МЕТОДЫ

Определения

- Метод — функциональный элемент класса, реализующий вычисления или другие действия. Методы определяют поведение класса и составляют его **интерфейс**.
- Метод — законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо.
- Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

```
double a = 0.1;
double b = Math.Sin(a);
Console.WriteLine(a);
```

Синтаксис метода

[модификаторы] тип_возвращаемого_значения название_метода ([параметры])

{

// тело метода

}

1. Все члены класса - поля, методы, свойства - все они имеют модификаторы (**спецификаторы**) доступа. *Модификаторы доступа* позволяют задать допустимую область видимости для членов класса. То есть контекст, в котором можно употреблять данную переменную или метод.

Спецификаторы: new, **public**, protected, internal, protected internal, private, static, virtual, sealed, override, abstract, extern.

```
static void Print(int[] a)
{
  for (int i = 0; i < a.Length; ++i) Console.Write("{0} ", a[i]);
  Console.WriteLine();
}
```

Спецификаторы методов класса

Спецификатор	Описание
public	Доступ к элементу не ограничен
protected	Доступ только из данного и производных классов
internal	Доступ только из данной сборки
protected internal	Доступ только из данного и производных классов и из данной сборки
private	Доступ только из данного класса

Между модификатором и типом может стоять ключевое слово static, что означает, что функция будет статичной. Из статичной функции можно вызывать другие функции, если они тоже статичные, и обращаться только к статическим полям класса.
Главная функция main – всегда static.

Статический метод вызывается через имя класса, а обычный — через имя экземпляра.

Пример

Пример:

```
class Demo {
    double y;                                  // закрытое поле класса

    public void Sety( double z ) {     // открытый метод класса
        y = z;
    }
}
… Demo x = new Demo();           // где-то в методе другого класса
   x.Sety(3.12); …                       // вызов метода
```

Синтаксис метода

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры])
{
   // тело метода
}
```

2. *Тип_возвращаемого_результата* определяет тип значения, возвращаемого методом. Это может быть любой тип, включая типы классов, создаваемые программистом.

Метод, который возвращает значение, называют функция (возвращает значение, в теле метода присутствует оператор `return`)

Метод, который не возвращает значение, называют процедура (указан тип `void` и в теле метода отсутствует оператор `return`)

```
static int [] Input ()
{
  Console.WriteLine("введите размерность массива");
  int n=int.Parse(Console.ReadLine());
  int []a=new int[n];
  for (int i = 0; i < n; ++i)
  {
    Console.Write("a[{0}]= ", i);
    a[i]=int.Parse(Console.ReadLine());
  }
  return a;
}
```

```
static void Change(int[] a)
{
  for (int i = 0; i < a.Length; ++i)
    if (a[i] > 0) a[i] = -a[i];
}
```

Примеры методов

```
1  using System;
2
3  public class Program
4  {
5      static void Func()  //дополнительный метод
6       {
7          Console.Write("x= ");
8          double x=double.Parse(Console.ReadLine());
9          double y = 1 / x;
10         Console.WriteLine("y({0})={1:f3}", x,y );
11       }
12     public static void Main()
13     {
14         Func(); //первый вызов метода Func
15         Func(); //второй вызов метода Func
16     }
17 }
```

```
x= 5
y(5)=0.200
x= -6
y(-6)=-0.167
```

```
1  using System;
2
3  public class Program
4  {
5      static double Func( double x) //дополнительный метод
6       {
7         return 1 / x; //Возвращаемое значение
8       }
9      public static void Main()
10     {
11        Console.Write("a=");
12        double a=double.Parse(Console.ReadLine());
13        Console.Write("b=");
14        double b=double.Parse(Console.ReadLine());
15        for (double x = a; x <= b; x += 0.5)
16         {
17            double y = Func(x); //вызов метода Func
18            Console.WriteLine("y({0:f1})={1:f2}", x, y);
19         }
20     }
21 }
```

```
a=2
b=6
y(2.0)=0.50
y(2.5)=0.40
y(3.0)=0.33
y(3.5)=0.29
y(4.0)=0.25
y(4.5)=0.22
y(5.0)=0.20
y(5.5)=0.18
y(6.0)=0.17
```

