Пользовательские типы - типы который создает пользователь.

В C# поддерживаются пользовательские ссылочные типы и типы значений. Типы **перечисления**, **структура**, **массив**, **строка**, **класс** и **интерфейс** используются для создания собственных пользовательских типов.

Перечисление - тип, переменные которого могут принимать конечное множество возможных значений. У *перечислений* нет собственных методов. Объявление типа содержит список возможных значений. Перечисления - тип значений, который состоит из набора именованных констант.

Синтаксис объявления *перечислений*:

[атрибуты] [спецификаторы] enum <имя_перечисления> [:базовый тип]
{<список_возможных_значений>}

```
public enum MyColors { red, blue, yellow, black, white };
public enum Rainbow{красный,оранжевый,желтый,синий,фиолетовый};
public enum Sex : byte { man = 1, woman };
public enum DayOfWeek{Monday,Tuesday, Wednesday, Thursday, Friday};
public enum Days : long { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

Значения объектов перечисления в задаются значениями базового класса. По умолчанию, базовым классом является класс **int** (значения начинается с 0). Значение задается или неявно (первый элемент принимает значение 0, последующие элементы принимают значение предыдущего + 1).

```
public enum Nums { two = 2, three, ten = 10, eleven, fifty = ten + 40 };
```

Элементы перечисления могут иметь одно и то же значение. Элементы одного перечисления не могут иметь одинаковые имена.

С переменными перечисляемого типа можно выполнять:

- арифметические операции (+, –, ++, – –)
- логические поразрядные операции (^, &, |, ~)
- сравнить их с помощью операции отношения (<, <=, >, >=, = =, !=)

```
public enum Menu { Read, Write, Edit, Quit };
Menu n, m;   m = Menu.Read; n = m; n++;
if (n > m) Console.WriteLine(n); //Вывод: Write
```

При использовании переменных перечисляемого типа в целочисленных выражениях и операциях присваивания требуется явное преобразование типа. Переменной перечисляемого типа можно присвоить любое значение, представимое с помощью базового типа.

Базовый класс System.Enum. Все перечисления в C# являются потомками базового класса System.Enum.

Метод/операция	*Описание*
typeof	Операция typeof возвращает тип своего аргумента.
GetName	Статический метод GetName возвращает символическое имя константы по ее номеру.
GetNames	Статический метод GetValues возвращае массив имен констант, составляющих перечисление.
GetValues	Статический метод GetValues возвращае массив значений констант, составляющих перечисление.
IsDefined	Статический метод IsDefined возвращает значение true, если константа с заданным символическим именем описана в указанном перечислении, и false в противном случае.
GetUnderlyingType	Статический метод GetUnderlyingType возвращает имя базового типа, на котором построено перечисление.

```
public enum B_Звание{Рядовой,Сержант,Лейтенант,Майор,Полковник}
   Console.WriteLine(Enum.GetName(typeof(B_Звание), 1)); //Вывод:
сержант
Array names = Enum.GetNames(typeof(B_Звание));
Console.WriteLine("Количество элементов в перечислении:" + names.Length);
//Вывод: 6
        foreach(string elem in names)
              Console.Write(elem);
//Вывод:Рядовой Сержант Лейтенант Майор Полковник
```

```
        Array values = Enum.GetValues(typeof(B_Звание));
        foreach (B_Звание elem in values)
            Console.Write(" " + (byte)elem); //Вывод: 0 1 2 3 4
    if (Enum.IsDefined(typeof(B_Звание), "Сержант"))
            Console.WriteLine("Константа с таким именем существует");
     else  Console.WriteLine("Константа с таким именем не существует");
Console.WriteLine(Enum.GetUnderlyingType(typeof(B_Звание)));
//Вывод: System.Int32
```

Структуры - составной тип данных (значимого типа), в который входят элементы любых типов, в том числе и функции. В отличие от массива, который является однородным объектом, структура может быть неоднородной (структура объединяет несколько переменных разного типа данных). Переменные структуры называются членами или полями структуры (структура - тип данных, который позволяет определять в рамках единого целого поля разных типов и методы их обработки - аналогична классу).

Синтаксис объявления *структуры*:

```
[ атрибуты ] [ спецификаторы ] struct имя_структуры [ : интерфейсы ]
  {   тело_структуры   }
struct Book // Определение структуры
 {
        public string author;
        public string title;
        public int copyright;
        public Book (string a, string t, int c)
        {// Определение конструктора
            author = a;         title = t;         copyright = c;
        }
    }
```

Правила описания структур:

- Структуры не могут участвовать в иерархиях, для их элементов не может использоваться спецификатор **protected**.
- Структуры не могут наследовать другие структуры или классы. Структуры не могут использоваться в качестве базовых для других структур и классов.
- Структуры не могут быть абстрактными (**abstract**).
- Методы структур не могут быть абстрактными или виртуальными.
- Переопределяться (то есть описываться со спецификатором override) могут только методы, унаследованные от базового класса **object**.
- Параметр **this** интерпретируется как значение, поэтому его можно использовать для ссылок, но нельзя - для присваивания.
- При описании структуры нельзя задавать значения полей по умолчанию (к статическим полям это ограничение не относится). Значения полей задаются в конструкторе по умолчанию, создаваемом автоматически (конструктор присваивает значимым полям структуры нули, а ссылочным - значение **null**).
- В структурах нельзя переопределить конструктор по умолчанию. Все определенные в структуре конструкторы должны принимать параметры и обязательно должны быть инициализированы все поля.

Структура - частный случай классов, *объекты которых хранятся в стеке, а не в "куче", как у классов*. Объект структуры можно создать с помощью оператора **new**, подобно любому объекту класса (но это не обязательно). Если использовать оператор **new**, вызывается указанный конструктор, а если не использовать его, объект все равно будет создан, но не инициализирован. В этом случае необходимо выполнить инициализацию структуры вручную.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace _Структуры
{
   class Program // Демонстрация использования структуры Book
   {
      struct Book // Определение структуры
      {
         public string author;
         public string title;
         public int god;
         public Book(string a, string t, int c)
         {// Определение конструктора
            author = a;          title = t;        god = c;
         }
          public override string ToString() // Перегруженный метод ToString()
{ return (string.Format("Автор: {0}; Название: {1}; Год: {2};", author, title, god));}
      }
      static void Main(string[] args)
      {   //ДОСТУП к элементам структуры – через операцию “точка”
       Book bookl = new Book("Носов","Незнайка на луне",2001); //Вызов констр.
         Book book2 = new Book(); // Вызов конструктора по умолчанию
         Book Book3; // Создание объекта без вызова конструктора
         Console.Write("Структура bookl:\n   ");
         Console.WriteLine(bookl);
         Console.Write("Структура book2:\n   ");
        if(book2.title == null)   Console.WriteLine("Поле book2.title содержит null");
    //Console.WriteLine(Book3.title);//Ошибка: необходима инициализация Book3
         Book3.title = "Волков";
         Book3.author = "Волшебник изумрудного города";
         Book3.god = 1992;
         Console.Write("Структура book3:\n   ");
         Console.WriteLine(book3);
          book2.title = "Шейнин";// Задание данных структуры book2
         book2.author = " Записки следователя ";
         book2.god = 2012;
         Console.Write("Теперь структура book2 содержит:\n " ) ;
         Console.WriteLine(book2+"\n");
  Console.WriteLine(book2.title + ", автор " + book2.author + ", (c) " + book2.god);
      }
   }
}
```

Результат выполнения:
Структура book1:
Автор: Носов; Название: Незнайка на луне; Год: 2001;
Структура book2:
Поле book2.title содержит null
Структура book3:
Автор: Волков; Название: Волшебник изумрудного города; Год: 1992;
Теперь структура book2 содержит:
Автор: Шейнин; Название: Записки следователя; Год: 2012;
Записки следователя, автор Шейнин, (c) 2012

При выборе между структурой и классом пользуются правилами:
если у класса мало полей, а число возможных объектов будет велико, класс надо описывать как структуру (в этом случае память объектам отводиться в стеке, не будут создаваться лишние ссылки, что позволит повысить эффективность работы).

Встроенные (стандартные) структуры. *Все значимые типы реализованы структурами.* В библиотеке FCL имеются и другие встроенные структуры: **Point**, **PointF**, **Size**, **SizeF** и Rectangle, находящиеся в пространстве имен System.Drawing и используемые при работе с графическими объектами. Первые четыре структуры имеют два открытых поля **X** и **Y** (**Height** и **Width**), задающие для точек - структур **Point** и **PointF** - координаты, целочисленные или в форме с плавающей точкой. Для размеров - структур **Size** и **SizeF** - они задают высоту и ширину, целочисленными значениями или в форме с плавающей точкой. Структуры **Point** и **Size** позволяют задать прямоугольную область - структуру **Rectangle**. Конструктору прямоугольника можно передать в качестве аргументов две структуры - точку, задающую координаты левого верхнего угла прямоугольника, и размер - высоту и ширину прямоугольника.

