##########}",d1)=>1 ("{0:#}",d2)=========>56789000000 ("{0:##}",d2)=========>56789000000 ("{0:#####}",d2) =====>56789000000 ("{0:##########}",d2)=>56789000000 ("{0:#}",d3)=========> ("{0:##}",d3)=========> ("{0:#####}",d3) =====> ("{0:##########}",d3)=> ("{0:#}",d4)=========>123456789 ("{0:##}",d4)=========>123456789 ("{0:#####}",d4) =====>123456789 ("{0:##########}",d4)=>123456789
"." Разделитель	Определяет расположение разделителя целой и дробной частей в результирующей строке. Первый знак "." в строке формата определяет положение разделителя целой и дробной частей форматированного значения, дополнительные знаки "." игнорируются.	("{0:0.0}",d1)=========>1,2 ("{0:#.#}",d1)=========>1,2 ("{0:#0.0#}",d1)=======>1,23 ("{0:#0#0#.#0#0#}",d1)=>0001,23457 ("{0:0.0}",d2)=========>56789000000,0 ("{0:#.#}",d2)=========>56789000000 ("{0:#0.0#}",d2)=======>56789000000,0 ("{0:#0#0#.#0#0#}",d2)=>56789000000,0000 ("{0:0.0}",d3)=========>0,0 ("{0:#.#}",d3)=========> ("{0:#0.0#}",d3)=======>0,0 ("{0:#0#0#.#0#0#}",d3)=>0000,00001 ("{0:0.0}",d4)=========>123456789,0 ("{0:#.#}",d4)=========>123456789 ("{0:#0.0#}",d4)=======>123456789,0 ("{0:#0#0#.#0#0#}",d4)=>123456789,0000
"," Разделитель числовых разрядов и масштабирование чисел	Служит в качестве описателя разделителя групп и описателя масштабирования чисел. В качестве разделителя групп вставляет локализованный символ-разделитель групп между всеми группами. Спецификатор разделителя числовых разрядов: Если один или несколько символов "," указаны между двумя заместителями цифр (0 или #), которые форматируют целые разряды	("{0:0,0}",d1)=========>01 ("{0:#,#}",d1)=========>1 ("{0:#0,0#}",d1)=======>001 ("{0:#0#,0#,#0#0#}",d1)=>000 000 001 ("{0:0,0}",d2)=========>56 789 000 000 ("{0:#,#}",d2)=========>56 789 000 000 ("{0:#0,0#}",d2)=======>56 789 000 000 ("{0:#0#,0#,#0#0#}",d2)=>56 789 000 000 ("{0:0,0}",d3)=========>00 ("{0:#,#}",d3)=========> ("{0:#0,0#}",d3)=======>000 ("{0:#0#,0#,#0#0#}",d3)=>000 000 000 ("{0:0,0}",d4)=========>123 456 789