```
using System;

public class Program
{
        static void Func()
    {
      Console.Write("x= ");
      double x=double.Parse(Console.ReadLine());
      double y = 1 / x;
      Console.WriteLine("y({0})={1:f3}", x,y );
     }
        public static void Main()
        {
                Func();
                Func();
        }
}
```

```
using System;
public class Program
{
        static double Func( double x) //дополнительный
метод
    {
      return 1 / x; //Возвращаемое значение
    }
        public static void Main()
        {
          Console.Write("a=");
      double a=double.Parse(Console.ReadLine());
      Console.Write("b=");
      double b=double.Parse(Console.ReadLine());
      for (double x = a; x <= b; x += 0.5)
       {
          double y = Func(x); //вызов метода Func
          Console.WriteLine("y({0:f1})={1:f2}", x, y);
       }
        }
}
```

Синтаксис метода

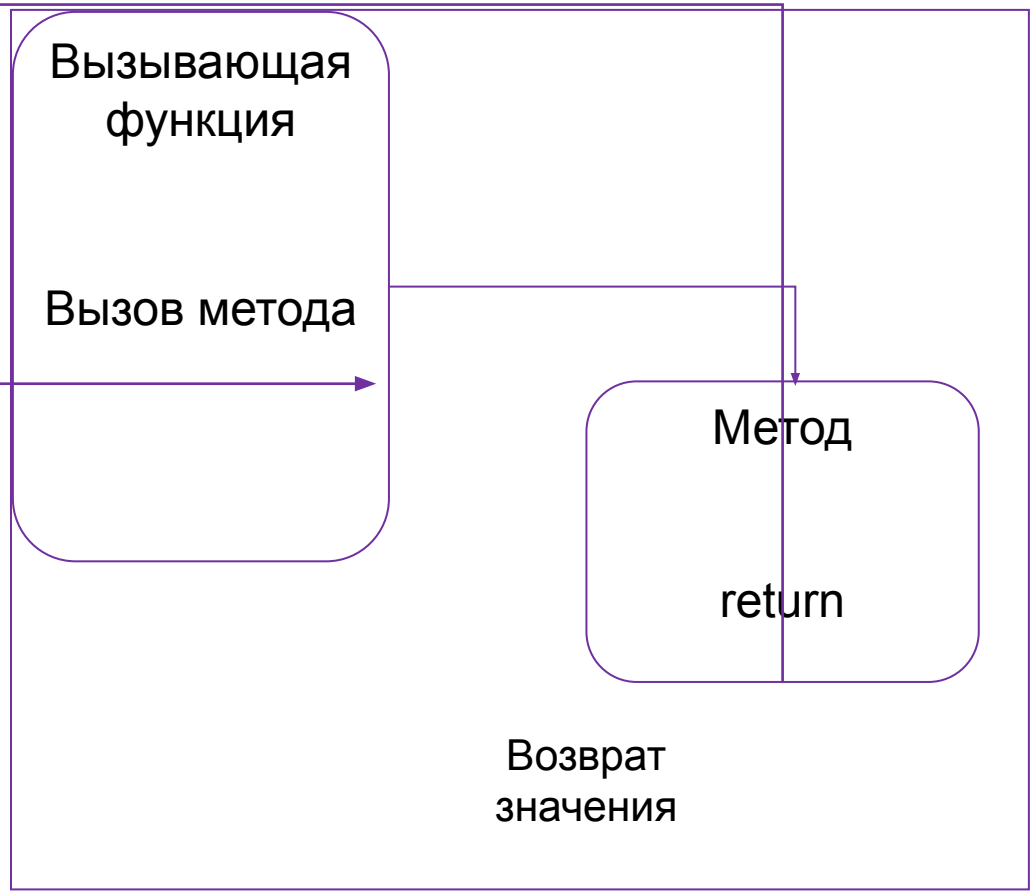
```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры])
{
   // тело метода
}
```

3. *Имя_метода* – идентификатор, заданный программистом с учетом требований, накладываемыми на идентификаторы в C#, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости.

4. *Список_параметров* представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми. Параметры — это переменные или константы, которые получают значения, передаваемые методу при вызове. Если метод не имеет параметров, то *список_параметров* остается пустым.

Примеры методов

```
public void Sety(double z)
{
    y = z;
}
public double Gety()
{
   return y;
}
```



Если метод не принимает параметров, то все равно при вызове после имени метода должны присутствовать пустые круглые скобки (!!!)