Структуры описания временных данных DateTime и TimeSpan. В пространстве имен System библиотеки FCL имеются структуры для работы с данными времени: **DateTime** и **TimeSpan**. Структура DateTime представляет

момент времени, который обычно задается в виде: даты (год, месяц, день) и времени дня (часы, минуты, секунды). Тип **DateTime** предназначен для хранения даты и времени в диапазоне от 01.01.0001 0:00:00 до 31.12.9999 23:59:59 в григорианском календаре. Значения типа измеряются в тиках - 100-наносекундных интервалах, прошедших от момента 01.01.0001 0:00:00. Примером конструктора является DateTime (int year, int month, int day).:

```
DateTime dTime = new DateTime(1980, 8, 15);
```

Пример использования - операции сложения (+):

```
public static DateTime operator + (DateTime d, TimeSpan t);
```

В операции используется еще одна структура TimeSpan, которая представляет интервал времени (в днях, часах, минутах и секундах). Пример создания экземпляра интервала времени в 17 дней, 4 часа, 2 минуты и 1 сек.:

```
TimeSpan tSpan = new TimeSpan(17, 4, 2, 1);
```

Тогда допустима операция:

```
DateTime result = dTime + tSpan;
```

(момент времени 01.09.1980 4:02:01)

Свойства структуры DateTime

Название	*Описание*
Date	Возвращает дату
Day	Возвращает день
Month	Возвращает месяц
Year	Возвращает год
DayOfWeek	Возвращает день недели
DayOfYear	Возвращает номер дня в году
Hour	Возвращает часы
Minute	Возвращает минуты
Second	Возвращает секунды
Millisecond	Возвращает миллисекунды
Ticks	Возвращает количество тиков
Now	Возвращает текущие дату и время, установленные на ПК
Today	Возвращает текущую дату
ToLongDateString	Возвращает дату с названием месяца
ToShortDateString	Возвращает дату с номером месяца
ToLongTimeString	Возвращает время с секундами

Название	*Описание*
ToShortTimeString	Возвращает время без секунд
DaysInMonth	Возвращает количество дней в месяце
IsLeapYear	Определяет высокосный год
AddDays	Добавляет к экземпляру соответствующее число дней
AddMinute	Добавляет к экземпляру соответствующее число минут

Пример использования структуры DateTime:

```
using System; using System.Collections.Generic; using System.Linq;
using System.Text;
namespace _Структуры
{  class Program
   {
      void Main(string[] args)
      {   //ДОСТУП к элементам структуры – через операцию “точка”
         Console.WriteLine("Тип System.DateTime:");
Console.WriteLine("Диапазон значений от {0} до {1}",DateTime.MinValue,
DateTime.MaxValue);
         DateTime d1 = DateTime.Now;
         Console.WriteLine("Текущие дата и время: {0}", d1.ToString());
         Console.WriteLine("Тики: {0}", d1.Ticks);
         Console.WriteLine("День недели: {0}", d1.DayOfWeek);
         Console.WriteLine("Номер дня в году {0}", d1.DayOfYear);
         Console.WriteLine("ToLongDateString: {0}", d1.ToLongDateString());
         Console.WriteLine("ToShortDateString; {0}", d1.ToShortDateString());
         Console.WriteLine("ToLongTimeString: {0}", d1.ToLongTimeString());
         Console.WriteLine("ToShortTimeString: {0}", d1.ToShortTimeString());
     }  }  }
```

Результат выполнения:

```
Тип System.DateTime:
Диапазон значений от 01.01.0001 0:00:00 до 31.12.9999 23:59:59
Текущие дата и время: 19.09.2012 23:43:44
Тики: 634836950242511673
День недели: Wednesday
Номер дня в году 263
ToLongDateString: 19 сентября 2012 г.
ToShortDateString; 19.09.2012
ToLongTimeString: 23:43:44
ToShortTimeString: 23:43
```

Присваивание структур, массивы структур. Структуры похожи на классы по своему описанию и ведут себя сходным образом, но имеют существенные различия в выполнении операции присваивания. Если при присваивании переменных класса происходит только присваивание ссылок на объекты, то при присваивании переменных структур - создается **новый объект в стеке**.

При присваивании структур и и при передаче структур в качестве параметров по значению создаются копия значений полей. В методы можно структуры передавать по ссылке с помощью ключевых слов **ref** или **out**.

Пример использования массива структур:

```
Point_St[] mas = new Point_St[4];
        for (int i = 0; i < 4; ++i)
        {
            mas[i].x = i;
            mas[i].y = 2 * i;
        }
    foreach ( Point_St elem in mas ) // вывод элементов массива структур
            Console.WriteLine(elem);
```

Результат выполнения

```
x = 0; y =  0;
x = 1; y =  2;
x = 2; y =  4;
x = 3; y =  6;
```

Массив - упорядоченная совокупность элементов одного типа. Каждый элемент массива имеет индексы, определяющие порядок элементов. Количество индексов характеризует размерность массива. Каждый индекс изменяется в диапазоне от **0** до **N**. Индексы задаются целочисленным типом. Массивы относятся к ссылочным типам данных - располагается только в *динамической памяти*, поэтому для создание массива нужно выделить память под его элементы (память хранения отводится в "куче"). Элементам массива присваиваются значения по умолчанию – 0 для значимых типов, и null для ссылочных. Элементами массива могут быть величины как *значимых (массив чисел)*, так и *ссылочных* (массив строк) типов (в том числе массивы).

Массив значимых типов хранит значения, массив ссылочных типов хранит ссылки на элементы. Количество элементов в массиве (размерность) не является частью его типа, это количество задается при выделении памяти и **не может быть изменено**. Для обращения к элементу массива после имени массива указывается номер (индекс) элемента в квадратных скобках. *Массивы одного типа можно присваивать друг другу*. Все массивы имеют общий базовый класс **System.Array** пространства имен System. В языке C# есть **одномерные**, **многомерные** и **ступенчатые** (**"зубчатые") массивы**.

Одномерный массив - фиксированное количество элементов одного типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в начинается с нуля (если массив состоит из 10 элементов, то его элементы будут иметь номера: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

Синтаксис объявления одномерного *массива*:

[атрибуты] [спецификаторы] <тип_элемента> [ ] <имя_массива>
[инициализаторы];

Для объявления одномерного массива используется одна форм описания:

Форма описания массива	***Пояснения***
тип_элемента[] **имя_массива;** ***Пример*:** **int[] a;**	Описана ссылка на одномерный массив, которая в дальнейшем может быть использована: - для адресации на уже существующий массив; - передачи массива в метод в качестве параметра; - последующего выделения памяти под элементы массива.
тип_элемента[] **имя_массива = new** **тип_элемента [размер];** ***Пример*:** **int[] a = new int[10];**	Объявлен одномерный массив заданного типа и выделена память под одномерный массив указанной размерности. Адрес данной области памяти записан в ссылочную переменную. Элементы массива равны нулю. В C# элементам массива присваиваются начальные значения по умолчанию в зависимости от базового типа. Для арифметических типов – нули, для ссылочных типов – null, для символов - пробел.

Форма описания массива	*Пояснения*
тип_элемента[] имя_массива = {список инициализации}; ***Пример*:** **int[] a = { 0, 1, 2, 3 };**	Выделена память под одномерный массив, размерность которого соответствует количеству элементов в списке инициализации. Адрес этой области памяти записан в ссылочную переменную. Значение элементов массива соответствует списку инициализации.

Описание и инициализация (присваивание значений) одномерных массивов:

```
int[] u, v;//объявление массивов с отложенной инициализацией
int[] x = { 5, 5, 6, 6, 7, 7 };//объявление массива с явной инициализацией
      u = new int[3];
      for (int i = 0; i < 3; i++) u[i] = i + 1;
      //v= {1,2,3}; //присваивание константного массива недопустимо
      v = new int[4];
      v = u; //допустимое присваивание
```

Действия над массивами. Массивы одного типа можно присваивать друг другу. При этом происходит присваивание ссылок, а не элементов:

```
int[] a = new int[10];      //  a [5] = 10
int[] b = a;       // b и a указывают на один и тот же массив  b [5] = 10
```

При присвоении: b [5] = 100 значение a [5] станет: a [5] = 100

Остальные действия выполняются с элементами массива по отдельности:

```
for (int i = 0; i < n; ++i) Console.Write("\t" + a[i]);
```

Хотя при инициализации массива нет необходимости использовать операцию **new**, массив можно инициализировать следующим образом:

```
int[] myArray = new int[] { 99, 10, 100, 18, 78, 23, 163, 9, 87, 49 };
    Console.Write("\nСтарый массив: \t");
int[] myArray = { 0, 1, 2, 3, 4, 5 };
for (int i = 0; i < 6; i++)    Console.Write(" " + myArray[i]);
Console.Write("\nНовый массив: \t");
myArray = new int[] { 99, 10, 100, 18, 78, 23, 163, 9, 87, 49 }; // 1
for (int i = 0; i < 10; i++)  Console.Write(" " + myArray[i]);
```

Массивы и исключения. Выход за границы массива в C# ошибка, в ответ на которую генерируется исключение - ***IndexOutOfRangeException***:

```
int[] myArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };        int i;
try    {  for (i = 0; i <= 10; i++) Console.WriteLine(myArray[i]); }
catch (IndexOutOfRangeException)
{  Console.WriteLine("Exception: Выход за границу диапазона");  }
```

при **i=10** будет сгенерировано исключение *IndexOutOfRangeException* (выход за границы) и выдано сообщение: "**Exception:** Выход за границу диапазона".

