Лабораторная работа №11

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И МЕХАТРОНИКИ

Кафедра информационных технологий и мехатроники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению лабораторных работ по дисциплине «Алгоритмизация и программирование»
для студентов специальности 6.050101 “Компьютерные науки”

Разработчик - доцент кафедры информационных технологий и мехатроники
кандидат технических наук, старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2015

Лабораторная работа №11
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений, использующие методы.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, использующие методы.

1. Теоретические сведения

Разработка программного обеспечения на практике является довольно непростым процессом. Программисту требуется учесть все тонкости и нюансы как всего программного комплекса в це лом, так и отдельных его частей. Системный подход к програм мированию основывается на том, что поставленная перед разработчиком задача предварительно разбивается на пару-тройку менее крупных вопросов, которые, в свою очередь, делятся еще на несколько менее сложных задач, и так "до тех пор, пока самые мелкие задачи не будут решены с помощью стандартных процедур. Таким образом осуществляется так называемая функциональная декомпозиция.

Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В Си задача может быть разделена на более простые и обозримые с помощью функций (в других языках программирования это процедуры, функции, методы), после чего программу можно рассмат ривать в более укрупненном виде — на уровне взаимодействия функций. Разделение программы на функции позволяет также избежать избыточности ко да, поскольку функцию записывают один раз, а вызывать ее на выполнение мож но многократно из разных точек программы. Процесс отладки программы, содер жащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. Таким образом создаются более простые в отлад ке и сопровождении программы.

Функция обеспечивает удобный способ отдельно оформить некоторое вычисление и пользоваться им далее, не заботясь о том, как оно реализовано. После того как функции написаны, можно забыть, как они сделаны, достаточно знать лишь, что они умеют делать. Механизм использования функций в Си удо бен, легок и эффективен.

1.1. Основные понятия функции

Функция — это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие. Функция может принимать пара метры и возвращать значение.

Любая программа на Си состоит из функций, одна из которых должна иметь имя Main() (рис.1). Программа начинает выполняться с функции Main() до вызова функции F1(). С этого момента управление программой пе редается в функцию F1(), далее до операто ра return выполняется тело функции F1(), после чего управ ление возвращается в тело функции Main(), а именно следующему за вызовом F1() оператору. После этого продолжается выполнение функции Main() до вызова функции F2(). После вызова функции F2() выполняется тело функции до оператора return, а затем передается управление в основную функцию Main(). После этого выполняется совокупность операторов до вызова функции F1(). Вызов функции F1() обеспечивает выполнение операторов данной функции, после чего управление передается в основную функцию Main() для завершения работы.

Функция начинает выпол няться в момент вызова. Любая функция должна быть объявлена и определена. Как и для других величин, объявлений может быть несколько, а определение только одно. Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова.

Объявление функции (прототип, заголовок, сигнатура) задает ее имя, тип возвра щаемого значения и список передаваемых параметров.