```
x.Sety(3.12);
double t = x.Gety();
```

Вызов метода в другом классе

```
1  using System;
2
3  class Demo
4  {
5          static string s = "Hello!";
6          double y;
7          public double Gety() { return y;  }          // метод получения y
8          public void Sety(double y_){  y = y_; }      // метод установки y
9          public static string Gets() { return s; }    // метод получения s
10 }
11 public class Program
12 {
13     public static void Main()
14     {
15         double c = 1.66;
16         Demo x = new Demo();
17         x.Sety(0.12);                    // вызов метода установки y
18         Console.WriteLine(x.Gety());     // вызов метода получения y
19         Console.WriteLine(Demo.Gets());  // вызов метода получения поля s
20         x.Sety(c);
21         Console.WriteLine(x.Gety());
22     }
23 }
```

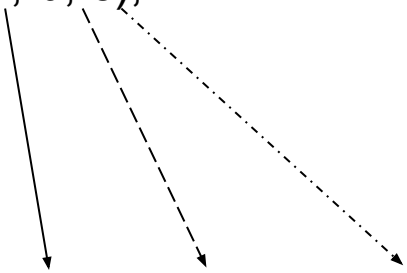
```
0.12
Hello!
1.66
```

```
using System;

class Demo
{
       static string s = "Hello!";
       double y;
       public double Gety() { return y;  }          // метод получения y
       public void Sety(double y_){  y = y_; }      // метод установки y
       public static string Gets() { return s; }    // метод получения s
}
public class Program
{
    public static void Main()
    {
      double c = 1.66;
      Demo x = new Demo();
       x.Sety(0.12);                       // вызов метода установки y
       Console.WriteLine(x.Gety());          // вызов метода получения y
       Console.WriteLine(Demo.Gets());        // вызов метода получения поля s
      x.Sety(c);
      Console.WriteLine(x.Gety());
    }
}
```

Параметры методов

- Параметры определяют множество значений аргументов, которые можно передавать в метод.
- Список аргументов при вызове как бы накладывается на список параметров, поэтому они должны попарно соответствовать друг другу.
- Для каждого параметра должны задаваться его тип, имя и, возможно, вид параметра.

Описание объекта: SomeObj obj = new SomeObj();
Описание аргументов: int b; double a, c;
Вызов метода: obj.P(a, b, c);



Заголовок метода P: public void P(double x, int y, double z);

Пример

Тип возвращаемого результата

Имя метода

Список параметров

Вызов метода

```
1  using System;
2
3  public class Program
4  {
5      static int Func( int x, int y)  //строка 1
6          {
7              return (x>y)? x:y;
8          }
9      public static void Main()
10     {
11        Console.Write("a=");
12             int a = int.Parse(Console.ReadLine());
13             Console.Write("b=");
14             int b = int.Parse(Console.ReadLine());
15             Console.Write("c=");
16             int c = int.Parse(Console.ReadLine());
17             int max = Func(Func(a, b), c);  //строка 2 - вызовы метода Func
18             Console.WriteLine("max({0}, {1}, {2})={3}", a, b, c, max);
19     }
20 }
```

```
a=5
b=-9
c=0
max(5, -9, 0)=5
```

Сигнатура метода

- Имя метода вкупе с количеством, типами и спецификаторами его параметров представляет собой сигнатуру метода — то, по чему один метод отличают от других.
- В классе не должно быть методов с одинаковыми сигнатурами.

Сигнатура складывается из следующих аспектов:

- Имя метода
- Количество параметров
- Типы параметров
- Порядок параметров
- Модификаторы параметров

```
public int Sum(int x, int y)
{
    return x + y;
}
```

Сигнатура данного метода: Sum(int, int)

названия параметров в сигнатуру НЕ входят

Здесь представлены четыре разных версии метода Add, то есть определены четыре перегрузки данного метода.
Первые три версии метода отличаются по количеству параметров. Четвертая версия совпадает с первой по количеству параметров, но отличается по их типу. При этом достаточно, чтобы хотя бы один параметр отличался по типу. Поэтому это тоже допустимая перегрузка метода Add.

Перегрузка метода заключается в том, что методы имеют разную сигнатуру, в которой совпадает только название метода.

Определение метода Add и его перегрузка

