

Массивы на языке C#

Массивы

- **Массивом** называют упорядоченную совокупность элементов одного типа.
- Каждый элемент *массива* имеет один или несколько индексов, определяющих порядок элементов.
- Количество индексов характеризует **размерность массива**. Каждый индекс изменяется в некотором диапазоне от нуля до некоторого числа $N > 0$. Индексы задаются целочисленным типом.
- **Массивы** относятся к ссылочным типам, а следовательно, память им отводится в "куче".
- В языке C# имеются одномерные массивы и многомерные массивы.
- Кроме них, в языке C# также имеется новый тип массивов – **ступенчатый**.

Одномерные массивы

Массивы соответствуют типу `System.Array`

Объявление массива `type [] <имя_массива> ;`

`int[] m;` массив `m`, состоящий из целых чисел,
число и значения элементов не заданы

`double[] x;` массив `x`, состоящий из вещественных чисел,
число и значения элементов не заданы

Создание массива `имя_массива = new type[n];`
`имя_массива = new type[n] {x1, x2, ..., xn};`

`m=new int[5];` создаётся в куче набор из 5 пустых элементов целого
типа `m[0]`, `m[1]`, `m[2]`, `m[3]`, `m[4]`

`x=new double[10];` создаётся в куче набор из 10 пустых элементов.
веществ. типа `x[0]`, `x[1]`, `x[2]`, ..., `x[9]`

`m=new int[5] {1, 2, 0, 5, -7};`
создаётся в куче набор из 5 элементов целого типа
равных указанным числам `m[0]=1`, ..., `m[4]=-7`

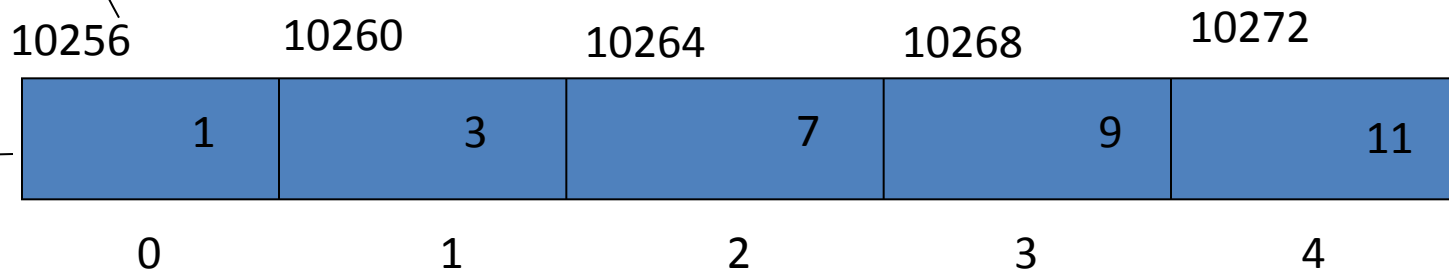
Массив - Array

`int [] a = new int [5];` // можно одним оператором

~~`a[] = 0;`~~ // нельзя использовать массив без индекса

`a[3] = 9;`

`a[0] = 1`



Инициализация массивов

- По умолчанию записывается нуль или `null`.
- Для работы с массивом необходимо заполнить его значения:

```
int[] a = new int [5] {1,2,3,4,5};
```

```
int[] b = {5,7,9};
```

```
char[] s = new char [7] {'s','t','u','d','e','n','t'};
```

```
char[] s = {'s','t','u','d','e','n','t'};
```

Пример разных способов объявления массивов

```
//объявляется одномерный массив A
int[] A = new int[5] {1,2,3,4,5};
//объявление массива x с явной инициализацией
int[] x = {5,5,6,6,7,7};
//объявление массивов с отложенной инициализацией
int[] u,v;
u = new int[3];
```

- Вначале объявляется одномерный массив *A*, создаваемый конструктором. Значения элементов этого массива имеет тип `int`. После такого объявления с инициализацией конструктором, все элементы получают указанные значения и могут участвовать в вычислениях.
- Массив *x* объявлен с явной инициализацией. Число и значения его элементов определяется константным массивом (в скобках `{...}`).
- Массивы *u* и *v* объявлены с отложенной инициализацией. В последующих операторах массив *u* инициализируется в объектном стиле – элементы получают его в цикле значения.

