**Arrays**

C# arrays are zero indexed; that is, the array indexes start at zero

When declaring an array, the square brackets ([]) must come after the type, not the identifier. Placing the brackets after the identifier is not legal syntax in C#.

int[] table; // not int table[];

C# supports

* single-dimensional arrays,
* multidimensional arrays (rectangular arrays), and
* array-of-arrays (jagged arrays)

**single-dimensional arrays**

int[] numbers;

**multidimensional arrays**

string[,] names;

**array-of-arrays (jagged arrays)**

byte[][] scores;

In C#, arrays are objects and must be instantiated.

int[] numbers = new int[5];

string[,] names = new string[5,4];

byte[][] scores = new byte[5][];

for (int x = 0; x < scores.Length; x++)

{

scores[x] = new byte[4];

}

Initializing Arrays

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

string[] names = new string[3] {"Matt", "Joanne", "Robert"};

You can omit the size of the array, like this:

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

string[] names = new string[] {"Matt", "Joanne", "Robert"};

You can also omit the **new**operator if an initializer is provided, like this:

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

string[] names = {"Matt", "Joanne", "Robert"};

### Accessing Array Members

int[] numbers = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

numbers[4] = 5;

The following code declares a multidimensional array and assigns 5 to the member located at [1, 1]:

int[,] numbers = { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10} };

numbers[1, 1] = 5;

The following is a declaration of a single-dimension jagged array that contains two elements. The first element is an array of two integers, and the second is an array of three integers:

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

};

The following statements assign 58 to the first element of the first array and 667 to the second element of the second array:

numbers[0][0] = 58;

numbers[1][1] = 667;

Assignment:

Create single dimensional, multidimensional and jagged arrays

**Tuples**

Tuple is a static class, It allows to store identifiers of logical group

Eg. PersonDetails – FirstName, MiddleName, LastName instead of creating a full fledged class

var PersonDetails = Tuple.Create("Mark", “J”, “Smith”);

// Display the first and last elements.

PersonDetails.Item1

PersonDetails.Item2

PersonDetails.Item3

You can create upto 7 items

For more that 8 you can create nested Tuple

var primes = new Tuple<int, int, int, int, int, int, int,

Tuple<int>>(2, 3, 5, 7, 11, 13, 16,

new Tuple<int>(19));

primes.Item1,

primes.Item2,

.

.

.

primes.Rest.Item1

primes.Rest.Item2..

Assignment:

1. Create Tuple with 3 items
2. Create Tuples with more than 7 items

**Enum**

The enum keyword is used to declare an enumeration, a distinct type that consists of a set of named constants called the enumerator list.

By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1. For example, in the following enumeration, Sat is 0, Sun is 1, Mon is 2, and so forth.+

enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};

 use initializers to override the default values, as shown in the following exam

enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

int x = (int)Days.Sun;

enum Days : short {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

Assignment: Create enum and use in your program

**Struct**

The struct type is suitable for representing lightweight objects such as Point, Rectangle, and Color

public struct CoOrds

{

public int x, y;

public CoOrds(int p1, int p2)

{

x = p1;

y = p2;

}

}

It is an error to define a default (parameterless) constructor for a struct. It is also an error to initialize an instance field in a struct body. You can initialize struct members only by using a parameterized constructor or by accessing the members individually after the struct is declared. Any private or otherwise inaccessible members can be initialized only in a constructor.

**Classes:**

* Can support inheritance
* Are reference (pointer) types
* The reference can be null

**Structs:**

* Cannot support inheritance
* Are value types
* Cannot contain parameterless constructors

Assignment:

* 1. Create Strut and assign one struct to another and see the result. Same way create class and assign the one class to another and see the values
  2. Check if you are able to inherit struct from another struct/class
  3. Check if you can create parameterless constructor in struct