подготовка специалистов
программы обучения

Керов Леонид Александрович

МАССИВЫ И КОЛЛЕКЦИИ В ЯЗЫКЕ C#

Аннотация

Данная статья является четвертой из серии статей, посвященных изложению «нулевого уровня» языка C#. Рассматриваются массивы и коллекции в языке C#.

Ключевые слова: язык C#, массивы, коллекции.

1. МАССИВЫ

1.1. ПОНЯТИЕ И ОБЪЯВЛЕНИЕ МАССИВА

Массив – это составной объект, состоящий из элементов одного и того же типа. Простейшей разновидностью массива является *одномерный массив*, который можно рассматривать как информационную модель вектора в многомерном пространстве. Чтобы определить *переменную* для одномерного массива, нужно указать тип элементов массива и имя переменной массива:

```
тип [] имя;
```

В результате такого объявления для переменной массива выделяется ячейка в области оперативной памяти, которая называется «стек». Эта ячейка предназначена для хранения адреса элементов массива.

Сами элементы массива хранятся в другой области оперативной памяти, которая называется «куча». Для выделения этой области используется оператор new, после которого указывается тип элементов массива и число этих элементов в квадратных скобках:

```
new тип [размер];
```

Значением оператора **new** является адрес области для хранения элементов массива. Этот адрес может быть присвоен переменной массива. Вместо оператора new можно перечислить в фигурных скобках значения элементов массива: компилятор подсчитает их количество, определит тип, выделит необходимую область в куче и инициализирует элементы массива указанными в фигурных скобках значениями. При использовании оператора new элементы массива инициализируются автоматически:

– числовые элементы инициализируются нулями,

– булевские элементы инициализируются значениями **false**,

– ссылочные элементы инициализируются пустой ссылкой **null**.

Приведем пример, в котором определяются три одномерных массива, затем элементы последнего массива выводятся в строчку на экран монитора (см. листинг 1, рис. 1, 2).

Массивы в языке C# являются статическими, то есть число элементов массива

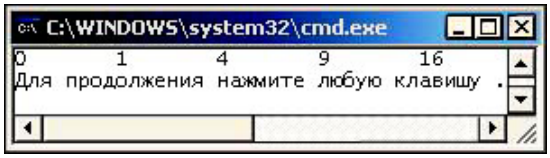

Рис. 1. Пример работы программы с одномерными массивами

нельзя изменить в процессе выполнения программы. Однако с помощью оператора new можно выделить для элементов массива другую область в куче с другим количеством элементов. Прежняя область памяти в куче при этом объявляется «мусором» и посредством «сборщика мусора» впоследствии передается в область свободной памяти. Из-за особенности выделения памяти для элементов массива их относят к ссылочным типам (**reference types**). Все рассмотренные ранее типы относятся к типам значений (**value types**). Для значений переменных этих типов выделяются ячейки памяти непосредственно в стеке.

1.2. МНОГОМЕРНЫЕ МАССИВЫ

Кроме векторов, в математике используются и другие объекты, например матрицы. Информационной моделью матрицы является двумерный массив. Определение и инициализация переменной для двумерного массива имеет вид:

```
тип [,] имя = new тип [число_строк,
                       число_столбцов];
```

Двумерный массив можно *инициализировать*, заключив список инициализаторов каждой размерности в собственный набор фигурных скобок. Если не указана явная инициализация, выполняется автоматическая инициализация по тем же правилам, что и для одномерного массива.

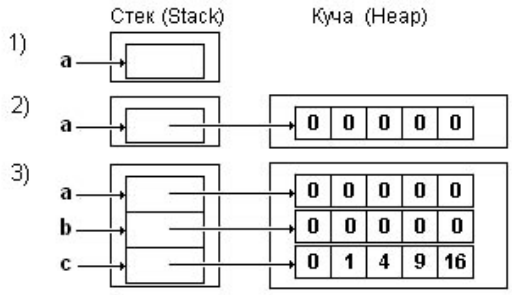

Рис. 2. Выделение памяти для массивов

Двумерный массив является частным случаем *многомерного массива*, определение и инициализация переменной которого имеет вид:

```
тип [,...,] имя = new тип
          [размер_0,...,размер_N];
```

Для массивов определено свойство **Length**, значением которого является общее число элементов в массиве. Для многомерных массивов можно определить число элементов по каждому измерению с помощью метода **GetLength**:

- **GetLength(0)** – число элементов по измерению 0,
- **GetLength(1)** – число элементов по измерению 1,
- **GetLength(2)** – число элементов по измерению 2 и т. д.