```
1  class Calculator
2  {
3      public void Add(int a, int b)
4      {
5          int result = a + b;
6          Console.WriteLine($"Result is {result}");
7      }
8      public void Add(int a, int b, int c)
9      {
10         int result = a + b + c;
11         Console.WriteLine($"Result is {result}");
12     }
13     public int Add(int a, int b, int c, int d)
14     {
15         int result = a + b + c + d;
16         Console.WriteLine($"Result is {result}");
17         return result;
18     }
19     public void Add(double a, double b)
20     {
21         double result = a + b;
22         Console.WriteLine($"Result is {result}");
23     }
24 }
```

```
Add(int, int)
Add(int, int, int)
Add(int, int, int, int)
Add(double, double)
```

сигнатуры методов

Вызов в программе

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Calculator calc = new Calculator();
6          calc.Add(1, 2); // 3
7          calc.Add(1, 2, 3); // 6
8          calc.Add(1, 2, 3, 4); // 10
9          calc.Add(1.4, 2.5); // 3.9
10
11         Console.ReadKey();
12     }
13 }
```

```
Result is 3
Result is 6
Result is 10
Result is 3.9
```

Пример

```
1  int Sum(int x, int y)
2  {
3      return x + y;
4  }
5  int Sum(int number1, int number2)
6  {
7      return x + y;
8  }
9  void Sum(int x, int y)
10 {
11     Console.WriteLine(x + y);
12 }
```

- Сигнатура у всех этих методов будет совпадать: Sum(int, int)
- Поэтому данный набор методов не представляет корректные перегрузки метода Sum и работать не будет.

Вызов метода

1. Вычисляются выражения, стоящие на месте аргументов.
2. Выделяется память под параметры метода.
3. Каждому из параметров сопоставляется соответствующий аргумент. При этом проверяется соответствие типов аргументов и параметров и при необходимости выполняется их преобразование. При несоответствии типов выдается диагностическое сообщение.
4. Выполняется тело метода.
5. Если метод возвращает значение, оно передается в точку вызова; если метод имеет тип void, управление передается на оператор, следующий после вызова.

Пример передачи параметров

```
1  using System;
2  public class Class1
3      {
4      static int Max(int a, int b)
5          {
6              if ( a > b ) return a;
7              else         return b;
8          }
9       public   static void Main()
10         {
11             int a = 2, b = 4;
12             int x = Max( a, b );              // вызов метода Max
13             Console.WriteLine( x );           // результат: 4
14             short t1 = 3, t2 = 4;
15             int y = Max( t1, t2 );            // пар-ры совместимого типа
16             Console.WriteLine( y );           // результат: 4
17             int z = Max( a + t1, t1 / 2 * b ); // выражения
18             Console.WriteLine( z );           // результат: 5
19         }
20     }
```

Пример передачи параметров

```
class Class1
   {   static int Max(int a, int b)      // выбор макс. значения
       {
          if ( a > b ) return a;
          else         return b;
       }
       static void Main()
       {
          int a = 2, b = 4;
          int x = Max( a, b );       // вызов метода Max
          Console.WriteLine( x );                 // результат: 4
          short t1 = 3, t2 = 4;
          int y = Max( t1, t2 );     // пар-ры совместимого типа
          Console.WriteLine( y );                 // результат: 4
          int z = Max( a + t1, t1 / 2 * b );   // выражения
          Console.WriteLine( z );                 // результат: 5
}}
```

Способы передачи параметров и их типы

Способы передачи параметров: по значению и по ссылке.

- *При передаче по значению* метод получает копии значений аргументов, и операторы метода работают с этими копиями.
- *При передаче по ссылке* (*по адресу*) метод получает копии адресов аргументов и осуществляет доступ к аргументам по этим адресам.

В C# четыре типа параметров:

- параметры-значения;
- параметры-ссылки (**ref**);
- выходные параметры (**out**);
- параметры-массивы (**params**).