Массив - параметр процедуры, функции. Имя массива - ссылка, поэтому он передается в метод по ссылке и, следовательно, все изменения элементов массива, являющегося формальным параметром, отразятся на элементах массива, являющимся фактическим параметром.

Пример передачи массива как параметра:

```
static void Print(int n, int[] a)//n-размерность массива, a-ссылка на массив
{  for (int i = 0; i < n; i++) Console.Write("{0} ", a[i]);  Console.WriteLine(); }
static void Che(int n, int[] a)//n-размерность массива, a-ссылка на массив
{  for (int i = 0; i < n; i++) if (a[i] > 0) a[i] = 0;  }//изменяются элем. массива
static void Main(string[] args)
{  int[] myArray = { 0, -1, -2, 3, 4, 5, -6, -7, 8, -9 };
 Print(10, myArray); Che(10, myArray);
 Print(10, myArray);
}
```

Результат выполнения

Старый массив: 0 -1 -2 3 4 5 -6 -7 8 -9
Новый массив: 0 -1 -2 0 0 0 -6 -7 0 -9

Пример передачи возвратоа массива в качестве результата функции:

```
public static int[]  Add_Mas (int[] a) // a – ссылка на массив
{
      for (int i = 0; i < a.Length; i++)  a[i] = Math.Abs(a[i] * i);    return a;
}
 static void Main(string[] args)
 {
   int[] myArray = { 0, -1, -2, 3, 4, 5, -6, -7, 8, -9 };
   Print(10, myArray);   Print(10, Add_Mas(myArray)); }
```

Результат выполнения

Старый массив: 0 -1 -2 3 4 5 -6 -7 8 -9
Новый массив: 0 1 4 9 16 25 36 49 64 81

Многомерные массивы. Разделение массивов на одномерные и многомерные носит исторический характер. Разницы между ними нет. Одномерные массивы - это частный случай многомерных массивов. Одномерные массивы позволяют задавать такие математические структуры как векторы, двумерные - матрицы, трехмерные - кубы данных, массивы большей размерности - многомерные кубы данных (при работе с базами данных многомерные кубы - кубы OLAP, встречаются сплошь и рядом).

Синтаксис объявления многомерного массива в общем случае:

[атрибуты] [спецификаторы] < тип_элемента > [ ] < имя_массива >

[инициализаторы];

Многомерный массив имеет более одного измерения. Число запятых (увеличенное на единицу) задает размерность массива.

Описание и инициализация (присваивание значений) многомерных массивов:

```
int[,] a; //элементов нет
int[,] b = new int[2, 3]; //элементы равны 0
int[,] c = { { 1, 2, 3 }, { 4, 5, 6 } }; //new подразумевается
int[,] d = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } }; //размерность вычисляется
string[, ,] e = new string[10, 10, 10];// создание 3-го массива элементы = null
int[,] ms = new int[2,3] {{1,2,3},{4,5,6}};
```

Доступ к элементам массива выполняется путем указания в квадратных скобках индексов по каждой размерности, через запятую: `int k=d[1,2]; //получим значение 6`

*Пример выполнения операций с **двухмерным** массивом:*

```
static void PrintArray(string a, int[,] mas)
{
    Console.WriteLine(a);
    for (int i = 0; i < mas.GetLength(0); i++)
    {
```

```
        for (int j = 0; j < mas.GetLength(1); j++)  Console.Write("{0} ", mas[i, j]);
        Console.WriteLine();
        }
    static void Change(int[,] mas)
    {
        for (int i = 0; i < mas.GetLength(0); i++)
            for (int j = 0; j < mas.GetLength(1); j++)
                if (mas[i, j] % 2 == 0) mas[i, j] = 0; //Обнуление четных значений
    }
     static void Main(string[] args)
    {
        try
        {
            int[,] MyArray = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
            Console.WriteLine("Введите значение элемента MyArray[0,0]");
            MyArray[0,0] = int.Parse(Console.ReadLine());
            PrintArray("Исходный массив:", MyArray);
            Change(MyArray);
            PrintArray("итоговый массив", MyArray);
        }
        catch (FormatException)
        {  Console.WriteLine("неверный формат ввода данных"); }
        catch (OverflowException)  {  Console.WriteLine("переполнение");  }
        catch (OutOfMemoryException)
{  Console.WriteLine("не хватает памяти для создания нового объекта"); }
        catch    {  Console.WriteLine("ошибка выполнения");  }
    }
```

Результат выполнения

```
Введите значение элемента MyArray[0,0]  asd
неверный формат ввода данных
Введите значение элемента MyArray[0,0]  1
Исходный массив:
1 2 3
4 5 6
7 8 9
Итоговый массив
1 0 3
0 5 0
7 0 9
```

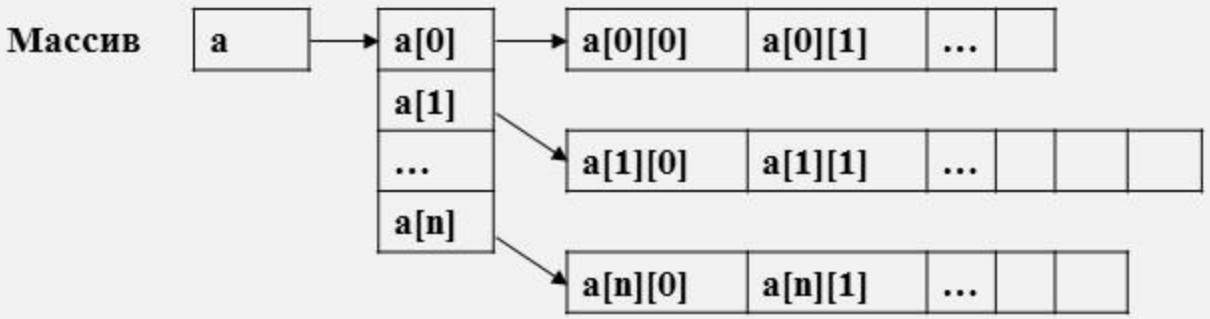
*Пример выполнения операций с **трехмерным** массивом:*

```
const int ROWS = 5, COLUMNS = 4, DEPTH = 3;// Границы
int[, ,] array = new int[ROWS, COLUMNS,DEPTH];//Создание массива [5,4,3]
        int count = 1;// Наполнение массива
        for (int i = 0; i < ROWS; i++)
           for (int j = 0; j < COLUMNS; j++)
              for (int k = 0; k < DEPTH; k++)        array[i, j, k] = count++;
        for (int i = 0; i < ROWS; i++)// Вывод массива
        {  // Работа с трехмерным массивом
           string strTab = "";// Сдвиг для новой строки
           for (int k = 0; k < DEPTH; k++)// Выборка глубины
           {
               string str = "";
               for (int j = 0; j < COLUMNS; j++)
              {str += String.Format("a[{0},{1},{2}]={3, -6}", i, j, k, array[i, j, k]);}
              Console.WriteLine(strTab + str);// Строка в глубину
              strTab += "  ";// Сдвиг 2 пробела
           }
        }
```

Результат выполнения

```
a[0,0,0]=1   a[0,1,0]=4   a[0,2,0]=7   a[0,3,0]=10
       a[0,0,1]=2   a[0,1,1]=5   a[0,2,1]=8   a[0,3,1]=11
              a[0,0,2]=3   a[0,1,2]=6   a[0,2,2]=9   a[0,3,2]=12
a[1,0,0]=13  a[1,1,0]=16  a[1,2,0]=19  a[1,3,0]=22
       a[1,0,1]=14  a[1,1,1]=17  a[1,2,1]=20  a[1,3,1]=23
              a[1,0,2]=15  a[1,1,2]=18  a[1,2,2]=21  a[1,3,2]=24
a[2,0,0]=25  a[2,1,0]=28  a[2,2,0]=31  a[2,3,0]=34
       a[2,0,1]=26  a[2,1,1]=29  a[2,2,1]=32  a[2,3,1]=35
              a[2,0,2]=27  a[2,1,2]=30  a[2,2,2]=33  a[2,3,2]=36
a[3,0,0]=37  a[3,1,0]=40  a[3,2,0]=43  a[3,3,0]=46
       a[3,0,1]=38  a[3,1,1]=41  a[3,2,1]=44  a[3,3,1]=47
              a[3,0,2]=39  a[3,1,2]=42  a[3,2,2]=45  a[3,3,2]=48
a[4,0,0]=49  a[4,1,0]=52  a[4,2,0]=55  a[4,3,0]=58
       a[4,0,1]=50  a[4,1,1]=53  a[4,2,1]=56  a[4,3,1]=59
              a[4,0,2]=51  a[4,1,2]=54  a[4,2,2]=57  a[4,3,2]=60
```

Ступенчатые массивы (массивы массивов) - массивы массивов, называемые также ступенчатыми (зубчатыми) массивами (jagged arrays). Массив массивов можно рассматривать как одномерный массив, элементы которого являются массивами, элементы которых, в свою очередь, снова могут быть массивами, и так может продолжаться до некоторого уровня вложенности. В ступенчатых массивах количество элементов в разных строках может быть различным. В памяти ступенчатый массив хранится в виде массива массивов. *Структура ступенчатого масива:*



Синтаксис объявления ступенчатого массива в общем случае :

[атрибуты] [спецификаторы] <тип_элемента>[ ][ ]<имя_массива>

[инициализаторы];

При объявлении типа многомерных массивов для указания размерности используются запятые, то для ступенчатых массивов применяется совокупности пар квадратных скобок ([ ][ ]). `int[][] a;` - фактически объявлен одномерный массив ссылок на целочисленные одномерные массивы. При таком описании необходимо не только выделять память под одномерный массив ссылок, но и выделять память под каждый из целочисленных одномерных массивов. Такое распределение памяти позволяет определять произвольную длину каждой строки массива (отсюда и название массива - ступенчатый массив). Для создания массива нельзя вызвать конструктор `int[][] a = new int[3][5];` , поскольку он не задает ступенчатый массив. Для создания массива нужно вызывать конструктор для каждого массива на самом нижнем уровне.

