Циклы

Циклы for и while

Цикл for

Цикл for в C# предоставляет механизм итерации, в котором определенное условие проверяется перед выполнением каждой итерации. Синтаксис этого оператора показан ниже:

for (инициализатор; условие; итератор) оператор (операторы)

Здесь:

инициализатор

это выражение, вычисляемое перед первым выполнением тела цикла (обычно инициализация локальной переменной в качестве счетчика цикла). Инициализация, как правило, представлена оператором присваивания, задающим первоначальное значение переменной, которая выполняет роль счетчика и управляет циклом;

условие

это выражение, проверяемое перед каждой новой итерацией цикла (должно возвращать true, чтобы была выполнена следующая итерация);

итератор

выражение, вычисляемое после каждой итерации (обычно приращение значения счетчика цикла).

Циклы for и while

Цикл while

Подобно for, **while** также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы while включают только одно выражение:

while(условие) оператор (операторы);

где *оператор* — это единственный оператор или же блок операторов, а *условие* означает конкретное условие управления циклом и может быть любым логическим выражением. В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Как и в цикле for, в цикле while проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

Циклы do while

Цикл do. . . while

Цикл do...while в C# — это версия while с постпроверкой условия. Это значит, что условие цикла проверяется после выполнения тела цикла. Следовательно, циклы do...while удобны в тех ситуациях, когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла do-while:

do { операторы; } while (условие);

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Тем не менее они зачастую используются для того, чтобы сделать конструкцию do-while более удобочитаемой и не путать ее с конструкцией цикла while. Цикл do-while выполняется до тех пор, пока условное выражение истинно.

Операторы перехода

Оператор break

С помощью **оператора break** можно специально организовать немедленный выход из цикла в обход любого кода, оставшегося в теле цикла, а также минуя проверку условия цикла. Когда в теле цикла встречается оператор break, цикл завершается, а выполнение программы возобновляется с оператора, следующего после этого цикла. Оператор break можно применять в любом цикле, предусмотренном в C#.

Операторы перехода

Оператор continue

С помощью **оператора continue** можно организовать преждевременное завершение шага итерации цикла в обход обычной структуры управления циклом. Оператор continue осуществляет принудительный переход к следующему шагу цикла, пропуская любой код, оставшийся невыполненным. Таким образом, оператор continue служит своего рода дополнением оператора break. В циклах while и do-while оператор continue вызывает передачу управления непосредственно условному выражению, после чего продолжается процесс выполнения цикла. А в цикле for сначала вычисляется итерационное выражение, затем условное выражение, после чего цикл продолжается

Операторы перехода

Оператор return

Оператор return организует возврат из метода. Его можно также использовать для возврата значения. Имеются две формы оператора return: одна — для методов типа void, т.е. тех методов, которые не возвращают значения, а другая — для методов, возвращающих конкретные значения.

Для немедленного завершения метода типа void достаточно воспользоваться следующей формой оператора return:

return;

Когда выполняется этот оператор, управление возвращается вызывающей части программы, а оставшийся в методе код пропускается.

Для возврата значения из метода в вызывающую часть программы служит следующая форма оператора return:

return значение;

Массивы в C#

Массивы в C#

В языке C# каждый индекс массива изменяется в диапазоне от 0 до некоторого конечного значения. Массивы в языке C# являются настоящими динамическими массивами. Как следствие этого, массивы относятся к ссылочным типам, память им отводится динамически в "куче". Массивы могут быть одномерными и многомерными.

Объявление одномерного массива

<тип>[] <объявители>;

каждый объявитель может быть именем или именем с инициализацией.

Объявление массива:

- с инициализацией
- с отложенной инициализацией

Объявление одномерного массива

Объявление массива с инициализацией:

- Явная инициализация константным массивом
- Создание массива с помощью операции new

Пример явной инициализации:

double[] x = {5.5, 6.6, 7.7};

в динамической памяти создаётся константный массив с заданными значениями, с которым и связывается ссылка.

Пример сознания с помощью операции new:

int[] d = new int[5];

массив создаётся в динамической памяти, его элементы получают начальные нулевые значения, и ссылка связывается с этим массивом.

