**Jagged Array in C#**

In C#, a jagged array is a multidimensional array in which every element of the main array is itself an array. A jagged array, unlike the traditional multidimensional array, allows the inner arrays to be of differing lengths, giving it a versatile and strong data structure for dealing with irregular or uneven data. One of the unique characteristics of jagged arrays is that their elements are reference types that are initialized to null by default. This means that when you create a jagged array, its elements are not assigned any specific values unless you manually initialize them with individual arrays.

**Declaration**

In jagged arrays, the user is only required to specify the number of rows and not the number of columns. If the user also specifies the number of columns, the array loses its jagged characteristic and becomes a traditional multi-dimensional array.

**Syntax:**

dataType[][] nameOfArray = new dataType[rows][];

**Example:**

int[][] jagged\_array = new int[3][]

In the above example, a jagged array is declared, where int is the array's data type, jagged\_array is its name, and [3][] is the number of elements (arrays) included within the jagged array.

We can set the size of the individual array since we know each member of a jagged array is itself an array. For example,

*// set size of the first array as 3*

jagged\_array[0] = new int[3];

*// set size of second array as 4*

jagged\_array[1] = new int[4];

**Initialization and Filling Elements in Jagged Array**

There are several ways to initialize and filling the elements of a jagged array in C#.

1. **Using the index number:** After declaring a jagged array, we can use the index number to initialize it. This implies that we can dynamically set values for each member in the array by accessing its specific index position. For example,

*// Declaring a jagged array*

int[][] jagged\_array = new int[2][];

*// Initialize the first array*

jagged\_array[0][0] = 3;

jagged\_array[0][1] = 2;

jagged\_array[0][2] = 5;

*// Initialize the second array*

jagged\_array[1][0] = 3;

jagged\_array[1][1] = 5;

In the above example, the index at the first square bracket indicates the index of the jagged array element, and the index at the second square bracket represents the index of the element inside each element of the jagged array.

1. **Initialize without setting the size of array elements:** When the size of the jagged array is not predetermined or needed, its elements can be directly initialized with specific values using inline initialization. For example,

*// Declaring string jagged array*

int[ ][ ] jagged\_array = new int[2] [ ];

*// Initializing each array*

jagged\_array[0] = new int[] {4, 3, 8};

jagged\_array[1] = new int[] {1, 5};

1. **Initialize while declaring Jagged Array:** In C#, we can use array initializers directly to initialize the jagged array upon declaration. This method allows you to declare and fill the jagged array's elements in one step, making the code more compact and expressive. For example,

int[ ][ ] jagged\_array = {

new int[ ] {10, 15, 20},

new int[ ] {12, 24},

new int[ ] {18, 36}

};

**C# Jagged Array Example**

**Example of Jagged Array in C# which declares, initializes and traverses jagged arrays**

Here is a C# jagged array program that declares, initializes, and traverses jagged arrays.

using System;

public class Jagged\_Array

{

public static void Main()

{

*// Declaring the jagged array*

int[][] jagged\_array = new int[2][];

*// Initializing each array*

jagged\_array[0] = new int[] {4, 3, 8, 6};

jagged\_array[1] = new int[] {1, 5, 7};

*// Traversing the array elements*

for (int i = 0; i < jagged\_array.Length; i++)

{

System.Console.Write("Row[{0}]: ", i);

for (int j = 0; j < jagged\_array[i].Length; j++)

{

System.Console.Write(jagged\_array[i][j]+" ");

}

System.Console.WriteLine();

}

}

}

**Output:**

Row[0]: 4 3 8 6

Row[1]: 1 5 7

**Example of Jagged Array which Initializes the Jagged Arrays Upon Declaration**

Here is a C# jagged array example that initializes the jagged arrays upon declaration.

using System;

public class Jagged\_Array

{

public static void Main()

{

int[][] jagged\_array = new int[3][]{

new int[ ] {10, 15, 20},

new int[ ] {12, 24, 36},

new int[ ] {18, 36}

};

*// Traversing the array elements*

for (int i = 0; i < jagged\_array.Length; i++)

{

System.Console.Write("Row[{0}]: ", i);

for (int j = 0; j < jagged\_array[i].Length; j++)

{

System.Console.Write(jagged\_array[i][j]+" ");

}

System.Console.WriteLine();

}

}

}

**Output:**

Row[0]: 10 15 20

Row[1]: 12 24 36

Row[2]: 18 36

**Accessing the Elements**

To access the elements of a jagged array, use the index notation as you would with any other multi-dimensional array. However, since a jagged array is an array of arrays, you need to specify both the row and column indices precisely within the array name to access a specific element.

