## Multi-Dimensional Arrays in C#

Multi-dimensional arrays in C# are arrays that have more than one dimension. They can be used to store data in a tabular format (like a grid or matrix) and are often used in mathematical computations, game development, and other areas where a structured representation of data is required.

## Types of Multi-Dimensional Arrays

### Rectangular Arrays

#### *Declaration and Initialization*

To declare a rectangular array, you specify the number of dimensions by using commas inside the square brackets.

```csharp
// Declare a 2D array
int[,] array2D;

// Initialize the array with dimensions 3x2
array2D = new int[3, 2];

// Declare and initialize in one step
int[,] array2D = new int[3, 2] {
  { 1, 2 },
  { 3, 4 },
  { 5, 6 }
};
```

#### *Accessing Elements*

You access elements in a rectangular array by specifying the row and column indices.

```csharp
int value = array2D[1, 1]; // Accesses the element in the second row and second column (value 4)
array2D[2, 0] = 7; // Sets the element in the third row and first column to 7
```

### Jagged Arrays

#### *Declaration and Initialization*

To declare a jagged array, you use an array of arrays.

```csharp
// Declare a jagged array
int[][] jaggedArray;

// Initialize the array with 3 rows
jaggedArray = new int[3][];

// Initialize each row separately
jaggedArray[0] = new int[2];
jaggedArray[1] = new int[3];
jaggedArray[2] = new int[1];

// Declare and initialize in one step
int[][] jaggedArray = new int[][] {
   new int[] { 1, 2 },
   new int[] { 3, 4, 5 },
   new int[] { 6 }
};
```

#### *Accessing Elements*

You access elements in a jagged array by specifying the row and column indices.

```csharp
int value = jaggedArray[1][2]; // Accesses the element in the second row and third column (value 5)
jaggedArray[0][1] = 7; // Sets the element in the first row and second column to 7
```

## Differences between Rectangular and Jagged Arrays

- **Memory Allocation**:
  - Rectangular arrays allocate memory for the entire grid upfront.
  - Jagged arrays allocate memory for each sub-array independently, which can be more efficient if the rows are of varying lengths.

- **Access Syntax**:

  - Rectangular arrays use a single set of square brackets with two indices: `array2D[row, column]`.
  - Jagged arrays use two sets of square brackets: `jaggedArray[row][column]`.

- **Flexibility**:
  - Rectangular arrays are simpler and more straightforward when dealing with a uniform grid.
  - Jagged arrays offer more flexibility when the structure is not uniform, such as representing a triangular matrix or other non-rectangular data structures.

## Practical Examples

### Example 1: Rectangular Array

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

for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        Console.Write(matrix[i, j] + " ");
    }
    Console.WriteLine();
}
```

### Example 2: Jagged Array

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

for (int i = 0; i < jaggedArray.Length; i++) {
    for (int j = 0; j < jaggedArray[i].Length; j++) {
        Console.Write(jaggedArray[i][j] + " ");
    }
```

```
    Console.WriteLine();
}
```