Приведем пример, в котором определяются три двумерных массива и один одномерный массив, затем выводятся на экран

Листинг 1

```
using System;
class P01
{
    public static void Main()
    {
        int[] a;
        a = new int[5];
        int[] b = new int[5];
        int[] c = { 0, 1, 4, 9, 16 };

        for (int j = 0; j < c.Length; j++)
            Console.Write(c[j] + "\t");
        Console.WriteLine();
    }
}
```

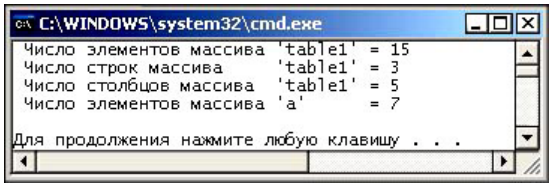

Рис. 3. Пример работы программы с двумерными массивами

монитора значения свойство **Length** и метода **GetLength** (см. листинг 2, рис. 3).

1.3. ДОСТУП К ЭЛЕМЕНТАМ МАССИВА

Доступ к элементам массива осуществляется с помощью имени массива и индекса элемента (порядкового номера элемента), который указывается в квадратных скобках после имени массива, например **a[i]**. Для многомерных массивов указываются индексы по каждой размерности, перечисленные через запятую, например **table[i, j]**. Нумерация элементов по каждой размерности начинается с нуля. Если в процессе работы программы индекс выйдет за границы массива, то это рассматривается как исключение **IndexOutOfRangeException**.

Приведем пример программы, в которой определяется одномерный массив, его элементы заполняются случайными числами и выводятся в столбик на экран монитора; затем определяется двумерный массив, его элементы также заполняются случайными числами и выводятся на экран монитора в виде таблицы (см. листинг 3, рис. 4).

1.4. СОРТИРОВКА МАССИВА

Если требуется найти в массиве индекс элемента, значение которого задано, то соответствующая программа работает значительно быстрее, если элементы массива упорядочены (отсортированы) по возрастанию или по убыванию значений. Рассмотрим один из простейших алгоритмов сортировки элементов одномерного массива, который называется *алгоритмом пузырьковой сортировки* (**bubble sort**). Символами слеш (**/**) выделен фрагмент программы,

Листинг 2

```
using System;
class P02
{
    public static void Main()
    {
        int[,] table1;
        table1 = new int[3, 5];
        int[,] table2 = new int[3, 5];
        int[,] table3 =  {  {0, 1, 2, 3,   4},
                            {0, 1, 4,  9, 16},
                            {0, 1, 8, 27, 64}
                         };
        int[] a = new int[7];
        Console.WriteLine(
          " Число элементов массива 'table1' = {0}\n" +
          " Число строк массива     'table1' = {1}\n" +
          " Число столбцов массива  'table1' = {2}\n" +
          " Число элементов массива 'a'      = {3}\n",
          table1.Length,
          table1.GetLength(0),
          table1.GetLength(1),
          a.Length);
    }
}
```

Листинг 3

```
using System;
class P03
{
    public static void Main()
    {
        Random r = new Random();
        int[] a = new int[5];
        for (int i = 0; i < a.Length; i++)
        {
            a[i] = r.Next(0, 11);
            Console.WriteLine("a[{0}] = {1}", i, a[i]);
        }
        Console.WriteLine();

        int[,] table = new int[3, 5];
        for (int i = 0; i < table.GetLength(0); i++)
        {
            for (int j = 0; j < table.GetLength(1); j++)
            {
                table[i, j] = r.Next(1, 11);
                Console.Write("{0}\t", table[i, j]);
            }
            Console.WriteLine();
        }
    }
}
```

который прорисовывает процесс протекания пузырьковой сортировки (см. листинг 4, рис. 5).