Описание , инициализация (присваивания значений) ступенчатых массивов:

Ступенчатый прямоугольный массив (описание и создание)

```
int[][] intArray = new int[3][]; //описание ступенчатого массива из 3 эл.
for (int i = 0; i < intArray.Length; i++)//создание массива из 3 эл.
intArray[i] =new int[5];//каждый эл. содержит одномерный массив из 5 эл.
```

Ступенчатый непрямоугольный массив (описание и создание)

```
float[][] floatArray = new float[3][]; //описание массива из 3 эл.
for (int i = 0; i < floatArray.Length; i++) //создание массива из 3 эл.
 floatArray[i] = new float[i + 1];//каждый эл. содержит массив из i + 1 эл.
```

Ступенчатый массив: инициализация нулевыми значениями

```
int[][] a = new int[3][];// описание ступенчатого массива из 3 эл.
a[0] = new int[2];// 0-ая строка ссылается на 2-х эл. одномерный массив
a[1] = new int[3];// 1-ая строка ссылается на 3-х эл. одномерный массив
a[2] = new int[10];//2-ая строка ссылается на 10-х эл. одномерный массив
```

Сокращенный вариант: `int[][] a = { new int[2], new int[3], new int[10] };`

Ступенчатый массив: инициализация значениями

```
int[][] intArray = new int[3][];
intArray[0] = new int[] { 1, 2, 3, 4, 5 };
intArray[1] = new int[] { 3, 4, 5 };
intArray[2] = new int[] { 1, 2, 3, 4, 5, 6, 7 };
```

Сокращенный вариант: `int[][] myArray = new int[][]`

```
{
  new int[] {1, 2, 3, 4, 5},
  new int[] {3, 4, 5},
  new int[] {1, 2, 3, 4, 5, 6, 7}
};//Конструктор new int[][] можно опустить
```

Ступенчатые и обычные массивы можно использовать совметно

```
// вариант ("зубчатые" и "обычные" массивы можно смешивать)
int[][,] myMixdArray=new int[3][,]//описание ступенчатого массива из 3 эл.
{
new int[,] { {1, 2}, {3, 4} },//0-ая строка ссылается на 2-й массив из 4-х эл.
new int[,] {{5,6}, {7,8}, {9,10}},//1-ая строка ссылается на 2-й массив из 6эл.
  new int[,] { {11, 12} } //2-ая строка ссылается на 2-й массив из 2-х эл.
};
```

Доступ к элементу массива: `Console.WriteLine(myMixdArray[0][0, 0]);// вывод 1`

Можно объявить ступенчатый массив без инициализации: `int[][] arr3;`

Массив нельзя использовать, пока он не создан с помощью new.

В ступенчатом двумерном массиве имеется свойство **Length**. Если свойство применить к самому массиву, то оно означает количество строк. При его применении к отдельной строке оно будет означать длину этой строки.

Работа со ступерчатыми массивами:

```
int[][] a = new int[3][];
```

```
        a[0] = new int[5] { 24, 50, 18, 3, 16 };
        a[1] = new int[3] { 7, 9, -1 };
        a[2] = new int[4] { 6, 15, 3, 1 };
        Console.WriteLine("Исходный массив - цикл FOR:");
        for (int i = 0; i < a.Length; ++i)
        {
            for (int j = 0; j < a[i].Length; ++j)
                Console.Write("\t" + a[i][j]);
            Console.WriteLine();
        }
        Console.WriteLine("Исходный массив - цикл FOREACH:");
        foreach (int[] mas1 in a)
        {
            foreach (int x in mas1)       Console.Write("\t" + x);
            Console.WriteLine();
        }
    Console.WriteLine(Array.IndexOf(a[0], 18)); //поиск числа 18 в 0-й строке
```

Класс System.Array - массив как объект. Массивы в C# реализованы как объекты на основе базового класса Array, определенного в пространстве имен System. Класс System.Array наследует интерфейсы: ICloneable, IList, ICollection, IEnumerable и соответственно реализует все их методы и свойства. Помимо наследования свойств и методов класса Object и интерфейсов, класс Array имеет большое число собственных методов и свойств. Благодаря этому, над массивами определены самые разнообразные операции - копирование, поиск, обращение, сортировка, получение различных характеристик. Массивы можно рассматривать как коллекции и устраивать циклы **foreach** для перебора всех элементов.

Основные свойства и методы класса Array

Свойства	*Описание*
IsFixedSize	True, если массив статический
IsReadOnly	Для всех массивов имеет значение **false**
IsSynchronized	True или False, в зависимости от того, установлена ли синхронизация доступа для массива
Length	Число элементов массива
Rank	Размерность массива
GetLength	Возвращает число элементов массива по указанному измерению
GetLowerBound, GetUpperBound	Возвращает нижнюю и верхнюю границу по указанному измерению. Для массивов нижняя граница всегда равна нулю.
GetValue,	Возвращает значение элемента массива с указанными индексами.
SetValue	Устанавливает значение элемента массива с указанными индексами.
Методы	**Описание**
Min	Получает минимальный элемент одномерного массива.
Max	Получает максимальный элемент одномерного массива.
Sum	Получает сумму элементов одномерного массива.

SyncRoot	Собственный метод синхронизации доступа к массиву. При работе с массивом его можно закрыть на время обработки, что запрещает его модификацию каким-либо потоком:
	`Array myMas = new int[];`
	`lock( myMas.SyncRoot ) {`
	`foreach (Object item in myMas){//безопасная обработка масс.}`

Clone	Позволяет создать плоскую или глубокую копию массива. В первом случае создаются только элементы первого уровня, а ссылки указывают на те же самые объекты. Во втором случае копируются объекты на всех уровнях. Для массивов создается только плоская копия.
CopyTo	Копируются все элементы одномерного массива в другой одномерный массив, начиная с заданного индекса: `myMas1.CopyTo(myMas2,0);`
Initialize	Можно применять только к массивам значимого типа. Инициализирует элементы, вызывая соответствующий конструктор (как правило, не используется в обычных программах).

Статические методы	Описание
Clear	Выполняет начальную инициализацию элементов. В зависимости от типа элементов устанавливает значение 0 для арифметического типа, false - для логического типа, Null для ссылок, "" - для строк.
BinarySearch	Двоичный поиск. Описание и примеры даны в тексте
Copy	Копирование части или всего массива в другой массив.
IndexOf	Индекс первого вхождения образца в массив. Описание и примеры даны в тексте
LastIndexOf	Индекс последнего вхождения образца в массив. Описание и примеры даны в тексте
Reverse	Обращение одномерного массива. Описание и примеры даны в тексте
Sort	Сортировка массива. Описание и примеры даны в тексте
CreateInstance	Класс Array, в отличие от многих классов, может создавать свои экземпляры не только с помощью конструктора new, но и при вызове метода CreateInstance: `Array myMas =Array.CreateInstance(typeof(double), 2, 2);`
GetValue,	Возвращает значение элемента массива с указанными индексами.
SetValue	Устанавливает значение элемента массива с указанными индексами.

Большинство методов класса Array можно применять только при использовании одномерных массивов. К элементам массива A, имеющего класс Array, нет прямого доступа в обычной манере - **A [<индексы>]**, но зато есть специальные методы **GetValue (<индексы>)** и **SetValue (<индексы>)**.