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением. Пример:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

Пример: параметры-значения и ссылки ref

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int a, ref int b )
        {
            a = 44; b = 33;
            Console.WriteLine( "внутри метода {0} {1}", a, b );
        }
        static void Main()
        {
            int a = 2, b = 4;
            Console.WriteLine( "до вызова     {0} {1}", a, b );
            P( a, ref b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Внимание! Для ссылочных типов данных не надо использовать **ref**

Результат работы программы:
до вызова 2 4
внутри метода 44 33
после вызова 2 33

Пример: параметры-значения и ссылки ref

- Что будет выведено в консоль?

```
1  using System;
2
3  public class Program
4  {
5      public static void Main()
6      {
7          int a = 5;
8          Console.WriteLine("Начальное значение переменной a = {0}", a);
9          IncrementVal(a);
10         Console.WriteLine("Переменная a после передачи по значению равна = {0}",a);
11     }
12     static void IncrementVal(int x)
13     {
14         x++;
15         Console.WriteLine("IncrementVal: {0}",x);
16     }
17 }
```

```
Начальное значение переменной a = 5

IncrementVal: 6

Переменная a после передачи по значению равна = 5
```

Пример: параметры-значения и ссылки ref

- Что будет выведено в консоль?

```
1  using System;
2
3  public class Program
4  {
5      public static void Main()
6      {
7          int a = 5;
8          Console.WriteLine("Начальное значение переменной a = {0}", a);
9          IncrementRef(ref a);
10         Console.WriteLine("Переменная a после передачи по значению равна = {0}",a);
11     }
12     static void IncrementRef(ref int x)
13     {
14         x++;
15         Console.WriteLine("IncrementVal: {0}",x);
16     }
17 }
```

```
Начальное значение переменной a = 5

IncrementRef: 6

Переменная a после передачи по ссылке равна = 6
```

Пример: выходные параметры out

```
1  using System;
2
3
4  public class Program
5   {
6      static void Sum(int x, int y, out int a)
7      {
8        a = x + y;
9      }
10     public static void Main()
11     {
12       int x = 10;
13       int z;
14       Sum(x, 15, out z);
15       Console.WriteLine(z);
16     }
17 }
18
19
```

результат возвращается не через оператор return, а через выходной параметр a

25

```
static void Sum(int x, int y, out int a)
{
    Console.WriteLine(x+y);
}
```

Методы, использующие выходные параметры out, обязательно должны присваивать им определенное значение.

Пример: выходные параметры out

Можно вернуть из метода не одно значение, а несколько.

```
1  using System;
2
3  public class Program
4  {
5      static void GetData(int x, int y, out int area, out int perim)
6      {
7          area= x * y;
8          perim= (x + y)*2;
9      }
10     public static void Main()
11     {
12         int x = 10;
13
14
15         GetData(x, 15, out area, out perimetr);
16         Console.WriteLine("Площадь : " + area);
17         Console.WriteLine("Периметр : " + perimetr);
18     }
19 }
20
21
```

```
Площадь : 150
Периметр : 50
```

Начиная с версии C# 7.0 можно определять переменные в непосредственно при вызове метода.

Ключевое слово **out** применяется для передачи аргументов по ссылке

Пример: выходные параметры out

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void P( int x, out int y )
        {
            x = 44; y = 33;
            Console.WriteLine( "внутри метода {0} {1}", x, y );
        }
        static void Main()
        {
            int a = 2, b;         // инициализация b не требуется

            P( a, out b );
            Console.WriteLine( "после вызова  {0} {1}", a, b );
}}}
```

Результат работы программы:

внутри метода 44 33
после вызова 2 33

Пример: передача массива

```
1  using System;
2
3  public class Program
4  {
5      class myClass
6      {
7          public void someMethod(double[] myArr, int i )
8          {
9              myArr[0] = 12.0;
10             i = 12;
11         }
12     }
13     public static void Main()
14     {
15         double[] arr1 = { 0, 1.5, 3.9, 5.1 };
16             int i = 0;
17             Console.WriteLine("Массив arr1 до вызова метода: ");
18             foreach (double d in arr1)
19                 Console.Write("{0}\t",d);
20             Console.WriteLine("\nПеременная i = {0}\n",i);
21
22             Console.WriteLine("Вызов метода someMethod ...");
23             myClass ss = new myClass();
24             ss.someMethod(arr1,i);
25             Console.WriteLine("Массив arr1 после вызова метода:");
26             foreach (double d in arr1)
27                 Console.Write("{0}\t",d);
28             Console.WriteLine("\nПеременная i = {0}\n",i);
29     }
30 }
```

Измененные значения в myArr также изменились в исходном массиве arr1, так как массивы являются ссылочными типами

```
Массив arr1 до вызова метода:
0       1,5     3,9     5,1
Переменная i = 0

Вызов метода someMethod ...
Массив arr1 после вызова метода:
12      1,5     3,9     5,1
Переменная i = 0
```

значение i осталось неизменным

Использование ключевого слова params

- Используя ключевое слово **params**, можно передавать неопределенное (переменное) количество параметров.