Сначала определяется одномерный массив из четырех элементов, значения которых задаются случайным образом. Затем выполняется сортировка элементов этого массива (по возрастанию значений). Сортировка выполняется с помощью двух циклов **for**, вложенных один в другой. Во внешнем цикле изменяется индекс **i** эле-

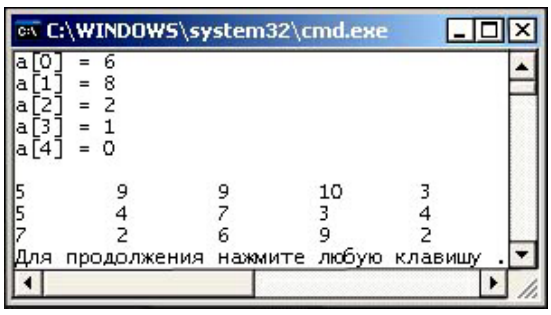

Рис. 4. Инициализация массивов случайными числами

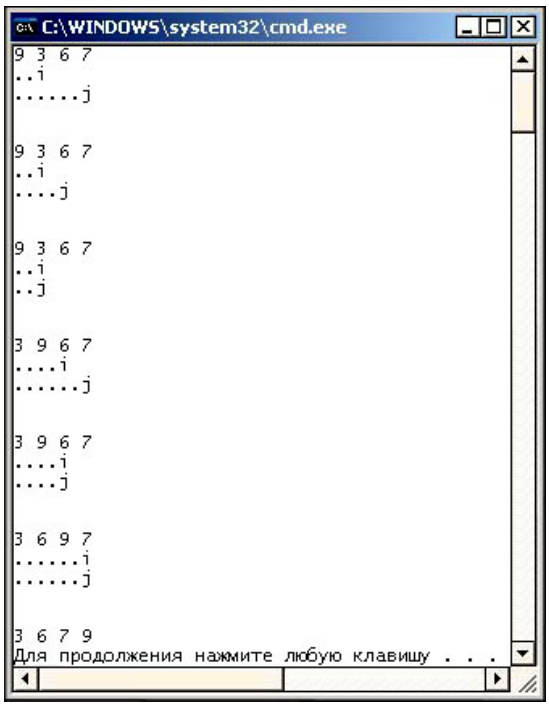

Рис. 5. Демонстрация протекания процесса пузырьковой сортировки массива

Листинг 4

```
using System;
class P04
{
    public static void Main()
    {
        Random r = new Random();
        int[] a = new int[4];
        for (int i = 0; i < a.Length; i++)
            a[i] = r.Next(0, 10);
        for (int i = 1; i < a.Length; i++)
        {
            for (int j = a.Length - 1; j >= i; j--)
            {   ////////////////////////////////////
                for (int k = 0; k < a.Length; k++)   //
                    Console.Write("{0} ", a[k]);     //
                Console.WriteLine();                 //
                for (int k = 0; k < i; k++)          //
                    Console.Write("..");             //
                Console.WriteLine("i");              //
                for (int k = 0; k < j; k++)          //
                    Console.Write("..");             //
                Console.WriteLine("j");              //
                Console.WriteLine("\n");             //
                ////////////////////////////////////
                if (a[j - 1] > a[j])
                {
                    //сортировка по возрастанию значений
                    int b = a[j - 1]; a[j - 1] = a[j];
                    a[j] = b;
                }
            }
        }
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0} ", a[i]);
        Console.WriteLine();
    }
}
```

ментов массива, начиная с единицы (индекс второго элемента массива) до индекса последнего элемента массива. Во внутреннем цикле изменяется индекс **j** элементов массива, начиная от номера последнего элемента массива до номера **i**.

В теле внутреннего цикла сравниваются значения пар **a[j-1]** и **a[j]** рядом расположенных элементов. Сначала просматриваются пары с самой последней и до первой. Если для какой-то пары требуемый порядок следования элементов нарушен, то элементы этой пары меняются местами. В итоге (если сортируем по возрастанию значений) на первую позицию в массиве будет «вытолкнут» (как пузырек) самый маленький элемент.

