Массивы

Массив – именованная структура данных, фиксированного размера, которая позволяет хранить, последовательность однотипных элементов, к которым можно обращаться с помощью индекса.

C# поддерживает следующие типы массивов:

- одномерные;
- многомерные.

Основные понятия массивов

Ранг (rank): количество измерений массива

Длина измерения (dimension length): длина отдельного измерения массива

Длина массива (array length): количество всех элементов массива

Одномерные массивы

Одномерный, или линейный массив – это конструкция фиксированной длины из набора элементов наперед заданного типа, доступ к которым осуществляется с использование одного индекса. Его можно рассматривать как *вектор*:

0	1	2	3	4	5	6	7	8	9

Синтаксис объявления массива, похожий на создание переменной, только после типа указываются квадратные скобки:

тип[] имя_массива;

Пример объявления целочисленного массива:

int[] array;

Одномерные массивы

Для создания массива используется следующий синтаксис:

```
тип[] имя_массива = new тип[длина_массива];
```

Присвоим, предварительно созданной, переменной массив из десяти элементов:

```
array = new int[10];
```

При этом все элементы массива будут иметь тип **int** и значения по умолчанию для этого типа данных - 0.

Для доступа к элементам массива, используются **индексы**, при этом начальный индекс равен **нулю**, соответственно последний индекс на единицу меньше длины массива.

Многомерные массивы

Многомерный массив, или массив массивов, объявляется путем задания последовательности константных выражений в квадратных скобках, следующей за *именем*:
<спецификация типа> <имя> [*<константное выражение>*][*<константное выражение>*] ... ;

Примеры объявлений массивов

```
double y[2][10];
// Двумерный массив вещественных чисел из
```

Свойство **Length**

Все массивы являются объектами и у них есть некоторые свойства.
Самым полезным для нас будет свойство Length, которое возвращает количество элементов в массиве (во всех размерностях)

```
static void Main(string[] args)
   {
      int[] numbers = new int[5];
      int size = numbers.Length; // size = 5
   }
```

Примеры инициализации массивов

```
int[] arr1 = new int[5] { 0, 2, 4, 6, 8 };
var a = new[] { 10, 20, 30 };
int t = a[2]; //t = 30, поскольку элемент с
   индексом 2 имеет значение 30
a[0] = 15; // 15, 20, 30
a[1] = 25; // 15, 25, 30
Console.WriteLine(a[1]);
Console.ReadKey(true);
```

Обычно для работы с массивами, используются циклы. Рассмотрим пример заполнения и вывода, на экран консоли, данных массива:

Цикл foreach предназначен для перебора элементов в контейнерах, в том числе в массивах. Формальное объявление цикла foreach:

```
foreach (тип_данных название_переменной in контейнер)
{ // действия
}
```

Например:

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in numbers)
{
    Console.WriteLine(i);
}
```

Подобные действия мы можем сделать и с помощью цикл for:

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.Length; i++)
{
    Console.WriteLine(numbers[i]);
}
```

В то же время цикл for более гибкий по сравнению с foreach. Если foreach последовательно извлекает элементы контейнера и только для чтения, то в цикле for мы можем перескакивать на несколько элементов вперед в зависимости от приращения счетчика, а также можем изменять элементы:

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.Length; i++)
{
    numbers[i] = numbers[i] * 2;
    Console.WriteLine(numbers[i]);
}
```

скопировать данные одного целочисленного массива в другой

```
var a = new[] { 1, 2, 3 };
var b = a; //a == {1, 2, 3}; b == {1, 2, 3}
a[0] = 9; //a == {9, 2, 3}; b == {9, 2, 3}
b[2] = 8; //a == {9, 2, 8}; b == {9, 2, 8}
```

Для создания физической копии массива, можно воспользоваться следующим синтаксисом:

```
int[] A = new[] { 10, 20, 30 };
int[] B = new int[A.Length];
for (int i = 0; i < B.Length; i++)
{
 B[i] = A[i];
}
```

