C# Types of Variables

A variable is a name given to a memory location and all the operations done on the variable effects that memory location. In C#, all the variables must be declared before they can be used. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution.

Types of Variables

- Local variables
- Instance variables or Non – Static Variables
- Static Variables or Class Variables
- Constant Variables
- Readonly Variables

Local Variables

A variable defined within a block or method or constructor is called local variable.

- These variables are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.

- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variables only within that block.

Example 1:

- C#

```
// C# program to demonstrate
// the local variables
using System;
class StudentDetails {

    // Method
    public void StudentAge()
    {

        // local variable age
        int age = 0;

        age = age + 10;
        Console.WriteLine("Student age is : " + age);
```

```
    }

    // Main Method
    public static void Main(String[] args)
    {

        // Creating object
        StudentDetails obj = new StudentDetails();

        // calling the function
        obj.StudentAge();
    }
}
```

Output:

Student age is : 10

Explanation : In the above program, the variable “age” is a local variable to the function StudentAge(). If we use the variable age outside StudentAge() function, the compiler will produce an error as shown in below program.

Example 2:

- C#

```
// C# program to demonstrate the error
// due to using the local variable
// outside its scope
using System;

class StudentDetails {

    // Method
    public void StudentAge()
    {

        // local variable age
        int age = 0;
        age = age + 10;
    }

    // Main Method
    public static void Main(String[] args)
```

```
    {


        // using local variable age outside it's scope
        Console.WriteLine("Student age is : " + age);
    }
}
```

Error:

prog.cs(22,43): error CS0103: The name `age' does not exist in the current context

Instance Variables or Non – Static Variables

Instance variables are non-static variables and are declared in a class but outside any method, constructor or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed. Unlike local variables, we may use access specifiers for instance variables.

Example:

- C#

```
// C# program to illustrate the
```

```
// Instance variables
using System;

class Marks {

    // These variables are instance variables.
    // These variables are in a class and
    // are not inside any function
    int engMarks;
    int mathsMarks;
    int phyMarks;

    // Main Method
    public static void Main(String[] args)
    {

        // first object
        Marks obj1 = new Marks();
        obj1.engMarks = 90;
```

```
obj1.mathsMarks = 80;
obj1.phyMarks = 93;

// second object
Marks obj2 = new Marks();
obj2.engMarks = 95;
obj2.mathsMarks = 70;
obj2.phyMarks = 90;

// displaying marks for first object
Console.WriteLine("Marks for first object:");
Console.WriteLine(obj1.engMarks);
Console.WriteLine(obj1.mathsMarks);
Console.WriteLine(obj1.phyMarks);

// displaying marks for second object
Console.WriteLine("Marks for second object:");
Console.WriteLine(obj2.engMarks);
Console.WriteLine(obj2.mathsMarks);
```

```
        Console.WriteLine(obj2.phyMarks);
    }
}
```

Output :

Marks for first object:

90

80

93

Marks for second object:

95

70

90

Explanation: In the above program the variables, engMarks, mathsMarks, phyMarksare instance variables. If there are multiple objects as in the above program, each object will have its own copies of instance variables. It is clear from the above output that each object will have its own copy of the instance variable.

Static Variables or Class Variables

Static variables are also known as Class variables. If a variable is explicitly declared with the static modifier or if a variable is declared under any static block then these variables are known as static variables.

- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.

Note: To access static variables, there is no need to create any object of that class, simply access the variable as:

class_name.variable_name;

Example:

- C#

```
// C# program to illustrate
// the static variables
```

```
using System;
class Emp {

    // static variable salary
    static double salary;
    static String name = "Aks";

    // Main Method
    public static void Main(String[] args)
    {

        // accessing static variable
        // without object
        Emp.salary = 100000;

        Console.WriteLine(Emp.name + "'s average salary:"
                                    + Emp.salary);
    }
}
```

Output:

Aks's average salary:100000

Note: Initialization of non-static variables is associated with instance creation and constructor calls, so non-static variables can be initialized through the constructor also. We don't initialize a static variable through constructor because every time constructor call, it will override the existing value with a new value.

Difference between Instance variable & Static variable

- Each object will have its own copy of instance variable whereas We can only have one copy of a static variable per class irrespective of how many objects we create.
- Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of instance variable. In the case of static, changes will be reflected in other objects as static variables are common to all object of a class.
- We can access instance variables through object references and Static Variables can be accessed directly using class name.
- In the life cycle of a class a static variable ie initialized one and only one time, whereas instance variables are

initialized for 0 times if no instance is created and n times if n instances are created.