После этого индекс **i** увеличивается на единицу и просматриваются пары с самой последней и до второй. При этом элементы «неправильных» пар меняются местами. В результате на вторую позицию в массиве будет «вытолкнут» (как пузырек) элемент, значение которого не больше значения пер-

вого элемента. После этого индекс **i** увеличивается на единицу и т. д., пока **i** не дойдет до последнего элемента массива. После завершения сортировки элементы массива выводятся на экран монитора.

1.5. РВАНЫЙ МАССИВ

Язык C# позволяет создавать двумерные массивы специального вида, именуемые рваными. Приведем пример, в котором определяется рваный массив, содержащий три строки (см. листинг 5, рис. 6).

Объявление рваного массива имеет вид:

```
тип [][] имя = new тип [число_строк][];
```

Для строк рваного массива память выделяется индивидуально, что позволяет строкам иметь разную длину. После создания рваного массива доступ к элементу осуществляется посредством задания индекса внутри собственного набора квадратных скобок.

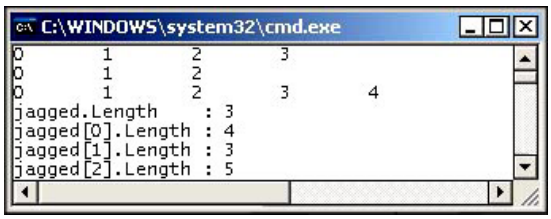

Рис. 6. Пример работы программы с рваным массивом

Для рваного массива свойство **Length** используется следующим образом. Выражение **jagged.Length** возвращает коли-

Листинг 5

```
using System;
class P05
{
    public static void Main()
    {
        int[][] jagged = new int[3][];
        jagged[0] = new int[4];
        jagged[1] = new int[3];
        jagged[2] = new int[5];
        for (int i = 0; i < 4; i++)
        {
            jagged[0][i] = i; Console.Write(jagged[0][i] + "\t");
        }
        Console.WriteLine();
        for (int i = 0; i < 3; i++)
        {
            jagged[1][i] = i; Console.Write(jagged[1][i] + "\t");
        }
        Console.WriteLine();
        for (int i = 0; i < 5; i++)
        {
            jagged[2][i] = i; Console.Write(jagged[2][i] + "\t");
        }
        Console.WriteLine(
            "\njagged.Length    : {0}" +
            "\njagged[0].Length : {1}" +
            "\njagged[1].Length : {2}" +
            "\njagged[2].Length : {3}",
            jagged.Length, jagged[0].Length,
            jagged[1].Length, jagged[2].Length
        );
    }
}
```

чество подмассивов, хранимых в массиве **jagged**. Чтобы получить длину отдельного подмассива, используются выражения вида: **jagged[0].Length**, **jagged[1].Length**, **jagged[3].Length**.

1.6. ЦИКЛ «FOREACH»

Цикл **foreach** позволяет реализовать в программе перебора элементов массива и имеет следующий вид:

```
foreach (тип переменная in
                имя_массива) оператор
```

Элемент **тип** – это тип элементов массива; элемент **переменная** – имя переменной, которая при функционировании цикла foreach будет получать значения элементов из массива с указанным именем; элемент **оператор** называется телом цикла; это обычно блок. Приведем пример, в котором определяется одномерный массив, затем с помощью цикла **foreach** элементы массива поочередно выводятся на экран монитора, и одновременно подсчитывается их сумма (см. листинг 6, рис. 7).

Цикл **foreach** работает и с многомерными массивами. В этом случае он возвращает элементы в порядке следования строк: от первой до последней строки.

2. МЕТОДЫ КЛАССА ARRAY

2.1. МЕТОД SORT

Массивы в языке C# – это объекты класса **Array**, который определен в пространстве имен **System**. В классе **Array** определено свойство **Length** и определен метод **GetLength**, которые были рассмотрены выше.

В классе **Array** определен статический метод **Sort()**, предназначенный для сортировки одномерных массивов по возрастанию значений. При этом исходный массив передается как параметр. Приведем пример, в котором определяется одномерный массив, который выводится на экран монитора, затем сортируется с помощью метода **Sort**, и результат сортировки снова выводится на экран (см. листинг 7, рис. 8).

