

# **Тема 7. Многомерные массивы. Библиотека LINQ.**

# **Учебные вопросы:**

- 1. Многомерные массивы.**
- 2. Свойства и методы массивов.**
- 3. Библиотека LINQ. Основные свойства и методы.**

# 1. Многомерные массивы.

Определение: Многомерный массив — это массив, элементы которого сами являются массивами.

Многомерные массивы позволяют хранить данные в форме матриц, таблиц или других многомерных структур.

Чаще всего используются двумерные массивы, которые можно представить как таблицу с рядами и столбцами. Однако можно создавать массивы с тремя и более измерениями.

# Создание и инициализация многомерных массивов

## Объявление и создание двумерного массива:

```
int[,] matrix = new int[3, 3]; // Создание двумерного массива размером 3x3
```

## Инициализация двумерного массива:

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

**Column** →

Row ↓

	0	1	2
0	1 $x[0, 0]$	2 $x[0, 1]$	3 $x[0, 2]$
1	3 $x[1, 0]$	4 $x[1, 1]$	5 $x[1, 2]$

## Создание трехмерного массива:

```
int[,,] threeDArray = new int[2, 2, 2]; // Создание трехмерного массива 2x2x2
```

## Доступ к элементам многомерного массива

### Обращение к элементу двумерного массива:

```
int value = matrix[1, 2]; // Доступ ко второму ряду, третьему столбцу (значение 6)
```

### Изменение значения элемента:

```
matrix[0, 0] = 10; // Изменение значения первого элемента (левый верхний угол)
```

## Перебор элементов многомерного массива

Использование вложенных циклов **for** для двумерного массива:

```
for (int i = 0; i < matrix.GetLength(0); i++) // Перебор рядов
{
    for (int j = 0; j < matrix.GetLength(1); j++) // Перебор столбцов
    {
        Console.Write(matrix[i, j] + " ");
    }
    Console.WriteLine();
}
```

## Использование цикла **foreach**:

Цикл **foreach** в C# можно использовать для перебора элементов двумерного массива, хотя при этом нельзя получить доступ к индексам элементов.

Вместо этого перебираются все элементы массива по порядку, начиная с первого элемента первой строки и заканчивая последним элементом последней строки.

Пример:

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

foreach (int element in matrix)
{
    // 4 5 6 1 2 3 7 8 9
    Console.WriteLine(element + " ");
}
```

## Примеры использования многомерных массивов

- **Матрицы и таблицы:** Двумерные массивы часто используются для хранения данных в виде таблиц. Например, таблица оценок студентов, где строки представляют студентов, а столбцы — предметы.
- **Графики и изображения:** Двумерные массивы могут использоваться для представления пикселей изображения, где каждый элемент содержит значения цвета пикселя.
- **Математические операции:** Многомерные массивы часто применяются в математических вычислениях, таких как операции с матрицами (сложение, умножение, транспонирование).

# **Преимущества и недостатки многомерных массивов**

## **Преимущества:**

- Возможность хранения и работы с данными в виде таблиц и многомерных структур.
- Удобство представления данных, имеющих более двух измерений.

## **Недостатки:**

- Сложность управления: Увеличение количества измерений усложняет работу с массивом, включая создание, доступ и перебор элементов.
- Повышенные требования к памяти: Многомерные массивы могут занимать значительно больше памяти, особенно при увеличении числа измерений и размеров массива.

## 2. Свойства и методы массивов.

### Основные свойства массивов

#### Длина массива (Length):

Свойство **Length** возвращает общее количество элементов в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };
int length = numbers.Length; // length будет равно 5
```

Применение: Часто используется для создания циклов, перебирающих элементы массива.

## Размерность массива (Rank):

Свойство **Rank** возвращает количество измерений в массиве.

Пример:

```
int[,] matrix = new int[3, 4];
int rank = matrix.Rank; // rank будет равно 2, так как это двумерный массив
```

Применение: Полезно при работе с многомерными массивами для определения их структуры.

## Тип элементов массива (GetType):

Метод **GetType()** возвращает тип данных массива, включая информацию о типе элементов, которые он содержит.

Пример:

```
int[] numbers = { 1, 2, 3 };
Type type = numbers.GetType(); // Возвращает System.Int32[]
```

Применение: Используется для получения информации о типе данных массива, что может быть полезно для отладки.

# Методы для работы с массивами

## Сортировка (Array.Sort()):

Метод **Array.Sort()** сортирует элементы массива по возрастанию.

Пример:

```
int[] numbers = { 3, 1, 4, 1, 5 };
Array.Sort(numbers); // После сортировки: { 1, 1, 3, 4, 5 }
```

Применение: Полезно для упорядочивания данных в массиве.

## Обратный порядок (Array.Reverse):

Метод **Array.Reverse()** переворачивает порядок элементов в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };
Array.Reverse(numbers); // После обращения: { 5, 4, 3, 2, 1 }
```

**Применение:** Полезно для упорядочивания данных в массиве.

## Поиск элементов (Array.IndexOf, Array.Find):

**Array.IndexOf** возвращает индекс первого вхождения элемента в массиве.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };
int index = Array.IndexOf(numbers, 3); // Возвращает 2 (индекс элемента со значением 3)
```

Применение: Эти методы полезны для поиска элементов в массиве по значению.

## Очистка массива (Array.Clear):

Метод **Array.Clear()** обнуляет все элементы массива, задавая им значение по умолчанию.

Пример:

```
int[] numbers = { 1, 2, 3, 4, 5 };
Array.Clear(numbers, 0, numbers.Length); // Все элементы массива станут равны 0
```

Применение: Применяется, когда нужно быстро очистить массив от всех значений.

### **3. Библиотека LINQ. Основные свойства и методы.**

LINQ (Language Integrated Query) — это мощная библиотека в C#, предоставляющая удобные инструменты для выполнения запросов к коллекциям данных, таким как массивы, списки, базы данных, XML-документы и многое другое.

LINQ позволяет писать запросы непосредственно в C# с использованием синтаксиса, напоминающего SQL-запросы.

## Пространство имен LINQ.

Для работы с LINQ необходимо подключить  
пространство имен:

```
using System.Linq;
```

## Основные свойства и методы LINQ.

**Sum()**. Возвращает сумму всех элементов числового массива.

Пример:

```
int[] numbers = { 1, 2, 3 };  
  
int sum = numbers.Sum();  
  
// sum = 6
```

**Min()**. Возвращает минимальное значение в коллекции.

Пример:

```
int[] numbers = { 5, 2, 8 };
int min = numbers.Min();
// min = 2
```

**Max().** Возвращает максимальное значение в коллекции.

Пример:

```
int[] numbers = { 5, 2, 8 };
int max = numbers.Max();
// max = 8
```

**Average()**. Возвращает среднее арифметическое элементов числового массива.

Пример:

```
int[] numbers = { 1, 2, 3 };
double average = numbers.Average();
// average = 2.0
```

## **Контрольные вопросы:**

- В чем разница между одномерными и многомерными массивами?
- Как осуществляется доступ к элементам массива и их изменение?
- Почему массивы считаются эффективными структурами данных для хранения и обработки информации?
- Объясните, как можно перебирать элементы многомерного массива с помощью цикла `for`.
- Какие альтернативные структуры данных можно использовать, если размер данных изменяется динамически?

# **Материалы лекций:**

<https://github.com/ShViktor72/Education2025>