Методы

Метод

- Метод – это функциональный элемент класса, который реализует вычисления или другие действия, выполняемые классом или его экземпляром (объектом).
- Метод представляет собой законченный фрагмент кода, к которому можно обратиться по имени.
- Он описывается один раз, а вызываться может многократно.
- Совокупность методов класса определяет, что конкретно может делать класс.

Синтаксис метода:

```
[атрибуты] [спецификторы] тип_возвращаемого_результаты
имя_метода ([список_параметров])
 {
    тело_метода;
     return значение
}
```

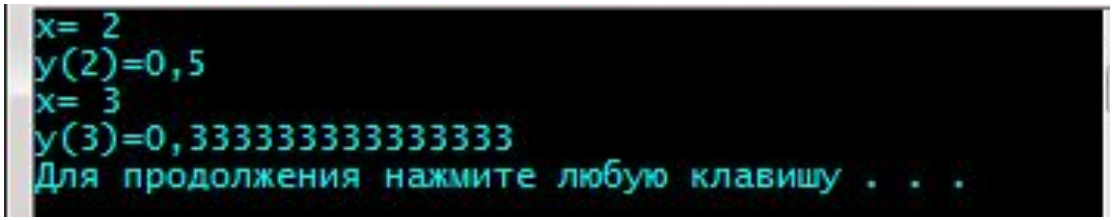
где:

1. ***Тип_возвращаемого_результата*** определяет тип значения, возвращаемого методом. Это может быть любой тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип void (в этом случае в теле метода отсутсвует оператор return).

2. ***Список_параметров*** представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми.

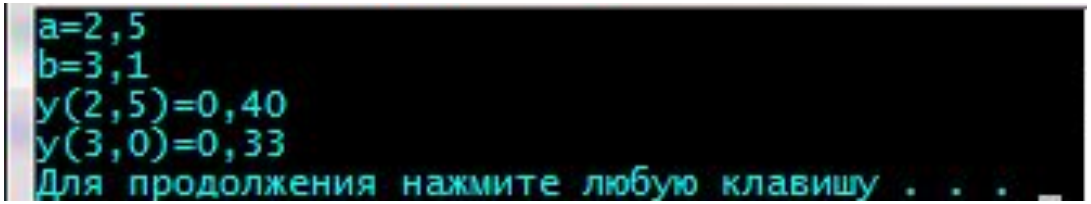
3. ***Значение*** определяет значение, возвращаемое методом. Тип значения должен соответствовать *типу_возвращаемого_результата* или приводится к нему.

```
class Program    {
      static void Func()          //дополнительный метод
      {
         Console.Write("x= ");
         double x = double.Parse(Console.ReadLine());
         double y = 1 / x;
         Console.WriteLine("y({0})={1}", x, y);
      }
      static void Main()          //точка входа в программу
      {
         Func();          //первый вызов метода Func
         Func();          //второй вызов метода Func
      }
   }
```

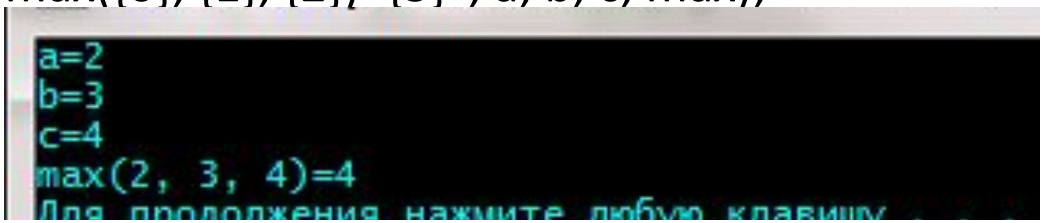


```
class Program    {
      static double Func(double x)     //дополнительный метод
      {
         return 1 / x;              //Возвращаемое значение
      }
      static void Main()         //точка входа в программу
      {
         Console.Write("a=");
         double a = double.Parse(Console.ReadLine());
         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);
         }
      }

   }
```



```
class Program    {
      static int Func(int x, int y)          /*указываются формальные
параметры, а в метод передаются фактические параметры,
которые по количеству и по типу совпадают с формальными
параметрами*/
      {
         return (x > y) ? x : y;
      }
      static void Main()       {
         Console.Write("a=");
         int a = int.Parse(Console.ReadLine());
         Console.Write("b=");
         int b = int.Parse(Console.ReadLine());
         Console.Write("c=");
         int c = int.Parse(Console.ReadLine());
         int max = Func(Func(a, b), c);//строка 2 - вызовы метода Func
         Console.WriteLine("max({0}, {1}, {2})={3}", a, b, c, max);
      }
   }
```



В C# для обмена предусмотрено четыре типа параметров:

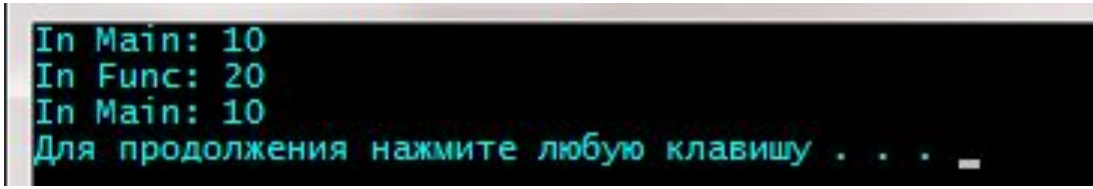
- *параметры-значения,*
- *параметры-ссылки,*
- *выходные параметры,*
- *параметры-массивы.*

При передаче параметра *по значению* метод получает копии параметров, и операторы метода работают с этими копиями

```
class Program
  {
    static void Func(int x)
    {
      x += 10; // изменили значение параметра
      Console.WriteLine("In Func: " + x);
    }

    static void Main()
    {
      int a = 10;
      Console.WriteLine("In Main: " + a);
      Func(a);
      Console.WriteLine("In Main: " + a);
    }
  }
```

значение формального параметра x было изменено в методе Func, но эти изменения не отразились на фактическом параметре *a* метода Main.