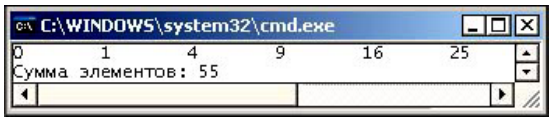

Рис. 7. Пример работы цикла **foreach**

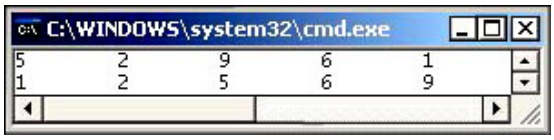

Рис. 8. Сортировка массива с помощью метода **Sort**

Листинг 6

```
using System;
class P06
{
    public static void Main()
    {
        double[] a = { 0, 1, 4, 9, 16, 25 };
        double sum = 0;
        foreach (double x in a)
        {
            Console.Write(x + "\t");
            sum += x;
        }
        Console.WriteLine("\nСумма элементов: " + sum);
    }
}
```

Листинг 7

```
using System;
class P07
{
    public static void Main()
    {
        double[] a = { 5, 2, 9, 6, 1 };
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Array.Sort(a);
        Console.WriteLine();
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();

    }
}
```

2.2. МЕТОД CLEAR

Статический метод **Clear** из класса **Array** позволяет «обнулить» указанные элементы массива:

– числовые элементы устанавливаются в 0,

– ссылочные элементы устанавливаются в **null**,

– булевские элементы устанавливаются в **false**.

Метод **Clear** имеет три параметра: имя массива; индекс, начиная с которого происходит «обнуление», и число «обнуляемых» элементов. Приведем пример, в котором определяется одномерный массив, который выводится на экран монитора, затем, начиная со второго элемента, обнуляются три элемента массива, и полученный в результате массив выводится на экран (см. листинг 8, рис. 9).

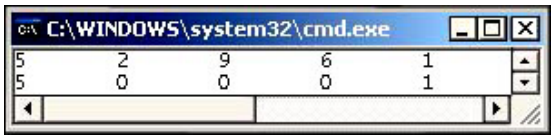

Рис. 9. Применение метода **Clear**

Листинг 8

```
using System;
class P08
{
    public static void Main()
    {
        double[] a = { 5, 2, 9, 6, 1 };
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Array.Clear(a, 1, 3);
        Console.WriteLine();
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();

    }
}
```

2.3. МЕТОД CLONE

Статический метод **Clone** из класса **Array** создает копию массива как объект типа **object**. Приведем пример, в котором определяется одномерный массив. Это массив, который выводится на экран монитора, затем создается его клон, который также выводится на экран. После этого все элементы исходного массива обнуляются с помощью метода **Clear**, и полученный в результате массив также выводится на экран (см. листинг 9, рис. 10).

Заметим, что если массив содержит ссылки на объекты, то копируются только ссылки; сами объекты не копируются.

2.4. МЕТОД INDEXOF

Метод **IndexOf** из класса **Array** возвращает номер первого вхождения указанного элемента. Если элемент не найден, то возвращается значение –1.

Приведем пример, в котором определяется одномерный массив, который выводится на экран монитора. Затем демонстрируется результат определения индекса существующего в массиве элемента и результат определения индекса элемента, которого в массиве нет (см. листинг 10, рис. 11).

3. КОЛЛЕКЦИИ

Коллекция – это, как и массив, составной объект. В отличие от массивов, элементы коллекции могут быть разного типа. Кроме того, коллекция может изменять свой размер во время работы программы. Все коллекции определены в виде классов в пространстве имен **System.Collections**.

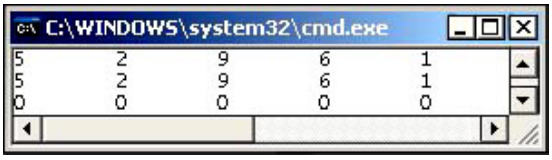

Рис. 10. Применение метода **Clone**

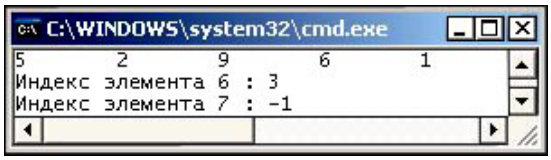

Рис. 11. Применение метода **IndexOf**

Листинг 9

```
using System;
class P09
{
    public static void Main()
    {
        int[] a = { 5, 2, 9, 6, 1 };
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();

        int[] b = (int[])a.Clone();
        for (int i = 0; i < b.Length; i++)
            Console.Write("{0}\t", b[i]);
        Console.WriteLine();

        Array.Clear(a, 0, a.Length);
        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();
    }
}
```