Использование свойств и методов класса Array:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace _класс_ARRAY
{
   class Program
   {
      public static Random rnd = new Random();
      public static void Create_1_Mas(Array A)
```

```
{
        int i = 0;
        foreach (object item in A)
            A.SetValue(rnd.Next(1, 20), i++);
    }
    public static void Create_2_Mas(int[,] A)
    {
        Random rnd = new Random();
        for (int i = 0; i < A.GetLength(0); i++)
            for (int j = 0; j < A.GetLength(1); j++)
                A[i, j] = rnd.Next(1, 100);
    }
    public static string Vivod_Min_Max_Sum(object A)
    {//Только для одномерного массива
        string s = "";
        Type t = A.GetType();
        switch (t.Name)
        {
            case "Int32[]":
                int[] m1 = (int[])A;
    s = s + " Min = " + m1.Min() + ", Max = " + m1.Max() + ", Sum = " + m1.Sum();
                break;
            case "Double[]":
                double[] m2 = (double[])A;
    s = s + " Min = " + m2.Min() + ", Max = " + m2.Max() + ", Sum = " + m2.Sum();
                break;
        }//Console.WriteLine(t.Name);
        return s;
    }
    public static void Vivod(string name, Array A)
    {
        string s = Vivod_Min_Max_Sum(A);
        Console.WriteLine("Массив " + name + " " + s);
        switch (A.Rank)
        {
            case 1:
                for (int i = 0; i < A.GetLength(0); i++)
                    Console.Write("   " + name + "[{0}]={1}",i, A.GetValue(i));
```

```
            Console.WriteLine();
            break;
        case 2:
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = 0; j < A.GetLength(1); j++)
                    Console.Write("  " + name + "[{0},{1}]={2}", i, j, A.GetValue(i, j));
                Console.WriteLine();
            }
            break;
        default: break;
      }
    }
static void Main(string[] args)
{//операции над массивами:   сортировка, поиск, копирование,  поиск элемента
      int nc = 7;
      int[] M1 = new int[nc], M2 = new int[nc];
      double[] M3 = new double[nc];
      int[,] M4 = new int[2, 2];
      Create_1_Mas(M1);
      Vivod("M1", M1);
      Create_1_Mas(M2);
      Vivod("M2", M2);
      Create_1_Mas(M3);
      Vivod("M3", M3);
      Create_2_Mas(M4);
     Vivod("M4", M4);
     int first = Array.IndexOf(M1, 2);
      int last = Array.LastIndexOf(M1, 2);
      if (first == -1)
         Console.WriteLine("\tНет вхождений значения 2 в массив M1");
      else
         if (first == last)
            Console.WriteLine("\tОдно вхождение значения 2 в массив M1");
         else      Console.WriteLine("\tНесколько вхождений 2 в массив M1");
     Array.Reverse(M1);//только одномерный массив
      Console.WriteLine("\tОбращение массива M1:Array.Reverse(M1)");
      Vivod("M1", M1);
       Array.Copy(M1, M3, M1.Length); );//Копирование
```

```
Console.WriteLine("\tМассив M3 после копирования в него
                          M1:Array.Copy(M1,M3, )");
        Vivod("M3", M3);
        Array.Copy(M1, 1, M2, 1, 2);//Копирование
        Console.WriteLine("\tКопирование двух элементов M1 в
                          M2:Array.Copy(M1,1,M2,1,2)");
        Vivod("M1", M1);
        Vivod("M2", M2);
        M1.CopyTo(M2, 0);//копирование CopyTo
       Console.WriteLine("\tМассив M2 после копирования M1.CopyTo(M2, 0):");
        Vivod("M2", M2);
        Array.Sort(M1); //быстрая сортировка Хоара
        Console.WriteLine("\tОтсортированный массив:Array.Sort(M1)");
        Vivod("M1", M1);
        int v = M1[3];
        first = Array.BinarySearch(M1, v);
        Console.WriteLine("\tИндекс значения {0}в M1:
                          Array.BinarySearch(M1,{0}){1}-", v, first);
        Console.WriteLine("\M5 после создания
                          M4:Array.CreateInstance(typeof(double),2,3)");
 Array M5=Array.CreateInstance(typeof(double),2,3);//Создание экземпляра (мас.)
        Vivod("M5", M5);
        M5 = (Array)M4.Clone();//клонирование
        Console.WriteLine("\tМассив M5 после клонирования
                          M4:(Array)M4.Clone()");
        Vivod("M5", M5);
    }
  }
}
```

Результат выполнения:

```
Массив M1  Min = 8, Max = 15, Sum = 81
  M1[0]=14  M1[1]=13  M1[2]=12  M1[3]=10  M1[4]=15  M1[5]=8  M1[6]=9
Массив M2  Min = 2, Max = 15, Sum = 61
  M2[0]=10  M2[1]=2  M2[2]=7  M2[3]=13  M2[4]=6  M2[5]=15  M2[6]=8
Массив M3  Min = 3, Max = 18, Sum = 62
  M3[0]=3  M3[1]=9  M3[2]=5  M3[3]=13  M3[4]=6  M3[5]=18  M3[6]=8
Массив M4
```

```
M4[0,0]=45  M4[0,1]=57
  M4[1,0]=12  M4[1,1]=17
      Нет вхождений значения 2 в массив M1
      Обращение массива M1:Array.Reverse(M1)
Массив M1  Min = 8, Max = 15, Sum = 81
  M1[0]=9  M1[1]=8  M1[2]=15  M1[3]=10  M1[4]=12  M1[5]=13  M1[6]=14
      Массив M3 после копирования в него массива M1:Array.Copy(M1, M3, )
Массив M3  Min = 8, Max = 15, Sum = 81
  M3[0]=9  M3[1]=8  M3[2]=15  M3[3]=10  M3[4]=12  M3[5]=13  M3[6]=14
      Копирование двух элементов M1 в M2:Array.Copy(M1, 1, M2, 1, 2)
Массив M1  Min = 8, Max = 15, Sum = 81
  M1[0]=9  M1[1]=8  M1[2]=15  M1[3]=10  M1[4]=12  M1[5]=13  M1[6]=14
Массив M2  Min = 6, Max = 15, Sum = 75
  M2[0]=10  M2[1]=8  M2[2]=15  M2[3]=13  M2[4]=6  M2[5]=15  M2[6]=8
      Массив M2 после копирования M1.CopyTo(M2, 0):
Массив M2  Min = 8, Max = 15, Sum = 81
  M2[0]=9  M2[1]=8  M2[2]=15  M2[3]=10  M2[4]=12  M2[5]=13  M2[6]=14
      Отсортированный массив:Array.Sort(M1)
Массив M1  Min = 8, Max = 15, Sum = 81
  M1[0]=8  M1[1]=9  M1[2]=10  M1[3]=12  M1[4]=13  M1[5]=14  M1[6]=15
      Индекс вхождения значения 12 в M1:Array.BinarySearch(M1, 12) 3 -
      Массив M5 после создания M4:Array.CreateInstance(typeof(double),2,3)
Массив M5
  M5[0,0]=0  M5[0,1]=0  M5[0,2]=0
  M5[1,0]=0  M5[1,1]=0  M5[1,2]=0
      Массив M5 после клонирования M4:(Array)M4.Clone()
Массив M5
  M5[0,0]=45  M5[0,1]=57
  M5[1,0]=12  M5[1,1]=17
```

Для массивов можно создавать ***общие*** процедуры обработки (внутри такой процедуры необходим анализ - какой реальный тип массива. Допустимо преобразований между классами-массивами и классом **Object**. В процедуре вывода массива формальный аргумент процедуры имеет тип **object** (прародитель всех классов). В процедуру передается массив, процедура определяет его тип и работает с ним уже как с массивом, а не как с переменной класса **Object**.

```
public static void ВыводObj(object A)
{
   Console.WriteLine("A.GetType()={0})", A.GetType());
   if (A.GetType() == typeof(System.Int32[]))
   {
      int[] temp = (int[])A;
      for (int i = 0; i < temp.GetLength(0); i++)
         Console.Write("  temp[{0}]={1}", i, temp[i]);
      Console.WriteLine();
   }
   else  Console.WriteLine("Аргумент не является массивом целых");
}
```

Результат выполнения

```
      A.GetType()=System.Int32[])
temp[0]=3  temp[1]=4  temp[2]=6  temp[3]=8  temp[4]=14  temp[5]=14  temp[6]=16
      A.GetType()=System.Double[])
Аргумент не является массивом целых
```

Строки и символы. Когда говорят о строковом типе, то обычно различают:

- отдельные символы - его называют типом **char**;
- строки постоянной длины - они представляются массивом символов;
- строки переменной длины - тип **string**.

Обработка текстовой информации одна из самых распространенных задач в программировании. Для ее решения в C# есть широкий набор средств: символы **char**, неизменяемые строки **string**, изменяемые строки **StringBuider** и регулярные выражения **Regex**.

Символы char (класс System.Char) - частный случай строк длиной 1.

Основные операции над строками - это разбор и сборка. При их выполнении приходится, чаще всего, доходить до каждого символа строки.

*Символьный тип **char** предназначен для хранения:*

- символов, заключенным в одинарные кавычки;
- символа escape-последовательности, задающей код символа;
- символа в кодировке Unicode.

Примеры объявления символьных переменных и работы с ними:

```
char C1 = 'A', C2 = '\x5A', C3 = '\u0058';
char C = new Char();
int code; string s;
Ch = C1;     code = Ch;//преобразование символьного типа в тип int
C1 = (char)(code + 1);//преобразование типа int в символьный
s = C1.ToString()+C2.ToString()+C3.ToString();//преобразование сим. типа
в строку     // Вывод: Строка = BZX, Символ = A, Код = 65
```

Две процедуры, выполняющие взаимно-обратные операции - получение по коду символа и получение символа по его коду:

```
static public int Получение_кода_по_символу (char sym)
  {   return (sym);  }//Получение_символа_по_коду
  static public char Получение_по_коду_символа (object code)
  {    return ((char)((int)code)); }//Получение_по_коду_символа
```

Использование процедур:

