Массивы

Массив представляет фиксированное количество элементов конкретного типа. Элементы в массиве всегда хранятся в непрерывном блоке памяти, обеспечивая высокоэффективный доступ.

Массив обозначается квадратными скобками после типа элементов. Например, ниже объявлен массив из 10 символов:

```
char[] vowels = new char[10];
```

С помощью квадратных скобок также указывается *индекс* в массиве, что позволяет получать доступ к элементам по их позициям:

```
vowels[0] = 'а'; vowels[1] = 'о'; vowels[2] = 'и';
vowels[3] = 'е'; vowels[4] = 'ё'; vowels[5] = 'э';
vowels[6] = 'ы'; vowels[7] = 'у'; vowels[8] = 'ю';
vowels[9] = 'я';

Console.WriteLine (vowels [1]); // о
```

Этот код приведет к выводу буквы “о”, поскольку массив индексируется, начиная с 0. Оператор цикла `for` можно использовать для прохода по всем элементам в массиве. Цикл `for` в следующем примере выполняется для целочисленных значений `i` от `0` до `4`:

```
for (int i = 0; i < vowels.Length; i++)
  Console.Write (vowels [i]);              // аоиеёэыуюя
```

Массивы также реализуют интерфейс `IEnumerable<T>` (Краткий справочник C#, Албахари стр. 111), так что по элементам массива можно проходить посредством оператора `foreach`:

```
foreach (char c in vowels) Console.Write (c);    // аоиеёэыуюя
```

Во время выполнения все обращения к индексам массивов проверяются на предмет выхода за границы. В случае применения некорректного значения индекса генерируется исключение `IndexOut- OfRangeException`:

```
vowels[10] = 'а';    // Ошибка времени выполнения
```

Свойство `Length` массива возвращает количество элементов в массиве. После создания массива изменить его длину невозможно. Пространство имен `System.Collection` и вложенные в него пространства имен предоставляют такие высокоуровневые структуры данных, как массивы с динамически изменяемыми размерами и словари.

Выражение инициализации массива позволяет объявлять и заполнять массив в единственном операторе:

```
char[] vowels = new char[] {'а','о','и','е','ё','э','ы','у','ю','я'};
```

или проще:

```
char[] vowels = {'а','о','и','е','ё','э','ы','у','ю','я'};
```

Все массивы унаследованы от класса `System.Array`, в котором определены общие

методы и свойства для всех массивов. Сюда входят свойства экземпляра вроде `Length` и `Rank` и статические методы для выполнения следующих действий:

- динамическое создание массива (`CreateInstance()`);
- извлечение и установка элементов независимо от типа массива (`GetValue()/SetValue()`);
- поиск в сортированном (`BinarySearch()`) или несортированном (`IndexOf()`, `LastIndexOf()`, `Find()`, `FindIndex()`, `FindLastIndex()`) массиве;
- сортировка массива (`Sort()`);
- копирование массива (`Copy()`).

Стандартная инициализация элементов

При создании массива всегда происходит инициализация его элементов стандартными значениями. Стандартное значение для типа представляет собой результат побитового обнуления памяти. Например, предположим, что создается массив целых чисел. Поскольку `int` — тип значения, будет выделено пространство под 1000 целочисленных значений в непрерывном блоке памяти. Стандартным значением для каждого элемента будет `0`:

```
int[] a = new int[1000);
Console.Write (a[123]);   // 0
```

Для элементов ссылочного типа стандартным значением будет `null`.
Независимо от типа элементов массив *сам по себе* всегда является объектом ссылочного типа. Например, следующий оператор допустим:

```
int[] a = null;
```

Многомерные массивы

Многомерные массивы имеют две разновидности: *прямоугольные* и *зубчатые*. Прямоугольный массив представляет *n*-мерный блок памяти, а зубчатый массив — это массив, содержащий массивы.

Прямоугольные массивы

Прямоугольные массивы объявляются с использованием запятых для отделения каждого измерения друг от друга. Ниже приведено объявление прямоугольного двумерного массива размерностью 3×3:

```
int[,] matrix = new int [3, 3];
```

Метод `GetLength()` массива возвращает длину для заданного измерения (начиная с 0):

```
for (int i = 0; i < matrix.GetLength(0); i++)
    for (int j = 0; j < matrix.GetLength(1); j++)
       matrix [i, j] = i * 3 + j;
```

Прямоугольный массив может быть инициализирован следующим образом (для создания массива, идентичного предыдущему примеру):

```
int[,] matrix = new int[,]
```

```
{
  {0,1,2},
  {3,4,5},
  {6,7,8}
};
```

(В операторах объявления такого рода код, выделенный полужирным, может быть опущен.)

Зубчатые массивы

Зубчатые массивы объявляются с применением последовательно идущих пар квадратных скобок для каждого измерения. Ниже показан пример объявления зубчатого двумерного массива, в котором самое внешнее измерение составляет 3:

```
int[][] matrix = new int[3][];
```

Внутренние измерения в объявлении не указываются, т.к. в отличие от прямоугольного массива каждый внутренний массив может иметь произвольную длину. Каждый внутренний массив неявно инициализируется null, а не пустым массивом. Каждый внутренний массив должен быть создан вручную:

```
for (int i = 0; i < matrix.Length; i++)
{
  matrix[i] = new int [3]; // Создать
  внутренний массив for (int j = 0; j <
  matrix[i].Length; j++)
    matrix[i][j] = i * 3 + j;
}
```

Зубчатый массив можно инициализировать следующим образом (для создания массива, идентичного предыдущему примеру, но с дополнительным элементом в конце):

```
int[][] matrix = new int[][]
{
  new int[] {0,1,2},
  new int[] {3,4,5},
  new int[] {6,7,8,9}
};
```

(В операторах объявления подобного рода код, выделенный полужирным, может быть опущен.)

Упрощенные выражения инициализации массивов

Ранее уже было показано, как упростить выражения инициализации массивов, опуская ключевое слово new и объявление типа:

```
char[] vowels = new char[] {'а','о','и','е','ё','э','ы','у','ю','я'};
char[] vowels = {'а','о','и','е','ё','э','ы','у','ю','я'};
```

При другом подходе после ключевого слова new имя типа не указывается и компилятор должен будет самостоятельно вывести тип массива. Это удобное сокращение при передаче массивов в качестве аргументов.