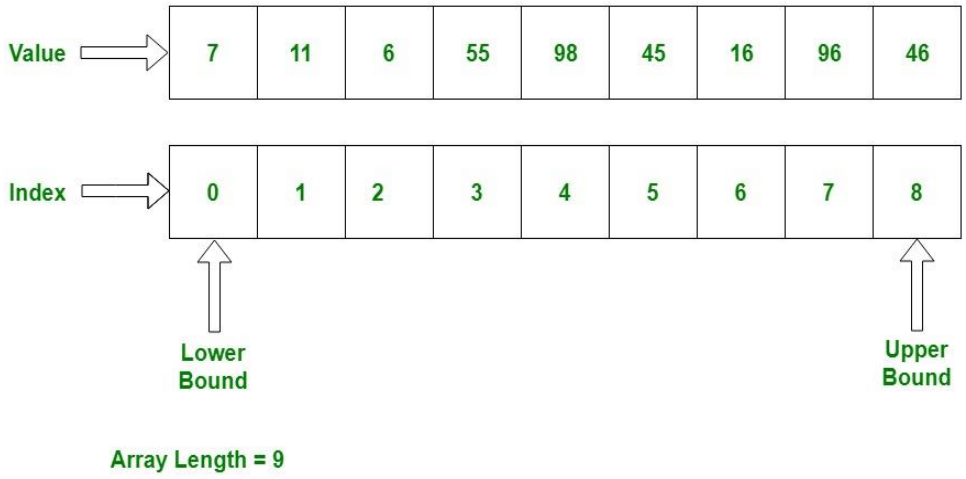
1.6 Handling Arrays

Array

- An array is the collection of elements of same datatype stored in contiguous (continuous) memory locations.
- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- A particular value is indicated by writing a number called ***index value*** or ***subscrip***t.



Depending upon dimension arrays there are following types:

- One Dimensional Array
- Two dimensional Array

One Dimensional Array

A list of items can be given one variable name using only one subscript and such a variable is called a ***single subscripted*** variable or a ***one-dimensional array***.

Like other variables, arrays must be declared and created in the computer memory before they are used. Creation of an array involves three steps:

1. Declaring the array
2. Creating memory locations
3. Putting values into the memory locations.

Declaring the array

Arrays in C# are declared as follows:

type[ ] arrayname;

Examples:

```
int[ ] counter;          //declare int array reference
```

```
float[ ]  marks;          //declare float array reference
int[ ]    x,y;            //declare two int array reference
```

Remember, we do not enter the size of the arrays in the declaration.

Creation of Array

After declaring an array, we need to create it in the memory. C# allows us to create arrays using **new** operator only, as shown below:

```
arrayname = new type[size];
```

Examples:

```
number  = new int[5];          //create a 5 element int array
average = new float[10];       // create a 10 element float array
```

These lines create the necessary memory locations for the arrays **number** and **average** and designate them as **int** and **float** respectively. Now, the variable **number** refers to an array of five integers and **average** refers to an array of ten floating-point values.

It is also possible to combine the two steps, declaration and creation, into one as shown below:

```
Int[ ] number = new int[5]; //declare and create 5 element int array
```

Initialisation of Array

The final step is to put values into the array created. This process is known as *initialization*. This is done using the array subscripts as shown below.

```
arrayname[subscript] = value ;
```

Example:

```
number[0] = 35;
number[1] = 40;
.....
.....
number[4] = 19;
```

Note that C# creates arrays starting with a subscript of 0 and ends with a value one less than the *size* specified.

C# protects arrays from overruns and underruns. Trying to access an array beyond its boundaries will generate an error message.

```
int [ ] number = new int [5] { 35, 40, 20, 57, 19 };
```

This combines all the three steps, namely declaration, creation and initialization.

We can also initialize arrays automatically in the same way as the ordinary variables when they are declared, as shown below:

```
type [ ] arrayname = {list of values};
```

Array Length

It is used to get the number of elements, or the length, of an array. It can be used to get the length of an array of any dimension.

Example

```
using System;
public class ArrayClearance
{
public static void Main()
{
      int[] arr1 = {10, 20, 30, 40};
      char[] arr2 = {'E', 'd', 'p', 'r', 'e', 's', 's', 'o'};
      bool[] arr3 = {true, false};
      double[] arr4 = {};

      Console.WriteLine(arr1.Length);
      Console.WriteLine(arr2.Length);
      Console.WriteLine(arr3.Length);
      Console.WriteLine(arr4.Length);
   }
}
```

