# ****C# Arrays****

## ****Introduction****

In C#, an array is a collection of elements of the same type that are stored in contiguous memory locations. Arrays are objects of the base type System.Array. The array index in C# starts at 0, which means the first element is accessed with an index of 0.

### ****Advantages of C# Arrays:****

* **Code Optimization:** Arrays help reduce the amount of code needed to manage multiple variables of the same type.
* **Random Access:** Elements can be accessed randomly using an index, making data retrieval efficient.
* **Easy Traversal:** Arrays can be easily traversed using loops, allowing iteration over each element.
* **Data Manipulation:** Data in arrays can be easily manipulated, such as sorting, searching, and modifying values.
* **Ease of Sorting:** Arrays can be sorted with built-in methods, making it simple to organize data.

### ****Disadvantages of C# Arrays:****

* **Fixed Size:** Once an array's size is defined, it cannot be changed. This may lead to either wasted memory or the need for reallocation.

## ****C# Array Types****

C# supports three main types of arrays:

1. **Single Dimensional Array**
2. **Multidimensional Array**
3. **Jagged Array**

## ****1. Single Dimensional Array****

### ****Declaration and Initialization****

A single-dimensional array is declared by specifying the type followed by square brackets []. The array can be created and initialized in several ways:

int[] arr = new int[5]; // Creating an array with 5 elements

Note: Placing square brackets after the identifier (e.g., int arr[]) will result in a compile-time error.

### ****Example: Declaring, Initializing, and Traversing an Array****

Here is a simple example of a single-dimensional array in C#. This example demonstrates how to declare, initialize, and traverse an array.

using System;

public class ArrayExample

{

public static void Main(string[] args)

{

int[] arr = new int[5]; // Creating an array with 5 elements

arr[0] = 10; // Initializing array

arr[2] = 20;

arr[4] = 30;

// Traversing array

for (int i = 0; i < arr.Length; i++)

{

Console.WriteLine(arr[i]);

}

}

}

In this example:

* The array arr is created with a size of 5.
* Only elements at indices 0, 2, and 4 are initialized.
* The other elements remain at their default value of 0.

### ****Example: Declaration and Initialization at the Same Time****

Arrays can be declared and initialized simultaneously in several ways:

int[] arr = new int[5] { 10, 20, 30, 40, 50 }; // Declaring with size

int[] arr = new int[] { 10, 20, 30, 40, 50 }; // Size is omitted

int[] arr = { 10, 20, 30, 40, 50 }; // Both size and new operator are omitted

### ****Example: Declaring, Initializing, and Traversing****

using System;

public class ArrayExample

{

public static void Main(string[] args)

{

int[] arr = { 10, 20, 30, 40, 50 }; // Declaration and initialization

// Traversing array

for (int i = 0; i < arr.Length; i++)

{

Console.WriteLine(arr[i]);

}

}

}

### ****Traversal Using**** foreach ****Loop****

Arrays can also be traversed using the foreach loop, which iterates over each element in the array:

using System;

public class ArrayExample

{

public static void Main(string[] args)

{

int[] arr = { 10, 20, 30, 40, 50 }; // Creating and initializing array

// Traversing array using foreach loop

foreach (int i in arr)

{

Console.WriteLine(i);

}

}

}

# ****C# Multidimensional Arrays****

## ****Introduction****

A multidimensional array, also known as a rectangular array in C#, is an array that contains multiple rows and columns, forming a matrix-like structure. Multidimensional arrays can be two-dimensional or three-dimensional, where the data is stored in a tabular form (rows \* columns).

### ****Declaration of Multidimensional Arrays****

To create a multidimensional array in C#, you use commas inside the square brackets to define the dimensions.

* **2D Array:** int[,] arr = new int[3,3];
* **3D Array:** int[,,] arr = new int[3,3,3];

## ****C# Multidimensional Array Example****

This section demonstrates how to declare, initialize, and traverse a two-dimensional array in C#.

### ****Example: Declaring, Initializing, and Traversing a 2D Array****

using System;

public class MultiArrayExample

{

public static void Main(string[] args)

{

int[,] arr = new int[3,3]; // Declaration of a 2D array

arr[0,1] = 10; // Initialization

arr[1,2] = 20;

arr[2,0] = 30;

// Traversal

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

Console.Write(arr[i,j] + " ");

}

Console.WriteLine(); // New line at each row

}

}

}

### ****Output:****

0 10 0

0 0 20

30 0 0

### ****Explanation:****

* The 2D array arr is declared with dimensions [3,3], meaning it has 3 rows and 3 columns.
* Only specific elements are initialized with values, while the rest default to 0.
* The traversal is done using nested for loops to access each element in the matrix.

## ****C# Multidimensional Array Example: Declaration and Initialization at the Same Time****

There are multiple ways to declare and initialize a multidimensional array at the same time:

### ****Method 1: Specifying Array Size****

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

### ****Method 2: Omitting Array Size****

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

### ****Method 3: Omitting**** new ****Operator****

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

### ****Example: Declaration, Initialization, and Traversal****

using System;

public class MultiArrayExample

{

public static void Main(string[] args)

{

int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }; // Declaration and initialization

// Traversal

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

Console.Write(arr[i,j] + " ");

}

Console.WriteLine(); // New line at each row

}

}

}

### ****Output:****

1 2 3

4 5 6

7 8 9

### ****Explanation:****

* The array arr is initialized at the time of declaration with specific values.
* The traversal is done in the same way as the previous example, printing each element of the matrix.

### Two-Dimensional Array (2D):

For the 2D array int[,] array2D = new int[3, 2] { { 4, 5 }, { 5, 0 }, { 3, 1 } };, imagine a table with 3 rows and 2 columns:

| Row/Column | Column 0 | Column 1 |

|------------|----------|----------|

| Row 0 | 4 | 5 |

| Row 1 | 5 | 0 |

| Row 2 | 3 | 1 |

### Three-Dimensional Array (3D):

For the 3D array int[,,] array3D = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } }, { { 7, 8, 9 }, { 10, 11, 12 } } };, you can think of it as two 2D tables stacked on top of each other:

**First Layer (Depth 0):**

| Row/Column | Column 0 | Column 1 | Column 2 |

|------------|----------|----------|----------|

| Row 0 | 1 | 2 | 3 |

| Row 1 | 4 | 5 | 6 |

**Second Layer (Depth 1):**

| **Row/Column** | **Column 0** | **Column 1** | **Column 2** |
| --- | --- | --- | --- |
| **Row 0** | 7 | 8 | 9 |
| **Row 1** | 10 | 11 | 12 |

This layout represents the two layers of the 3D array, where each layer is a 2x3 matrix. The different layers are separated by depth (or "stacked" along the third dimension).