Спецификатор	Описание	Пример использования
	числа, то символ разделителя групп вставляется между каждой группой числа в составной части выходных данных. Если строка "#,#" и язык и региональные параметры используются для форматирования числа 1000, то результатом является "1,000".	("{0:#,#}",d4)==========>123 456 789 ("{0:#0,0#}",d4)==========>123 456 789 ("{0:#0#,0#,#0#0#}",d4)=>123 456 789 В качестве описателя масштабирования чисел делит число на 1000 для всех указанных запятых. Если один или несколько символов "," указаны непосредственно слева от явной или неявной десятичной точки, то форматируемое число делится на 1000 каждый раз, когда встречается спецификатор масштабирования числа. Например, если строка "0,," используется для форматирования числа 100 миллионов, то результатом является "100".
"%" Заместитель процентов	Умножает число на 100 и вставляет локализованный символ процента в результирующую строку.	("{0:0,0%}",d1)==========>123% ("{0:#,#%}",d1)==========>123% ("{0:#0,0#%}",d1)==========>123% ("{0:#0#,0#,#0#0#%}",d1)=>000 000 123% ("{0:0,0%}",d2)==========>5 678 900 000 000% ("{0:#,#%}",d2)==========>5 678 900 000 000% ("{0:#0,0#%}",d2)==========>5 678 900 000 000% ("{0:#0#,0#,#0#0#%}",d2)=>5 678 900 000 000% ("{0:0,0%}",d3)==========>00% ("{0:#,#%}",d3)==========>% ("{0:#0,0#%}",d3)==========>000% ("{0:#0#,0#,#0#0#%}",d3)=>000 000 000% ("{0:0,0%}",d4)==========>12 345 678 900% ("{0:#,#%}",d4)==========>12 345 678 900% ("{0:#0,0#%}",d4)==========>12 345 678 900% ("{0:#0#,0#,#0#0#%}",d4)=>12 345 678 900%
"‰" Местозаполнитель промилле Используемый знак промилле определяется значением свойства NumberFormatInfo.PerMilleSymbol объекта, содержащего сведения о форматировании для заданного языка и региональных параметров.	Умножает число на 1000 и вставляет локализованный символ промилле в результирующую строку. При использовании в строке формата знака "‰" (\u2030) перед форматированием число будет умножено на 1000. В соответствующую позицию возвращаемой строки будет вставлен знак промилле.	("{0:0,0‰}",d1)==========>1 235‰ ("{0:#,#‰}",d1)==========>1 235‰ ("{0:#0,0#‰}",d1)==========>1 235‰ ("{0:#0#,0#,#0#0#‰}",d1)=>000 001 235‰ ("{0:0,0‰}",d2)==========>56 789 000 000 000‰ ("{0:#,#‰}",d2)==========>56 789 000 000 000‰ ("{0:#0,0#‰}",d2)==========>56 789 000 000 000‰ ("{0:#0#,0#,#0#0#‰}",d2)=>56 789 000 000 000‰ ("{0:0,0‰}",d3)==========>00‰ ("{0:#,#‰}",d3)==========>‰ ("{0:#0,0#‰}",d3)==========>000‰ ("{0:#0#,0#,#0#0#‰}",d3)=>000 000 000‰ ("{0:0,0‰}",d4)==========>123 456 789 000‰ ("{0:#,#‰}",d4)==========>123 456 789 000‰ ("{0:#0,0#‰}",d4)==========>123 456 789 000‰ ("{0:#0#,0#,#0#0#‰}",d4)=>123 456 789 000‰