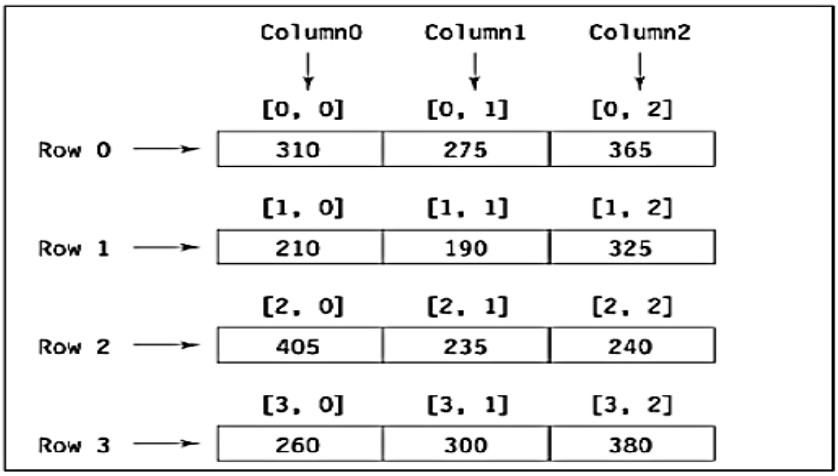
Output

```
4
8
2
0
```

Two Dimensional Array

In case of two dimensional array data is represented in form of rows and columns.

Two- dimensional arrays are stored in memory as shown below. First index selects the row and the second index selects the column within that row.



For creating two-dimensional arrays, we must follow the same steps as that of simple arrays. We may create a two-dimensional array like this

```
int [,] myArray;

myArray= new int[3,4];

or

int[,] myArray= new int[3,4];
```

This creates a table that can store twelve integer values, four across and three down.

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. For example,

```
int[,] table ={{0,0,0},{1,1,1}};
```

initializes the elements of the first row to zero and the second row to one. The initialization is done row by row. We can also initialize a two-dimensional array in the form of a matrix as shown below:

```
int[,] table = {
                    {0,0,0},
                    {1,1,1}
               };
```

Commas are required after each brace that closes off a row expect in then each if the last row

We can refer to a value stored in a two-dimensional array by using subscripts for both the column and row of the corresponding element. For example:

```
int value = table[1,2];
```

This retrieves the value stored in the second row and third column of the **table** matrix.

Example

```
using System;
namespace MultiDArray
{
  class Program
{
    public static void Main(string[] args)
{
      int[ , ] numbers = {{2, 3}, {4, 5}};
Console.WriteLine("Element at index [0, 0] : "+numbers[0, 0]);
Console.WriteLine("Element at index [1, 0] : "+numbers[1, 0]);
    }
  }
}
```

Output

```
Element at index [0, 0] : 2

Element at index [1, 0] : 4
```

Jagged Array

Jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.

Declaration of Jagged array

int[][] arr = **new int**[2][];

Initialization of Jagged array

arr[0] = **new int**[4];
arr[1] = **new int**[6];

Initialization and filling elements in Jagged array.

arr[0] = **new int**[4] { 11, 21, 56, 78 };
arr[1] = **new int**[6] { 42, 61, 37, 41, 59, 63 };

Here, size of elements in jagged array is optional. So, you can write above code as given below:

arr[0] = **new int**[] { 11, 21, 56, 78 };

arr[1] = **new int**[] { 42, 61, 37, 41, 59, 63 };

Example

```
public class JaggedArrayTest
{
  public static void Main()
   {
      int[][] arr = new int[2][];// Declare the array

      arr[0] = new int[] { 11, 21, 56, 78
      arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };

      for (int i = 0; i < arr.Length; i++)
      {
         for (int j = 0; j < arr[i].Length; j++)
         {
            System.Console.Write(arr[i][j]+" ");
         }
         System.Console.WriteLine();
      }
   }
}
```

Output:

```
11 21 56 78
42 61 37 41 59 63
```

System.Array Class

In C#, every array we create is automatically derived from the **System.Array** class. This class defines a number of methods and properties that can be used to manipulate arrays more efficiently.

Table 9.1 *Some commonly used methods of System.Array class*

METHOD/PROPERTY	PURPOSE
Clear ( )	Sets a range of elements to empty values
CopyTo ( )	Copies elements from the source array into the destination array
GetLength ( )	Gives the number of elements in a given dimension of the array
GetValue ( )	Gets the value for a given index in the array
Length	Gives the length of an array
SetValue ( )	Sets the value for a given index in the array
Reverse ( )	Reverses the contents of a one-dimensional array
Sort ( )	Sorts the elements in a one-dimensional array

Clear()

- The **Array.clear()** method is used to clear the contents of an array.
- **array**: This is the array we want to clear.
- **index**: This is the index position of the element from which we need to start the clearance.
- **length**: This is the number of elements that we need to delete from the array array.

```
Array.Clear(arr, 0, arr.Length);
```

```
using System;
public class ArrayClearance
{
   public static void Main()
   {
      double[] myArr = {10, 20, 30, 40};
      Console.WriteLine("Array Before Clearing:");
      for(int i = 0; i < myArr.Length ; i++)
{
         Console.WriteLine(myArr[i]);
      }
```

```
        Array.Clear(myArr, 1, 3);

        Console.WriteLine("Array After Clearing:");
         for(int i = 0; i < myArr.Length ; i++)
{
            Console.WriteLine(myArr[i]);
        }
    }
}
```

Output

```
Array Before Clearing:
10
20
30
40
Array After Clearing:
10
0
0
0
```

CopyTo()

- The CopyTo() method in C# is used to copy elements of one array to another array.
- In this method, you can set the starting index from where you want to copy from the source array.

```
CopyTo(dest, index);
```