Пример работы с массивами

```
int[] u,v;
```

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

```
for(int i=0; i<3; i++) u[i] = i+1;
```

```
v = new int[4];
```

```
v=u; //допустимое присваивание
```

- Оператора присваивания `v = u` является правильным ссылочным присваиванием: хотя `u` и `v` имеют разное число элементов, но они являются объектами одного класса.
- Теперь обе ссылки `u` и `v` будут теперь указывать на один и тот же *массив*, так что изменение элемента одного *массива* немедленно отразится на другом *массиве*.
- На *массив* `v` теперь никто ссылаться не будет, и он будет считаться мусором, который автоматически удаляется с помощью сборщика мусора.

Использование цикла foreach с массивами

Традиционный способ обработки массивов:

Найти сумму элементов массива

```
int[] ar = { 1, 2, 3, 4, 5 };
```

```
int s=0;
```

```
for (int i = 0; i<ar.Length; i++) s += ar[i];
```


Оператор `foreach`

`foreach`(`<тип>` `<переменная>` `in` `<коллекция>`) оператор

- `<тип>` переменной должен быть согласован с типом элементов, хранящихся в коллекции данных.
- предполагается, что элементы коллекции (массива, коллекции) упорядочены.
- на каждом шаге цикла идентификатор, задающий текущий элемент коллекции, получает значение очередного элемента в соответствии с порядком, установленным на элементах коллекции.
- с этим текущим элементом и выполняется тело цикла - выполняется столько раз, сколько элементов находится в коллекции.
- цикл заканчивается, когда полностью перебраны все элементы коллекции.

Пример использования foreach

Найти сумму элементов массива

```
int[] arr = new int[5] {1,2,3,4,5}
int s = 0;
foreach(int c in arr)
{
    s += c;
}
```

Методы и свойства массивов

Статические методы класса `System.Array`

- `IndexOf()` – индекс заданного элемента в массиве
- `Reverse()` – изменение порядка элементов на обратный
- `Copy()` – создание копии массива
- `Clear()` – удаление всех значений из массива

Свойства массива

- `Rank` – количество размерностей
- `Length` – количество элементов массива (32 битное значение)

Методы массива

- `CopyTo()` – копирование
- `GetLength (n)` – количество элементов в размерности n

Пример

```
int[] At = new int [5] { 1, 2, 3, 4, 5 };  
// изменение одного элемента  
At[2] = 12;  
// перестановка в обратном порядке  
Array.Reverse(At, 0, 5);  
// замена всех элементов нулём  
Array.Clear(At, 0, 5);
```

Многомерные массивы – двухмерные (таблицы)

```
int [,] pp = new int [3,4];
```

```
pp [1,2] = 4;
```

```
int [,] pp = {{1,0,1,0}, {2,3,4,5}, {0,3,0,3}};
```