```
1  using System;
2
3  public class Program
4  {
5      static void Addition(params int[] integers)
6  {
7      int result = 0;
8      for (int i = 0; i < integers.Length; i++)
9      {
10         result += integers[i];
11     }
12     Console.WriteLine(result);
13 }
14     public static void Main()
15     {
16         Addition(1, 2, 3, 4, 5);
17
18     int[] array = new int[] { 1, 2, 3, 4 };
19     Addition(array);
20
21     Addition();
22     }
23 }
```

При объявлении метода ключевое слово params можно использовать только один раз

Передача отдельных значений

```
15
10
0
```

Передача массива значений

Вызов метода без параметров

```
using System;

public class Program
{
        static void Addition(params int[] integers)
{
   int result = 0;
   for (int i = 0; i < integers.Length; i++)
   {
      result += integers[i];
   }
   Console.WriteLine(result);
}
        public static void Main()
        {
                Addition(1, 2, 3, 4, 5);

   int[] array = new int[] { 1, 2, 3, 4 };
   Addition(array);

   Addition();
   }
}
```

Использование ключевого слова params

- Использование ключевого слова **params** указывает на то, что параметр, передаваемый методу, состоит из одного или более значений. Все эти значения имеют один и тот же тип.
- При объявлении метода ключевое слово **params** можно использовать только один раз
- Если же надо передать какие- то другие параметры, то они должны указываться до параметра с ключевым словом **params**

```
//Так работает
static void Addition( int x, string mes, params int[] integers)
{}
```

```
// Вызов метода
Addition(2, "hello", 1, 3, 4);
```

```
//Так НЕ работает
static void Addition(params int[] integers, int x, string mes)
{}
```

ↄ слова params

```
1 2 3 4
1 a test

5 6 7 8 9
2 b test again
```

```
using System;
public class MyClass
    {
        public static void UseParams(params int[] list)
        {
            for (int i = 0; i < list.Length; i++)
            {
                Console.Write(list[i] + " ");
            }
            Console.WriteLine();
        }

        public static void Us
        {
            for (int i = 0; i
            {
                Console.Write
            }
            Console.WriteLine
        }
```

```
    public static void Main()
    {
            // Можно отправить разделенный запятыми список
            //аргументов указанного типа.
            UseParams(1, 2, 3, 4);
            UseParams2(1, 'a', "test");

            // Следующая инструкция вызова отображает
            //только пустую строку.
            UseParams2();

            // Аргумент массива может быть передан, если
            //тип массива соответствует типу параметра вызываемого метода.
            int[] myIntArray = { 5, 6, 7, 8, 9 };
            UseParams(myIntArray);

            object[] myObjArray = { 2, 'b', "test", "again" };
            UseParams2(myObjArray);

            // Следующий вызов вызывает ошибку компилятора,
            //так как object массив не может быть преобразован в массив integer.
            UseParams(myObjArray);


    }
```

```
using System;
public class MyClass
   {
      public static void UseParams(params int[] list)
      {
         for (int i = 0; i < list.Length; i++)
         {
            Console.Write(list[i] + " ");
         }
         Console.WriteLine();
      }

      public static void UseParams2(params object[] list)
      {
         for (int i = 0; i < list.Length; i++)
         {
            Console.Write(list[i] + " ");
         }
         Console.WriteLine();
      }

          public static void Main()
          {
                        // Можно отправить разделенный запятыми список
                        //аргументов указанного типа.
         UseParams(1, 2, 3, 4);
         UseParams2(1, 'a', "test");

         // Следующая инструкция вызова отображает
                        //только пустую строку.
         UseParams2();

         // Аргумент массива может быть передан, если
                        //тип массива соответствует типу параметра вызываемого метода.
         int[] myIntArray = { 5, 6, 7, 8, 9 };
         UseParams(myIntArray);

         object[] myObjArray = { 2, 'b', "test", "again" };
         UseParams2(myObjArray);

         // Следующий вызов вызывает ошибку компилятора,
                        //так как object массив не может быть преобразован в массив integer.
         //UseParams(myObjArray);
```

Правила применения параметров

1. Для параметров-значений используется передача по значению. Этот способ применяется для исходных данных метода.

- При вызове метода на месте параметра, передаваемого по значению, может находиться выражение (а также его частные случаи — переменная или константа). Должно существовать неявное преобразование типа выражения к типу параметра.

