Функции

Тема 6

Функции (методы)

- **Функция** – это именованная последовательность описаний и операторов, выполняющая законченное действие, например:
  - формирование массива,
  - печать массива
  - сортировка массива,
  - и т. д.
- В хорошей программе одна функция выполняет только **одну** задачу.

Преимущества использования функций

- Разделение задачи на функции позволяет сократить сложность задачи.
- Разделение задачи на функции позволяет избежать избыточности кода, т. к. функцию записывают один раз, а вызывают многократно.
- Можно использовать одну функцию в разных задачах, т.е. сокращается время работы программиста и повышается качество работы, т.к. эта функция уже будет работать без ошибок.
- Программу, которая содержит функции, легче отлаживать, т.к можно сначала протестировать каждую функцию отдельно (простые задачи), а затем уже интегрировать их в сложную задачу.

Упрощенный формат записи функции

```
тип имя_функции([список_формальных
    параметров])
{
    тело_функции
}
```

Возврат значения из функции

- В теле функции должен быть оператор, который **возвращает** полученное значение функции в точку вызова. Он может иметь две формы:
- return выражение; //тип результата
- return;//тип void

Параметры функции

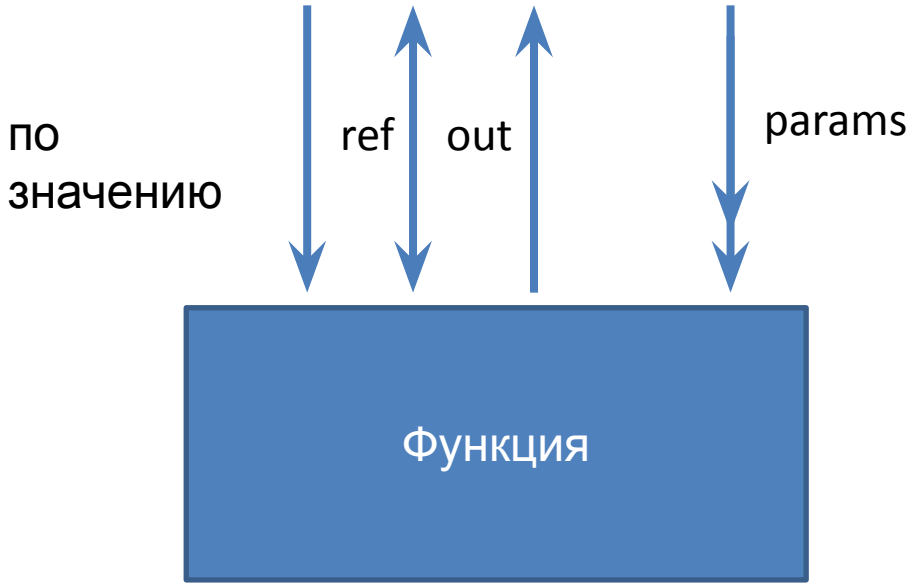
- **Список формальных параметров** – это те величины, которые требуется передать в функцию. Элементы списка разделяются запятыми. Для каждого параметра указывается тип и имя.
- **Фактические параметры** заменяют формальные параметры при выполнении операторов тела функции. Фактические и формальные параметры должны совпадать по количеству и типу.

Обмен данными между функциями

- **Исходные данные** могут быть получены:
  - как параметры метода;
  - как глобальные переменные (по отношению к методу);
  - от внешних устройств (файлы, потоки ввода).
- **Результаты** метод может передавать:
  - в точку вызова, как возвращаемое функцией значение;
  - в глобальные по отношению к методу объекты (переменные);
  - внешним устройствам (файлы, потоки вывода);
  - через параметры метода.

Способы передачи параметров в функцию:

- по значению;
- по ссылке (ref);
- выходные параметры (out);
- массив-параметр (params).



Задача

- Заданы координаты сторон треугольника, если такой треугольник существует, то найти его площадь.

Передача параметров по значению

- При **передаче по значению** выполняются следующие действия:
  - вычисляются значения выражений, стоящие на месте фактических параметров;
  - в стеке выделяется память под формальные параметры функции;
  - каждому фактическому параметру присваивается значение формального параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.

Передача параметров по значению

```
class Program
   {
      public static void Change(int a, int b)
      {
         int r = a;
         a = b;
         b = r;
      }
       static void Main(string[] args)
      {
         int x = 10, y = 5;
         Console.WriteLine("x=" + x + "  y=" + y);
         Change(x,y);
         Console.WriteLine("x=" +x + " y=" + y);
      }
```

Передача параметров по ссылке

```
class Program
   {
      public static void Change(ref int  a, ref int  b)
      {
         int r = a;
         a = b;
         b = r;
      }
static void Main(string[] args)
      {
         int x = new int(), y = new int();
         x = 5; y = 10;
         Console.WriteLine("x=" + x + "  y=" + y);
         Change(ref x, ref y);
         Console.WriteLine("x=" + x + " y=" + y);
    }
}
```

Передача выходных параметров

- **Выходные параметры** снабжаются модификатором out и позволяют присвоить значения объектам вызывающего метода даже в тех случаях, когда эти объекты значений еще не имели.
- В вызываемом методе операция присваивания должна выполняться обязательно.