Объявление одномерного массива

Объявление с отложенной инициализацией выполняется в 2 этапа:

1. Объявление массива
2. Инициализация массива

Объявление одномерного массива

Объявление массива

Пример:

int[ ] a;

При объявлении с отложенной инициализацией сам массив не формируется, а создаётся только ссылка на массив, имеющая неопределённое значение Null. Поэтому пока массив не будет реально создан и его элементы инициализированы, использовать его в вычислениях нельзя!!!

Объявление одномерного массива

Инициализация массива:

Пример:

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

Выражение, задающее границу изменения индексов, в динамическом случае может содержать переменные. Единственное требование – значения переменных должны быть определены в момент объявления!

Пример:

```
n=Convert.ToInt32(Console.ReadLine());
a = new int[n];
```

Работа с одномерным массивом

Ввод элементов

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {   int[]  a = new int[10];
            for (int i = 0; i < 10; i++)
            {
                Console.Write("a[{0}]=", i);
               a[i] = Convert.ToInt32(Console.ReadLine());
            }
        }
    }
}
```

Работа с одномерным массивом

Вывод элементов одномерного массива

```
Console.WriteLine("Массив a");
        for (int i = 0; i < 10; i++)
            Console.Write(a[i] + "\t");
        Console.ReadKey();
```

Работа с одномерным массивом

Новый оператор цикла:

foreach(тип идентификатор in массив) оператор

Цикл работает в полном соответствии со своим названием – тело цикла выполняется для каждого элемента в контейнере. Тип идентификатора должен быть согласован с типом элементов, хранящихся в массиве данных. На каждом шаге цикла идентификатор, задающий текущий элемент массива, получает значение очередного элемента в соответствии с порядком, установленным на элементах массива. С этим текущим элементом и выполняется тело цикла - выполняется столько раз, сколько элементов находится в массиве. Цикл заканчивается, когда полностью перебраны все элементы массива.

Работа с одномерным массивом

Вычисление суммы элементов:

```
Console.WriteLine("Массив a");
int s=0;
foreach (int i in a)
s+=a[i];
```

Многомерные массивы

Объявление многомерных массивов

<тип>[, ... ,] <объявители>;

Число запятых, увеличенное на единицу, задаёт размерность массива. Всё, что сказано для одномерных массивов, справедливо и для многомерных.

Пример явной инициализации двумерного массива:

