Лабораторная работа №9

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 №9
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений по обработке одномерных массивов данных.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, обрабатывающие одномерные массивы данных.

1. Теоретические сведения

1.1. Объявление одномерных массивов

Массив (array) — это коллекция переменных одинакового типа (int, char, double), обращение к которым происходит с использованием общего для всех имени. Имя массива является константой и содержит адрес первого элемента массива. В массивах хранятся отдельные значения, которые называются элементами. Все элементы массива хранятся в памяти последовательно, и первый элемент имеет нулевое смещение адреса, то есть нулевой индекс.

В C# массивы могут быть одномерными или многомерными.

Одномерным называется такой массив, который характеризуется одним измерением, а доступ к отдельному элементу осуществляется с помощью одного индекса (иногда такой массив называют вектором).

Для объявления одномерного массива используется следующая форма записи:

тип[ ] имя_массива = new тип [размер];

где **тип** - определяет тип данных элементов, составляющего массив; одна пара квадратных скобок за типом означает, что определяется одномерный массив; **имя_массива** – идентификатор(имя) массива; количество элементов, которые будут храниться в массиве, определяется **размером**. Поскольку массивы реализуются как объекты, их создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, а затем для него выделяется память, и переменной массива присваивается ссылка на эту область памяти. Таким образом, в C#-массивы динамически размещаются в памяти с помощью оператора **new**.

Например, оператор:

int[ ] mas = new int [10];

создается int-массив (состоящий из 10 целочисленных элементов), который связывается со ссылочной переменной массива **mas**. Переменная **mas** содержит ссылку на область памяти, выделенную оператором **new**.

Доступ к отдельному элементу массива осуществляется посредством индекса. Индекс описывает позицию элемента внутри массива. Индексами могут служить любые целые выражения — константы, переменные, результаты функций и т.п.

В C# первый элемент массива имеет нулевой индекс. Поскольку массив **mas** содержит 10 элементов, его индексы изменяются от 0 до 9. Чтобы получить доступ к элементу массива по индексу, достаточно указать нужный номер элемента в квадратных скобках. Так, первым элементом массива **mas** является **mas[0]**, а последним — **mas[9]**.

Пример 1. Нижеприведенный пример демонстрирует размещение чисел от 0 до 9 в массив **mas**.

```
public static void Main()
  {
   int[ ] mas = new int[10];
   int i ;
   for (i = 0 ; i < 10; i = i+1) mas[i] = i;
```

```
    for (i = 0 ; i < 10; i = i+1) Console.WriteLine("mas[" + i + "] : " + mas[i]);
  }
```

Так как массивы реализованы как объекты, то с каждым массивом связано свойство **Length**, содержащее количество элементов, которое может хранить массив. Поскольку каждый массив сопровождается информацией о собственной длине, то используя эту информацию можно отслеживать размер массива не «вручную». Следует иметь в виду, что свойство **Length** никак не связано с реально используемым количеством элементов массива. Оно содержит количество элементов, которое массив способен хранить.

Пример 2. Нижеприведенный пример демонстрирует использование свойства **Length**. Цикл **for** использует свойство **Length** для управления количеством итераций. Результат выполнения примера приведен на рис. 1.

```
public static void Main()
  {
   Console.Title = "   Квадраты чисел";
   int[] kvadrat = new int[10];
   Console.WriteLine("  Длина массива kvadrat равна " + kvadrat.Length);
   // Используем Length для инициализации массива kvadrat
   for (int i = 0; i < kvadrat.Length; i++) kvadrat[i] = i * i;
   // Теперь используем Length для отображения массива kvadrat
   Console.Write("  Содержимое массива kvadrat: ");
   for (int i = 0; i < kvadrat.Length; i++) Console.Write(kvadrat[i] + " ");
   Console.WriteLine();
   Console.Write("Для завершения работы приложения нажмите
                  клавишу <Enter>");
   Console.Read();
  }
```