- The Syntax for static and instance variables are :

```
class Example
    {
        static int a; // static variable
        int b;        // instance variable
    }
```

Constants Variables

If a variable is declared by using the keyword "**const**" then it as a constant variable and these constant variables can't be modified once after their declaration, so it's must initialize at the time of declaration only.

Example 1: Below program will show the error because no value is provided at the time of constant variable declaration.

- C#

```
// C# program to illustrate the
// constant variables
using System;
```

```
class Program {

    // constant variable max
    // but no value is provided
    const float max;

    // Main Method
    public static void Main()
    {

        // creating object
        Program obj = new Program();

        // it will give  error
        Console.WriteLine("The value of b is = " + Program.b);
    }
}
```

Error:

prog.cs(8,17): error CS0145: A const field requires a value to be provided

Example 2: Program to show the use of constant variables

- C#

```
// C# program to illustrate the
// constant variable
using System;
class Program {

    // instance variable
    int a = 10;

    // static variable
    static int b = 20;

    // constant variable
    const float max = 50;
```

```
    // Main Method
    public static void Main()
    {

        // creating object
        Program obj = new Program();

        // displaying result
        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
    }
}
```

Output:

The value of a is = 10

The value of b is = 20

The value of max is = 50

Important Points about Constant Variables:

- The **behavior of constant variables will be similar to the behavior of static variables** i.e. initialized one and only one time in the life cycle of a class and doesn't require the instance of the class for accessing or initializing.
- The **difference between a static and constant variable** is, static variables can be modified whereas constant variables can't be modified once it declared.

Read-Only Variables

If a variable is declared by using the **readonly keyword** then it will be read-only variables and these variables can't be modified like constants but after initialization.

- It's not compulsory to initialize a read-only variable at the time of the declaration, they can also be initialized under the constructor.
- The behavior of read-only variables will be similar to the behavior of non-static variables, i.e. initialized only after creating the instance of the class and once for each instance of the class created.

Example 1: In below program, read-only variables k is not initialized with any value but when we print the value of the variable the default value of int i.e 0 will display as follows :

- C#

```
// C# program to show the use
// of readonly variables
// without initializing it
using System;
class Program {

    // instance variable
    int a = 80;

    // static variable
    static int b = 40;

    // Constant variables
    const float max = 50;

    // readonly variables
    readonly int k;

    // Main Method
```

```
    public static void Main()
    {

        // Creating object
        Program obj = new Program();

        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
        Console.WriteLine("The value of k is = " + obj.k);
    }
}
```

Output:

The value of a is = 80

The value of b is = 40

The value of max is = 50

The value of k is = 0

Example 2: To show the initialization of readonly variable in the constructor.

- C#

```
// C# program to illustrate the
// initialization of readonly
// variables in the constructor
using System;
class Geeks {

    // instance variable
    int a = 80;

    // static variable
    static int b = 40;

    // Constant variables
    const float max = 50;

    // readonly variables
    readonly int k;
```

```
// constructor
public Geeks()
{

    // initializing readonly
    // variable k
    this.k = 90;
}

// Main Method
public static void Main()
{

    // Creating object
    Geeks obj = new Geeks();

    Console.WriteLine("The value of a is = " + obj.a);
    Console.WriteLine("The value of b is = " + Geeks.b);
    Console.WriteLine("The value of max is = " + Geeks.max);
```

```
        Console.WriteLine("The value of k is = " + obj.k);
    }
}
```

Output :

The value of a is = 80

The value of b is = 40

The value of max is = 50

The value of k is = 90

Example 3: Program to demonstrate when the readonly variable is initialized after its declaration and outside constructor :

- C#

```
// C# program to illustrate the
// initialization of readonly
// variables twice
using System;
class Geeks {


    // instance variable
```

```
int a = 80;

// static variable
static int b = 40;

// Constant variables
const float max = 50;

// readonly variables
readonly int k;

// constructor
public Geeks()
{

    // first time initializing
    // readonly variable k
    this.k = 90;
}
```

```
    // Main Method
    public static void Main()
    {

        // Creating object
        Geeks obj = new Geeks();

        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Geeks.b);
        Console.WriteLine("The value of max is = " + Geeks.max);

        // initializing readonly variable again
        // will compile time error
        obj.k = 55;

        Console.WriteLine("The value of k is = " + obj.k);
    }
}
```

Error:

prog.cs(41,13): error CS0191: A readonly field `Geeks.k' cannot be assigned to (except in a constructor or a variable initializer)

Important Points about Read-Only Variables:

- The only **difference between read-only and instance variables** is that the instance variables can be modified but read-only variable can't be modified.
- Constant variable is a fixed value for the whole class whereas read-only variables is a fixed value specific to an instance of class.