```
int[,] table1 = new int[3, 3] { { 1, 2, 3 },
   { 4, 5, 6 }, { 7, 8, 9 } };
int[,] table2 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7,
   8, 9 } };
foreach (int i in table1)
                Console.Write($"{i} ");
Console.WriteLine();
```

Доступ к элементам осуществляется посредством двух индексов:

```
int[,] tbl = new int[2, 3];
tbl[0, 0] = 10;
tbl[1, 1] = 20;
foreach (int i in tbl1)
        Console.Write($"{i} ");
Console.WriteLine();
```

Но что если мы хотим отдельно пробежаться по каждой строке в таблице? В этом случае надо получить количество элементов в размерности. В частности, у каждого массива есть
метод GetUpperBound(dimension), который возвращает индекс последнего элемента в определенной размерности. И если мы говорим непосредственно о двухмерном массиве, то первая размерность (с индексом 0) по сути это и есть таблица. И с помощью выражения mas.GetUpperBound(0) + 1 можно получить количество строк таблицы, представленной двухмерным массивом.

А через mas.Length / rows можно получить

```
int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };
int rows = mas.GetUpperBound(0) + 1;
int columns = mas.Length / rows;

for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < columns; j++)
    {
        Console.Write($"{mas[i, j]} \t");
    }
    Console.WriteLine();
}
```

Найдем количество положительных чисел в массиве:

```
int[] numbers = { -4, -3, -2, -1, 0, 1, 2, 3, 4 };
int result = 0;
foreach(int number in numbers)
{
    if(number > 0)
    {
        result++;
    }
}
Console.WriteLine($"Число элементов больше нуля:
    {result}");
```

Циклы

Цикл – управляющая конструкция, предназначенная для многоразового выполнения одной или нескольких инструкций

Цикл while

Цикл с предусловием **while** – проверяет условие **в начале** выполнения, и имеет следующий синтаксис:

```
while(условное_выражение) { // инструкции }
```

Выполняется до тех пор, пока условие – истинно(равно **true**).

Пример: вывод в консоль цифр от 1 до 9:

```
var n = 1;
while (n < 10) {
Console.WriteLine(n);
n += 1;
}
```

Цикл do while

Цикл с постусловием **do while** – осуществляет проверку условия **после выполнения** одной итерации, то есть тело цикла, не зависимо от условий, выполниться хотя бы один раз.

Пример: вычисление суммы чисел от 1 до 10:

```
int k = 1;
int s = 0;
do {
s += k;
++k;
}
while (k <= 10);
Console.WriteLine("Sum = {0}", s);
```

Цикл for

Цикл со счетчиком **for** – это цикл в котором счетчик изменяет значение от начального до конечного значения, с заданным шагом, и для каждого значения счетчика выполняется одна итерация.

Имеет следующий синтаксис:

for(инициализатор; логическое условие; итератор) { // инструкции; }

Рассмотрим пример вычисления произведения чисел от 1 до 10:

```
int p = 1;
for (int i = 1; i <= 10; i++)
{
p *= i;
 }
```

Цикл foreach

Цикл foreach, или цикл просмотра – применяется к коллекциям(массивы, списки), перебирает все элементы коллекции, для каждой итерации берет один элемент.

```
var array = new[] { 1, 2, 3, 4, 5 }; //массив целых чисел
//в каждой итерации переменной item присваивается следующее значение массива
foreach (var item in array)
{ Console.WriteLine(item);
```

Операторы break и continue

Иногда необходимо выйти из цикла, не дождавшись его завершения, для этого используется оператор прерывания цикла - **break**.

```
int x = 0;
while (x < 100) { // при значении x больше 7 состоится досрочное прерывание цикла
if (x > 7) break;
Console.WriteLine(x);
x++;
}
```

Операторы break и continue

Если нужно пропустить одну или несколько инструкций, и сразу перейти к следующей итерации, используется оператор – **continue**.