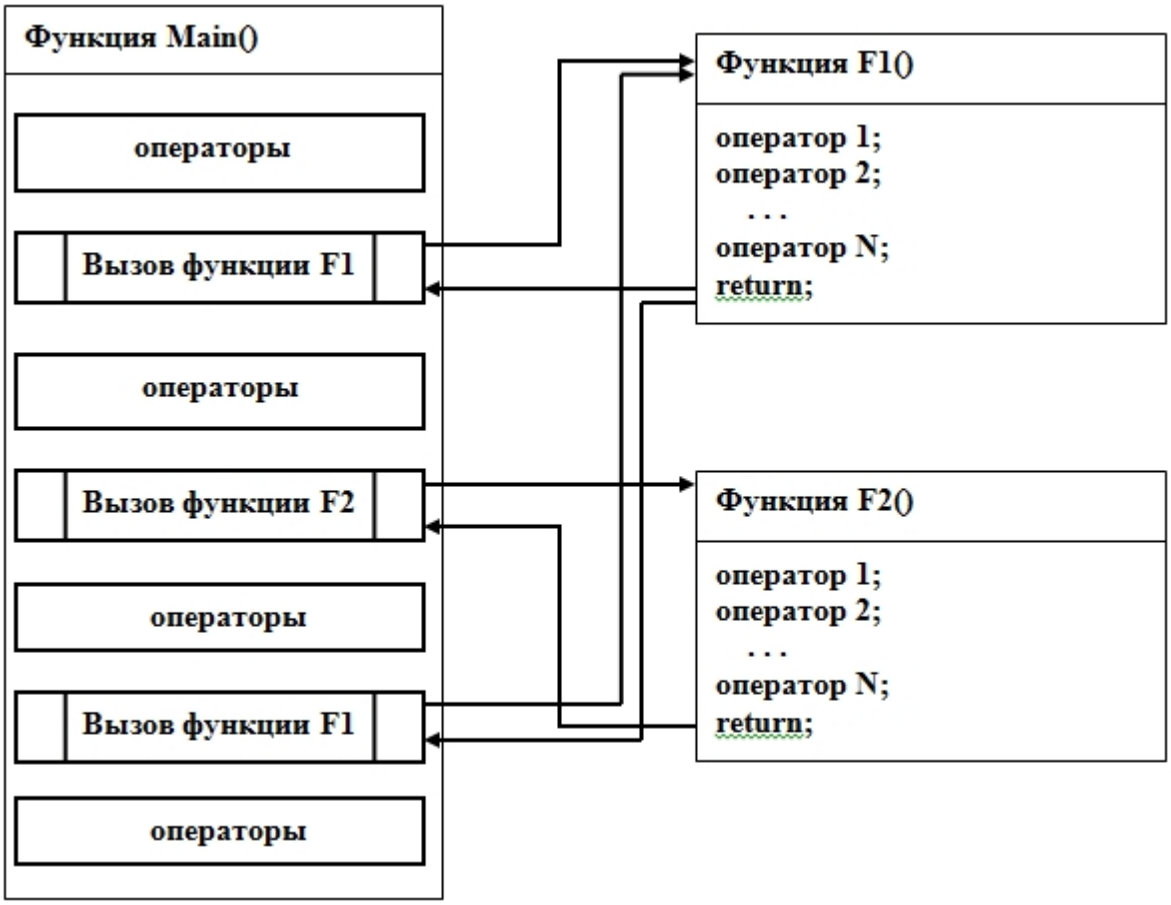

Рис. 1. Схема работы программы с функциями

Так как в объектно-ориентированном программировании вместо понятия «функция» используется понятие «метод», то в дальнейшем будем использовать понятие «метод». Методы — это функции (процедуры, подпрограммы), которые манипулируют данными, определенными в классе, и во многих случаях обеспечивают доступ к этим данным. Обычно различные части программы взаимодействуют с классом посредством его методов.

1.2. Описание метода

Любой метод содержит один или несколько операторов В хорошей C#-программе один метод выполняет только одну задачу. Каждый метод имеет имя, и именно это имя используется для его вызова. В общем случае методу можно присвоить любое имя. Но помните, что имя Main() зарезервировано для метода, с которого начинается выполнение программы. Кроме того, в качестве имен методов нельзя использовать ключевые слова C#.

Формат записи метода имеет вид:

модификатор_доступа тип_возврата имя (список_параметров)

```
{
  // тело метода
}
```

где **модификатор_доступа** означает модификатор доступа (public, protected, private), который определяет, какие части программы могут получить доступ к методу. Модификатор доступа необязателен, и в этом случае подразумевается, что метод закрыт (private) в рамках класса, где он определен. Пока все методы будем объявлять какpublic-члены, чтобы их могли вызывать все остальные составные части программного кода, даже те, которые определены вне класса.

С помощью элемента **тип_возврата** указывается тип значения, возвращаемого методом. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип **void**.

Имя метода задается элементом **имя**. В качестве имени метода можно использовать любой допустимый идентификатор, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости.

Элемент **список_параметров** представляет собой последовательность параметров (состоящих из модификатора, типа данных и идентификатора), разделенных запятыми. Параметры — это переменные, которые получают значения аргументов, передаваемых методу при вызове. Если метод не имеет параметров, **список_параметров** остается пустым. Существует несколько модификаторов, с помощью которых можно управлять способом передачи аргументов интересующему методу(см. табл. 1).