Спецификатор	Описание	Пример использования
"E0" "E+0" "E-0" "e0" "e+0" "e-0" Экспоненциальная нотация	Если в строке формата присутствует один из знаков "E", "E+", "E-", "e", "e+" или "e-", за которым следует по крайней мере одна цифра "0", число представляется в экспоненциальной форме; между числом и экспонентой вставляется знак "E" или "e". Минимальная длина экспоненты при выводе определяется количеством нулей, расположенных за знаком формата. Знаки "E+" и "e+" устанавливают обязательное отображение знака "плюс" или "минус" перед экспонентой.	("{0:0.###E+0}",d1)=====>1,235E+0 ("{0:#,#0.###E+000}",d1)=>123,457E-002 ("{0:#000.###E-00}",d1)=>1234,568E-03 ("{0:#0#,00.###e00}",d1)=>12 345,678e-04 ("{0:0.###E+0}",d2)=====>5,679E+10 ("{0:#,#0.###E+000}",d2)=>567,89E+008 ("{0:#000.###E-00}",d2)=>5678,9E07 ("{0:#0#,00.###e00}",d2)=>56 789e06 ("{0:0.###E+0}",d3)=====>9,877E-6 ("{0:#,#0.###E+000}",d3)=>987,654E-008 ("{0:#000.###E-00}",d3)=>9876,54E-09 ("{0:#0#,00.###e00}",d3)=>98 765,4e-10 ("{0:0.###E+0}",d4)=====>1,235E+8 ("{0:#,#0.###E+000}",d4)=>123,457E+006 ("{0:#000.###E-00}",d4)=>1234,568E05 ("{0:#0#,00.###e00}",d4)=>12 345,679e04 Знаки "E", "e", "E-" и "e-" устанавливают отображение знака только для отрицательных чисел
\ Escape - символ	В C# и C++ символ, следующий в строке формата за обратной косой чертой, воспринимается как escape-последовательность. Этот символ используется с обычными последовательностями форматирования (например, \n — новая строка). Чтобы отобразить обратную косую черту, необходимо использовать строку с "\\".	("{0:\#\# ##0 руб\. \0\0 коп\. \#\#}",d1)=>## 1 руб. 00 коп. ## ("{0:\\ ##0 руб\. 00 коп\. \\}",d1)=====>\ 0 руб. 01 коп. \ ("{0:\#\# ##0 руб\. \0\0 коп\. \#\#}",d2)=>## 56789000000 руб. 00 коп. ## ("{0:\\ ##0 руб\. 00 коп\. \\}",d2)=====>\ 567890000 руб. 00 коп. \ ("{0:\#\# ##0 руб\. \0\0 коп\. \#\#}",d3)=>## 0 руб. 00 коп. ## ("{0:\\ ##0 руб\. 00 коп\. \\}",d3)=====>\ 0 руб. 00 коп. \ ("{0:\#\# ##0 руб\. \0\0 коп\. \#\#}",d4)=>## 123456789 руб. 00 коп. ## ("{0:\\ ##0 руб\. 00 коп\. \\}",d4)=====>\ 1234567 руб. 89 коп. \
'строка' Разделитель строк-литералов	Указывает на то, что заключенные в разделители символы должны быть скопированы в результирующую строку без изменений. Символы, заключенные в одинарные или двойные кавычки, копируются в выходную строку без форматирования.	("{0:'Зн-'0.0}",d1)=======>Зн-1,2 ("{0:'Зн-'#.#}",d1)=======>Зн-1,2 ("{0:'Зн-'#0.0#}",d1)======>Зн-1,23 ("{0:'Зн-'#0#0#.#0#0#}",d1)=>Зн-0001,23457 ("{0:'Зн-'0.0}",d2)=======>Зн-56789000000,0 ("{0:'Зн-'#.#}",d2)=======>Зн-56789000000 ("{0:'Зн-'#0.0#}",d2)======>Зн-56789000000,0 ("{0:'Зн-'#0#0#.#0#0#}",d2)=>Зн-56789000000,0000 ("{0:'Зн-'0.0}",d3)=======>Зн-0,0 ("{0:'Зн-'#.#}",d3)=======>Зн- ("{0:'Зн-'#0.0#}",d3)======>Зн-0,0

Спецификатор	Описание	Пример использования
		("{0:'Зн-'#0#0#.#0#0#}",d3)=>Зн-0000,00001 ("{0:'Зн-'0.0}",d4)========>Зн-123456789,0 ("{0:'Зн-'#.#}",d4)========>Зн-123456789 ("{0:'Зн-'#0.0#}",d4)=======>Зн-123456789,0 ("{0:'Зн-'#0#0#.#0#0#}",d4)=>Зн-123456789,0000
; Разделитель секций	Определяет секции с раздельными строками формата для положительных чисел, отрицательных чисел и нуля. Знак ";" служит для разделения секций положительных, отрицательных и нулевых чисел в строке формата.	("{0:<Полож\.= ##>;<##>}",d1)=><Полож.= 1> ("{0:##;[##];[**Ноль**]}",d1) ==>1 ("{0:<Полож\.= ##>;<##>}",d2)=><Полож.= 56789000000> ("{0:##;[##];[**Ноль**]}",d2) ==>56789000000 ("{0:<Полож\.= ##>;<##>}",d3)=><Полож.= > ("{0:##;[##];[**Ноль**]}",d3) ==>[**Ноль**] ("{0:<Полож\.= ##>;<##>}",d4)=><Полож.= 123456789> ("{0:##;[##];[**Ноль**]}",d4)==>123456789 Если в строке пользовательского формата две секции, то крайняя левая секция определяет форматирование положительных чисел и нулей, а крайняя правая – форматирование отрицательных чисел.
другой Все остальные символы	Символ копируется в результирующую строку без изменений. Любой другой символ копируется в выходную строку и не влияет на форматирование.	("{0:Значение-0.0}",d1)=>Значение-1,2 ("{0:Значение-0.0}",d2)=>Значение-56789000000,0 ("{0:Значение-0.0}",d3)=>Значение-0,0 ("{0:Значение-0.0}",d4)=>Значение-123456789,0