	0	1	2	3
0	1	0	1	0
1	2	3	4	5
2	0	3	0	3

10256

1	0	1	0	2	3	4	5	0	3	0	3
---	---	---	---	---	---	---	---	---	---	---	---

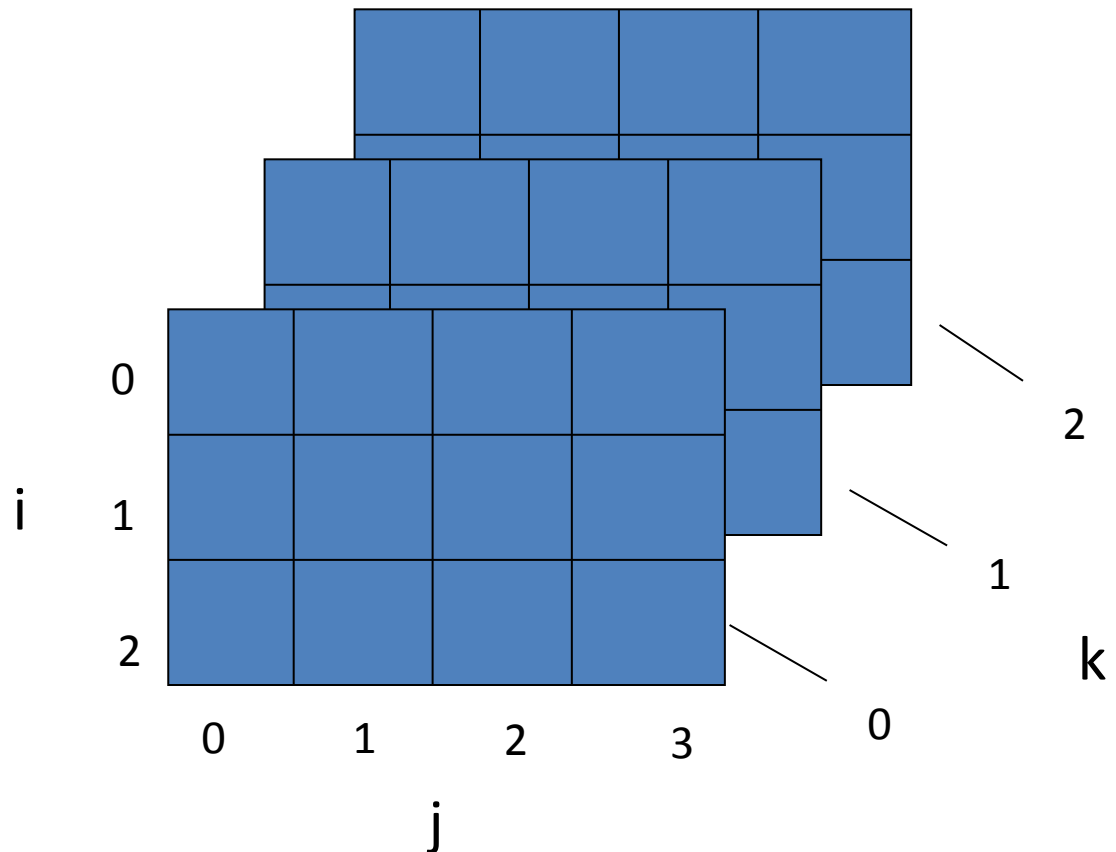
Пример создания двумерного массива

- Пример объявления инициализации двумерного массива:
`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, разделенных запятой. Например:
`int k = ms[1,2]; // получим значение 6`
- Ниже приведен пример использования двумерного массива:
`int s = 0;
for (int i = 0; i < 2; i++)
 for (int j = 0; j < 3; j++)
 s += ms[i, j];`
- В результате выполнения этого кода `s` получит значение 21 – сумму всех элементов массива `ms`.

Многомерные массивы - трехмерные

```
float [, ,] m = new float [3,4,3];
```

```
m [i,j,k] = 3;
```



Ступенчатые массивы

Массив массивов – количество элементов по каждому измерению непостоянно.

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

```
matrix[0] = new int [3];
```

```
matrix[1] = new int [2];
```

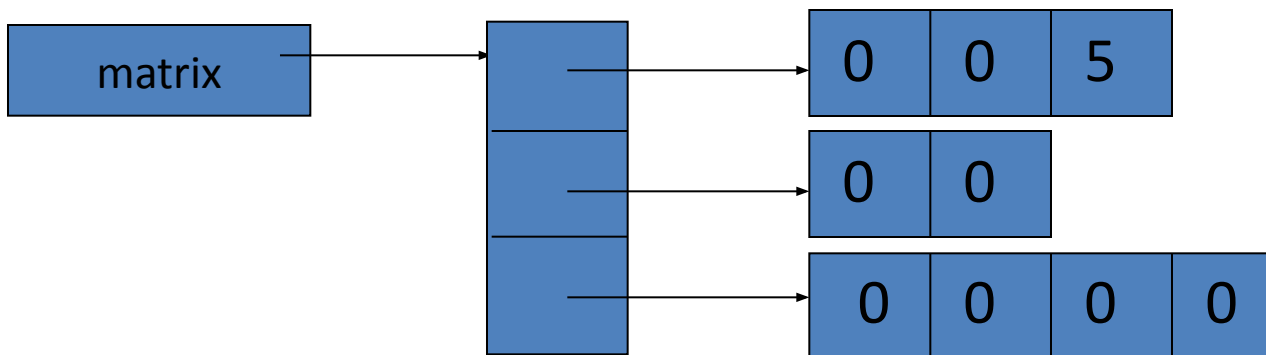
```
matrix[2] = new int [4];
```

ИЛИ

```
int [ ] [ ] matrix = {new int [3], new int [2], new int [4]}
```

...

```
matrix [0] [2] = 5;
```



Ступенчатые массивы

Массив массивов – количество элементов по каждому измерению непостоянно.