Таблица 1. Модификаторы параметров в C#

Модификатор параметра	Описание
-	Если параметр не сопровождается модификатором, предполага ется, что он должен передаваться по значению, т.е. вызываемый метод должен получать копию исходных данных
out	Выходные параметры должны присваиваться вызываемым мето дом (и, следовательно, передаваться по ссылке). Если парамет рам **out** в вызываемом методе значения не присвоены, компиля тор сообщит об ошибке
ref	Это значение первоначально присваивается вызывающим кодом и при желании может повторно присваиваться в вызываемом методе (поскольку данные также передаются по ссылке). Если параметрам **ref** в вызываемом методе значения не присвоены, компилятор никакой ошибки генерировать не будет
params	Этот модификатор позволяет передавать в виде одного логическо го параметра переменное количество аргументов. В каждом мето де может присутствовать только один модификатор **params** и он должен обязательно указываться последним в списке параметров. В реальности необходимость в использовании модификатора **params** возникает не особо часто, однако он применяется во мно гих методах внутри библиотек базовых классов

1.3. Вызов метода

Для того чтобы метод выполнил определенные действия, он должна быть вызван в программе. Вызов метода представляет собой указание идентификатора метода (его имени), за которым в круглых скобках следует список параметров, разделенных запятыми:

имя_метода (параметр1, параметр2 …, параметрN);

Каждый параметр метода представляет собой переменную, выражение или константу, передаваемые в тело метода для дальнейшего использования в программе. Список параметров может быть пустым.

Вызов метода может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует метод. Если тип возвра щаемого методом значения не **void**, то метод может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.

При обращении к методу он выполняет поставленную задачу, а по завершению работы возвращает в качестве результата некоторое значение.

Все величины, описанные внутри метода, а также его параметры, являются локальными. Областью их действия является метод. При вызове метода, как и при входе в любой блок, в стеке выделяется память под локальные автоматиче ские переменные. Кроме того, в стеке сохраняется содержимое регистров процес сора на момент, предшествующий вызову метода, и адрес возврата из метода для того, чтобы при выходе из нее можно было продолжить выполнение вызы вающего метода.

1.4. Передача параметров

1.4.1. Передача параметров по умолчанию

По умолчанию параметр передается по значению, т.е., если аргумент не снабжается каким-то конкретным модификатором параметра, методу передается копия данных, которая зависит от того, к какому типу относится параметр — типу значения или ссылочному типу.

Пример 1. Нижеприведенный пример демонстрирует создание внутри класса **Program** метода **Add**, который оперирует двумя числовыми типами данных, передаваемыми по значению:

```
public static int Add(int x, int y)
        {
          int ans = x + y;
          x = 100;
          y = 500;
          return ans;
        }
static void Main(string[] args)
        {
          // Передача двух переменных по значению
          int x = 9, y = 10;
          Console.WriteLine("Значение переменных до вызова метода
                                    X: {0}, Y: {1}", x, y);
          int z = Add(x,y);
          Console.WriteLine("Результат выполнения метода: {0}", z);
          Console.WriteLine("Значение переменных после вызова метода
                                    X: {0}, Y: {1}", x, y);
          Console.Write("Для завершения работы приложения нажмите
                                 клавишу <Enter>");
          Console.Read();
        }
```

Числовые данные подпадают под категорию типов значения. Поэтому в случае изменения значений параметров **x, y** внутри метода **Add()** значения параметров вызывающего метода будут оставаться без изменения, поскольку значения будут изменяться лишь в копии исходных данных Обмен значениями переменных представлен на рис.2.

Рис. 2. Обмен значениями при вызове функции Add(x,y)

1.4.2. Передача параметров с помощью модификатора out

Методы, которым при определении (с помощью ключевого слова **out**) указано принимать выходные параметры, должны перед выходом обязательно присваивать им соответствующие значения (в противном случае компилятор сообщит об ошибке).

Пример 2. Нижеприведенный пример демонстрирует модифицированный вариант метода **Add()**, который предусматривает возврат суммы двух целых чисел с использованием модификатора **out** (необходимо обратить внимание, что возвращаемым значением метода является тип **void**). В вызове метода с выходными параметрами тоже должен использоваться модификатор**out**. Локальным переменным, передаваемым в качестве выходных параметров, присваивать начальные значения не требуется (после вызова эти значения все равно будут утрачены). Обмен значениями переменных представлен на рис.3.

```
public static void Add (int x, int y, out int ans)
        {
          ans = x + y;
        }
static void Main(string[] args)
       {
         int x=90, y=30, z;
         Add(x, y, out z);
         Console.WriteLine("90 + 30 = {0}", z);
         Console.Write("Для завершения работы приложения нажмите
                              клавишу <Enter>");
         Console.Read();
       }
```

