**Arrays:**

* An *array* is a collection of variables of the same type that are referred to by a common name.
* In C#, arrays can have one or more dimensions, although the one-dimensional array is the most common.
* Arrays are used for a variety of purposes because they offer a convenient

means of grouping together related variables

* The principal advantage of an array is that it organizes data in such a way that it can be easily manipulated.
* Also, arrays organize data in such a way that it can be easily sorted.
* Although arrays in C# can be used just like arrays in many other programming languages, they have one special attribute: They are implemented as objects.

**One-Dimensional Arrays:**

* A *one-dimensional array* is a list of related variables. Such lists are common in programming.
* arrays in C# are implemented as objects, two steps are needed to obtain an

array for use in your program.

* First, you must declare a variable that can refer to an array.
* Second, you must create an instance of the array by use of **new**.
* Therefore, to declare a onedimensional array, you will typically use this general form:

**type[ ] array-name = new type[size];**

* Here is an example. The following creates an **int** array of ten elements and links it to an array reference variable named **sample**.

int[] sample = new int[10];

* The **sample** variable holds a reference to the memory allocated by **new**.
* This memory is large enough to hold ten elements of type **int**.
* As is the case when creating an instance of a class, it is possible to break the preceding declaration in two.

For example:

int[] sample;

sample = new int[10];

In this case, when **sample** is first created, it refers to no physical object.

It is only after the second statement executes that **sample** refers to an array.

* An individual element within an array is accessed by use of an index.
* An *index* describes the position of an element within an array.
* In C#, all arrays have 0 as the index of their first element.

using System;

class ArrayDemo

{

public static void Main()

{

int[] sample = new int[10];

int i;

for(i = 0; i < 10; i = i+1)

sample[i] = i;

for(i = 0; i < 10; i = i+1)

Console.WriteLine("sample[" + i + "]: " + sample[i]);

}

}

**Initializing an Array:**

* The general form for initializing a onedimensional array is shown here:

**type[ ] array-name = { val1, val2, val3, ..., valN }**

**Example:**

// Compute the average of a set of values.

**using System;**

**class** Average

{

**public static void Main()**

{

int[] nums = { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };

int avg = 0;

for(int i=0; i < 10; i++)

avg = avg + nums[i];

avg = avg / 10;

Console.WriteLine("Average: " + avg);

}

}

* As a point of interest, although not needed, you can use **new** when initializing an array.

**For example,** this is a proper, but redundant, way to initialize **nums** in the foregoing program:

int[] nums = new int[] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };

* Although redundant here, the **new** form of array initialization is useful when you are assigning a new array to an already-existent array reference variable.

**For example:**

int[] nums;

nums = new int[] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };

* In this case, **nums** is declared in the first statement and initialized by the second.

One last point: It is permissible to specify the array size explicitly when initializing an array, but the size must agree with the number of initializers

**For example**, here is another way to initialize **nums**:

int[] nums = new int[10] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };

In this declaration, the size of **nums** is explicitly stated as 10.

**Multidimensional Arrays:**

* Although the one-dimensional array is the most commonly used array in programming, multidimensional arrays are certainly not rare.
* A *multidimensional array* is an array that has two or more dimensions, and an individual element is accessed through the combination of two or more indices.

**Two-Dimensional Arrays:**

* The simplest form of the multidimensional array is the two-dimensional array.
* In a twodimensional array, the location of any specific element is specified by two indices.
* To declare a two-dimensional integer array **table** of size 10, 20, you would write

int[,] table = new int[10, 20];

* In the first part of the declaration, the syntax

[,]

indicates that a two-dimensional array reference variable is being created.

* When memory is actually allocated for the array using **new**, this syntax is used:

int[10, 20]

This creates a 10×20 array, and again, the comma separates the dimensions.

**Example:**

// Demonstrate a two-dimensional array.