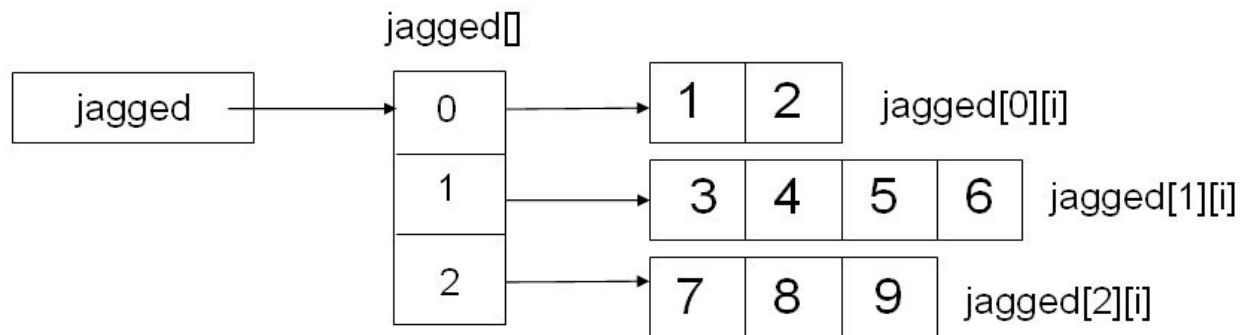
Например:

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

Для создания самих строк массива нужно создать объекты соответствующих типов:

```
jagged[0] = new int[2] { 1, 2 };  
jagged[1] = new int[6] { 3, 4, 5, 6 };  
jagged[2] = new int[3] { 7, 8, 9 };
```

Схема данного ступенчатого массива показана ниже:



Класс System.String (тип string)

- Тип `string` соответствует классу `System.String`

- Создание строк

```
int myInteger = 5;
```

```
string intString = myInteger.ToString( );
```

- **ТЕКСТОВЫЕ КОНСТАНТЫ**

```
string literalOne = "\\MySystem\\ProgrammingC#.cs"
```

```
string verbatimLiteralOne = @"\\MySystem\\ProgrammingC#.cs";
```

Методы класса System.String

- статические методы (string.метод)
 - `Compare()` – сравнение двух строк
 - `Concat()` - объединение
 - `Copy()` - копирование
 - `Format()` – форматирование
- СВОЙСТВА КЛАССА
 - `Length` – количество символов
- методы класса (<строка>.метод)
 - `string Insert(int StartIndex, string value)` – вставляем подстроку в строку;
 - `string Remove(int StartIndex, string value)` – удаляем подстроку из строки;
 - `string Trim()` (`TrimEnd`, `Trim Start`) – удаление пробелов в начале и конце строки;
 - `string ToLower()` (`ToUpper()`) – преобразование символов строки в нижний (верхний) регистр;
 - `string [] Split(params char[] separator)` – разделить строку на массив строк, с использованием заданных разделяющих символов.
 - `bool Contains(string s)` – проверка, содержит ли строка заданную подстроку.

Перегруженные операции класса String

- Операция + (совпадает с `string.Concat()`)

Например:

```
string s1="Привет, ", s2="Мир!", s3;  
s3 = s1 + s2; // s3 = "Привет, Мир!"
```

- Операция += (добавление в конец строки)
- Операции == и != (сравниваются значения, а не ссылки)
- [] получение символа (начиная с 0)

Примеры использования

// сравнение двух строк с учетом регистра

```
result = string.Compare(s1, s2);
```

// сравнение двух строк без учета регистра

```
result = string.Compare(s1,s2, true);
```