```
int Cod = 48;    char A = Получение_по_коду_символа(Cod);
Console.WriteLine("Код ={0} Символ = {1}", Cod, A);  // Код = 48 Символ = 0
A = 'Я';          Cod = Получение_кода_по_символу(A);
Console.WriteLine("Символ = {0} Код = {1}", A, Cod);//Символ= я Код= 1103
```

Символьный тип относится к встроенным типам данных и соответствует классу **Char** библиотеки из пространства имен **System**. В классе определены методы, позволяющие задавать вид и категорию символа, преобразовывать символ в верхний или нижний регистр, в число.

Свойства и методы класса System.Char

Свойства	*Описание*
MinValue	Возвращает символы с максимальным кодом. Возвращаемые символы не имеют видимого образа
MaxValue	Возвращает символы с минимальным кодом. Возвращаемые символы не имеют видимого образа
Метод	*Описание*
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае.
GetUnicodeCategory	Возвращает категорию Unicode-символа. В Unicode символы разделены на категории, например цифры (DecimalDigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и т.д.
IsControl	Возвращает true, если символ является управляющим.
IsDigit	Возвращает true, если символ является десятичной цифрой.
IsLetter	Возвращает true, если символ является буквой.
IsLetterOrDigit	Возвращает true, если символ является буквой или десятичной цифрой.
IsLower	Возвращает true, если символ задан в нижнем регистре.
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным).
IsPunctuation	Возвращает true, если символ является знаком препинания.
IsSeparator	Возвращает true, если символ является разделителем.
IsUpper	Возвращает true, если символ задан в нижнем регистре.
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки, возврат каретки).
IsSurrogate	Некоторые символы Unicode с кодом в интервале [0x1000, 0x10FFF] представляются двумя 16-битными «суррогатными» символами. Метод возвращает true, если символ является суррогатным.
Parse	Преобразует строку в символ (строка состоит из 1символа).
ToLower	Преобразует символ в нижний регистр.
ToUpper	Преобразует символ в верхний регистр.
CompareTo	Позволяет проводить сравнение символов.

Методы могут применяться как к отдельному символу, так и к строке, для которой указывается номер символа для применения метода. Основную группу составляют методы Is, которые используются при разборе строки.

Используя символьный тип можно организовать массив символов и работать с ним на основе базового класса Array:

```
static void PrintArrayChar(string line, Array a)
{
   Console.Write(line);
   foreach (object x in a)
      Console.Write(x);
}
static void Main(string[] args)
{
   char[] M1 = { 'm', 'a', 'X', 'i', 'M', 'u', 'M', ' ', 'ф', 'у', 'н', 'к', 'ц', 'и', 'и' };
   char[] M2 = "Около колодца".ToCharArray();
   //преобразование строки в массив символов
   PrintArrayChar("Исходный массив  M1:", M1);
   for (int x = 0; x < M1.Length; x++)
      if (char.IsLower(M1[x]))
        M1[x] = char.ToUpper(M1[x]);
   ArChar("Измененный массив M1:", M1);
   ArChar("Исходный массив  M2:", M2);
    Array.Reverse(M2);
    ArChar("Измененный массив M2:", M2);
}
```

Результат выполнения

Исходный массив M1: maXiMuM функции
Измененный массив M1: MAXIMUM ФУНКЦИИ
Исходный массив M2: Около колодца
Измененный массив M2: ацдолок олокО

Неизменяемые строки string (класс System.String). Основным типом при работе со строками в C# является тип **string**, задающий строки переменной длины. Над строками определен широкий набор операций.

Каждый объект string - это неизменяемая последовательность символов Unicode, т.е. методы, предназначенные для изменения строк, возвращают измененные копии, исходные же строки остаются неизменными. Конструкторы класса создают строку из:

- символа, повторенного заданное число раз;
- массива символов char[];
- части массива символов

Примеры создания строк:

```
string s1;   // инициализация строки отложена
string s2 = "Около колокола"; // инициализация строковым литералом
string s3 = @"Привет!// символ @ сообщает конструктору, что строку
Сегодня хорошая погода!!!";// нужно воспринимать буквально
                                  // занимает  несколько строк, даже если она
string s4 = new string(' ', 20); // конструктор создает строку из 20 пробелов
int x = 12344556; // инициализация целочисленной переменной
string s5 = x.ToString();           // преобразование ее к типу string
char[] a1 = { 'a', 'b', 'c', 'd', 'e' };// создание массив символов
string s6 = new string(a1);         // создание строки из массива символов
char[] a2 = " Yes ".ToCharArray();// создание массив символов
string s7 = new string(a2);          // создание строки из массива символов
char[] a3 = {'a','b','c','d','e'};//создание строки из части массива символов:
string s8 = new string(a3, 0, 2);  // 0-показывает с какого символа,
        // 2 – сколько символов  использовать для инициализации строки
```

Над строками определены следующие операции:

- присваивание (**=**);
- две операции проверки эквивалентности (**==**) и (**!=**);
- конкатенация или сцепление строк (**+**);
- взятие индекса (**[]**).

Присваивание. **string** - ссылочный тип, поэтому в результате присваивания создается ссылка на константную строку, хранимую в "куче". Со строковой константой в "куче" может быть связано несколько переменных строкового типа. Строковые константы в "куче" не меняются, поэтому когда переменная получает новое значение, она связывается с новым константным объектом в "куче". Остальные переменные сохраняют свои связи и значения.

Эквивалентность. В отличие от других ссылочных типов операции, проверяющие эквивалентность строк, сравнивают значения строк, а не ссылки (операция выполняется как над значимыми типами).

Бинарная операция "+" сцепляет строки, добавляя 2-ю строку к концу 1-й.

Возможность взятия индекса строки отражает факт, что строку можно рассматривать как массив и получать каждый ее символ. Каждый символ строки имеет тип **char**, доступный только для **чтения**, но не для **записи**.

Примеры выполнения операций над строками:

```
{//операции над строками
    string s1 = "РОМ", s2 = "КОД";
    string s3 = s2;
    Console.WriteLine("   Значения: s1 = \"{0}\", s2 = \"{1}\"", s1, s2);
    Console.WriteLine(" Присваивание: s3 = s1 = \"{0}\"", s3);
    bool b1 = (s2 == s3);
    Console.WriteLine(" Эквивалентность: (s2 == s3) = {0}", b1);
    b1 = (s1 == s2);
    Console.WriteLine(" Эквивалентность: (s1 == s2) = {0}", b1);
    b1 = (s3 != s2);
    Console.WriteLine(" Неэквивалентность: (s3 != s2) = {0}", b1);
    b1 = (s1 != s2);
    Console.WriteLine(" Неэквивалентность: (s1 != s2) = {0}", b1);
```

```
        string s4 = s1 + s2 + "рак" + s3;
Console.WriteLine(" Операция сцепления:  s1 + s2 + \"рак\" + s3 = {0}", s4);
        char c1 = s1[1], c2 = s4[8];
 Console.WriteLine("  Взятие индекса:  s1[1] = '{0}', s4[8] = '{1}'", c1, c2);
}
```

Результат выполнения

```
   Значения: s1 = "РОМ", s2 = "КОД"
 Присваивание: s3 = s1 = "КОД"
 Эквивалентность: (s2 == s3) = True
 Эквивалентность: (s1 == s2) = False
 Неэквивалентность: (s3 != s2) = False
 Неэквивалентность: (s1 != s2) = True
 Операция сцепления:  s1 + s2 + "рак" + s3 = “РОМКОДракКОД”
 Взятие индекса:  s1[1] = 'О',     s4[8] = 'к'
```

Свойства и методы класса System.String. В языке C# есть неизменяемые (***immutable***) классы. Для таких классов нельзя изменить значение объекта при вызове его методов. Динамические методы могут создавать новый объект, но не могут изменить значение существующего объекта. К таким неизменяемым классам относится класс **String**. Методы этого класса не меняет значения существующих объектов. Некоторые методы создают новые значения и возвращают в качестве результата новые строки. Аналогично, при работе со строкой как с массивом разрешается только чтение отдельных символов, но не их замена. Оператор присваивания **s1[0] = 'L'** - недопустим.

Название	*Вид*	*Описание*
Empty	Статическое поле	Открытое статическое поле, представляющее пустую строку
Length	Свойство	Возвращает длину строки
Compare	Статический метод	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки с (без) учетом регистра.
CompareTo	Экземплярный метод	Сравнение текущего экземпляра строки с другой строкой.
Concat	Статический метод	Слияние произвольного числа строк.

Название	*Вид*	*Описание*
Copy	Статический метод	Создание копии строки
Format	Статический метод	Форматирование строки в соответствии с заданным форматом
IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Экземплярные методы	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора в данную строку.
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители.
PadLeft, PadRigth	Экземплярные методы	Выравнивают строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки.
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом.
Split	Экземплярный метод	Разделяет строку на элементы, используя разные разделители. Результат помещается в массив строк.
StartWith, EndWith	Экземплярные методы	Возвращают true или false в зависимости от того, начинается или заканчивается строка заданной подстрокой.
Substring	Экземплярный метод	Выделение подстроки, начиная с заданной позиции
ToCharArray	Экземплярный метод	Преобразует строку в массив символов
ToLower, ToUpper	Экземплярные методы	Преобразование строки к нижнему или верхнему регистру
Trim, TrimStart, TrimEnd	Экземплярные методы	Удаление пробелов в начале и конце строки или только с одного ее конца.

Методы Join и Split выполняют над строкой текста взаимно обратные преобразования. *Динамический* метод **Split** позволяет осуществить разбор текста на элементы. *Статический* метод **Join** выполняет обратную операцию - собирает строку из элементов.

Примеры использования методов Split и Join

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace _Строки
{
   class Program
   {
     static void Main(string[] args)
     {//операции над строками
        string Текст = "А это пшеница, которая в темном чулане хранится,"
                       + " в доме, который построил Джек!";
        Console.WriteLine("Исходный текст = (Текст =)");
        Console.WriteLine(" \"{0}\"", Текст);
       Console.WriteLine("Разделение текста на простые предложения:
                          Предложения=Текст.Split(',')");
       string[] Предложения, Слова; //размерность массивов
                    // Предложения и Слова устанавливается в соответствии с
                    // размерностью массива, возвращаемого методом Split
        Предложения = Текст.Split(',');
        for (int i = 0; i < Предложения.Length; i++)
           Console.WriteLine("  Предложения[{0}]= \"{1}\"", i, Предложения[i]);
         Console.WriteLine("  Объединение текста:
                             Текст_Join = string.Join(\",\", Предложения)");
        string Текст_Join = string.Join(",",Предложения);
        Console.WriteLine(" \"{0}\"", Текст_Join);
        Console.WriteLine("Разделение текста на простые слова: Слова =
                           Текст.Split(',',' ')");
        Слова = Текст.Split(',',' ');
        for (int i = 0; i < Слова.Length; i++)
           Console.WriteLine("  Слова[{0}]= \"{1}\"", i, Слова[i]);
        Console.WriteLine("Объединение текста: Текст_Join = string.Join(\" \",
                           Слова)");
        Текст_Join = string.Join(" ", Слова);
        Console.WriteLine(" \"{0}\"", Текст_Join);
     }
   }
}
```