Применение стандартных и пользовательских числовых форматов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Globalization;
using System.Threading;
namespace Примеры_форматирования
{   class Program
    {  static void Числовые_форматы()
{CultureInfo FrCulture = new CultureInfo("ru-RU"); //Для вывода в качестве
FrCulture.NumberFormat.CurrencyGroupSeparator = " "; //денеж.един. - р.
Thread.CurrentThread.CurrentCulture = FrCulture;//раздел. тысяч - пробел
        Console.WriteLine("Стандартные числовые форматы");
```

```
        Console.WriteLine(  //Использование метода WriteLine для формат.
            "(C) Денежный:_____{0:C}\n" +
            "(D) Целочисленный_____{0:D}\n" +
            "(E) Научный:_____{1:E}\n" +
            "(F) Фиксированная точка_____{1:F}\n" +
            "(G) Общий:_____{0:G}\n" +
            "(N) Число:_____{0:N,}\n"+
            "(P) Процент:_____{1:P}\n" +
            "(X) Шестнадцатеричный:_____{0:X}\n", -123456789, -123.45f);
        Int32 v1 = 1204567890;     double v2 = 103.23456;
    Console.WriteLine(  //Использование метода ToString() для формат.
          "v1 = " + v1.ToString("c6") + "\n" +   "v2 = " + v2.ToString("E4"));
    Console.WriteLine(  //Использование метода string.Format() для формат.
"v1 = "+string.Format("{0:N4}", v1)+"\n"+ "v2 ="+string.Format("{0:F3}",v2));
string s1 =string.Format("v1= {0:F5}", v1);//Использование строки формат.
string s2 =string.Format("\nv2={0:e2}",v2);//Использование строки формат.
        Console.WriteLine(s1 + s2); //Использование строки формат.
        Console.WriteLine("Пользовательские  числовые форматы");
        Console.WriteLine(  //Использование метода WriteLine для формат.
            "(000.0000)_____{0:000.0000}\n" +
            "(####.####)_____{0:####.##}\n" +
            "(#...)_____{0:#...}\n" +
            "(##,##.##)_____{0:##,##.##}\n" +
            "(##,##.###%)_____{0:##,##.###%}\n" +
            "(0:##;[##];[**Ноль**]_____{0:0:##;[##];[**Ноль**]}\n" +
            "(0:##;[##];[**Ноль**]_____{1:0:##;[##];[**Ноль**]}\n" +
            "(0:##;[##];[**Ноль**]_____{2:0:##;[##];[**Ноль**]}\n",+
            -120456789.123, 103.45f,0);
}
    static void Main(string[] args)
    {  Числовые_форматы();
        Console.ReadLine();
     }
  }
}
```

Результат выполнения:

Стандартные числовые форматы
(C) Денежный:_____-123 456 789,00р.
(D) Целочисленный_____-123456789
(E) Научный:_____-1,234500E+002