**Example:**

*// Accessing and Assigning 45 to the second element ([1]) of the second array ([1]):*

jagged\_arrary[1][1] = 45;

*// Accessing and Assigning 32 to the first element ([0]) of the third array ([2]):*

jagged\_arrary[2][0] = 32;

Here's a C# program that demonstrates how to access the elements of a jagged array:

using System;

public class Jagged\_Array

{

public static void Main()

{

int[][] jagged\_array = new int[2][]{

new int[ ] {95, 45, 20},

new int[ ] {32, 64},

};

*// Accessing and printing elements of the jagged array*

for (int i = 0; i < jagged\_array.Length; i++)

{

for (int j = 0; j < jagged\_array[i].Length; j++)

{

System.Console.Write("jagged\_array[{0}][{1}]: {2} ", i , j, jagged\_array[i][j]);

}

System.Console.WriteLine();

}

}

}

**Output:**

jagged\_array[0][0]: 95 jagged\_array[0][1]: 45 jagged\_array[0][2]: 20

jagged\_array[1][0]: 32 jagged\_array[1][1]: 64

**Jagged Arrays With Multidimensional Arrays**

In C#, you can use jagged arrays in conjunction with multidimensional arrays to create complex data structures. This allows the creation of arrays of arrays, with some of the elements being multi-dimensional arrays. The definition and initialization of a 1-D jagged array containing three two-dimensional array elements of varying sizes are shown below.

int[][, ] jagged\_array = new int[3][, ]

{

new int[, ] { {1, 8}, {2, 6} },

new int[, ] { {5, 2}, {14, 12}, {13, 23} },

new int[, ] { {71, 28}, {13, 41}, {10, 34} },

};

To access the individual elements, as seen in the following example, which displays the value of the second array's element [1, 0].

System.Console.Write("{0}", jagged\_array[1][1, 0]);

**Example:**

Here is a C# program that demonstrates the mixing of a 1-D Jagged Array with three 2-D arrays.

using System;

public class Jagged\_Array

{

public static void Main()

{

*// Declaration and Initialization of Jagged array with three 2-D arrays*

int[][, ] jagged\_array = new int[3][, ]

{

new int[, ] { {1, 8}, {2, 6} },

new int[, ] { {5, 2}, {14, 12}, {13, 23} },

new int[, ] { {71, 28}, {13, 41}, {10, 34} },

};

*// Accessing and printing elements of the jagged array*

for (int i = 0; i < jagged\_array.Length; i++)

{

Console.WriteLine($"Elements of the Jagged Array {i+1}:");

*// GetLength method takes an integer that indicates the array's dimension.*

for (int j = 0; j < jagged\_array[i].GetLength(0); j++)

{

for (int k = 0; k < jagged\_array[i].GetLength(1); k++)

{

Console.Write("{0} ",jagged\_array[i][j, k]);

}

Console.WriteLine();

}

Console.WriteLine();

}

}

}

**Output:**

Elements of the Jagged Array 1:

1 8

2 6

Elements of the Jagged Array 2:

5 2

14 12

13 23

Elements of the Jagged Array 3:

71 28

13 41

10 34

**Conclusion**

* An array is a fundamental data structure in programming that allows you to store a group of elements of the same data type in contiguous memory locations.
* In C#, a jagged array is a multidimensional array in which every element of the main array is itself an array.
* A jagged array, unlike the traditional multidimensional array, allows the inner arrays to be of differing lengths, giving it a versatile and strong data structure for dealing with irregular or uneven data.
* In the case of jagged arrays, the user is only required to specify the number of rows and not the number of columns.
* To access the elements of a jagged array, use the index notation as you would with any other multi-dimensional array.
* In C#, you can use jagged arrays in conjunction with multidimensional arrays to create complex data structures.