Jagged Arrays in C#

In C#, a jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. Unlike multidimensional arrays, where the arrays form a matrix-like structure, jagged arrays allow for more flexibility as each row (or element array) can have a different length.

Declaration and Initialization

A jagged array is declared using square brackets. Here's an example of how to declare and initialize a jagged array:

```
// Declaring a jagged array with 3 elements
int[][] jaggedArray = new int[3][];

// Initializing the jagged array
jaggedArray[0] = new int[] { 1, 2, 3 };
jaggedArray[1] = new int[] { 4, 5 };
jaggedArray[2] = new int[] { 6, 7, 8, 9 };
```

Accessing Elements

You can access the elements of a jagged array using the index of the sub-arrays and the index within those sub-arrays:

```
int value = jaggedArray[0][1]; // Accesses the second element of the first array, which is 2
```

Example

Here's a more complete example demonstrating the use of a jagged array:

```
using System;

class Program
{
    static void Main()
    {
        // Declare and initialize a jagged array
        int[][] jaggedArray = new int[3][];
        jaggedArray[0] = new int[] { 1, 2, 3 };
        jaggedArray[1] = new int[] { 4, 5 };
        jaggedArray[2] = new int[] { 6, 7, 8, 9 };
```

```
        // Iterate through the jagged array
        for (int i = 0; i < jaggedArray.Length; i++)
        {
            Console.Write("Element " + i + ": ");
            for (int j = 0; j < jaggedArray[i].Length; j++)
            {
                Console.Write(jaggedArray[i][j] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

Key Points

-
  - Flexibility: Jagged arrays offer flexibility in terms of different row sizes.
  - Declaration: Declared with double square brackets (e.g., int[][]).
  - Initialization: Each row is individually initialized.
  - Access: Elements are accessed using two indices.

Use Cases

Jagged arrays are useful in scenarios where data structures with non-uniform dimensions are needed, such as:

-
  - Representing a table with rows of varying lengths.
  - Storing data that naturally fits a hierarchical structure, such as a list of lists.