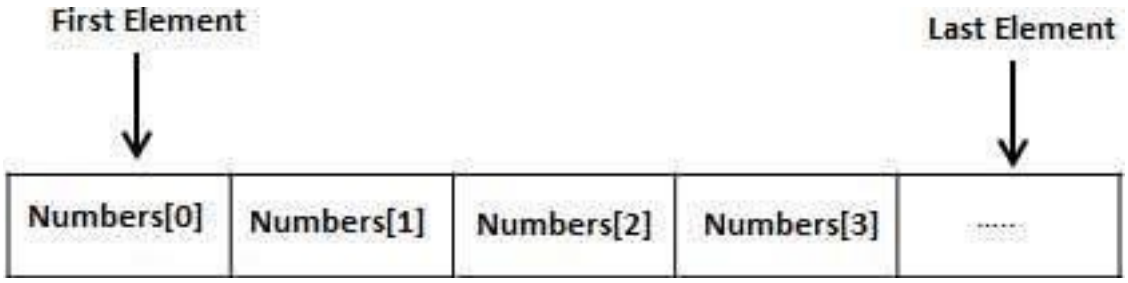
Chapter-3 Arrays

Introduction to Array

An array is a fundamental data structure that allows you to store a collection of elements of the same type in a contiguous block of memory. 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.

- Arrays are commonly used to:
  - ➢ Store and manipulate collections of data efficiently.
  - ➢ Perform operations on a set of values, such as sorting or searching.
  - ➢ Represent matrices and grids in mathematical computations.
  - ➢ Store data retrieved from databases, files, or user input.

- **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

A multidimensional array is a data structure that allows you to store elements in a grid-like format, with multiple dimensions. The most common multidimensional array is a two-dimensional array, which can be thought of as a matrix with rows and columns.
The multidimensional array is also known as rectangular arrays in C#. It can be two dimensional or three dimensional.

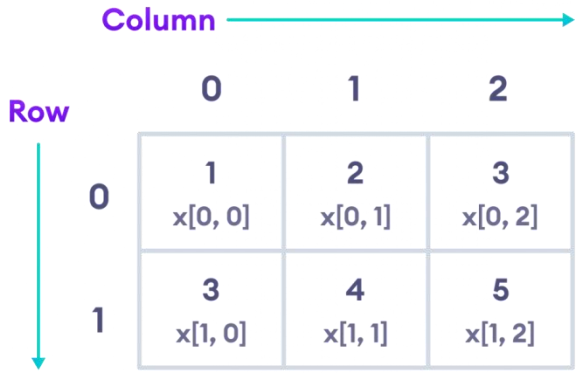
To create multidimensional array, we need to use comma inside the square brackets. For example:

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



➢ **Two-Dimensional Array Declaration**

int[,] x = new int [2, 3];

➢ **Two-Dimensional Array initialization**

int[,] x = { { 1, 2 ,3}, { 3, 4, 5 } };

➢ **Declaration and initialization at same time**

int [,] x = new int[2, 3]{ {1, 2, 3}, {3, 4, 5} };

➢ **Access Elements from 2D Array**

// access first element from first row
x[0, 0]; // returns 1

// access third element from second row
x[1, 2]; // returns 5

// access third element from first row
x[0, 2]; // returns 3

Thus, every element in the array x 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.

Example:

```
using System;

namespace ArrayApplication
{
   class MyArray
   {
      static void Main(string[] args)
      {
         int[,] twoDimensionalArray = new int[3, 4];

               // Create a 2D array with 3 rows and 4 columns

         int value = 1;

         // Initialize elements of the 2D array
         for (int row = 0; row < 3; row++)
         {
            for (int col = 0; col < 4; col++)
            {
               twoDimensionalArray[row, col] = value;
               value++;
            }
         }

         // Output each array element's value
         for (int row = 0; row < 3; row++)
         {
            for (int col = 0; col < 4; col++)
            {
               Console.WriteLine("Element[{0},{1}] = {2}", row, col,
twoDimensionalArray[row, col]);
            }
         }
      }
   }
}
```

- **C# Jagged Arrays:** A jagged array is an **array of array** such that member arrays can be of different sizes. Jagged arrays store arrays instead of literal values.

A jagged array is initialized with two square brackets [][]. The first bracket specifies the size of an array, and the second bracket specifies the dimensions of the array which is going to be stored.
Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.

Ex.

```
int[][] jArray1 = new int[2][]; // can include two single-dimensional arrays
int[][,] jArray2 = new int[3][,]; // can include three two-dimensional arrays
```

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.

```
int[][] participants = new int[3][];
```

Ex:

```
using System;

namespace JaggedArrayExample
{
   class Program
   {
      static void Main(string[] args)
      {
         // 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 };

         // Access and display elements from the jagged array
         Console.WriteLine("Elements of the jagged array:");
```

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

         }
      }
}
```

- **Params Array(Parameter array):**

Sometimes, you are unaware 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.You can combine a params argument with other optional parameters, but it must be the last argument in the parameter list, and there can be only one in the function definition.

Syntax:

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

Example:

```
using System;

namespace params_array
{
   class Program
   {
      static int Add(params int[] allNumbers)
      {
         int sum = 0;
         foreach (int n in allNumbers)
```

```
            {
                sum += n;
            }
            return sum;
        }

        static void Main(string[] args)
        {
            int sum;

            // Passing three parameters
            sum = Add(1, 2, 3);
            Console.WriteLine("Sum of 1, 2, 3 is:\t{0}", sum);

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