Передача выходных параметров

```
static void MakePoint(int number, out double x, out
    double y)
    {
       try
       {
          Console.WriteLine("Введите координаты
    {0} точки:", number);
          Console.Write("Первая координата:");
          x = Convert.ToDouble(Console.ReadLine());
          Console.Write("Вторая координата:");
          y = Convert.ToDouble(Console.ReadLine());
       }
       catch (FormatException)
       {
          Console.WriteLine("Ошибка при вводе
    координат {0} точки!", number);
          x = 0;
          y = 0;
       }
    }
```

```
static void Main(string[] args)
     {
   double x1, y1, x2, y2, x3, y3;
         . . . .
         MakePoint(1, out x1, out y1);
         MakePoint(2, out x2, out y2);
         MakePoint(3, out x3, out y3);
         . . .
    }
```

Отличия между ссылочными и выходными параметрами

- Выходные параметры (out) не нужно инициализировать перед передачей методу, т.к. метод сам должен присваивать значения выходным параметрам перед выходом. Т.е. этих данных не было до выполнения функции и функция их должна создать.
- Ссылочные параметры (ref) нужно обязательно инициализировать перед передачей методу, т.к. они подразумевают передачу ссылки на уже существующую переменную. Т.е. эти данные существовали в памяти компьютера до выполнения функции и функция их должна изменить.

Функции с переменным числом параметров

- Ключевое слово **params** позволяет передавать методу переменное количество аргументов одного типа в виде единственного логического параметра.
- Аргументы, помеченные ключевым словом params, могут обрабатываться, если вызывающий код на их месте передает строго типизированный массив или разделенный запятыми список элементов.
- C# требует, чтобы в любом методе поддерживался только один аргумент params, который должен быть последним в списке параметров.

Функции с переменным числом параметров

```
static double CalcAverage
(params double[] values)
      {
         double avarage = 0;
         if (values.Length == 0) return 0;
    foreach (double x in values)
            avarage += x;
    avarage /= values.Length;
         return avarage;
      }
```

```
static void Main(string[] args)
       {
       //1 способ
      Console.WriteLine("Среднее
       арифметическое="+CalcAverage
       (0.2, 0.3, 0.5, 0.7));
       //2 способ
      double[] mas = { 0.2, 0.3, 0.5, 0.7 };
      Console.WriteLine("Среднее
       арифметическое=" +
       CalcAverage(mas));
       //3 способ
       Console.WriteLine("Среднее
       арифметическое=" +
       CalcAverage());
       }
```

Необязательные параметры (параметры по умолчанию)

- Умалчиваемое значение параметра используется, если при вызове функции соответствующий параметр опущен.
- Все параметры, описанные справа от такого параметра, также должны быть умалчиваемыми.
- Значение, присваиваемое необязательному параметру, должно быть известно во время компиляции и не может вычисляться во время выполнения.

Необязательные параметры (параметры по умолчанию)

```
static  void  Print(string s = "номер дома", int value = 1)
        {
            Console.WriteLine(s + "  " + value);
        }
static void Main(string[] args)
        {
             Print();
            Print("номер квартиры");
            Print(,2);
            Print("номер квартиры",15);
        }
```

Необязательные параметры (параметры по умолчанию)

```
static void Print(double [] mas,int size=mas.Length)
      {
        // int size = mas.Length;
         for (int i = 0; i < size; i++)
             Console.Write(mas[i] + " ");
       Console.WriteLine();
      }
       static void Main(string[] args)
       {
          double[] mas = { 0.2, 0.3, 0.5, 0.7 };
          Print(mas);
      }
```

Перегрузка методов

- Цель перегрузки состоит в том, чтобы функция с одним именем по-разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров.
- Для обеспечения перегрузки необходимо для каждой перегруженной функции определить возвращаемые значения и передаваемые параметры так, чтобы каждая перегруженная функция отличалась от другой функции с тем же именем.
- Компилятор определяет, какую функцию выбрать по типу фактических параметров.

Перегрузка методов

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

    static double
   MaxValue(double a, double b)
    {
       if (a > b)
          return a;
       else
          return b;
    }
```

```
 static string MaxValue(string a, string b)
     {
        if (String.Compare(a,b)>0)
           return a;  else  return b;
     }
static void Main(string[] args)
     {
        Console.WriteLine("максимальное
     значение из {0} и {1} = {2}",1,5,
     MaxValue(1,5));
 Console.WriteLine("максимальное значение из
     {0} и {1} = {2}", 1.5, 5.1, MaxValue(1.5, 5.1));
Console.WriteLine("максимальное значение из
     {0} и {1} = {2}", "111","555",
     MaxValue("111","555"));
      }
  }
```

Рекурсивные функции

- **Рекурсией** называется ситуация, когда какой-то алгоритм вызывает себя прямо (прямая рекурсия) или через другие алгоритмы (косвенная рекурсия) в качестве вспомогательного. Сам алгоритм называется рекурсивным.
- Рекурсивное решение задачи состоит из двух этапов:
  - 1. исходная задача сводится к новой задаче, похожей на исходную, но несколько проще;
  - 2. подобная замена продолжается до тех пор, пока задача не станет тривиальной, т. е. очень простой.

Рекурсивные функции

```
class Program
   {
      static  int fact(int n)
       {
            if (n==0)return 1; //база рекурсии
            return (n*fact(n-1));
       }
        static void Main(string[] args)
       {
          Console.WriteLine("Введите число для вычисления
    факториала");
          int k = Convert.ToInt32(Console.ReadLine());
          Console.WriteLine("{0}!={1}",k,fact(k));
        }
   }
```