Arrays in C#

An array is a collection of similar type of values which are stored in sequential order i.e. they stored in a contiguous memory location. C# supports 2 types of arrays. They are as follows:

1. Single dimensional array
2. Multi-dimensional array

In the Single dimensional array, the data is arranged in the form of a row whereas in the Multi-dimensional arrays in C# the data is arranged in the form of rows and columns. Again the multi-dimensional arrays are of two types:

1. Jagged array: whose rows are fixed but columns varies row-by-row
2. Rectangular array: whose rows and columns are fixed

The values of an array can be accessed using the index positions whereas the array index starts from 0 which means the first item of an array will be stored at 0th position and the position of the last item of an array will be the total number of the item -1.

`Array` class in C#

The Arrays in C# are reference types that are derived from the `System.Array` class. The `Array` class is a predefined abstract class that is defined inside the `System` namespace. This class is working as the base class for all the arrays in C#. The `Array` class provides a set of members (methods and properties) to work with the arrays such as creating, manipulating, searching and sorting the elements of an array.

The `Array` class in C# is not a part of the `System.Collections` namespace, it is a part of the `System` namespace. But still, we considered it a collection because it is based on the `IList` interface.

The `Array` class provides the following methods and properties:

1. `Sort(<array>)`
2. `Reverse (<array>)`
3. `Copy (src, dest, n)`
4. `GetLength(int)`
5. `Length`

Example:

```
using System;
namespace ArrayDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            //Declaring an array
            int[] arr = { 17, 23, 4, 59, 27, 36, 96, 9, 1, 3 };
            //Printing the array elements
            Console.WriteLine("The Array Contains the Below Elements:");
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write(arr[i] + " ");
            }
            Console.WriteLine();
            //Sorting the array elements
            Array.Sort(arr);
            //Printing the array elements after sorting
            Console.WriteLine("The Array Elements After Sorting:");
            foreach (int i in arr)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
            //Reversing the array elements
            Array.Reverse(arr);
            //Printing the array elements in reverse order
            Console.WriteLine("The Array Elements in the Reverse Order :");
            foreach (int i in arr)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
            //Creating a new array
            int[] brr = new int[10];
            //Copying some of the elements from old array to new array
            Console.WriteLine("The new array elements:");
            Array.Copy(arr, brr, 5);
            //We declare the array with size 10. we copy only 5 elements. so the
            //rest
            //5 elements will take the default value. In this array, it’ll take 0
            foreach (int i in brr)
            {
                Console.Write(i + " ");
            }
            Console.ReadKey();
        }
    }
}
```

Multidimensional array in C#

1. Rectangular 2D array

Syntax:

```
<type> [ , ] arrayName = new <type>[rowSize, columnSize]
```

Example:

For declaration and initialization,

```
int[,] arr = new int[4, 5];
//assigning values to the array by using nested for loop
for (int i = 0; i < arr.GetLength(0); i++)
{
    for (int j = 0; j < arr.GetLength(1); j++)
    {
        a += 5;
        arr[i, j] = a;
    }
}
//Assigning the array elements at the time of declaration
int[,] arr1 = {{11,12,13,14},
              {21,22,23,24},
              {31,32,33,34}};
```

2. Jagged 2D array

In the jagged array, the column size will differ from row to row. The jagged array in C# is also called the array of arrays. This is because in the case of the jagged array each row is a single dimensional array. So a combination of multiple single-dimensional arrays with different column sizes form a jagged array in C#.

Syntax:

```
<type> [][] <name> = new <type> [rowSize][];
```

Example:

```
//Creating an jagged array with four rows
int[][] arr = new int[4][];
//Initializing each row with different column size
// Uisng one dimensional array
arr[0] = new int[5];
arr[1] = new int[6];
arr[2] = new int[4];
arr[3] = new int[5];
//assigning values to the jagged array by using nested for loop
for (int i = 0; i < arr.GetLength(0); i++)
{
      for (int j = 0; j < arr[i].Length; j++)
      {
            arr[i][j] = j++;
      }
}
// Assigning the values of the jagged array
// at the time of its declaration
int[][] arr1 = {
            new int[4]{11,12,13,14},
            new int[5]{21,22,23,24,25},
            new int[3]{31,32,33}
            };
```

Advantages of using an array

- It is used to represent similar types of multiple data items using a single name.
- We can use arrays to implement other data structures such as linked lists, trees, graphs, stacks, queues, etc.
- The two-dimensional arrays in C# are used to represent matrices.
- The Arrays in C# are strongly typed. That means they are used to store similar types of multiple data items using a single name. As the arrays are strongly typed so we are getting two advantages. First, the performance of the application will be much better because boxing and unboxing will not happen. Secondly, runtime errors will be prevented because of a type mismatch. In this case, at compile time it will give you the error if there is a type mismatch.

Disadvantages of using arrays

- The array size is fixed. So, we should know in advance how many elements are going to be stored in the array. Once the array is created, then we can never increase the size of an array. If you want then we can do it manually by creating a new array and copying the old array elements into the new array.
- As the array size is fixed, if we allocate more memory than the requirement then the extra memory will be wasted. On the other hand, if we allocate less memory than the requirement, then it will create the problem.
- As the elements of an array are stored in contiguous memory locations. So insertions and deletions of array elements are very difficult and also time-consuming. So the time complexity increase in insertion and deletion operation.
- We can never insert an element into the middle of an array. It is also not possible to delete or remove elements from the middle of an array.

To overcome the above problems Collections are introduced in C#.