Результат выполнения:

Исходный текст = (Текст =)
"А это пшеница, которая в темном чулане хранится, в доме, который построил Джек!"
Разделение текста на простые предложения: Предложения = Текст.Split(',')
Предложения[0]= "А это пшеница"
Предложения[1]= " которая в темном чулане хранится"
Предложения[2]= " в доме"
Предложения[3]= " который построил Джек!"
Объединение текста: Текст_Join = string.Join(",", Предложения)
"А это пшеница, которая в темном чулане хранится, в доме, который построил Джек!"
Разделение текста на простые слова: Слова = Текст.Split(',', ' ')
Слова[0]= "А"
Слова[1]= "это"
Слова[2]= "пшеница"
Слова[3]= ""
Слова[4]= "которая"
Слова[5]= "в"
Слова[6]= "темном"
Слова[7]= "чулане"
Слова[8]= "хранится"
Слова[9]= ""
Слова[10]= "в"
Слова[11]= "доме"
Слова[12]= ""
Слова[13]= "который"
Слова[14]= "построил"
Слова[15]= "Джек!"
Объединение текста: Текст_Join = string.Join(" ", Слова)
"А это пшеница которая в темном чулане хранится в доме который построил Джек!"

При работе с объектами string нужно учитывать их свойство неизменяемости.

```
string a=""; for (int i = 1; i <= 7; i++)
{
    a += "*"; Console.WriteLine(a + ",");
}
```

Результат выполнения

*, **, ***, ****, *****, ******, *******

Примеры использования свойств и методов класса System. String

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace _Строки
{
   class Program
   {
     static void Main(string[] args)
     {//операции над строками
        string str1 = "Первая строка";
         string str2 = string.Copy(str1);
         string str3 = "Вторая строка";
         string str4 = "ВТОРАЯ строка";
         string strUp, strLow;
         int result, idx;
         Console.WriteLine("Исходная строка: str1 = \"{0}\"", str1);
         Console.WriteLine("Исходная строка: str3 = \"{0}\"", str3);
         Console.WriteLine("Исходная строка: str4 = \"{0}\"", str4);
         Console.WriteLine("Длина строки str1: " + str1.Length);
  Console.WriteLine("Копирование строки: str2=string.Copy(str1)=\"{0}\"", str2);
  strLow = str1.ToLower();// Создание прописную и строчную версии строки
str1.
         strUp = str1.ToUpper();
  Console.WriteLine(" Строчная версия строки str1.ToLower()=\"{0}\"", strLow);
  Console.WriteLine(" Прописная версия строки str1:.ToUpper()=\"{0}\"", strUp);
         result = str1.CompareTo(str3);// Сравнение строк
         Console.WriteLine(" Сравнение строк: result = str1.CompareTo(str3)");
         if (result == 0)
            Console.WriteLine("   str1 и str3 равны.");
         else if (result < 0)
            Console.WriteLine("   str1 меньше, чем str3");
         else
            Console.WriteLine("   str1 больше, чем str3");
    result = String.Compare(str3, str4, true);// Сравнение строк без учета регистра
         Console.WriteLine("Сравнение строк без  уч.рег.:result=
                                String.Compare(str3,str4,true)");
    if (result == 0)    Console.WriteLine("   str3 и str4 равны без учета регистра.");
```

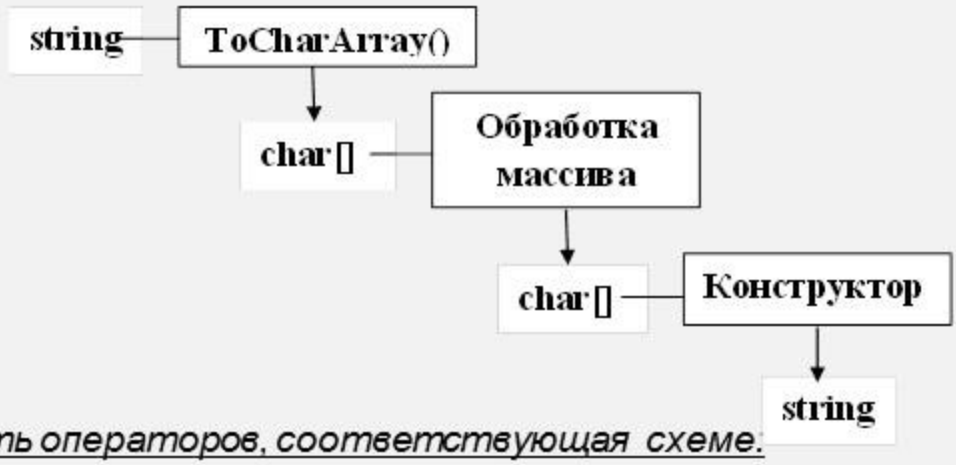
```
else                Console.WriteLine("str3 и str4 не равны без учета регистра.");
        result = String.Compare(str1, 4, str2, 4, 2);// Сравнение части строк
        Console.WriteLine(" Сравнение части строк: result =
                                String.Compare(str1, 4, str2, 4, 2)");
        if (result == 0)
           Console.WriteLine("   Часть str1 и str2 равны");
        else
           Console.WriteLine("   Часть str1 и str2 не равны");
        Console.WriteLine(" Поиск строк: idx = str2.IndexOf(\"строка\")");
        idx = str2.IndexOf("строка");// Поиск строк
        Console.WriteLine("Индекс первого вхождения
                                подстроки:str2.IndexOf(\"строка\")="+idx);
        idx = str2.LastIndexOf("о");
        Console.WriteLine("Индекс последнего вхождения символа
                                str2.LastIndexOf(\"о\")="+idx);
        string str = String.Concat(str1, str2, str3, str4); //конкатенация
  Console.WriteLine("Конкатенация: str = String.Concat(str1, str2, str3, str4) =");
        Console.WriteLine("   " + str);
        str = str.Remove(0, str1.Length);//удаление подстроки
    Console.WriteLine("Удаление подстроки: str = str.Remove(0, str1.Length) =");
        Console.WriteLine("   " + str);
str =str.Replace("строка","");//замена подстроки "строка" на пустую подстроку
 Console.WriteLine("Замена подстроки: str = str.Replace(\"строка\", \"\") =");
        Console.WriteLine("   " + str);
      }
   }
}
```

Результат выполнения:

Исходная строка: str1 = "Первая строка"
Исходная строка: str3 = "Вторая строка"
Исходная строка: str4 = "ВТОРАЯ строка"
Длина строки str1: 13
Копирование строки: str2 = string.Copy(str1) = "Первая строка«
Строчная версия строки str1.ToLower() = "первая строка"
Прописная версия строки str1:.ToUpper() = "ПЕРВАЯ СТРОКА"
Сравнение строк: result = str1.CompareTo(str3)
str1 больше, чем str3
Сравнение строк без учета регистра: result = String.Compare(str3, str4, true)