При передаче параметров *по ссылке* метод получает копии адресов параметров, что позволяет осуществлять доступ к ячейкам памяти по этим адресам и изменять исходные значения параметров(*ref*)

```
class Program
  {
    static void Func(int x, ref int y)
    {
      x += 10; y += 10;          //изменение параметров
      Console.WriteLine("In Func: {0}, {1}", x, y);
     }

    static void Main()
    {
      int a=10, b=10;  // строка 1
      Console.WriteLine("In Main: {0}, {1}", a, b);
      Func(a, ref b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
    }
  }
```

изменения не отразились на фактическом параметре a, т.к. он передавался по значению, но значение b было изменено, т.к. он передавался по ссылке

```
In Main: 10, 10
In Func: 20, 20
In Main: 10, 20
Для продолжения нажмите любую клавишу .
```

Если невозможно инициализировать параметр до вызова метода. Тогда параметр следует передавать как выходной, используя спецификатор *out*

```
class Program
  {
    static void Func(int x, out int y)
    {
      x += 10; y = 10;      // определение значения выходного параметра y
      Console.WriteLine("In Func: {0}, {1}", x, y);
     }

    static void Main()
    {
      int a=10, b;
      Console.WriteLine("In Main: {0}", a);
      Func(a, out b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
    }
  }
```

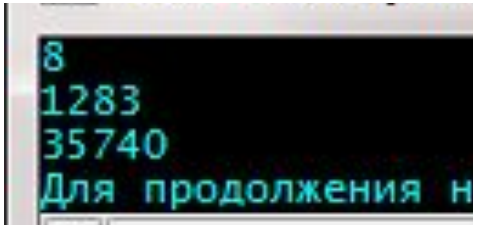
в методе Func формальный параметр y и соответствующий ему фактический параметр b метода Main помечены спецификатором out, поэтому значение b до вызова метода Func можно было не определять, но изменение параметра y отразились на изменении значения параметра b

Перегрузка методов

- Использование нескольких методов с одним и тем же именем, но различными типами и количеством параметров называется *перегрузкой методов.*

- Компилятор определяет, какой именно метод требуется вызвать, по типу и количеству фактических параметров

```
class Program    {
      static int max(int a)          //первая версия метода max
      {
         int b = 0;
         while (a > 0)
         {
            if (a % 10 > b) b = a % 10;
            a /= 10;          }
         return b;       }
      static int max(int a, int b)  //вторая версия метода max
      {
         if (a > b) return a;
         else return b;       }
            static int max(int a, int b, int c)  //третья версия метода max
      {          if (a > b && a > c) return a;
                 else if (b > c) return b;
                 else return c;       }
       static void Main()       {
         int a = 1283, b = 45, c = 35740;
         Console.WriteLine(max(a));
         Console.WriteLine(max(a, b));
         Console.WriteLine(max(a, b, c));       }    }
```



- При вызове метода компилятор выбирает вариант, соответствующий типу и количеству передаваемых в метод аргументов.
- Если точного соответствия не найдено, выполняются неявные преобразования типов в соответствии с общими правилами.
- Если преобразование невозможно, выдается сообщение об ошибке.
- Если выбор перегруженного метода возможен более чем одним способом, то выбирается «лучший» из вариантов (вариант, содержащий меньшие количество и длину преобразований в соответствии с правилами преобразования типов).
- Если существует несколько вариантов, из которых невозможно выбрать лучший, выдается сообщение об ошибке.

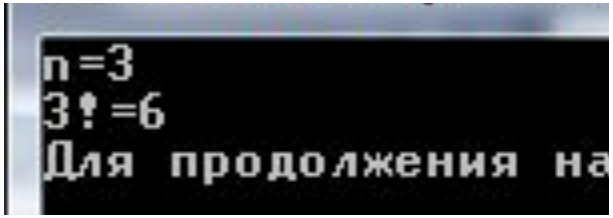
Рекурсивные методы

- Рекурсивным называют метод, если он вызывает сам себя в качестве вспомогательного

Прямая рекурсия

```
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
   }

   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);
    }
}
```



Метод с прямой рекурсией

if (<условие>)

<оператор>;

else <вызов данного метода с другими параметрами>;

- Со входом в рекурсию осуществляется вызов метода, а для выхода необходимо помнить точку возврата, т.е. то место программы откуда мы пришли и куда нам нужно будет возвратиться после завершения метода.

- Место хранения точек возврата называется стеком вызовов и для него выделяется определенная область оперативной памяти.

- В этом стеке запоминаются не только адреса точек возврата, но и копии значений всех параметров.

- По этим копиям восстанавливается при возврате вызывающий метод. При развертывании рекурсии за счет создания копий параметров возможно переполнение стека(недостаток рекурсивного метода).

Косвенная рекурсия

- метод вызывает себя в качестве вспомогательного не непосредственно, а через другой вспомогательный метод

Косвенная рекурсия

```
class Program      {
        static bool Prim (int j)          {
                int k=2;       //первое простое число
                        //значение k «пробегает» последовательность простых чисел,
                        //начиная с 2 до корня из j, при этом проверяется делится ли j на
                        // одно из таких простых чисел
                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))   //вызов метода Prim
                        ++p;
                return p;          }

        static void Main(){
                Console.Write("n=");
                int n=int.Parse(Console.ReadLine());
                Console.WriteLine("Следующее за {0} простое число равно {1}.", n, NextPrim(n));
        }
      }
```