```
(F) Фиксированная точка_____-123,45
(G) Общий:_____-123456789
(N) Число:_____-123 456 789,00
(P) Процент:_____-12 345,00%
(X) Шестнадцатеричный:_____F8A432EB
v1 = 1 204 567 890,000000р.
v2 = 1,0323E+002
v1 = 1 204 567 890,0000
v2 = 103,235
v1= 1204567890,00000
v2= 1,03e+002
Пользовательские числовые форматы
(000.0000)_____-120456789,1230
(####.####)_____-120456789,12
(#...)_____-120456789
(##,##.##)_____-120 456 789,12
(##,##.###%)_____-12 045 678 912,3%
(0:##;[##];[**Ноль**]_____[120456789]
(0:##;[##];[**Ноль**]_____1:03
(0:##;[##];[**Ноль**]_____[**Ноль**]
```

Классы Math, Random

Класс **System.Math** библиотеки FCL предоставляет константы и статические методы для тригонометрических, логарифмических и других математических функций. Для того, чтобы в программе на C# использовать математические функции, необходимо подключить пространство имён **System**, а при вызове метода, реализующего математическую функцию явно указывать название класса **Math**.

Основные поля и методы класса Math

Название	Описание
E	Константа, равная значению натурального числа **e** = 2,71828182845905
PI	Константа, равная значению натурального числа π = 3,14159265358979
	Тригонометрические функции: Sin, Cos, Tan
Sin(x)	Функция синуса (возвращает синус указанного угла). Аргумент задается в радианах
Cos(x)	Функция косинуса (возвращает косинус указанного угла). Аргумент задается в радианах
Tan(x)	Функция тангенса (возвращает тангенс указанного угла). Аргумент задается в радианах

Название	Описание
Обратные тригонометрические функции: ASin, ACos, ATan, ATan2	
Asin(x)	Функция арксинуса (возвращает угол, синус которого равен указанному числу). Значение аргумента должно находиться в диапазоне от -1 до +1
Acos(x)	Функция арккосинуса (возвращает угол, косинус которого равен указанному числу). Значение аргумента должно находиться в диапазоне от -1 до +1
Atan(x)	Функция арктангенса (возвращает угол, тангенс которого равен указанному числу.). Аргумент (угол) задается в радианах
Atan2(x,y)	Функция арктангенса (возвращает угол, тангенс которого равен отношению двух указанных чисел).
Гиперболические функции: Sinh, Cosh, Tanh	
Sinh(x)	Функция гиперболического синуса (возвращает гиперболический синус указанного угла). Аргумент задается в радианах
Cosh(x)	Функция гиперболического косинуса (возвращает гиперболический косинус указанного угла). Аргумент задается в радианах
Tanh(x)	Функция гиперболического тангенса (возвращает гиперболический тангенс указанного угла). Аргумент задается в радианах
Экспонента и логарифмические функции: Exp, Log, Log10	
Exp(x)	Вычисляет значение e^X (экспоненциальная функция)
Log(x)	Возвращает значение натурального логарифма (с основанием **e**) - $\log_E x$
Log10(x)	Возвращает значение десятичного логарифма - $\log_{10} x$
Log(x,y)	Возвращает значение логарифма числа **x** (с основанием **y**) - $\log_Y x$
Модуль, корень, знак: Abs, Sqrt, Sign	
Abs(x)	Вычисляет модуль (абсолютное значение) числа **x**. Перегружен для всех числовых типов (**int**, **double** и т.д.)
Sqrt(x)	Возвращает положительное значение квадратного корня - $\sqrt{x}$
Sign(x)	Возвращает значение, определяющее знак числа x. Перегружен для всех числовых типов (**int**, **double** и т.д.)
Функции округления - Ceiling, Floor, Round, Truncate	
Ceiling(x)	Возвращает наименьшее целое число, которое больше или равно заданному числу **x**. Перегружен для числовых типов (**decimal**, **double**)
Floor(x)	Возвращает наибольшее целое число, которое меньше или равно указанному числу **x**. Перегружен для числовых типов (**decimal**, **double**)
Round(x)	Округляет десятичное значение **x** до ближайшего целого. Перегружен для числовых типов (**decimal**, **double**)
Round(x,y)	Округляет десятичное значение **x** до указанного числа дробных разрядов **y**. Перегружен для числовых типов (**decimal**, **double**)
Round(x,y)	Округляет десятичное значение **x** до ближайшего целого. Параметр **y** задает правило округления значения, если оно находится ровно посредине между двумя другими числами. Перегружен для числовых типов (**decimal**, **double**)
Round(x,y,z)	Округляет десятичное значение **x** до указанного числа дробных разрядов **y**. Параметр задает **z** правило округления значения, если оно находится ровно посредине между двумя другими числами. Перегружен для числовых типов (**decimal**, **double**)

Название	Описание
Truncate(x)	Вычисляет целую часть заданного числа x. Перегружен для числовых типов (**decimal**, **double**)
Минимум, максимум, степень, остаток - Min, Max, Pow, IEEERemainder	
Min(x, y)	Возвращает минимум из двух чисел **x** и **y**
Max(x, y)	Возвращает максимум из двух чисел **x** и **y**
Pow(x, y)	Возвращает значение $\mathbf{x^y}$ (возводит число **x** в степень **y**)
IEEERemainder(x,y)	Возвращает остаток от деления числа **x** на число **y**. Число, равное $x - (y*Q)$, где Q является частным x / y, округленным до ближайшего целого числа Если x / y находится на равном расстоянии от двух целых чисел, выбирается четное число. Если значение $x - (y*Q)$ равно нулю, возвращается значение +0 при положительном x, или значение -0 при отрицательном x. Если значение параметра y равно 0, возвращается значение NaN.
BigMul(x,y)	Умножает два числа **x** и **y**. Перегружен для числовых типов (**int32**, **int64**)
DivRem(x,y,z)	Вычисляет частное двух целых чисел **x** , **y** и возвращает остаток в выходном параметре **z**. Перегружен для числовых типов (**int32**, **int64**)

Генерация случайных чисел требуется во многих приложениях. Класс **System.Random** - класс для формирования псевдослучайных чисел, выбираемых с одинаковой вероятностью из заданного множества чисел.

Название	Описание
Конструкторы	
Random()	Инициализирует новый экземпляр класса Random с помощью зависимого от времени начального значения по умолчанию. Созданные числа распределены равномерно; все числа возвращаются с одинаковой вероятностью. Начальное значение по умолчанию извлекается из значения системных часов, поэтому различные объекты Random, создаваемые путем вызова конструктора по умолчанию, будут иметь одинаковые начальные значения по умолчанию, и в результате будут выдавать идентичные (одинаковые) наборы случайных чисел.
Random(x)	Инициализирует новый экземпляр класса Random с помощью указанного начального значения. Число **x** - используется для вычисления начального значения последовательности псевдослучайных чисел. Если для приложения требуются различные последовательности случайных чисел, нужно несколько раз подряд вызвать конструктор с различными начальными значениями (если **x** -отрицательное число, то используется его абсолютное значение).
Методы для получения случайных чисел	
Nex	Возвращает неотрицательное случайное число.
Next(x)	Возвращает неотрицательное случайное число, не превышающее максимально допустимое значение **x**.
Next(x, y)	Возвращает случайное число в диапазоне от **x** до **y**.
NextBytes(byte[] x)	Заполняет элементы массива байтов **x** случайными числами.
NextDouble	Возвращает случайное число в диапазоне от **0,0** до **1,0**.

Применение классов Math и Random

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Пример_Math_Random
{
   class Program
   {
      static void Main(string[] args)
      {
         const int p = 5;  //Класс Math
          double a, b, t, t0, dt  = 0.2,  y;
          string vvod, NameF, NF;
          string[] buf;
  Console.WriteLine("Ввод: имя функции a*F(b*t)"+"(sin,cos,tan,cotan,ln)");
          NameF = Console.ReadLine();    //ввод через пробел
   Console.WriteLine("Введите параметр a,b и нач. время t0(тип-double)");
          vvod=Console.ReadLine();  buf = vvod.Split(' ');
           a  = double.Parse(buf[0]);
           b  = double.Parse(buf[1]);
           t0 = double.Parse(buf[2]);
           for (int i = 1; i <= p; i++)
            { NF = "t = {0:#.0}; {1}*{2}({3}*t)= {4};";
              t = t0 + (i-1)* dt;
              switch (NameF)
              { case ("sin"):y = a*Math.Sin(b*t);  break;
                case ("cos"):  y = a*Math.Cos(b*t); break;
                case ("tan"):   y = a*Math.Tan(b*t); break;
                case ("cotan"):  y = a/Math.Tan(b*t);  break;
                case ("ln"):      y = a*Math.Log(b*t); break;
                case ("tanh"):   y = a*Math.Tanh(b*t); break;
                default: NF = "t = {0:#.0}; {1}*{2}*t[{3}]={4:.00};";
                         y=a*b*t; break;}
                Console.WriteLine(NF,t,a,b,NameF,y);
              }
                 double u = 2.5, v = 1.5, pp, w;
                 pp= Math.Pow(u,v);
                 w = Math.IEEERemainder(u,v);
```

```
 Console.WriteLine("U = {0:.##}; V= {1:.##}; Степень(U**V)= {2:##.####};
                                    Остаток(U/V)= {3}", u, v, pp, w);
                const int iRnd = 77; //Класс Random
                Random rRnd = new Random();
Random arrayRnd = new Random(iRnd);// случайные числа в диап. [0,1)
                Console.WriteLine(" Случайные числа в диапазоне: [0,1]");
              for(int i =1; i <= 5; i++)
                { Console.WriteLine("Число " + i + "= " + rRnd.NextDouble() ); }
                int min = -100, max = -10; //  случайные числа в диап.min,max]
       Console.WriteLine("Случайные числа в диап.: ["+min+","+max+"]");
       for(int i =1; i <= 5; i++)
              {Console.WriteLine("Число " + i + "= " + rRnd.Next(min,max) ); }
                byte[] bar = new byte[10]; // случайный массив байтов
                arrayRnd.NextBytes(bar);
         Console.WriteLine(" Массив случайных чисел в диап.: [0, 255]");
                    for(int i =0; i < 10; i++)
                    {Console.WriteLine("Число " + i + "= " +bar[i]); }
          Console.ReadKey();
    }
  }
}
```