// объединение строк

```
string s6 = string.Concat(s1,s2);
```

// использование перегруженного оператора

```
string s7 = s1 + s2;
```

// копирование строки

```
string s8 = string.Copy(s7);
```

Примеры использования

```
// определить длину строки  
n = s9.Length
```

```
// получить номер символа с которого начинается слово  
n = s3.IndexOf("Training");
```

```
// вставка слова excellent начиная со 101 символа  
string s10 = s3.Insert(101,"excellent ");
```

```
// получить часть строки – строку состоящую из 10 первых  
СИМВОЛОВ строки s3  
string s12 = s3.Substring(0, 9);
```

Пример

```
string s1 = "Раз,Два,Три Группа студентов, АВТФ";  
char[ ] delimiters = new char[ ] { ' ', ',' };  
string[ ] strings = s1.Split(delimiters);
```

```
int ctr = 1;  
string output = "";  
foreach (string ss in strings)  
{  
    output += ctr++;  
    output += ": ";  
    output += ss;  
    output += "\n";  
}  
Console.WriteLine(output);
```

Результат:

```
1: Раз  
2: Два  
3: Три  
4: Группа  
5: студентов  
6:  
7: АВТФ
```

Преобразование строк в другие ТИПЫ

С помощью объекта Convert:

```
N = Convert.ToInt32(s1);
```

```
M = Convert.ToDouble(s2);
```

```
F = Convert.ToBoolean(s3);
```

```
B = Convert.ToByte(s4);
```

```
C = Convert.ToChar(k); // по коду
```

```
S = Convert.ToString(x);
```

С помощью метода Parse:

```
N = int.Parse(s1);
```

```
N = int.Parse(Console.ReadLine());
```

```
M = Double.Parse(s2);
```

```
F = bool.Parse(s3);
```


Методы на языке C# (функции)

Методы

Метод представляет собой блок кода, содержащий набор инструкций. В C# все инструкции выполняются в контексте (в теле) метода.

Методы объявляются в классе или в структуре путем указания уровня доступа, возвращаемого значения, имени метода и списка параметров этого метода. Параметры заключаются в круглые скобки и разделяются запятыми. Пустые скобки указывают на то, что у метода нет параметров.

Методы имеют смысл подпрограмм. Если метод не возвращает никаких значений (тип метода **void**) - его применение аналогично *процедуре*.

В случае, если метод возвращает какое-либо значение (тип, отличный от **void**), его использование схоже с использованием *функции*.

Методы

Если тип возвращаемого значения, указываемый перед именем метода, не равен **void**, для возвращения значения используется оператор **return <значение>**, после которого указанное значение нужного типа будет передано объекту, вызвавшему метод. Без этого оператора такой метод является некорректным.

Оператор **return** всегда останавливает выполнение метода.

Если тип возвращаемого значения **void**, инструкция **return** не нужна - выполнение метода завершится, когда будет достигнут конец его блока кода. Но оператор **return;** можно использовать для завершения выполнения метода.

Чтобы использовать возвращаемое методом значение в вызываемом методе, вызов метода можно поместить в любое место кода, где требуется значение соответствующего типа.

Пример использованием возвращаемого значения:

```
static string my_func()
{
    return "Hello world!";
}
static void Main(string[] args)
{
    string s = "*** "+my_func()+" ***";
    Console.WriteLine(s);
}
```

На экране появится

```
*** Hello world! ***
```

Пример с использованием **void**:

```
static void my_proc()
{
    Console.WriteLine( "Hello world!");
}
static void Main(string[] args)
{
    my_proc();
    my_proc();
    my_proc();
}
```

На экране появится
Hello world!