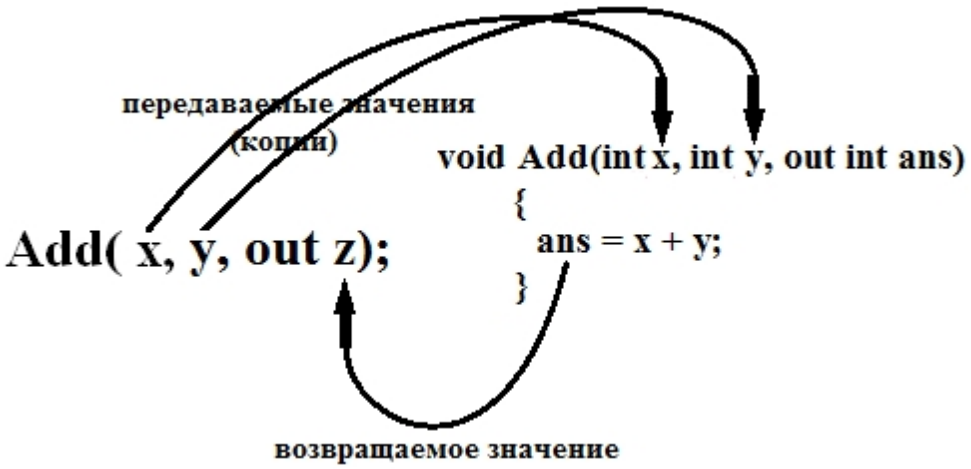

Рис. 3. Обмен значениями при вызове функции Add(x,y,out z)

Модификатор **out** в C# позволяет вызывающему коду получать в результате одного вызова метода сразу несколько значений

Пример 3. Нижеприведенный пример демонстрирует использование модификатора **out** для возврата в качестве одного вызова метода несколько значений переменных, при этом в вызывающем коде обращение к методу осуществляется оператором **MultiParams(out i, out str, out logic);**. Модификатор **out** используется как при вызове, так и при реализации данного метода.

```
public static void MultiParams(out int a, out string b, out bool c)
       {
         a = 9;
```

```
        b = "Это строка";
        c = true;
    }
static void Main(string[] args)
    {
        int i; string str; bool logic;
        MultiParams(out i, out str, out logic);
        Console.WriteLine("Это целое число: {0}", i);
        Console.WriteLine("Это строка: {0}", str);
        Console.WriteLine("Это логическая величина: {0}", logic);
        Console.Write("Для завершения работы приложения нажмите
                        клавишу <Enter>");
        Console.Read();
    }
```

В любом методе, в котором определяются выходные параметры, перед выходом им обязательно должны быть присвоены действительные значения.

1.4.3. Передача параметров с помощью модификатора ref

Параметры, сопровождаемые модификатором **ref** (от "reference" - ссылка), называются ссылочными и применяются, когда нужно позволить методу выполнять операции и изменять значения различных элементов данных, объявляемых в вызывающем коде (например, в процедуре сортировки или обмена). Отличия между ссылочными и выходными параметрами являются:

• Выходные параметры не нужно инициализировать перед передачей методу, так как метод сам должен присваивать значения выходным параметрам перед выходом;

• Ссылочные параметры нужно обязательно инициализировать перед передачей методу, так как они подразумевают передачу ссылки на уже существующую переменную. Если первоначальное значение ей не присвоено, это будет равнозначно выполнению операции над неинициализированной локальной переменной.

Пример 4. Нижеприведенный пример демонстрирует использование модификатора **ref** на примере метода, меняющего две строки местами. Здесь в вызывающем коде производится присваивание первоначальных значений локальным строкам (s1 и s2). После выполнения вызова **SwapStrings()** в строке s1 будет содержаться значение"Вторая строка", а в строке s2 — значение "Первая строка". Обмен ссылочными параметрами представлен на рис.4.

```
public static void SwapStrings(ref string s1, ref string s2)
    {
        string tempStr = s1;
        s1 = s2;
        s2 = tempStr;
    }
static void Main(string[] args)
    {
        string s1 = "Первая строка";
        string s2 = "Вторая строка";
        Console.WriteLine("Содержимое строк до перестановки: {0}, {1} ",
                          s1, s2);
        SwapStrings(ref s1, ref s2);
        Console.WriteLine("Содержимое строк после перестановки: {0}, {1} ",
                          s1, s2);
        Console.Write("Для завершения работы приложения нажмите
```

```
                клавишу <Enter>");
        Console.Read();
    }
```

