**ARRAY**

**What is an 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.

For example,

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;
```

- 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();

    }

  }
```

}

- When the above code is compiled and executed, it produces the following result –

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

- Using the *foreach* Loop

In the previous example, we used a for loop for accessing each array element. You can also use a **foreach** statement to iterate through an array.

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

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

      /* output each array element's value */
      foreach (int j in n ) {
        int i = j-100;
        Console.WriteLine("Element[{0}] = {1}", i, j);
      }
```

```
        Console.ReadKey();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

- C# Arrays
- There are following few important concepts related to array which should be clear to a C# programmer –

***Multi-dimensional arrays***

C# supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.

C# allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array. You can declare a 2-dimensional array of strings as –

```
string [,] names;
```

or, a 3-dimensional array of int variables as –

```
int [ , , ] m;
```

- 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 –

- 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 */
};
```

- Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts. That is, row index and column index of the array.

For example,

```
int val = a[2,3];
```

The above statement takes 4th element from the 3rd row of the array. You can verify it in the above diagram. Let us check the program to handle a two dimensional array –

```
using System;
namespace ArrayApplication {
  class MyArray {
    static void Main(string[] args) {
      /* an array with 5 rows and 2 columns*/
      int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
      int i, j;

      /* output each array element's value */
      for (i = 0; i < 5; i++) {

        for (j = 0; j < 2; j++) {
          Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
        }
      }
```

```
        Console.ReadKey();
      }
   }
}
```

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

***Jagged arrays***

C# supports multidimensional arrays, which are arrays of arrays.

A Jagged array is an array of arrays. You can declare a jagged array named *scores* of type **int** as –

```
int [][] scores;
```

Declaring an array, does not create the array in memory. To create the above array –

```
int[][] scores = new int[5][];
for (int i = 0; i < scores.Length; i++) {
   scores[i] = new int[4];
}
```

You can initialize a jagged array as –

```
int[][] scores = new int[2][]{new int[]{92,93,94},new int[]{85,66,87,88}};
```

Where, scores is an array of two arrays of integers - scores[0] is an array of 3 integers and scores[1] is an array of 4 integers.

- The following example illustrates using a jagged array

```
using System;
namespace ArrayApplication {
   class MyArray {
```

```
    static void Main(string[] args) {

      /* a jagged array of 5 array of integers*/
      int[][] a = new int[][]{new int[]{0,0},new int[]{1,2},
        new int[]{2,4},new int[]{ 3, 6 }, new int[]{ 4, 8 } };
      int i, j;

      /* output each array element's value */
      for (i = 0; i < 5; i++) {
        for (j = 0; j < 2; j++) {
          Console.WriteLine("a[{0}][{1}] = {2}", i, j, a[i][j]);
        }
      }
      Console.ReadKey();
    }
  }
}
```

***Passing arrays to functions***

You can pass to the function a pointer to an array by specifying the array's name without an index.

You can pass an array as a function argument in C#. The following example demonstrates this –

```
using System;

namespace ArrayApplication {
  class MyArray {
    double getAverage(int[] arr, int size) {
      int i;
      double avg;
      int sum = 0;

      for (i = 0; i < size; ++i) {
        sum += arr[i];
```

```
      }
      avg = (double)sum / size;
      return avg;
    }
    static void Main(string[] args) {
      MyArray app = new MyArray();

      /* an int array with 5 elements */
      int [] balance = new int[]{1000, 2, 3, 17, 50};
      double avg;
      /* pass pointer to the array as an argument */
      avg = app.getAverage(balance, 5 ) ;
      /* output the returned value */
      Console.WriteLine( "Average value is: {0} ", avg );
      Console.ReadKey();
    }
  }
}
```

***Param arrays***

This is used for passing unknown number of parameters to a function.

At times, while declaring a method, you are not sure of the number of arguments passed as a parameter. C# param arrays (or parameter arrays) come into help at such times.

The following example demonstrates this –

```
using System;
namespace ArrayApplication {
  class ParamArray {
    public int AddElements(params int[] arr) {
      int sum = 0;

      foreach (int i in arr) {
        sum += i;
```

```
        }
        return sum;
      }
    }
    class TestClass {
      static void Main(string[] args) {
        ParamArray app = new ParamArray();
        int sum = app.AddElements(512, 720, 250, 567, 889);

        Console.WriteLine("The sum is: {0}", sum);
        Console.ReadKey();
      }
    }
}
```

***The Array Class***

Defined in System namespace, it is the base class to all arrays, and provides various properties and methods for working with arrays.

The Array class is the base class for all the arrays in C#. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.

- Properties of the Array Class

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

- METHODS OF THE ARRAY CLASS
- METHODS OF ARRAY CLASS PART 2

```
using System;

namespace ArrayApplication {
  class MyArray {
    static void Main(string[] args) {
      int[] list = { 34, 72, 13, 44, 25, 30, 10 };
      int[] temp = list;
      Console.Write("Original Array: ");
```

```
        foreach (int i in list) {
          Console.Write(i + " ");
        }
        Console.WriteLine();

        // reverse the array
        Array.Reverse(temp);
        Console.Write("Reversed Array: ");

        foreach (int i in temp) {
          Console.Write(i + " ");
        }
        Console.WriteLine();

        //sort the array
        Array.Sort(list);
        Console.Write("Sorted Array: ");

        foreach (int i in list) {
          Console.Write(i + " ");
        }
        Console.WriteLine();
        Console.ReadKey();
      }
    }
}
```