Передача параметров - значений

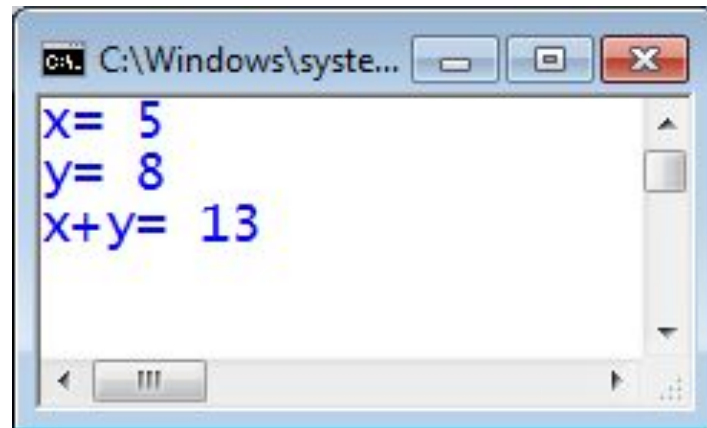
При описании метода можно указать, что ему передаётся набор значений любого типа, указав их список с указанием типа в скобках после имени метода (**формальные параметры**).

```
static void Print(string Msg, int k)
{
    k++;
    Console.WriteLine(Msg + " {0}", k);
}
```

При вызове такого метода следует в скобках указать набор выражений соответствующего типа, значения которых будут переданы в процедуру (**фактические параметры**).

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

```
    int x = 5, y=8;
    Print("x=", x);
    Print("y=", y);
    Print("x+y=", x+y);
    ...
}
```

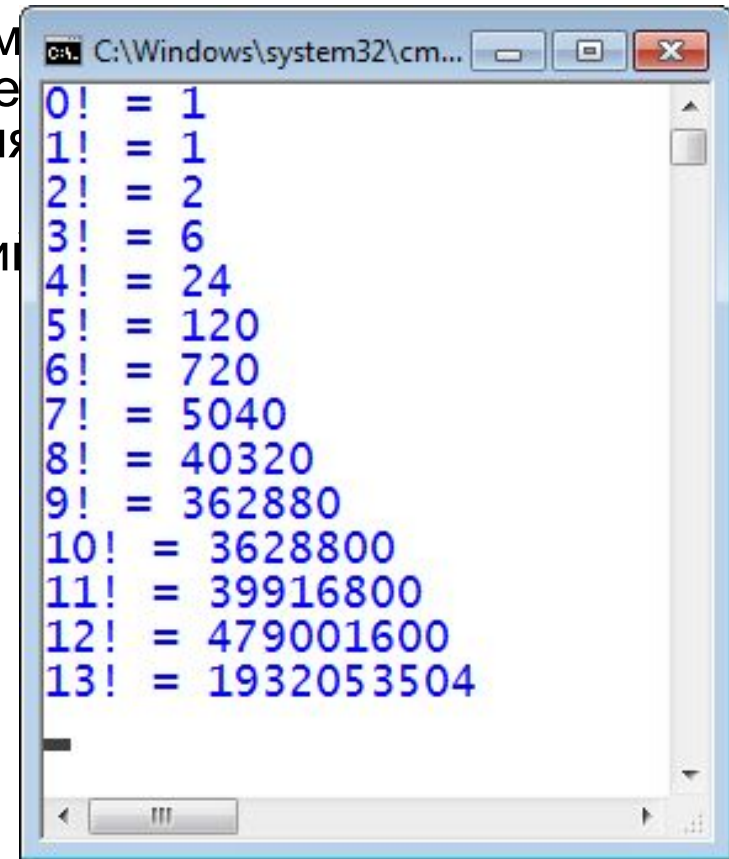


Передача переменной типа значения м
методу **копии** переменной. Любые
выполняемые внутри метода, не влия
хранимые в переменной.

Пример: написать метод вычисляющий
возвращающий это значение.

```
static int factorial (int n)
{
    int i, res = 1;
    for (i = 1; i <= n; i++)
        res = res * i;
    return res;
}
```

```
static void Main(string[] args)
{
    for (int i = 0; i <= 13; i++)
        Console.WriteLine("{0}! = {1}", i, factorial(i));
    Console.ReadKey();
}
```



```
C:\Windows\system32\cm...
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
```

Передача параметров – ссылок

Если требуется, чтобы изменения формального параметра внутри метода привели к аналогичным изменениям фактического параметра, требуется передавать не копии значений, а **ссылки** на фактические параметры. В этом случае фактические параметры должны быть только переменными.

Для передачи параметров по ссылке перед типом таких параметров указываются **ref** или **out**.

Слово **ref** используется, когда значение передаётся от объекта к методу и обратно. При этом значение фактического параметра должно быть определено к моменту вызова метода.