```
int b = 0;
 for (int i = 1; i < 10; i++)
 { b += 10;
//при значении i = 4 в консоль ничего не выводиться
if (i == 4) continue;
Console.WriteLine("{0}. {1}", i, b);
}
```

Бесконечные циклы

Безусловные или бесконечные циклы, это циклы которые повторяются бесконечное количество раз. Часто используются на практике, когда необходимо выполнить неизвестное в начале цикла количество операций. Для выхода из цикла используется оператор **break**.

Варианты реализации:

```
while(true) { //тело цикла }
for(;;) { //тело цикла }
do { //тело цикла } while(true);
```

Пример использования бесконечного цикла для чтения текстовых строк из клавиатуры:

```
string text = "";
do {
Console.Write("Введите строку, или exit для
   прерывания ввода: ");
var s = Console.ReadLine();
if (s == "exit") break;
else text += s + "\r\n"; }
while (true);
```

Вывести на экран, консольного приложения, прямоугольный треугольник из звездочек “*”, с длиной катета m.

```
using System;
class Program
{
static void Main(string[] args)
{
int m = 13;
for (int i = 0; i <= m; i++)
{
int j = 0;
while (j < i)
{
Console.Write("*");
j++;
}
Console.WriteLine(); }
Console.ReadLine(); } }
```

Строки

Создание строк:

```
string text1 = null; //строка без
   значения
string text2 = ""; //пустая строка
string text3 = "Some text";
string text4 = new string('x', 4); //xxxx
string text5 = new string(new[] { 'M',
   'e', 's', 's', 'a', 'g', 'e' }); //Message
```

IsNullOrEmpty - проверка на пустоту

Для проверки на null и пустоту используется метод IsNullOrEmpty, он возвращает **true** если строка пуста или равна **null**.

```
string nullString = null;
var b1 = string.IsNullOrEmpty(nullString);
  //true
var b2 = string.IsNullOrEmpty("test"); // false
```

Объединение строк

Для **конкатенации**(объединения) строк можно использовать метод **Concat** или операторы “+” и “+=”:

```
string h = "Hello";
string w = "World";
h += " "; //добавляем пробел
w = w + "!"; //добавляем восклицательный знак
Console.WriteLine(string.Concat(h, w));
```

Для объединения последовательности(массива, коллекции) с разделителем, можно использовать метод **Join**:

```
//первый аргумент - разделитель для частей
var t1 = string.Join(",", "1", "2", "3"); //1,2,3
//целочисленный массив преобразуется в текст
var t2 = string.Join("; ", new[] { 7, 8, 9 }); //7; 8; 9
```

Смена регистра

Для преобразования строк в нижний регистр используется метод **ToLower**, в верхний – **ToUpper**.

```
var t = "Abdc";
var lower = t.ToLower(); //abcd
var upper = t.ToUpper(); //ABCD
```

Сравнение строк

```
var b1 = "abc" == "abc"; //true
var b2 = "bcd" != "Bcd"; //true
```

Compare

Сравнивает значения двух строк, возвращает целое число:

Число	Значение
Отрицательное	Первая строка меньше второй, или имеет значение null
Ноль	Строки равны
Положительное	Вторая строка меньше первой, или имеет значение null

```
int x = string.Compare( "qwerty", "asdfg"); //
    > 0
```

Замена

Для замены символов или части строки, используется метод **Replace**, как аргументы принимает искомую подстроку и текст на который ее нужно заменить(аналог операции найти и заменить в текстовых редакторах):

```
//замена слова
string n = "United States";
string m = n.Replace("States", "Kingdom");
Console.WriteLine(m); //United Kingdom
 //множественная замена символов
var f = "2;4;6;8;10";
Console.WriteLine(f.Replace(";", ", "));
//2, 4, 6, 8, 10
```

Вставка

Метод **Insert** предназначен для вставки подстроки с указанной в качестве аргумента позиции.

```
string v = "Some message"; //вставка слова
string p = v.Insert(5, "text");
Console.WriteLine(p); //Some textmessage
//вставляем пробел между словами
Console.WriteLine(p.Insert(9, " "));
//Some text message
```

Удаление

Метод **Remove** позволяет удалить часть строки. В качестве аргументов принимает позицию с которой начинается удаление, и количество символов, которые нужно удалить(если не указано, удаляются все символы до конца строки):

```
string text = "London is the Capital of Great Britain";
//удаляем слово Great
Console.WriteLine(text.Remove(25, 6));
   //оставляем только London
   Console.WriteLine(text.Remove(6));
```