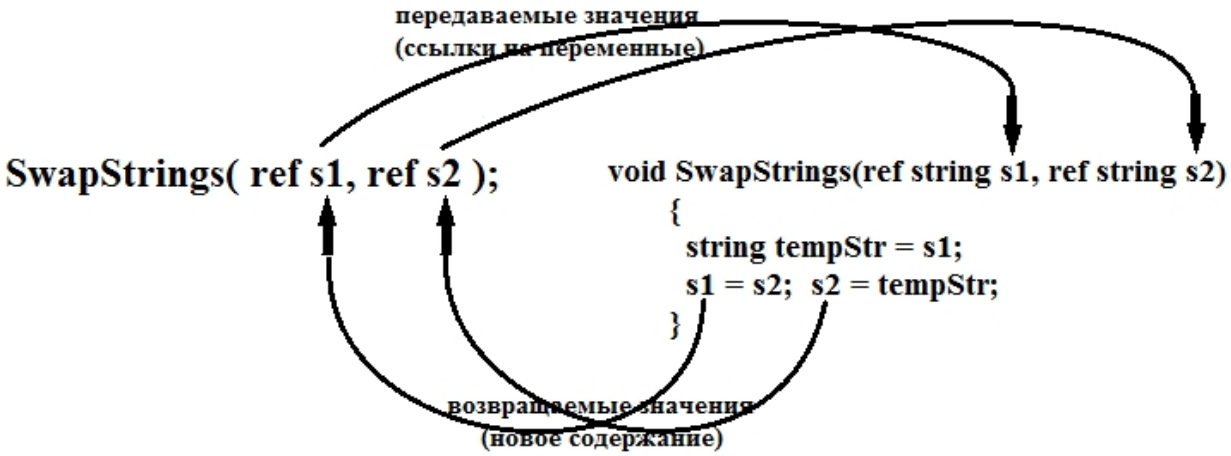

Рис. 4. Обмен параметрами при вызове функции SwapStrings(ref s1, ref s2)

1.4.4. Передача параметров с помощью модификатора params

В C# поддерживается использование массивов параметров за счет применения ключевого слова **params**. 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.

Пример 5. Нижеприведенный пример приложения «Вычисление среднего значения» демонстрирует использование модификатора **params**.

Пусть требуется создать метод, который бы позволил вызывающему коду передавать любое количество аргументов и возвращать их среднее значение. Для этого необходимо типизировать соответствующий метод так, чтобы он принимал массив значений типа double, а вызывающий код должен сначала определить массив, затем заполнить его значениями и только потом передать Если определить метод **Average()** так, чтобы он принимал массив параметров типа double **(params double[])**, тогда вызывающий код может просто передать разделенный запятыми список значенийdouble, а исполняющая среда .NET автоматически упакует этот список в массив типа double.

Метод определен таким образом, чтобы принимать массив параметров со значениями типа double, т.е. метод ожидает произвольное количество (включая ноль) значений double и вычисляет по ним среднее значение. Благодаря этому, он может вызываться любым из показанных ниже способов:

```
static double Average(params double[] values)
    {
     // Вывод количества значений
     Console. WriteLine ("Введено {0} значений.", values.Length) ;
     double sum = 0;
     if (values.Length == 0)
       return sum;
     for (int i = 0; i < values.Length; i++)
         sum += values [i];
     return (sum / values.Length);
    }

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