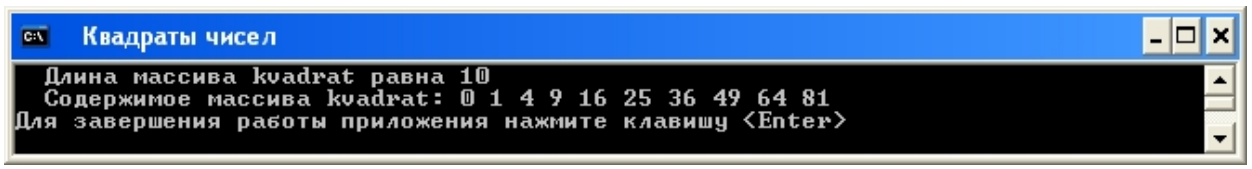

Рис. 1. Результат выполнения примера

1.2. Описание метода

Методы — это 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.

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.

1.4. 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 3. 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 значений double, 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();
      }
```

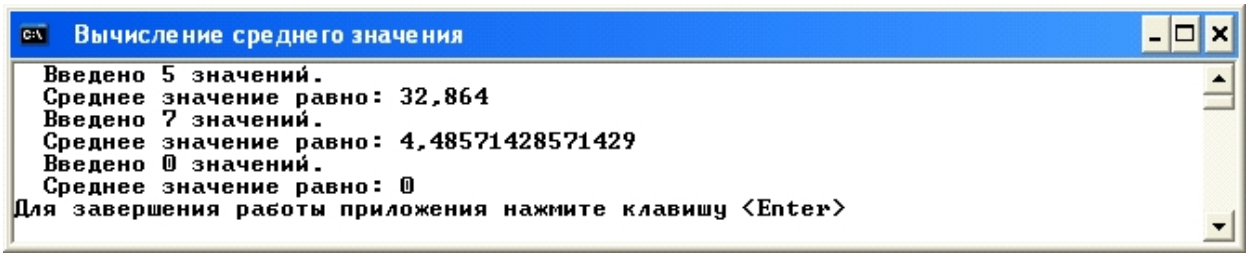

Рис. 2. Результат выполнения приложения«Вычисление среднего значения»

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, обрабатывающие одномерные массивы данных, и выполнить практически все примеры, описанные в этом разделе.

Задание 2. Разработать приложение с заголовком «Среднее арифметическое значение», которое вычисляет среднее арифметическое значение элементов массива без учета минимального и максимального элементов массива. Нахождение минимального значения, максимального значения и среднего арифметического значения оформить в виде отдельных методов. Размерность массива и значения его элементов вводятся с клавиатуры.
По окончании выполнения приложения на экран монитора должны быть выведены минимальное, максимальное и среднее арифметическое значения (см. рис.3).