Листинг 10

```
using System;
class P10
{
    public static void Main()
    {
        int[] a = { 5, 2, 9, 6, 1 };

        for (int i = 0; i < a.Length; i++)
            Console.Write("{0}\t", a[i]);
        Console.WriteLine();

        Console.WriteLine("Индекс элемента 6 : {0}",
            Array.IndexOf(a, 6));

        Console.WriteLine("Индекс элемента 7 : {0}",
            Array.IndexOf(a, 7));

    }
}
```

3.1.КОЛЛЕКЦИЯ ARRAYLIST

Коллекция **ArrayList** представляет собой упорядоченную группу объектов, доступ к которым можно получить посредством индекса. Она имеет следующие методы:

- **Add** – добавляет новый элемент в конец коллекции;
- **AddRange** – добавляет в конец коллекции набор элементов из другой коллекции;
- **Remove** – удаляет первое вхождение указанного элемента;
- **RemoveAt** – удаляет элемент по указанному индексу;
- **RemoveRange** – удаляет, начиная с указанного индекса, заданное количество элементов;
- **Clear** – удаляет все элементы из коллекции;
- **Insert** – вставляет по указанному индексу элемент в коллекцию;
- **Sort** – сортирует элементы коллекции по возрастанию значений элементов;
- **Reverse** – изменяет порядок следования элементов на противоположный.

Приведем пример программы, которая работает с объектом класса **ArrayList**. Работа программы подразделяется на 10 шагов, в конце каждого из которых состав элементов коллекции выводится на экран монитора с помощью метода **ShowValues** (см. листинг 11, рис. 12).

Листинг 11

```
using System;
using System.Collections;
public class P11
{
    public static void Main()
    {
        //Создается коллекция aL, в нее добавляются три элемента,
        //и содержимое коллекции aL выводится на экран
        arrayList aL = new ArrayList();
        aL.Add(1); aL.Add("two"); aL.Add(3);
        ShowValues(1, aL);
```

```
        //Создается коллекция aL2, в нее добавляются два элемента,
        // и содержимое коллекции aL2 выводится на экран
        ArrayList aL2 = new ArrayList();
        aL2.Add("four"); aL2.Add(3);
        ShowValues(2, aL2);

        //В конец коллекции aL добавляется набор элементов из aL2,
        //и содержимое коллекции aL выводится на экран
        aL.AddRange(aL2); ShowValues(3, aL);

        //Из коллекции aL удаляется первое вхождение элемента "3",
        //и содержимое коллекции aL выводится на экран
        aL.Remove(3); ShowValues(4, aL);

        //Из коллекции aL удаляется третий элемент (его индекс 2) ,
        //и содержимое коллекции aL выводится на экран
        aL.RemoveAt(2);
        ShowValues(5, aL);

        //Из коллекции aL удаляются два элемента, начиная с первого,
        //и содержимое коллекции aL выводится на экран
        aL.RemoveRange(0, 2); ShowValues(6, aL);

        //Из коллекции aL удаляются все элементы,
        //и содержимое коллекции aL выводится на экран
        aL.Clear(); ShowValues(7, aL);

        //В коллекцию aL вставляеюся элементы по указанному индексу,
        //и содержимое коллекции aL выводится на экран
        aL.Insert(0, "Hello");
        aL.Insert(1, "world");
        aL.Insert(1, ",");
        ShowValues(8, aL);

        //Элементы коллекции aL сортируются по возрастанию значений,
        //и содержимое коллекции aL выводится на экран
        aL.Sort(); ShowValues(9, aL);

        //Порядок следования элементов aL изменяется на
        //противоположный, и содержимое коллекции aL выводится
        //на экран
        aL.Reverse(); ShowValues(10, aL);
    }

    public static void ShowValues(int i, IEnumerable lst)
    {
        System.Collections.IEnumerator en = lst.GetEnumerator();
        Console.Write("\n{0} : ", i);
        while (en.MoveNext())
        {
            Console.Write(" {0}", en.Current);
        }
        Console.WriteLine();
    }
}
```