Слово **out** используется, когда значение передаётся только от метода к объекту. При этом значение фактического параметра может быть не определено к моменту вызова метода.

Пример подпрограммы ввода, обработки и вывода массива. "Ввести массив, заменить его элементы факториалами, вывести массив".

```
static void enter(out int[] m1)
{
    Console.Write("Введите число элементов массива N=");
    string s = Console.ReadLine();
    int N = Convert.ToInt32(s);

    m1 = new int[N]; // создаём массив в куче

    for (int i = 0; i < N; i++)
    {
        //вводим i-ый элемент
        Console.Write("Введите {0}-й элемент массива ", i);
        s = Console.ReadLine();
        m1[i] = Convert.ToInt32(s);
    }
}
```

```
static void process(ref int[] m1)
{
    for(int j=0; j<m1.Length; j++)
    {
        int f = 1;
        for (int i=2; i<=m1[j]; i++) f=f*i;
        m1[j] = f;
    }
}
static void output(int[] mas)
{
    Console.WriteLine("Список элементов массива");
    foreach (int i in mas) Console.Write("{0} ", i);
}
static void Main(string[] args)
{
    int[] my_massiv;
    enter(out my_massiv);
    process(ref my_massiv);
    output(my_massiv);
    Console.ReadKey();
}
```

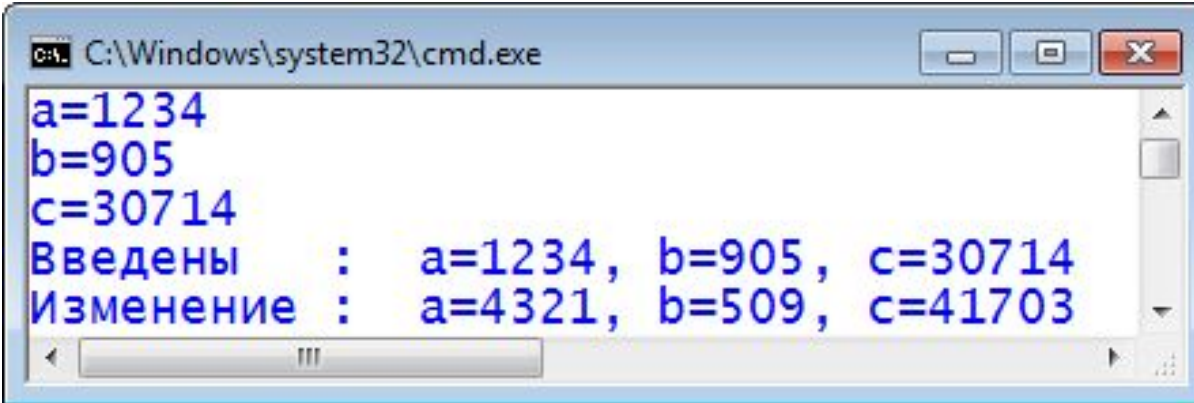
Пример: использование методов для упрощения текста программы.

Требуется ввести три целых числа и вывести их, переставив у каждого цифры в обратном порядке.

```
static int ReadInt(string Msg)
{
    Console.Write(Msg);
    string w = Console.ReadLine();
    int res = Convert.ToInt32(w);
    return res;
}
```

```
static void Obr(ref int k) // переставляет цифры числа k>0
{ // в обратном порядке
    int m=0, d;
    while (k > 0) // пока в числе есть цифры
    {
        d = k % 10; // выделяем последнюю цифру
        k = k / 10; // вырезаем из k посл. цифру
        m = m * 10 + d; // дописываем посл. цифру к m
    }
    k = m; // изменяем переменную-параметр
}
```

```
static void Main(string[] args)
{
    int a = ReadInt("a=");
    int b = ReadInt("b=");
    int c = ReadInt("c=");
    Console.WriteLine("Введены      : a={0}, b={1}, c={2}", a, b, c);
    Obr(ref a);
    Obr(ref b);
    Obr(ref c);
    Console.WriteLine("Изменение   : a={0}, b={1}, c={2}", a, b, c);
}
```



```
C:\Windows\system32\cmd.exe
a=1234
b=905
c=30714
Введены      : a=1234, b=905, c=30714
Изменение   : a=4321, b=509, c=41703
```