**using** System;

**class** TwoD

{

**public static void Main()**

{

int t, i;

int[,] table = new int[3, 4];

for(t=0; t < 3; ++t)

{

for(i=0; i < 4; ++i)

{

table[t,i] = (t\*4)+i+1;

Console.Write(table[t,i] + " ");

}

Console.WriteLine();

}

}

}

**Arrays of Three or More Dimensions:**

* C# allows arrays with more than two dimensions.
* Here is the general form of a multidimensional array declaration:

**type[, ...,] name = new type[size1, size2, ..., sizeN];**

* For example, the following declaration creates a 4×10×3 three-dimensional integer array:

int[,,] multidim = new int[4, 10, 3];

* To assign element 2, 4, 1 of **multidim** the value 100, use this statement:

multidim[2, 4, 1] = 100;

* Here is a program that uses a three-dimensional array that holds a 3×3×3 matrix of values.

It then sums the value on one of the diagonals through the cube.

// Sum the values on a diagonal of a 3x3x3 matrix.

**using** System;

**class** ThreeDMatrix

{

public static void Main()

{

int[,,] m = new int[3, 3, 3];

int sum = 0;

int n = 1;

for(int x=0; x < 3; x++)

for(int y=0; y < 3; y++)

for(int z=0; z < 3; z++)

m[x, y, z] = n++;

sum = m[0, 0, 0] + m[1, 1, 1] + m[2, 2, 2];

Console.WriteLine("Sum of first diagonal: " + sum);

}

}

**The output is shown here:**

Sum of first diagonal: 42

**Jagged Arrays:**

* C# allows you to create a special type of two-dimensional array called

a *jagged array.*

* A jagged array is an *array of arrays* in which the length of each array can differ.
* Thus, a jagged array can be used to create a table in which the lengths of the rows are not the same.
* Jagged arrays are declared by using sets of square brackets to indicate each dimension.
* For example, to declare a two-dimensional jagged array, you will use this **general form:**

**type[ ] [ ] array-name = new type[size][ ];**

Here, *size* indicates the number of rows in the array

* For example, the following code allocates memory for the first dimension of **jagged** when it is declared. It then allocates the second dimensions manually.

int[][] jagged = new int[3][];

jagged[0] = new int[4];

jagged[1] = new int[3];

jagged[2] = new int[5];

* The following program demonstrates the creation of a jagged two-dimensional array:

// Demonstrate jagged arrays.

**using** System;

class Jagged

{

public static void Main()

{

int[][] jagged = new int[3][];

jagged[0] = new int[4];

jagged[1] = new int[3];

jagged[2] = new int[5];

int i;

// Store values in first array.

for(i=0; i < 4; i++)

jagged[0][i] = i;

// Store values in second array.

for(i=0; i < 3; i++)

jagged[1][i] = i;

// Store values in third array.

for(i=0; i < 5; i++)

jagged[2][i] = i;

// Display values in first array.

for(i=0; i < 4; i++)

Console.Write(jagged[0][i] + " ");

Console.WriteLine();

// Display values in second array.

for(i=0; i < 3; i++)

Console.Write(jagged[1][i] + " ");

Console.WriteLine();

// Display values in third array.

for(i=0; i < 5; i++)

Console.Write(jagged[2][i] + " ");

Console.WriteLine();

}

}

**The output is shown here:**

0 1 2 3

0 1 2

0 1 2 3 4

* Jagged arrays are not used by all applications, but they can be effective in some situations.
* For example, if you need a very large two-dimensional array that is sparsely

populated (that is, one in which not all of the elements will be used), then a jagged array might be a perfect solution jagged arrays are arrays of arrays, there is no restriction that requires that the arrays be one-dimensional.

* For example, the following creates an array of two-dimensional arrays:

int[][,] jagged = new int[3][,];

The next statement assigns **jagged[0]** a reference to a 4×2 array:

jagged[0] = new int[4, 2];

The following statement assigns a value to **jagged[0][1,0]**:

jagged[0][1,0] = i;