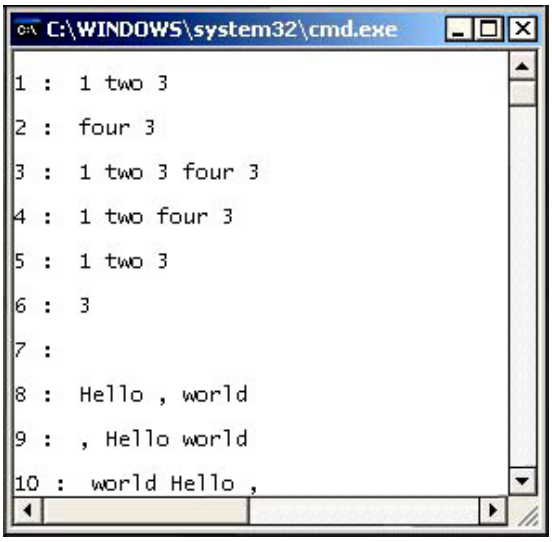

Рис. 12. Пример работы программы с объект класса **ArrayList**

Метод **ShowValues** имеет два параметра: номер шага и параметр типа **IEnumerable**. Класс **ArrayList** наследует интерфейс **IEnumerable**:

```
public class ArrayList:IList,
ICollection, IEnumerable, ICloneable
```

Язык C# разрешает ссылке на объект базового класса (и перечисления) присвоить ссылку на объект производного класса, поэтому ссылка на объект типа **ArrayList** передается как аргумент в метод **ShowValues**. В интерфейсе **IEnumerable** определен метод **GetEnumerator**, который возвращает объект типа **IEnumerator**, который называется перечислителем и позволяет перебирать элементы коллекции.

Перечислители могут использоваться для чтения данных коллекции; они не могут использоваться для изменения коллекции. Изначально перечислитель располагается перед первым элементом коллекции. Метод **Reset** снова устанавливает перечислитель в данную позицию. В этой позиции вызов свойства **Current** приводит к выдаче исключения. Поэтому необходимо вызвать метод **MoveNext** до считывания значения свойства **Current**, чтобы переместить перечислитель к первому элементу коллекции.

Свойство **Current** возвращает один и тот же объект, пока не будет вызван метод **MoveNext** или **Reset**. Метод **MoveNext** задает следующий элемент в качестве значения свойства **Current**.

Если метод **MoveNext** проходит конец коллекции, то перечислитель помещается в ней после последнего элемента, а метод **MoveNext** возвращает значение **false**. Когда перечислитель находится в этой позиции, последующие вызовы метода **MoveNext** также возвращают значение **false**. Если при последнем вызове метода **MoveNext** было возвращено значение **false**, обращение к свойству **Current** приводит к выдаче исключения. Чтобы снова задать в качестве значения свойства **Current** первый элемент коллекции, можно последовательно вызвать методы **Reset** и **MoveNext**.

3.2. КОЛЛЕКЦИЯ QUEUE

В отличие от коллекции **ArrayList**, которая допускает обращение к своим элементам в произвольном порядке, коллекция **Queue** обеспечивает исключительно последовательный доступ к своим элементам и обслуживается по принципу «первым поступил – первым обслужен». Класс **Queue** наследует интерфейс **IEnumerable**:

```
public class Queue : ICollection,
                IEnumerable, ICloneable
```

Как следствие, для вывода элементов коллекции **Queue** на экран монитора может использоваться тот же метод **ShowValues**, что и в предыдущем примере. Для доступа к элементам коллекции **Queue** используются методы:

- **Enqueue** – добавляет объект в конец коллекции **Queue**,
- **Dequeue** – извлекает и удаляет объект, находящийся в начале коллекции **Queue**.

Приведем пример программы, которая работает с объектом класса **Queue** (см. листинг 12, рис. 13).