```
static int Max(int a, int b)
    {
        if ( a > b ) return a;
        else         return b;
    }

  public static void Main()
  {
          int a = 2, b = 4;
          int x = Max( a, b );
          Console.WriteLine( x );
          short t1 = 3, t2 = 4;
          int y = Max( t1, 5 );
          Console.WriteLine( y );
          int z = Max( a + t1, t2 / 2 * b );
          Console.WriteLine( z );

  }
```

Правила применения параметров

2. Параметры-ссылки и выходные параметры передаются по адресу. Этот способ применяется для передачи побочных результатов метода.

- При вызове метода на месте параметра-ссылки **ref** может находиться только имя инициализированной переменной точно того же типа. Перед именем параметра указывается ключевое слово ref.
- При вызове метода на месте выходного параметра **out** может находиться только имя переменной точно того же типа. Ее инициализация не требуется. Перед именем параметра указывается ключевое слово out.

```
class Program
{
    static void AddValue(ref int value)
    {
        value = 5; // присваиваем новое значение
    }

    static void Main(string[] args)
    {
        int price = 7; // обязательно нужно инициализировать

        AddValue(ref price);

        Console.WriteLine(price);
        // на экране увидим 5
    }
}
```

```
class Program
{
    static void AddValue(out int value)
    {
        value = 5; // обязательно нужно присвоить значение
    }

    static void Main(string[] args)
    {
        int price; // можно не инициализировать

        AddValue(out price);

        Console.WriteLine(price);
        // на экране увидим 5
    }
}
```

РЕКУРСИВНЫЕ МЕТОДЫ

Определение

- Рекурсивным называют метод, если он вызывает сам себя в качестве вспомогательного.
- Классическим примером *рекурсивного метода* является метод, вычисляющий *факториал*.

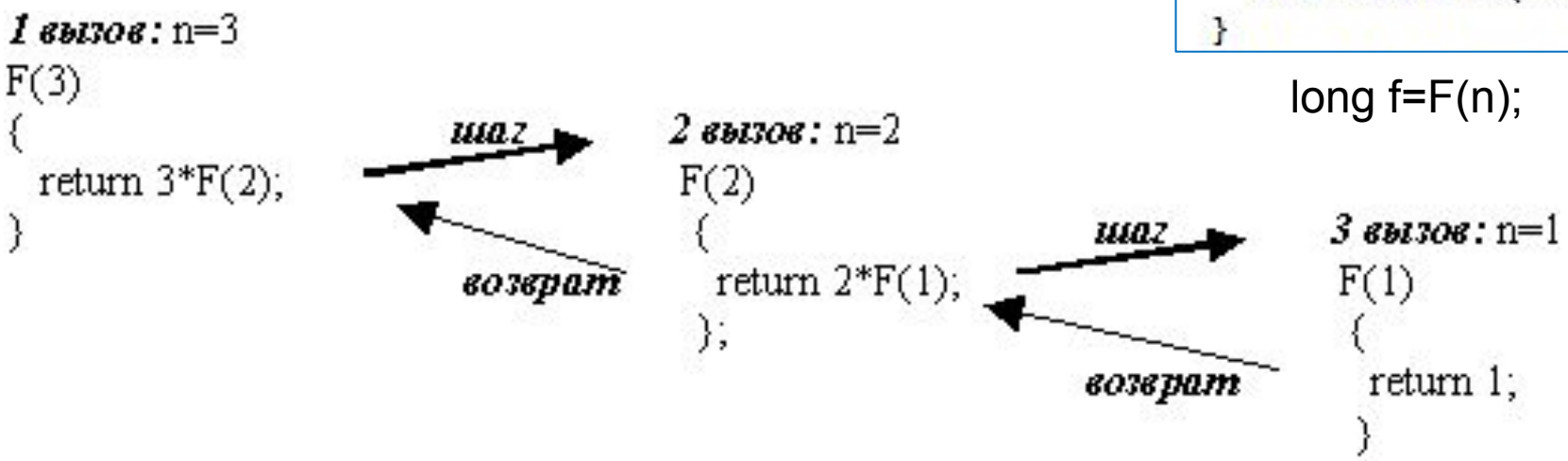
```
using System;