```
  str3 и str4 равны без учета регистра.
Сравнение части строк: result = String.Compare(str1, 4, str2, 4, 2)
  Часть str1 и str2 равны
Поиск строк: idx = str2.IndexOf("строка")
  Индекс первого вхождения подстроки: str2.IndexOf("строка") = 7
  Индекс последнего вхождения символа str2.LastIndexOf("о") = 10
Конкатенация: str = String.Concat(str1, str2, str3, str4) =
  Первая строкаПервая строкаВторая строкаВТОРАЯ строка
Удаление подстроки: str = str.Remove(0, str1.Length) =
  Первая строкаВторая строкаВТОРАЯ строка
Замена подстроки: str = str.Replace("строка", "") =
  Первая Вторая ВТОРАЯ
```

*Схема преобразовавший объекта класса **string**:*



Последовательность операторов, соответствующая схеме:

```
string row1 = "0123456789";
char[] row2;
row2 = row1.ToCharArray();
Array.Reverse(row2);
row1 = new string(row2);
Console.WriteLine(row1);
```

Результат выполнения

```
9876543210
```

Изменяемые строки StringBuilder (класс System.Text.StringBuilder) Для создания изменяемой строки C# использует класс **StringBuilder**. Объекты класса всегда объявляются с явным вызовом конструктора класса (через операцию **new**). *Синтаксис конструкторов класса StringBuilder:*

public StringBuilder (string str, int cap); Параметр **str** задает строку инициализации, cap - емкость объекта;

public StringBuilder (int curcap, int maxcap); Параметры **curcap** и **maxcap** задают начальную и максимальную емкость объекта;

public StringBuilder (string str, int start, int len, int cap); Параметры **str**, **start**, **len** задают строку инициализации, cap - емкость объекта.

Первая группа задает строку, значением которой будет инициализироваться создаваемый объект класса **StringBuilder**. Вторая группа задает емкость объекта (объем памяти, отводимой экземпляру класса **StringBuilder)**. Каждая из этих групп параметров не является обязательной и может быть опущена (конструктор без параметров создает объект, инициализированный пустой строкой с емкостью по умолчанию, значение которой зависит от реализации).

Примеры создания изменяемых строк:

```
//создание пустой строки, размер по умолчанию 16 символов
StringBuilder a = new StringBuilder();
//инициализация строки и выделение необходимой памяти
StringBuilder b = new StringBuilder("abcd");
//создание пустой строки и выделение памяти под 100 символов
StringBuilder c = new StringBuilder(100);
//инициализация строки и выделение памяти под 100 символов
StringBuilder d = new StringBuilder("abcd", 100);
//инициализация подстрокой "bcd", и выделение памяти под 100 симв.
StringBuilder e = new StringBuilder("abcd", 1, 3, 100);
```

Над строками класса **StringBuilder** определены (как у класса **String**):

- присваивание (=);
- две операции проверки эквивалентности (==) и (!=);
- взятие индекса ([]).

Операция конкатенации (+) не определена над строками **StringBuilder**, ее роль играет метод **Append**, дописывающий новую строку в конец строки. Со строкой этого класса можно работать как с массивом, но, в отличие от класса **String**, здесь допускается не только чтение символа, но и его изменение.

Пример операций над строками класса StringBuilder:

```
StringBuilder s1 = new StringBuilder("ABC"),
s2 = new    StringBuilder("CDE");
StringBuilder s3 = new StringBuilder();
s3 = s1.Append(s2);//s3= s1+s2; Ошибка
bool b1 = (s1 == s3);      char ch1 = s1[0], ch2 = s2[0];
Console.WriteLine("s1={0},s2={1},b1(s1==s3)={2},«
                  +"ch1={3},ch2={4}",  s1,s2,b1,ch1,ch2);
s2 = s1;
b1 = (s1 != s2);
ch2 = s2[0];
Console.WriteLine("s1={0},s2={1},b1(s1!=s2)={2},«
                  +"ch1={3},ch2={4}",s1,s2,b1,ch1,ch2);
StringBuilder s = new StringBuilder("Zenon");
s[0] = 'L';
```

Результат выполнения

```
s1=ABCCDE, s2=CDE, b1(s1==s3)=True,ch1=A, ch2=C
s1=ABCCDE, s2=ABCCDE, b1(s1!=s2)=False,ch1=A, ch2=A
s(new StringBuilder("Zenon");s[0] = 'L';)=Lenon
```

Свойства и методы класса System.Text.StringBuilder

Название	*Вид*	*Описание*
Capacity	Свойство	Получение и установка емкости буфера. Если устанавливаемое значение меньше текущей длины строки или больше максимального, то генерируется исключение **ArgumentOutOfRangeException**

Название	*Вид*	*Описание*
Length	Изменяемое свойство	Возвращает длину строки. Присвоение ему значения 0 сбрасывает содержимое и очищает строку
Chars	Изменяемое свойство	Возвращает из массива или устанавливает в массиве символ с заданным индексом. Вместо него можно пользоваться квадратными скобками []
Append	Экземплярный метод	Добавление данных в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки string.
AppendFormat	Экземплярный метод	Добавление форматированной строки в конец строки
Insert	Экземплярный метод	Вставка подстроки в заданную позицию
MaxCapacity	Неизменяемое свойство	Возвращает наибольшее количество символов, которое может быть размещено в строке (результат один и тот же для всех экземпляров класса)
Remove	Экземплярный метод	Удаление подстроки из заданной позиции
Replace	Экземплярный метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
ToString	Экземплярный метод	Преобразование в строку типа **string**
Equals	Экземплярный метод	Возвращает true, только если объекты имеют одну и ту же длину и состоят из одних и тех же символов
CopyTo	Экземплярный метод	Копирует подмножество символов строки в массив **char**
EnsureCapacity (int capacity)	Экземплярный метод	Метод пытается сначала установить емкость, заданную параметром capacity; если это значение меньше размера хранимой строки, то емкость устанавливается такой, чтобы гарантировать размещение строки. Это число и возвращается в качестве результата работы метода.

Методы класса **StringBuilder** позволяют более эффективно использовать память за счет работы с изменяемыми строками.

```
StringBuilder strbuild = new StringBuilder();//Insert, Append, AppendFormat
string Str = "это  это не ";
strbuild.Append(Str); strbuild.Append(true);
strbuild.Insert(4, false); strbuild.Insert(0, "2*2=5 - ");
Console.WriteLine(strbuild);
string txt = "А это пшеница, которая в темном чулане хранится,
                в доме, который  построил Джек!";
StringBuilder txtbuild = new StringBuilder();
int num = 1;
foreach (string sub in txt.Split(','))
{     txtbuild.AppendFormat(" {0}: {1} ", num++, sub);     }
Str = txtbuild.ToString();
```

Результат выполнения

2*2=5 - это False это не True
1: А это пшеница 2: которая в темном чулане хранится 3: в доме
4: который построил Джек!

Примеры использования свойств и методов класса StringBuilder

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace _Строки
{
   class Program
   {
     static void PrintString(StringBuilder a)
     {
        Console.WriteLine("Строка: " + a);
        Console.WriteLine("  Текущая длина строки " + a.Length);
        Console.WriteLine("  Объем буфера " + a.Capacity);
```

```
        Console.WriteLine(" Максимальный объем буфера " + a.MaxCapacity);
    }
    static void Main(string[] args)
    {
        try
        {
            StringBuilder str = new StringBuilder("Площадь");
            PrintString(str);
            str.Append(" треугольника равна");
            PrintString(str);
            str.AppendFormat(" {0:f2} см ", 123.456);
            PrintString(str);
            str.Insert(8, "данного ");
            PrintString(str);
            str.Remove(7, 21);
            PrintString(str);
            str.Replace("a", "o");
            PrintString(str);
            StringBuilder str1 = new StringBuilder(Console.ReadLine());
            StringBuilder str2 = new StringBuilder(Console.ReadLine());
            Console.WriteLine((" str1.Equals(str2) = ");,str1.Equals(str2));
        }
        catch
        {
            Console.WriteLine("Вознико исключение");
        }
    }
  }
}
```

Результат выполнения:

```
Строка: Площадь
 Текущая длина строки 7
 Объем буфера 16
 Максимальный объем буфера 2147483647
Строка: Площадь треугольника равна
 Текущая длина строки 26
 Объем буфера 32
 Максимальный объем буфера 2147483647
```

```
Текущая длина строки 45
  Объем буфера 72
  Максимальный объем буфера 2147483647
Строка: Площадь равна 123,46 см
  Текущая длина строки 24
  Объем буфера 51
  Максимальный объем буфера 2147483647
Строка: Площодь ровно 123,46 см
  Текущая длина строки 24
  Объем буфера 51
  Максимальный объем буфера 2147483647
```

На практике часто комбинируют одновременное использование изменяемых и неизменяемых строк. *Пример*. Вывести все слова текста, которые начинаются и заканчиваются на одну и ту же букву

```
Console.WriteLine("Введите строку: ");
StringBuilder a = new StringBuilder(Console.ReadLine());
for (int i = 0; i < a.Length; )
   if (char.IsPunctuation(a[i]))
      a.Remove(i, 1);
   else
      ++i;
Console.WriteLine("Исходная строка: " + a);
string st = a.ToString();
string[] S = st.Split(' ');
Console.Write("Найденные слова: ");
for (int i = 0; i < S.Length; ++i)
      Console.Write(" " + S[i]);
```

Результат выполнения

```
Введите строку: /арфа/ [завод] {афиша} .акр. ,варан, !арба!
Исходная строка: арфа завод афиша акр варан арба
Найденные слова:  арфа афиша арба
```