3.3. КОЛЛЕКЦИЯ STACK

Коллекция **Stack** похожа на коллекцию **Queue**, только элементы извлекаются из стека по принципу «первым поступил –

последним обслужен». Класс **Stack** наследует интерфейс **IEnumerable**:

```
public class Stack : ICollection,
          IEnumerable, ICloneable
```

Как следствие, для вывода элементов коллекции **Stack** на экран монитора может использоваться тот же метод **ShowValues**, что и в предыдущем примере. Для доступа к элементам коллекции **Stack** используются методы:

- **Push** – вставляет объект как верхний элемент стека **Stack**,
- **Pop** – возвращает и удаляет и верхний объект стека **Stack**.

Приведем пример программы, которая работает с объектом класса Stack (см. листинг 13, рис. 14).

Рис. 13. Пример работы программы с объект класса **Queue**

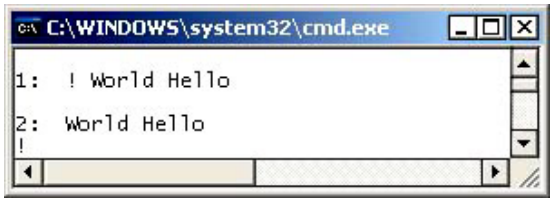

Рис. 14. Пример работы программы с объект класса **Stack**

Листинг 12

```
using System;
using System.Collections;
public class P12
{
    public static void Main()
    {
        //Создается коллекция q, в конец очереди поочередно
        //добавляются три элемента, затем содержимое коллекции
        //выводится на экран
        Queue q = new Queue();
        q.Enqueue("Hello"); q.Enqueue("World"); q.Enqueue("!");
        ShowValues(1, q);

        //Из начала очереди извлекается элемент, затем
        //содержимое коллекции выводится на экран
        object ob = q.Dequeue();
        ShowValues(2, q);

        //Извлеченный из очереди  элемент выводится на экран
        Console.WriteLine(ob);
    }
    public static void ShowValues(int i, IEnumerable Q)
    {
        System.Collections.IEnumerator en = Q.GetEnumerator();
        Console.Write("\n{0}: ", i);
        while (en.MoveNext())
        {
            Console.Write(" {0}", en.Current);
        }
        Console.WriteLine();
    }
}
```

Листинг 13

```
using System;
using System.Collections;
public class P13
{
    public static void Main()
    {
        //Создается стек s, в него поочередно добавляются
        //три элемента, затем содержимое стека выводится
        //на экран
        Stack s = new Stack();
        s.Push("Hello"); s.Push("World"); s.Push("!");
        ShowValues(1, s);

        //Из стека извлекается элемент, затем
        //содержимое стека выводится на экран
        object ob = s.Pop();
        ShowValues(2, s);

        //Извлеченный из стека элемент выводится на экран
        Console.WriteLine(ob);
    }
    public static void ShowValues(int i, IEnumerable Q)
    {
        System.Collections.IEnumerator en = Q.GetEnumerator();
        Console.Write("\n{0}: ", i);
        while (en.MoveNext())
        {
            Console.Write(" {0}", en.Current);
        }
        Console.WriteLine();
    }
}
```

Литература

1. *Керов Л.А.* Методы объектно-ориентированного программирования на C# 2005: Учебное пособие. СПб: Издательство «ЮТАС», 2007. 164 с.

2. *Нэш Т.* C# 2008: ускоренный курс для профессионалов: Пер. с англ. М.: ООО «И.Д. Вильямс», 2008. 576 с.

3. *Павловская Т.А.* C#. Программирование на языке высокого уровня. Учебник для вузов. СПб: Питер, 2007. 432 с.

4. *Троелсен Э.* Язык программирования C# 2005 и платформа .NET 2.0. 3-е издание.: Пер. с англ. М.: ООО «И.Д. Вильямс», 2007. 1168 с.

5. *Шилдт Г.* C#: учебный курс. СПб: Питер; К.: Издательская группа BHV, 2003. 512 с.

Abstract

The article is third of a series of articles, devoted to «a zero level» of language C#. Arrays and collections in C# are considered.

Керов Леонид Александрович, кандидат технических наук, старший научный сотрудник, доцент, заведующий кафедрой бизнес-информатики СПб филиала ГУ–ВШЭ при Правительстве РФ,

kerov@hse.spb.ru