Mukunda Paudel
paudelmuku@gmail.com

C# Statements

- The actions that a program takes are expressed in statements. Common actions include declaring variables, assigning values, calling methods, looping through collections, and branching to one or another block of code, depending on a given condition.
- The order in which statements are executed in a program is called the flow of control or flow of execution.
- The flow of control may vary every time that a program is run, depending on how the program reacts to input that it receives at run time.
- A statement can consist of a single line of code that ends in a semicolon, or a series of single-line statements in a block.
- A statement block is enclosed in {} brackets and can contain nested blocks. The following code shows two examples of single-line statements, and a multi-line statement block

Types of Statements

- The following table lists the various types of statements in C# and their associated keywords.

Category	**C# keywords / notes**
Declaration statements	It introduces a new variable or constant. A variable declaration can optionally assign a value to the variable. In a constant declaration, the assignment is required.
Expression statements	Expression statements that calculate a value must store the value in a variable.
Selection statements	Selection statements enable you to branch to different sections of code, depending on one or more specified conditions. **if, else, switch case**
Iteration statements	Iteration statements enable you to loop through collections like arrays, or perform the same set of statements repeatedly until a specified condition is met. **do, for, foreach, in, while**

Category	C# keywords / notes
Jump statements	Jump statements transfer control to another section of code. **break, continue, default, goto, return, yield**
Exception handling statements	Exception handling statements enable you to gracefully recover from exceptional conditions that occur at run time. **throw, try-catch, try-finally, try-catch-finally**
Checked and unchecked	Checked and unchecked statements enable you to specify whether numerical operations are allowed to cause an overflow when the result is stored in a variable that is too small to hold the resulting value.
The empty statement	The empty statement consists of a single semicolon. It does nothing and can be used in places where a statement is required but no action needs to be performed.

Object and Classes:

- ***Class and Object*** are the basic concepts of Object Oriented Programming which revolve around the real-life entities.
- A **class** is a user-defined blueprint or prototype from which objects are created.
- Basically, a class combines the fields and methods (member function which defines actions) into a single unit.
- In C#, classes support the polymorphism, inheritance and also provide the concept of derived classes and base classes.

Declaration of Class:

- Generally, a class declaration contains only keyword **class**, followed by an **identifier (name)** of the class. Constructors in class are used for initializing new objects.
- Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.
- The default access modifier of a **class**is Internal.
- The default access modifier of methods and variables is Private.

Mukunda Paudel
paudelmuku@gmail.com

Objects:

- It is a basic unit of Object Oriented Programming and represents the real-life entities.

Declaring Objects (Also called instantiating a class)

- When an object of a class is created, the class is said to be instantiated.
- All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object.
- A single class may have any number of instances.

```
class Animal
{
  string Name = "TIGER";

  static void Main(string[] args)
  {
    Animal _animal = new Animal();
    Console.WriteLine (_animal.Name);
  }
}
```

EverestEngineeringCollege.cs

```
// C# program to illustrate the Initialization of an object
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClssObj
{
    // Class Declaration
    class EverestEngineeringCollege
    {
        String name;
        String Address;
        int Total_Student;
        String Principle_Name;
```