Результат выполнения:

```
Введите имя функции a*F(b*t) (sin, cos, tan, cotan, ln)  sin
Введите параметр a,b и начальное время t0 (тип - double) 2,5  3,6  1,0
t = 1,0; 2,5*3,6(sin*t)= -1,10630110823713;
t = 1,2; 2,5*3,6(sin*t)= -2,30999539680797;
t = 1,4; 2,5*3,6(sin*t)= -2,36703443898152;
t = 1,6; 2,5*3,6(sin*t)= -1,24910470779225;
t = 1,8; 2,5*3,6(sin*t)= 0,488866287751361;
U = 2,5; V= 1,5; Степень(U**V)= 3,9528; Остаток(U/V)= -0,5
 Случайные числа в диапазоне: [0,1]
Число 1= 0,281415222809378
Число 2= 0,06275194839702 54
Число 3= 0,593723216836212
Число 4= 0,921412883755478
Число 5= 0,387342177977479
  Случайные числа в диапазоне: [-100,-10]
Число 1= -25   Число 2= -55  Число 3= -80  Число 4= -70   Число 5= -79
 Массив случайных чисел в диапазоне: [0, 255]
Число 0= 49    Число 1= 202  Число 2= 111  Число 3= 54   Число 4= 198
Число 5= 165  Число 6= 186  Число 7= 14    Число 8= 50   Число 9= 203
```