```
int[,] matrix = {{1,2},{3,4}};
```

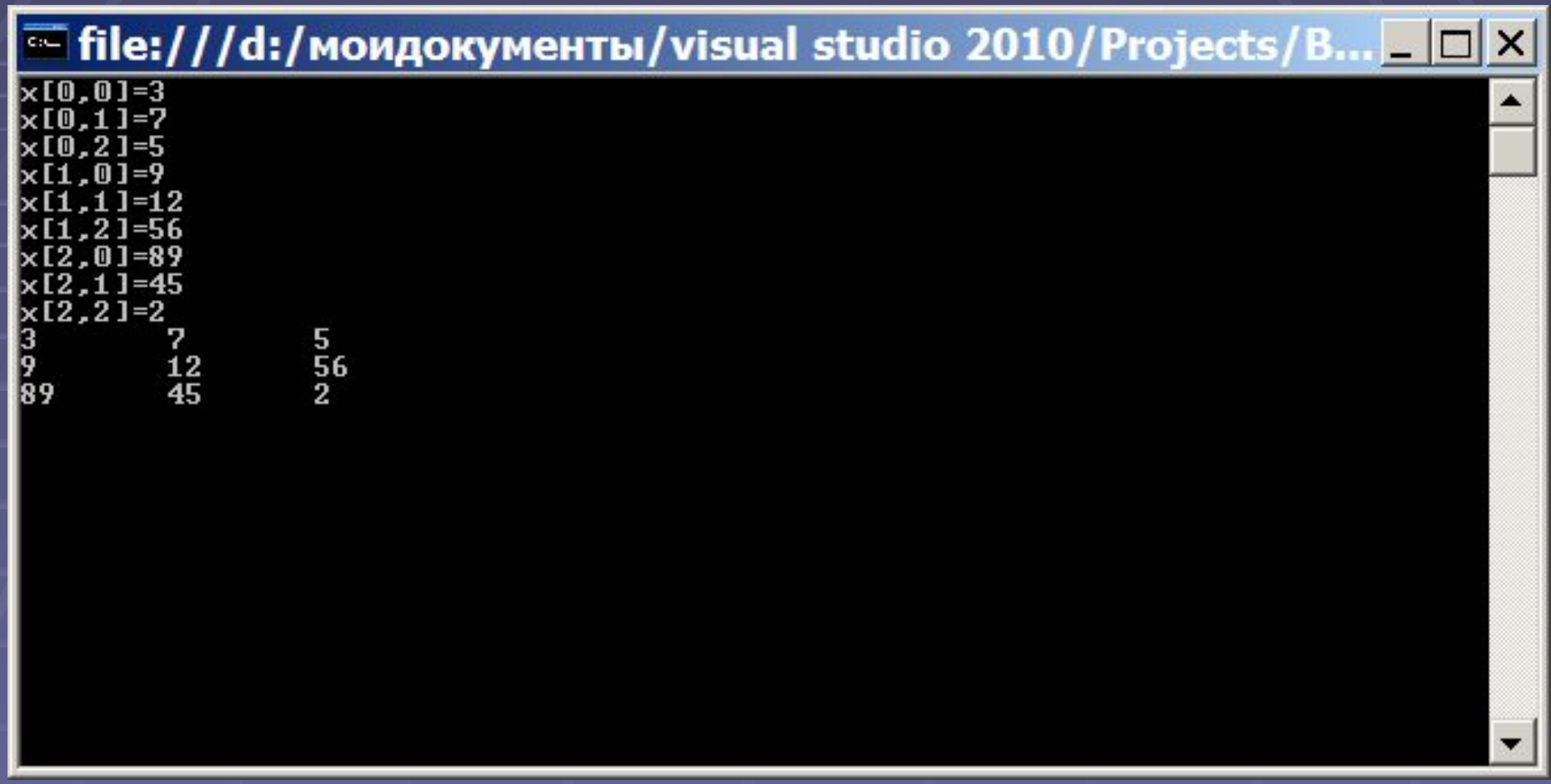
Многомерный массив

```
namespace Ввод_двумерного_массива
{
    class Program
    {
        static void Main(string[] args)
        {
            double[,] x = new double[3,3];
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                {
                    Console.Write("x[{0},{1}]=", i, j);
                    x[i, j] = Convert.ToDouble(Console.ReadLine());
                }
```

Многомерный массив

```
// Вывод элементов двумерного массива
for (int i = 0; i < 3; i++, Console.WriteLine())
                for (int j = 0; j < 3; j++)
                    Console.Write(x[i, j] + "\t");
            Console.ReadKey();
        }
    }
}
```

Многомерный массив



Многомерный массив

```
// Сумма элементов двумерного массива
int sum = 0;
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                sum = sum + x[i, j];
```

Класс Array

Все классы, являющиеся массивами, имеют много общего, поскольку все они являются потомками класса System.Array из библиотеки FCL .

Класс Array имеет довольно большое число собственных методов и свойств.

Класс Array

Методы класса Array:

1. Copy – позволяет копировать весь массив или его часть в другой массив.
2. IndexOf, LastIndexOf – определяют индексы первого и последнего вхождения образца в массив, возвращая значение -1, если такового вхождения не обнаружено.
3. Reverse – выполняет обращение массива, переставляя элементы в обратном порядке.
4. Sort – осуществляет сортировку массива.
5. BinarySearch – определяет индекс первого вхождения образца в отсортированный массив, используя алгоритм двоичного поиска.
6. Clear – обнуление элементов массива в заданном диапазоне

Класс Array

Примеры использования методов:

1. Array.Copy(a, b, 4);
2. int k= Array.IndexOf(a,100);
3. Array.Reverse(a);
4. Array.Sort(a);
5. int k = Array.BinarySearch(a, 100);
6. Array.Clear(a, 3, 4);