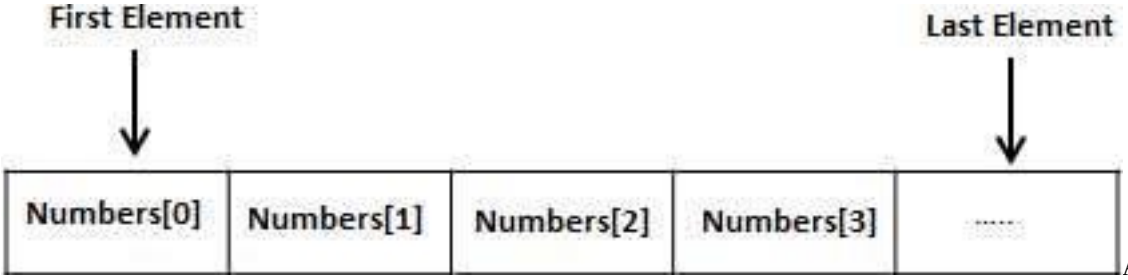
Chapter-3
Arrays

Introduction to Array

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.



All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays:

To declare an array in C#, you can use the following syntax –

```
datatype[] arrayName;
```

where,

- datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.
- arrayName specifies the name of the array.

Eg:

```
double[] balance;
```

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the **new** keyword to create an instance of the array. For example,

```
double[] balance = new double[10];
```

Assigning Values to an Array

You can assign values to individual array elements, by using the index number, like −

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

You can assign values to the array at the time of declaration, as shown −

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

You can also create and initialize an array, as shown −

```
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
```

You may also omit the size of the array, as shown −

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location −

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
int[] score = marks;
```

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example,

```
double salary = balance[9];
```

The following example, demonstrates the above-mentioned concepts declaration, assignment, and accessing arrays −

```
using System;

namespace ArrayApplication {
  class MyArray {
     static void Main(string[] args) {
        int []  n = new int[10]; /* n is an array of 10 integers */
        int i,j;

        /* initialize elements of array n */
        for ( i = 0; i < 10; i++ ) {
           n[ i ] = i + 100;
        }

        /* output each array element's value */
        for (j = 0; j < 10; j++ ) {
           Console.WriteLine("Element[{0}] = {1}", j,n[j]);
        }
        Console.ReadKey();
     }
  }
}
```

C# Multidimensional Arrays

The multidimensional array is also known as rectangular arrays in C#. It can be two dimensional or three dimensional. The data is stored in tabular form (row * column) which is also known as matrix.

To create multidimensional array, we need to use comma inside the square brackets. For example:

1. int[,] arr=new int[3,3];//declaration of 2D array
2. int[,,] arr=new int[3,3,3];//declaration of 3D array

Two-Dimensional Arrays

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.

A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns −

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i , j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in array a.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

```
int [,] a = new int [3,4] {
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

Jagged Array:

Jagged array is a **array of arrays** such that member arrays can be of different sizes. In other words, the length of each array index can differ. The elements of Jagged Array are reference types and initialized to null by default. Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.

Declaration

In Jagged arrays, user has to provide the number of rows only. If the user is also going to provide the number of columns, then this array will be no more Jagged Array.

Syntax:

```
data_type[][] name_of_array = new data_type[rows][]
```

Params Array:

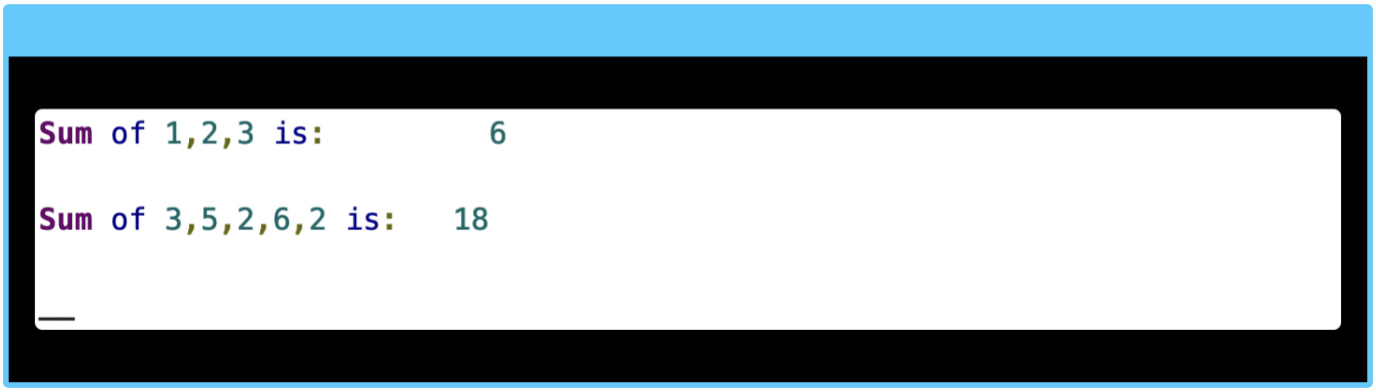
Sometimes, you are not assured about a number of parameters or you want to create a method that can accept n number of parameters at runtime. This situation can be handled with params type array in C#. The params keyword creates an array at runtime that receives and holds n number of parameters.

```
using System;
namespace params_array
{
   class Program
   {
       static int add(params int[] allnumber)
       {
           int sum = 0;
           foreach (int n in allnumber)
           {
               sum = sum + n;
           }
           return sum;
       }
       static void Main(string[] args)
       {
           int sum;
           // passing three parameters
           sum = Program.add(1, 2, 3);
           Console.WriteLine("Sum of 1,2,3 is:\t{0}", sum);

           // passing five parameters
           sum = Program.add(3, 5, 2, 6, 2);
           Console.WriteLine("Sum of 3,5,2,6,2 is:\t{0}", sum);
           Console.ReadLine();
       }
   }
}
```

`static int add(params int[]` allnumber)

Output



In above example, we are creating a function add() that will receive any number of integer parameters at runtime and returns the sum of all those numbers. We will use params array to achieve this goal in C#.

Array class:

C# provides an Array class to deal with array related operations. It provides methods for creating, manipulating, searching, and sorting elements of an array. This class works as the base class for all arrays in the .NET programming environment.

Characteristics of Array Class:

- In Array, the elements are the value of the array and the length of the array is the total number of item present in the array.
- The lower bound of an Array is the index of its first element and the default value of the lower bound is 0.
- The default size of an Array is 2GB.
- Array objects with the same array type share the same Type object.

Properties of the Array Class:

The following table describes some of the most commonly used properties of the Array class –

Sr.No.	Property & description
1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.