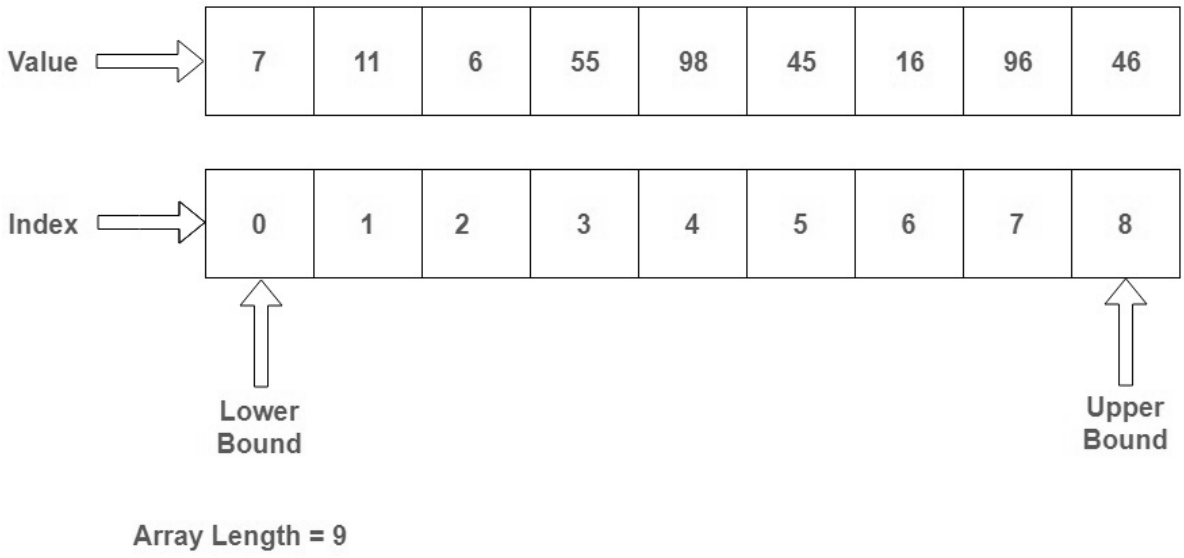
```
        // Constructor Declaration of Class
                public EverestEngineeringCollege(String name, String Address,
                                int Total_Student, String Principle_Name)
                {
                    this.name = name;
                    this.Address = Address;
                    this.Total_Student = Total_Student;
                    this.Principle_Name = Principle_Name;
                }

                // method 1
                public String getName()
                {
                    return name;
                }

                // method 2
                public String getAddress()
                {
                    return Address;
                }

                // method 3
                public int getTotalStudent()
                {
                    return Total_Student;
                }

                // method 4
                public String getPrincipleName()
                {
                    return Principle_Name;
                }

                public String toString()
                {
                    return ("Hi My College Name is " + this.getName()
                            + ".\nCollege Address Total NUmber of Students,and Name of Our
        Principle  are " + this.getAddress()
                            + ", " + this.getTotalStudent() + ", " + this.getPrincipleName());
                }
            }
        }
```

Program.cs

```
static void Main(string[] args)
        {
            // Creating object
            EverestEngineeringCollege _eec = new EverestEngineeringCollege("Everest
Engineering College", "Sanepa -2 Lalitpur", 500, "Er.Umesh Thani");
            Console.WriteLine(_eec.toString());
            Console.Read();
        }
```

Array and String:

- 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.
- **Length** of the array specifies the number of elements present in the array.
- In C# the allocation of memory for the arrays is done dynamically. And arrays are kind of objects, therefore it is easy to find their size using the predefined functions.
- Arrays in C# work differently than they do in C/C++.
- Since arrays are objects in C#, we can find their length using member length. This is different from C/C++ where we find length using sizeof().
- C# array is an object of base type **System.Array**.
- A C# array variable can also be declared like other variables with [] after the data type.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Explanation:

The index is starting from 0, which stores value. We can also store a fixed number of values in an array. Array index is to be increased by 1 in sequence whenever it's not reach the array size.

Declaring Arrays

To declare an array in C#,

Syntax:

< Data Type > [ ] < Name_Array >

where,

< Data Type > → It define the type of the array element.
[ ] → It define the size of the array.
< Name_Array > →it is the Name of array.

E.g. double[] BankBlance;

Initializing an Array

- Declaring an array does not initialize / allocate 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.

E.g. double[] BankBlance = new double[15];

Assigning Values to an Array

- ✓ We can assign values to individual array elements, by using the index number of an array,

E.g. double[] BankBalance = new double[15];

BankBalance[0] = 1000.0;

- ✓ Also, values to the array can be assigned at the time of declaration,

E.g. double[] BankBalance = { 1000.0, 2000.00, 2525.0};

- ✓ An array can also create and initialize as:

E.g. int [] marks = new int[6] { 10, 15, 20, 25, 30};

✓ Similarly, also you can omit the size of the array as:

E.g. int [] marks = new int[] { 10, 15, 20, 25, 30};

- 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.

E.g. double salary = balance[3];

Example:

```
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();
        }
```

O/P of the program:

```
file:///c:/users/paude/documents/visual studio 2015/Projects/C
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
```

Types of Array

1. One Dimensional Array

- One dimensional array contains only one row for storing the values.
- All values of this array are stored contiguously starting from 0 to the array size.
- For example, declaring a single-dimensional array of 5 integers:

  int[] arrayint = new int[5];

This array contains the elements from arrayint[0] to arrayint[4].

Here, the new operator has to create the array and also initialize its element by their default values.