public class Program
{
     static long F(int n)       //рекурсивный метод
   {
      if (n==0 || n==1)
       return 1;                //нерекурсивная ветвь
       else return n*F(n-1);    //шаг рекурсии - повторный вызов метода с другим параметром
    }
    public static void Main()
    {
       Console.Write("n=");
       int n =int.Parse( Console.ReadLine());
       long f=F(n);             //нерекурсивный вызов метода F
       Console.WriteLine("{0}!={1}",n, f);
    }
}
```

Вызов рекурсивного метода

Рассмотрим работу *рекурсивного метода,* вычисляющего *факториал,* для n=3.

```
static long F(int n)
{
  if (n==0 || n==1)
   return 1;
   else return n*F(n-1);
}
```

long f=F(n);



Первый *вызов метода* осуществляется из метода `Main` командой `f=F(3)`. Этап вхождения в рекурсию обозначим жирными стрелками. Он продолжается до тех пор, пока *значение* переменной `n` не становится равной `1`. После этого начинается *выход* из рекурсии (тонкие стрелки). В результате вычислений получается, что `F(3)=3*2*1`.

Прямая рекурсия

Рассмотренный вид рекурсии называют прямой.

Метод с прямой рекурсией обычно содержит следующую структуру:

if (<условие>)

<оператор>;

else <вызов данного метода с другими параметрами>;

В качестве <условия> обычно записываются некоторые граничные случаи параметров, передаваемых рекурсивному методу, при которых результат его работы заранее известен, поэтому далее следует простой оператор или блок, а в ветви else происходит рекурсивный вызов данного метода с другими параметрами.

```
if (n==0 || n==1)
 return 1;
 else return n*F(n-1);
```

Косвенная рекурсия

- В косвенной *рекурсии* метод вызывает себя в качестве вспомогательного не непосредственно, а через другой вспомогательный метод.
- Метод **P** будет методом косвенной рекурсии, если в теле P вызывается метод **Q** (эта цепочка может быть продолжена), в теле которого вызывается метод **P**.

Косвенную рекурсию демонстрирует следующая программа которая для заданного значения n выводит на экран следующее за ним простое число.

Пример косвенной рекурсии

```
static bool Prim (int j)
    {
        int k=2;                    //первое простое число
        while (k*k<=j && j%k!=0)
            k=NextPrim(k);          //вызов метода NextPrim
        return (j%k==0)?false:true;
    }
  static int NextPrim(int i)
      {
          int p=i+1;
          while (!Prim(p))
              ++p;
          return p;
      }
public static void Main()
{
   Console.Write("n=");
        int n=int.Parse(Console.ReadLine());
        Console.WriteLine("Следующее за {0} простое число равно {1}.", n, NextPrim(n));
}
```

метод `Prim` возвращает `true`, если его *параметр* является простым числом, `false` – в противном случае

метод `NextPrim` возвращает следующее за значением ее параметра *простое число* метод `NextPrim` обращается к методу `Prim` для того, чтобы определить является ли заданное число простым.

Пример косвенной рекурсии

- Данная *программа* содержит метод Prim, который возвращает true, если его *параметр* является простым числом, false – в противном случае.
- Чтобы установить, является ли число j простым, нужно проверить делимость числа j на все простые числа, не превышающие квадратный корень из j. Перебор таких простых чисел можно организовать так: рассмотреть первое *простое число* – 2, а затем, используя метод NextPrim, возвращающий следующее за значением ее параметра *простое число*, получить все простые числа, не превышающие квадрата числа j.
- В свою *очередь* метод NextPrim обращается к методу Prim для того, чтобы определить является ли заданное число простым.
- Таким образом методы Prim и NextPrim перекрестно вызывают друг друга. В этом и проявляется косвенная *рекурсия*.