Введение

При разработке программного обеспечения разработчики сталкиваются с необходимостью обрабатывать наборы однородных данных: список студентов в группе, финансовые показатели за месяц, координаты точек на графике. Использование отдельных переменных для каждого значения в таком наборе является неэффективным и не масштабируемым. Для решения этой задачи в C# и других языках программирования существует фундаментальная структура данных - **массив**.

Массив в C# - это ссылочный тип данных, представляющий собой индексированную коллекцию элементов одного и того же типа, размещенных в непрерывной области памяти.

Ключевые характеристики массива:

1. Массив может содержать элементы только одного, строго определенного при его создании, типа (например, `int[]` для целых чисел или `string[]` для строк).
2. Размер массива определяется в момент его создания (инстанцирования) и не может быть изменен в дальнейшем.
3. Доступ к элементам массива осуществляется по целочисленному индексу. Нумерация индексов всегда начинается с нуля. Первый элемент имеет индекс `0`, второй — `1`, и так далее, до `Length - 1`.

1. Декларация и инициализация массива

Работа с массивом состоит из двух основных этапов: декларации (объявления) переменной массива и его инициализации (создания экземпляра и выделения памяти).

1.1. Декларация

На этом этапе мы объявляем переменную, которая будет ссылаться на объект массива в памяти.

```
// Декларация переменной 'numbers', которая может ссылаться на массив целых чисел.
int[] numbers;
```

На данный момент переменная `numbers` имеет значение `null`, так как память под сам массив еще не выделена.

1.2. Инициализация

Инициализация - процесс создания экземпляра массива с помощью оператора `new`. В этот момент мы обязаны указать его размер.

```
// Инициализация: выделение в памяти места для 5 элементов типа int.
numbers = new int[5];
```

Эти два шага чаще всего объединяют в одну строку:

```
int[] numbers = new int[5];
```

После выполнения этого кода в управляемой куче (managed heap) будет выделен непрерывный блок памяти, достаточный для хранения пяти значений типа `int`. Все элементы массива будут автоматически инициализированы значениями по умолчанию для их типа (`0` для числовых типов, `false` для `bool`, `null` для ссылочных типов).

1.3. Инициализатор массива

Существует синтаксический сахар для одновременного объявления, создания и заполнения массива данными.

```
// Размер массива (3) будет выведен компилятором автоматически.
string[] daysOfWeek = { "Monday", "Tuesday", "Wednesday" };
```

Эта запись эквивалентна следующей:

```
string[] daysOfWeek = new string[3];
daysOfWeek[0] = "Monday";
daysOfWeek[1] = "Tuesday";
daysOfWeek[2] = "Wednesday";
```

2. Доступ к элементам массива

Доступ для чтения или записи элемента осуществляется через указание имени массива и индекса элемента в квадратных скобках `[]`.

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

// Запись значения в элемент с индексом 3 (четвертый по счету элемент).
data[3] = 99;

// Чтение значения из элемента с индексом 3.
int value = data[3]; // value будет равно 99

Console.WriteLine(data[0]); // Выведет 0 (значение по умолчанию).
```

Попытка обратиться к элементу по индексу, который находится за пределами допустимого диапазона (т.е. меньше `0` или больше либо равен `Length`), приведет к

генерации исключения `System.IndexOutOfRangeException` во время выполнения программы.

3. Свойство `Length` и итерация по массиву

Каждый массив имеет свойство `Length`, которое возвращает общее количество элементов, которое может содержать массив.

```
double[] measurements = new double[250];
Console.WriteLine(measurements.Length); // Выведет 250
```

Свойство `Length` является ключевым для организации перебора элементов массива.

3.1. Цикл `for`

Классический способ итерации, который предоставляет полный контроль над процессом, включая доступ к индексу элемента.

```
for (int i = 0; i < measurements.Length; i++)
{
    // i будет последовательно принимать значения от 0 до 249.
    measurements[i] = i * 1.5; // Пример операции с элементом.
}
```

3.2. Цикл `foreach`

Более высокоуровневый и синтаксически простой способ перебора элементов, когда индекс не требуется. Он обеспечивает итерацию по коллекции только для чтения.

```
string[] fruits = { "Apple", "Orange", "Banana" };

foreach (string fruit in fruits)
{
    // Переменная 'fruit' на каждой итерации будет содержать следующий
элемент массива.
    Console.WriteLine(fruit);
}
```

Цикл `foreach` защищает от ошибок выхода за пределы массива и делает код более читаемым.

4. Многомерные массивы

C# поддерживает многомерные массивы, которые полезны для представления табличных данных, матриц или сеток.
Массивы характеризуются таким понятием как ранг или количество измерений. Выше мы рассматривали массивы, которые имеют одно измерение (то есть их ранг равен 1) - такие массивы можно представлять в виде ряда (строки или столбца) элемента. Но

массивы также бывают многомерными. У таких массивов количество измерений (то есть ранг) больше 1.

Прямоугольные массивы (Rectangular arrays)
Это массивы, где каждая строка имеет одинаковую длину.

```
// Декларация и инициализация двумерного массива (матрицы 3x4).
int[,] matrix = new int[3, 4];

// Доступ к элементу на пересечении строки с индексом 1 и столбца с индексом 2.
matrix[1, 2] = 5;
```

Определенную сложность может представлять перебор многомерного массива. Прежде всего надо учитывать, что длина такого массива - это совокупное количество элементов.

```
int[,] numbers = { { 1, 2, 3 }, { 4, 5, 6 }};
foreach (int i in numbers)
    Console.Write($"{i} ");
```

В данном случае длина массива `numbers` равна 6. И цикл foreach выводит все элементы массива в строку:

```
1 2 3 4 5 6
```

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(номер_размерности), 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.

Для получения размера каждого измерения используется метод `GetLength(dimension)`.

```
int rows = numbers.GetLength(0);    // Вернет 3
int cols = numbers.GetLength(1);    // Вернет 4
```