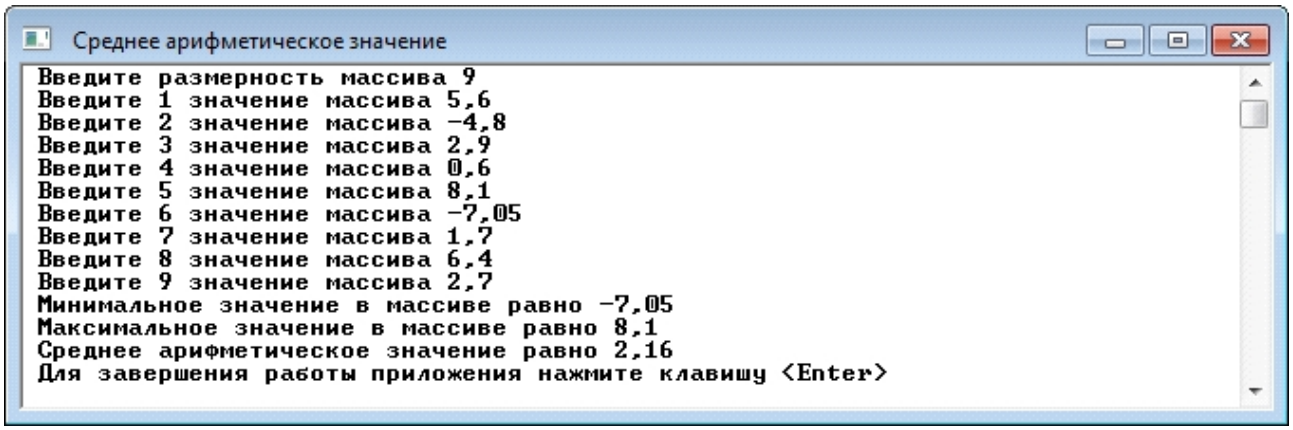

Рис. 3. Результат выполнения приложения«Среднее арифметическое значение»

Задание 3. Разработать приложение с заголовком «Нахождение минимального элемента», которое осуществляет поиск минимального элемента среди максимальных элементов в **N** одномерных массивах.

Количество массивов, размерность массивов и значения элементов соответствующих массивов вводятся с клавиатуры. Ввод значений массивов с поиском максимального значения в массиве оформить в виде отдельного метода. Поиск минимального элемента среди максимальных элементов оформить в виде отдельного метода с использованием модификатора **params**. По окончании выполнения приложения на экран монитора должны быть выведены максимальные значения всех массивов и номер массива со значением минимального элемента (см. рис.4).

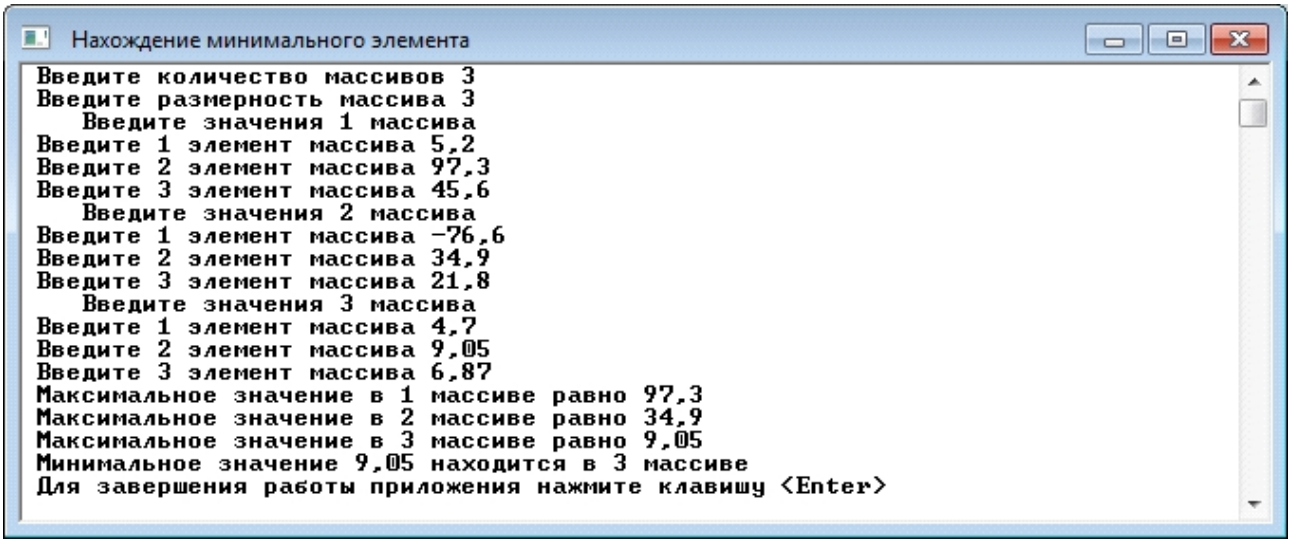

Рис. 4. Результат выполнения приложения«Нахождение минимального элемента»

3. Контрольные вопросы

1. Что такое массив?
2. Как объявляются одномерные массивы?
3. С помощью чего осуществляется доступ к элементам массива?
4. Для чего используется свойство массива Length?
5. Что такое метод и его назначение.
6. Как объявляется метод?
7. Как осуществляется вызов метода?
8. 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.