Программа на языке C# - *это текстовый файл (один или несколько), состоящий из* ***операторов*** *языка и* ***комментариев****.*

Операторы *могут содержать* ***зарезервированные*** *слова,* ***идентификаторы****,* ***константы*** *(литералы) и* ***выражения****. Отдельные части оператора отделяются друг от друга пробелами. Несколько следующих друг за другом пробелов считаются одним пробелом. Каждый оператор заканчивается точкой с запятой, что позволяет располагать в одной строке несколько операторов (или один длинный оператор на нескольких строках - в этом случае перенос части оператора на другую строку возможен в любом месте, где может располагаться пробел, кроме строковых выражений).*

Лабораторная работа №1

Написать программу для работы с одномерными, двухмерными и "ступенчатыми" массивами, выполняющую следующие действия:

1. Работа с одномерными массивами:

а). – вывод элементов массива;
– найти Max, Min элементы (вывести номер и значение элемента);
– выполнить прямую и обратную сортировку;
– заполнить новый массив четными элементами из исходного.

б). использовать свойств и методов класса System.Array
– вывод элементов массива;
– найти Max, Min элементы;
– выполнить прямую и обратную сортировку;
– заполнить новый массив четными элементами из исходного.

2. Работа с двухмерными массивами:
– вывод элементов массива;
– найти Max, Min элементы (вывести номер и значение)
– выполнить произведение, сумму, разность 2-х массивов.

3. Работа со "ступенчатыми" массивами:
– вывод элементов массива;
– изменить элементы массива; найти Max, Min элементы.

Разработать меню для функционирования программы.
Предусмотреть ввод данных массивов:
1). с клавиатуры, 2). из файла.