Here **dest** = destination array

index= starting index

Example

```
using System;
class Program

 public  static void Main()
 {
     int[] arrSource = {5,9,1,3};
```

```
    int[] arrTarget = new int[4];
    arrSource.CopyTo(arrTarget,0 );
    Console.WriteLine("Destination Array ...");
    foreach (int value in arrTarget)
{
       Console.WriteLine(value);
    }
  }
}
```

Output

```
Destination Array ...
5
9
1
3
```

GetLength()

- **GetLength(Int32) Method** is used to find the total number of elements present in the specified dimension of the Array.
- *dimension* is a zero-based dimension of the Array whose length needs to be determined.

```
using System;
public class Program
{
 public  static void Main()
   {
int[] myarray = {445, 44, 66, 6666667, 78, 878, 1};
 Console.WriteLine("The elements of myarray :");

       foreach(int i in myarray)
       {
          Console.WriteLine(i);
       }
       int result = myarray.GetLength(0);
       Console.WriteLine("Total Elements: {0}", result);
   }
}
```

Output:

```
The elements of myarray :
```

```
445
44
66
6666667
78
878
1
Total Elements: 7
```

GetValue()

- Array.GetValue() Method in C# is used to gets the value of the specified position element in the current Array.
- Method is used to get the value at the specified position in the one-dimensional Array

```
using System;
public class Program
{
  public static void Main()
   {
      char[] arr = new char[]{'A', 'B', 'C', 'D'};

      Console.WriteLine("element at index 3 is : " + arr.GetValue(3));
      Console.WriteLine("element at index 1 is : " + arr.GetValue(1));
      Console.WriteLine("element at index 2 is : " + arr.GetValue(2));
      Console.WriteLine("element at index 0 is : " + arr.GetValue(0));
   }
}
```

Output:

```
element at index 3 is : D
element at index 1 is : B
element at index 2 is : C
element at index 0 is : A
```

SetValue()

Sets a value to the element at the specified position in the one-dimensional Array.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Demo {
  class Program {
    static void Main(string[] args) {
      Array arr = Array.CreateInstance(typeof(String), 6);
      arr.SetValue("One", 0);
      arr.SetValue("Two", 1);
      arr.SetValue("Three", 3);
      arr.SetValue("Four", 4);
      arr.SetValue("Five", 5);
      Console.WriteLine("Length {0}",arr.GetLength(0).ToString());
      Console.ReadLine();
    }
  }
}
```

Output

Length 6

Reverse()

The Reverse() method in array class reverses the sequence of the elements in the entire one-dimensional Array.

```
using System;
namespace Demo
{
  class MyArray
{
    static void Main(string[] args)
{
      int[] list = { 29, 15, 30, 98};
      int[] temp = list;
      Console.Write("Original Array: ");
      foreach (int i in list)
 {
        Console.Write(i + " ");
      }
      Console.WriteLine();
            Array.Reverse(temp);
```

```
        Console.Write("Reversed Array: ");
        foreach (int i in temp)
{
          Console.Write(i + " ");
        }
        Console.ReadKey();
      }
   }
}
```

Output

```
Original Array: 29 15 30 98
Reversed Array: 98 30 15 29
```

Sort()

```
using System;
class Demo {
   static void Main() {
      int[] arr = { 99, 43, 86 };
      // sort
      Array.Sort(arr);
      Console.WriteLine("Sorted List");
      foreach (int res in arr) {
         Console.Write(" "+res);
      }
      Console.WriteLine();
   }
}
```

Output

```
Sorted List
43
86
99
```

ArrayList

- ArrayList is a non-generic collection of objects whose size increases dynamically. It is the same as Array except that its size increases dynamically.
- An ArrayList can be used to add unknown data where you don't know the types and the size of the data.

```
ArrayList   arr = new ArrayList(15);
```

Add() method is used to add elements in the ArrayList

```
arr.Add(“one”);

arr.Add(“two”);

arr.Add(“three”);
```

RemoveAt() method is used to remove the element

```
arr.RemoveAt(1);
```

```
using System;
using System.Collections;
class Program
{

   public static void Main()
   {

      ArrayList myList = new ArrayList(5);
        myList.Add("Hello");
      myList.Add("World");

      Console.WriteLine("Count : " + myList.Count);
      Console.WriteLine("Capacity : " + myList.Capacity);
   }
}
```

Output:

```
Count : 2

Capacity : 5
```

Difference between array and arraylist

Feature	**Array**	**ArrayList**
Memory	This has fixed size and can't increase or decrease dynamically.	Size can be increase or decrease dynamically.
Namespace	Arrays belong to **System.Array** namespace	ArrayList belongs to **System.Collection** namespace.
Data Type	In Arrays, we can store only one datatype either int, string, char etc.	In ArrayList we can store different datatype variables.
Operation Speed	Insertion and deletion operation is fast.	Insertion and deletion operation in ArrayList is slower than an Array.
Typed	Arrays are strongly typed which means it can store only specific type of items or elements.	ArrayList are not strongly typed.
null	Array cannot accept null.	ArrayList can accepts null.