```
        Console.Title = "  Вычисление среднего значения";
        Console.BackgroundColor = ConsoleColor.White;
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.Black;
        double average;
        // 1 способ - передача разделенного запятыми списка значений double
        average = Average(4.0, 3.2, 5.7, 64.22, 87.2);
        Console.WriteLine("Среднее значение равно: {0}", average);
        // 2 способ - передача проинициализированного массива значений double
        double[] data = { 4.0, 3.2, 5.7, 1.2, 2.9, 6.1, 8.3 };
        average = Average(data);
        Console.WriteLine("Среднее значение равно: {0}", average);
        // передача 0 значений
        Console.WriteLine("Среднее значение равно: {0}", Average());
        Console.Write("Для завершения работы приложения нажмите клавишу <Enter>");
        Console.Read();
    }
```

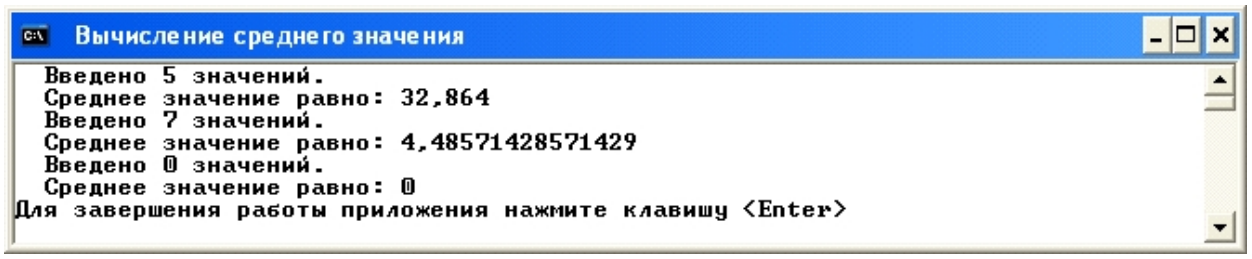

Рис. 5. Результат выполнения приложения «Вычисление среднего значения»

Если бы в определении Average() не было модификатора **params**, первый способ вызова этого метода приводил бы к ошибке на этапе компиляции, поскольку тогда компилятор искал бы вариант Average(), принимающий пять аргументов double.

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, использующие методы, и выполнить практически все примеры, описанные в этом разделе.

Задание 2. Разработать приложение с заголовком «Площадь кольца», которое вычисляет площадь кольца, внешний радиус которого – **R1**, внутренний – **R2**. Вычисление площади круга оформить в виде отдельного метода, параметрами которого являются радиус и площадь круга. Исходные данные вводятся с клавиатуры. Результат вычислений выводятся на экран монитора из метода **Main()**.

Задание 3. Разработать приложение с заголовком «Площадь офиса», с помощью которого можно определить площадь помещения. Схема и размеры представлены на рис. 6. Определение площади отдельного помещения оформить в виде отдельного метода. Количество и размеры помещений вводятся с клавиатуры. Результат выводится на экран монитора из метода **Main()**.

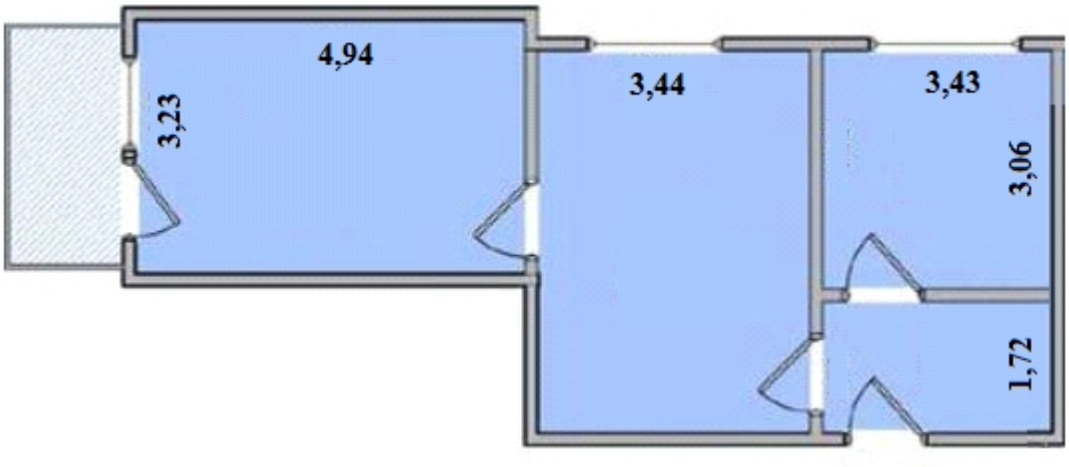

Рис. 6. План офиса

Задание 4. Разработать приложение с заголовком «Квадратное уравнение», с помощью которого обеспечивается решение квадратного уравнения $\mathbf{ax^2 + bx + c = 0}$. Решение квадратного уравнения оформить в виде отдельного метода, параметрами которого являются коэффициенты **a, b, c**, корни уравнения и сообщение о наличие или отсутствие корней. Исходные данные вводятся с клавиатуры. Результат решения квадратного уравнения выводятся на экран монитора из метода **Main()**.

3. Контрольные вопросы

1. Что такое метод и его назначение.
2. Как объявляется метод?
3. Как осуществляется вызов метода?
4. Как осуществляется передача параметров по умолчанию?
5. Как осуществляется передача параметров с помощью модификатора out?
6. Как осуществляется передача параметров с помощью модификатора ref?
7. Как осуществляется передача параметров с помощью модификатора params?

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011.– 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.