And here all elements are initialized by zero, because it is the **int** type.

```
// C# program to creating an array of the string as week days, store day values in the
weekdays, and prints each value.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EecTestPlatform
{
    class Program
    {
        class EEC
        {
// Main Method
            static void Main(string[] args)
            {
// declares 1D Array of string.
                string[] weekDays;
// allocating memory for days.
                weekDays = new string[] {"Sun", "Mon", "Tue", "Wed","Thu", "Fri", "Sat"};
// displaying Elements of array
                foreach (string day in weekDays)
                    Console.Write(day + " ");

            }
        }
    }
}
```

Output:

Sun Mon Tue Wed Thu Fri Sat

2. Multidimensional Arrays

- The multi-dimensional array is such type of array that contains more than one row to store data on it.
- Also known as Rectangular Array in C# because it has same length of each row
- Contains more than one coma (,) within single rectangular brackets ("[,]").
- It can be a **2D-array** or **3D-array** or more.
- To storing and accessing the values of the array, one required the nested loop.
- The multi-dimensional array declaration, initialization and accessing is as follows:

```
// Main Method
            static void Main(string[] args)
            {
// creates a two-dimensional array of four rows and two columns.
                int[,] intarray = new int[4, 2];

//creates an array of three dimensions, 4, 2, and 3
                int[,,] intarray1 = new int[4, 2, 3];

            }
```

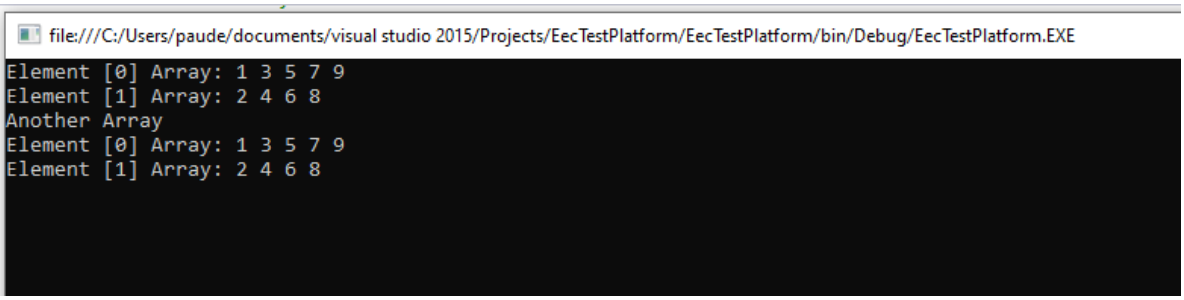
3. Jagged Arrays

- An array whose elements are arrays is known as jagged arrays it means "**array of arrays**".
- The jagged array elements may be of different dimensions and sizes.
- Example to show how to declare, initialize, and access the jagged arrays.
- It's possible to mix jagged and multidimensional arrays. The jagged array is an array of arrays, and therefore its elements are reference types and are initialized to null

```
// Main Method
            static void Main(string[] args)
            {
                /*----------2D Array---------------*/
                // Declare the array of two elements:
                int[][] arr = new int[2][];

                // Initialize the elements:
                arr[0] = new int[5] { 1, 3, 5, 7, 9 };
                arr[1] = new int[4] { 2, 4, 6, 8 };

                // Another way of Declare and Initialize of elements
                int[][] arr1 = { new int[] { 1, 3, 5, 7, 9 },
                         new int[] { 2, 4, 6, 8 } };

                // Display the array elements:
                for (int i = 0; i < arr.Length; i++)
                {
                    System.Console.Write("Element [" + i + "] Array: ");
                    for (int j = 0; j < arr[i].Length; j++)
                        Console.Write(arr[i][j] + " ");
                    Console.WriteLine();
                }
```

```
Console.WriteLine("Another Array");

            // Display the another array elements:
            for (int i = 0; i < arr1.Length; i++)
            {
                System.Console.Write("Element [" + i + "] Array: ");
                for (int j = 0; j < arr1[i].Length; j++)
                    Console.Write(arr1[i][j] + " ");
                Console.WriteLine();
            }
            Console.Read();

        }
```

Output:



Example: To Declare and initialization of a single-dimensional jagged array which contains three two-dimensional array elements of different sizes.

// C# program to single-dimensional jagged array that contains three two-dimensional array elements of different sizes.

```
        // Main Method
        static void Main(string[] args)
        {
            int[][,] arr = new int[3][,] {new int[, ] {{1, 3}, {5, 7}},
                                          new int[, ] {{0, 2}, {4, 6}, {8, 10}},
                                          new int[, ] {{11, 22}, {99, 88}, {0, 9}}};

            // Display the array elements:
```

```
for (int i = 0; i < arr.Length; i++)
                {
                    int x = 0;
                    for (int j = 0; j < arr[i].GetLength(x); j++)
                    {
                        for (int k = 0; k < arr[j].Rank; k++)
                            Console.Write(" arr[" + i + "][" + j + ", " + k + "]:"
                                                    + arr[i][j, k] + " ");
                        Console.WriteLine();
                    }
                    x++;
                    Console.WriteLine();
                    Console.Read();

                }
            }
```

Output: