C# Programming Language Practical Example

By Mohammad Mustafa Hayat Amarkhil

Syntax and Hello World Program

```
using System;// it's mean that we are using the classes from the system namespace(always
appear)


namespace Hello_World //organize the code and this is container for code.(always appear)
{
    class Program// this is class container named  program in what we want to do
                 //we put this inside the class.(always Appear)
    {
        static void Main(string[] args)// this is main function (alway apper)
        {
            Console.WriteLine("Hello World!");//
            // console is the class of system namespace whaich has writeline()
            //method that is use to output the text to the console.
            // Note:
            // if we omit the using system line we would have to write
            // System.Console.WriteLine() to print the data.

            // Note: Every C# statment end with ;
            // C# is case-sensitive.
        }
    }
}
```

Data Type and Variable in C#

```
using System;

namespace VARIABLE
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            //c# Variable:
            // int store integer data.
            //double: store floating point number
            //char: store single character
            //string: store array of character.
            // bool: store bolean value e.g. true and false.

            string name = "Mohammad Mustafa Hayat Amarkhil";// 2 bytes
            Console.WriteLine("Your Name is: " + name);

            int myNum = 1332;// 4 bytes
            Console.WriteLine("Your Num is: " + myNum);

            long myLong = 1332;// 8 bytes

            float myFloat = 23.5f; // 4 bytes
                                   // we must use (f)suffix to convert to double.
            Console.WriteLine("Your Float is : " + myFloat);

            double myDouble = 32.4;// 8 bytes
            Console.WriteLine("Your double is: " + myDouble);

            char myChar = 'A'; // 2 bytes
            Console.WriteLine("Your Char is: " + myChar);

            bool myBool = true; // 1 byte
            Console.WriteLine("Your Bool is: " + myBool);

            // Constants in C#:
            // which mean unchangable and read-only

            const int AGE = 20;
            Console.WriteLine("Your Age is : " + AGE);
           // AGE = 34;// IT WILL GIVE AN ERROR.


        }
    }
}
```

Type Casting In C#

```
using System;

namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            // C# TYPE CASTING:

            /*
              *In C#, there are two types of casting:
              *Implicit Casting (automatically)
              *- converting a smaller type to a larger type size
              *char -> int -> long -> float -> double

             * Explicit Casting (manually)
             * - converting a larger type to a smaller size type
              double -> float -> long -> int -> char
             */

            char myChar = 'A';
            int myInt = myChar;// Automatic casting: char to int
            Console.WriteLine("ASCII OF A is: " + myInt);

            int myAscii = 68;
            myChar = (char)myAscii;

            Console.WriteLine("Your Charater is: " + myChar);

            /*
            Type Conversion Methods:
            It is also possible to convert data types explicitly by using built-in 
methods, such as
            Convert.ToBoolean, Convert.ToDouble, Convert.ToString, Convert.ToInt32 (int) 
and Convert.ToInt64 (long):
             */

            Console.WriteLine("The int Conver to String: " + Convert.ToString(myInt));
            Console.WriteLine("The Char to Int: " + Convert.ToInt32(myChar));




        }
    }

}
```

C# User Input (Get Input From Screen)

```
using System;
namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            //C# User Input
            // use the Console.ReadLine() to get user inpute.

            Console.Write("Enter Your Name: ");
            string name = Console.ReadLine();
            // it will return the string.
            // so if we want to get integer number so we use the type casting.

            Console.WriteLine("Your Name Is: " + name);

            // so if we want get integer number we do like so:

            Console.Write("Enter Your Age: ");
            int age = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Your Age is: " + age);
        }
    }
}
```

C# Operations

Please go to the W3Schools C# Tutorial

C# Math

```
using System;

namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            // C# Math:
            /*
              The C# Math class has many methods that allows you to perform
             * mathematical tasks on numbers.
             */

            Console.WriteLine("The Max Value is: "+ Math.Max(5, 10));
            Console.WriteLine("The Min Value is: " + Math.Min(5, 10));
            Console.WriteLine("The Root of 64 is: "+ Math.Sqrt(64));
            Console.WriteLine("The Absolute Value is: " + Math.Abs(-4.7));
            Console.WriteLine("The Round value is: " + Math.Round(9.99));

        }
    }
}
```

C# String

```
using System;
namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            /*
             * C# Strings
Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes:
             */
            string name = "Mohammad Mustafa Hayat Amarkhil";

            Console.WriteLine("The length: "+ name.Length);
            Console.WriteLine("To Upper Case: " + name.ToUpper());
            Console.WriteLine("To Lower Case: " + name.ToLower());

            string firstName = "Ahmad", lastName = " Hayat";
            Console.WriteLine("Your Full name is: "+ firstName + lastName);

            // we have a function as well string.Concat();
            Console.WriteLine("Your Full name using Function: "+ string.Concat(firstName,
lastName));


             string firstName1 = "John";
            string lastName1 = "Doe";
        //    string name1 = $"My full name is: {firstName} {lastName}"; not working

            // the $ sign is use when we are usng the interpolation method.

            Console.WriteLine("In The third index is: "+ name[3]);

            Console.WriteLine("The index of 'H' is: " + name.IndexOf('H'));
           Console.WriteLine("Your last name is: "+ name.Substring(17));

            // Special Charaters in C#:
            Console.WriteLine("My Name is \'Hayat\' Amarkhil");
            Console.WriteLine("My Name is \"Hayat\" Amarkhil");
            Console.WriteLine("My Name is Hayat\\ Amarkhil");
            /*
             * \n mean new line
             * \t mean tab
             * \b mean backspace
             */
            Console.WriteLine();
            Console.WriteLine();

        }
    }
}
```

Condition in C# (if-else and switch)

```
using System;

namespace VARIABLE
{
    class Program
    {
```

```csharp
static void Main(string[] args)
{
    // Comparision of Three number.
    int a = 3, b = 4, c = 5;
    if (a > b)
    {
        if (a > c)
            Console.WriteLine("A is greater");
        else
            Console.WriteLine("C is Greater.");
    }
    else
    {
        if (b > c)
            Console.WriteLine("B is Greater.");
        else
            Console.WriteLine("C is Greater.");
    }

    // Ternary Operation
    // variable = (Condition ) ? ExperesionTrue : ExpressionFalse;
    int time = 30;
    string result = (time < 20) ? "Have a good Day!" : "Have a good Night!";
    Console.WriteLine(result);


    // C# Switch:
    // choese on of many code blocks to be executed.

    Console.Write("Enter The Day in Number: ");
    int day = Convert.ToInt32(Console.ReadLine());
    string dayText = null;

    switch (day)
    {
        case 0:
            dayText = "Today is Suturday!";
            break;
        case 1:
            dayText = "Today is Sanday!";
            break;
        case 2:
            dayText = "Today is Monday!";
            break;
        case 3:
            dayText = "Today is Tuesday!";
            break;
        case 4:
            dayText = "Today is Wednesday!";
            break;
        case 5:
            dayText = "Today is Thurseday!";
            break;
        case 6:
            dayText = "Today is Friday!";
            break;
        default:
            dayText = "Wrong Input!";
```

```
                break;
            }
            Console.WriteLine(dayText);

        }
    }
}
```

Loop in C# (for, while, do-while)

```
using System;
namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            // While loop:
            Console.WriteLine("The While Loop:");
            int i = 0;
            while (i < 5)
            {
                Console.WriteLine(i);
                i++;
            }

            Console.WriteLine("The Do-While Loop:");
            // do while loop:
            int j = 0;
            do
            {
                Console.WriteLine(j);
                j++;
            }
            while (j < 5);

            Console.WriteLine("The For Loop: ");
            for (int k = 0; k < 5; k++)
            {
                Console.WriteLine(k);
            }

            Console.WriteLine("The Foreach Loop:");
            // the foreach loop is use for accessing the array elements directly.
            string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };
            foreach (string car in cars)
                Console.WriteLine("The car is: " + car);
        }


    }
}
```

Break and Continue Statements

```
using System;

namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {
            // The Break and Continue Statement in loop:
            Console.WriteLine("The Break Statement:");
            for (int i = 0; i < 5; i++)
            {
                Console.Write(i + ", ");
                if (i == 3)
                    break;// exit from  the loop.
            }
            Console.WriteLine("\nThe Continue Statement:");
            int a = 0;
            while (a < 5)
            {
                Console.Write(a + ", ");
                a++;
                if (a % 2 == 0)
                    continue;// jump to the next
            }
        }
    }
}
```

Array and The Syste.Linq

```
using System;// this is used for Console.Write() and Console.Read()
using System.Linq;// this is used for array methods:
    // such as Min, Max,  and Sum.

namespace VARIABLE
{
    class Program
    {
        static void Main(string[] args)
        {


            // Array in C#:
            // and System.Linq.
            //  A varaible that store multiple variable instead of declaring saparate.

            // Declaration and initialization:
            int[] num = { 1, 4, 2, 55, 0, 43 };

            Console.WriteLine("The Array Elements are: ");
            for(int a=0; a<num.Length; a++)
```

```
                Console.Write(num[a] +", ");

            // for each loop:
            Console.WriteLine("\n\nThe Array Emements Using Foreach: ");
            foreach(int n in num)
                Console.Write(n + ", ");

            // sorting array elements: Array.Sort(array_name);
            Array.Sort(num);
            Console.WriteLine("\n\nThe Array Emements After Sort: ");
            foreach (int n in num)
                Console.Write(n + ", ");

            Console.WriteLine("\n\nThe System.Linq Namespace:");
            Console.WriteLine("The Max Value is: " + num.Max());
            Console.WriteLine("The Min Value is: "+ num.Min());
            Console.WriteLine("The sum of Values is: "+num.Sum());


            // Different Wa To create Array:
            // Create an array of four elements, and add values later
            string[] cars = new string[4];

            // Create an array of four elements and add values right away
            string[] cars = new string[4] { "Volvo", "BMW", "Ford", "Mazda" };

            // Create an array of four elements without specifying the size
            string[] cars = new string[] { "Volvo", "BMW", "Ford", "Mazda" };

            // Create an array of four elements, omitting the new keyword, and without
specifying the size
            string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };

            // Saparete Declarationa and Defination:
            // Declare an array
            string[] cars;

            // Add values, using new
            cars = new string[] {"Volvo", "BMW", "Ford"};

            // Add values without using new (this will cause an error)
            cars = {"Volvo", "BMW", "Ford"};

        }
    }
}
```

Functions of Methods

```
using System;// this is used for Console.Write() and Console.Read()
using System.Linq;// this is used for array methods:
    // such as Min, Max,  and Sum.

namespace VARIABLE
{
    class Program
```

```
{
    // the static is access modifier that show visibility.
    // simple show function
    static void showMessage()// declation and definatin of function.
    {
        Console.WriteLine("This is the Show Message Function");
    }

    // function with parameter
    static void showParameter(string name)// this is called parameters.
    {
        Console.WriteLine("Your Full Name is: " + name + " Amarkhil");
    }

    // default parament value:
    static void defaultFunction(int age = 20)
    {
        Console.WriteLine("Your Age is: " + age);
    }

    //The function with return value:
    static int returnFunction(int a, int b)
    {
        return a + b;
    }

    // The Name Arguments Function
    static void namedArgument(string childe1, string childe2)
    {
        Console.WriteLine("First Childe: " + childe1 + "\tSecond Child: " + childe2);
    }

    // The Function Overlaoding:
    // the function with int parameters.
    static int addNumber(int a, int b)
    {
        return a + b;
    }

    // function with double parameters
    static double addNumber(double a, double b)
    {
        return a + b;
    }


    static void Main(string[] args)// this is the main function0
    {
       // C# Methods:
        // A method is a block of code which only runs when it is called.
        // we can pass parament into the method.
        // The method is also know as function.
        Console.WriteLine("Function Without Paramenter:");
        showMessage(); // call the message.
        showMessage();

        Console.WriteLine("\n\nFunction With Parament:");
        showParameter("Ahmad Hayat");// this is called arguments
```

```
            showParameter("Yhaya Hayat");// this is called arguments.

            Console.WriteLine("\n\nThe Default Parament Value:");
            defaultFunction();// with no argument the default will used.
            defaultFunction(44);// with argument
            defaultFunction(55);// with arguments.

            Console.WriteLine("\n\nThe Return Value Function:");
            Console.WriteLine("The sum is: "+ returnFunction(3, 4));
            Console.WriteLine("The sum is: " + returnFunction(33, 4));

            Console.WriteLine("\n\nThe namedArgument Function:");
            namedArgument(childe1: "Mustafa", childe2: "Hayat");


            // Function Overloading:

Console.WriteLine("======================================================================
====");
            Console.WriteLine("The  C# Method Overlaoding:");
            // a methods that have same name with the different parameters.

            Console.WriteLine("The sum of int is: "+addNumber(2, 3));
            Console.WriteLine("The sum of double is: "+addNumber(4.9, 3.22));

        }
    }
}
```

OOPs in C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Hello_World
{
    class Car// create class
    {
       private string name;// memeber
       private int speed;
       // The default is also private if you not specify
       // Access Modifier.

        // The Constructer
       public Car()
       {
           Console.WriteLine("This is Constructer");
           name = null;
           speed = 0;
       }
```

```
         // Constructer Overloading
        public Car(string name, int speed)
        {
            this.name = name;
            this.speed = speed;
        }

         public void setValue(string name, int speed)// member function
         {
             this.name = name;
             this.speed = speed;
         }
         public string getValue()// member functions
         {
             return "Name of Car: " + this.name +
                     "\nSpeed of Car: " + this.speed;
         }
    }
    class Program
    {
        static void Main(string[] args)
        {
            /*
             Procedural programming is about writing procedures or methods that perform
operations on the data, while object-oriented programming is about creating objects that
contain both data and methods.
             Object-oriented programming has several advantages over procedural
programming:

            OOP is faster and easier to execute
            OOP provides a clear structure for the programs
            OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code
easier to maintain, modify and debug
            OOP makes it possible to create full reusable applications with less code and
shorter development time
             */

            Console.WriteLine("Welcome To OOPs in C#");
            Console.WriteLine("So, a class is a template for objects, and an object is an
instance of a class.\n\n");

            // creation of objects.
            Car myCar = new Car();
            myCar.setValue("Corola", 34);
            Console.WriteLine(myCar.getValue());

            Car yourCar = new Car();
            yourCar.setValue("TOYOTA", 66);
            Console.WriteLine(yourCar.getValue());

            Car thierCar = new Car("Ahmad", 493);
            Console.WriteLine( thierCar.getValue());
        }
    }

}
```

Access Modifier, Get and Set Method:

```
using System;

namespace Hello_World
{
    class Person
    {
        string name;// this is called field
        int id;
        public string Name// and this is called property.
        {
            get { return name; }
            set { name = value; }
        }
    }
    class Program
        {
            static void Main(string[] args)
            {
                // Access Modifier in C#:
/*
C# has the following access modifiers:

Modifier      Description
public The code is accessible for all classes
private       The code is only accessible within the same class
protected     The code is accessible within the same class, or in a class that is
inherited from that class. You will learn more about inheritance in a later chapter
internal      The code is only accessible within its own assembly, but not from another
assembly. You will learn more about this in a later chapter
There's also two combinations: protected internal and private protected.
 */
                //C# Properties (Get and Set)

                Person person = new Person();
                person.Name = "Ahmad";
                Console.WriteLine("Your Name is: " + person.Name);

                // the short hand form is

                /*
                 * class Person{
                 *  string name;
                 *  public string Name
                 *  {get; set;}
                 *
                 * }
                 */

            }
        }
}
```

Inheritance And Sealed Class

```
using System;
namespace Hello_World
{
    // Parent class persont
    class Person
    {
        string name;
        int age;
        public Person()
        {
            Console.WriteLine("This is Person Constructer");
            this.name = null;
            this.age = 0;
        }

        public void setValue(string name, int age)
        {
            this.name = name;
            this.age = age;
        }
        public string getValue()
        {
            return  "\nYour Name is: " + this.name +
                    "\nYour Age is: " + Convert.ToString(age);
        }

    }

    // Child Class Student:
    class Student : Person
    {
        int id;
        public Student()
        {
            // Here the Persont Constructer is also called.
            Console.WriteLine("This is Student Constructer.");
            this.id = 0;
        }

        public void setStudentInfo(int id, string name, int age)
        {
            this.setValue(name, age);// call the person.setValue() Function.
            this.id = id;
        }
        public string getStudentInfo()
        {
            return this.getValue() + "\nYour Id is: " + this.id;
        }
    }

    sealed class Library// this class connot be inherited.
    {

                /*
                The sealed Keyword:
```

```
                If you don't want other classes to inherit from a class,
                use the sealed keyword:
                In Above Example Like the Library Class is Sealed class.
                 */

         public string name = "Computer Science Library";
      }


      class Program
          {
              static void Main(string[] args)
              {
                  // C# Inheritance:(Drived an Base Class)
                  Student ahmad = new Student();
                  ahmad.setStudentInfo(1, "Ahmad", 13);
                  Console.WriteLine("\nThe Information About Student:");
                  Console.WriteLine(ahmad.getStudentInfo());

                  Student hamid = new Student();
                  hamid.setStudentInfo(2, "Hamid", 23);
                  Console.WriteLine("\nThe Information About Student:");
                  Console.WriteLine(hamid.getStudentInfo());

              }
          }
  }
```

Polymorphism, The Virtual and Override Keywords

```
using System;
namespace Mustafa_Hayat_Amarkhil
{

    /* class Animal  // Base class (parent)
{
  public void animalSound()
  {
    Console.WriteLine("The animal makes a sound");
  }
}

class Pig : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The pig says: wee wee");
  }
}

class Dog : Animal  // Derived class (child)
{
  public void animalSound()
  {
    Console.WriteLine("The dog says: bow wow");
  }
}

class Program
{
  static void Main(string[] args)
  {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object

    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}


The output will be:

The animal makes a sound
The animal makes a sound
The animal makes a sound
     *
     * Note: The output is not the same we are expected
     * so to make it a real dream we ar using the (virtual) keyword
     * with parent class function that wa going to overried this
     * and the (overried) keywork in the child class
     * */

    // we will do like so:
    // The Parent Class
    class Animal
    {
        public virtual void AnimalSound()
        {
            Console.WriteLine("The Animal makes a sound.");
        }
    }

    // The first Childe:
    class Pig : Animal
    {
        public override void AnimalSound()
        {
            //base.AnimalSound();// if you wanna call the base class method.
            Console.WriteLine("The  Pig says: wee wee");
        }
    }

    // The Second Childe:
    class Dog : Animal
    {
        public override void AnimalSound()
        {
            //base.AnimalSound();// if you wanna call the parent method.
            Console.WriteLine("The Dog say: bow bow");
        }
    }

    class Program
        {
            static void Main(string[] args)
            {
                // C# Polymorphism and Overriding a Methods:

                //Polymorphism means "many forms", and it occurs when we have
                //many classes that are related to each other by inheritance.

                Animal animal = new Animal();
                animal.AnimalSound();

                Pig pig = new Pig();
                pig.AnimalSound();

                Dog dog = new Dog();
                dog.AnimalSound();

            }
        }
}
```

C# Abstraction:

```
using System;
namespace Mustafa_Hayat_Amarkhil
{

    // C# Abstraction:
    //Data abstraction is the process of hiding certain details
    //and showing only essential information to the user.
    // we can achive this using abstract class or interface.

    /*The abstract keyword is used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects
     * (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have
a body. The body is provided by the derived class (inherited from).
     */
    // Note: The Abstract class can have both
    // abstract and regulare methods.

    // to access the abstract class, it must be inherite from another class
    // let's make the Animal class an abstract class.

    // abstract parent class
    abstract class Animal
    {
        public abstract void AnimalSound();
        public void Sleep()
        {
            Console.WriteLine("Zzz");
        }
    }

    // First Childe:
    class Pig : Animal
    {
        public override void AnimalSound()// we have to use the override
                                          // keyword here as well
                                          // otherwise it will give error.
        {
            Console.WriteLine("The Pig says: wee wee");
        }
    }

    // The Second Childe:
    class Dog : Animal
    {
        public override void AnimalSound()
        {
            Console.WriteLine("The Dog says: bow bow");
        }
    }


    class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("This is Dog Info");
                Dog dog = new Dog();
                dog.AnimalSound();
                dog.Sleep();

                Console.WriteLine("\nThis is A Pig Info:");
                Pig pig = new Pig();
                pig.AnimalSound();
                pig.Sleep();

                /*
                 * Why And When To Use Abstract Classes and Methods?
To achieve security - hide certain details and only show the important details of an
object.
                 */
            }
        }
}
```

Interface and Multiple Inheritance:

```
using System;
namespace Mustafa_Hayat_Amarkhil
{
    // C# Interface:
    /* Another way to achieve abstraction in C#, is with interfaces.
       An interface is a completely "abstract class", which can only contain
```

```
   abstract methods and properties (with empty bodies):

 By default, members of an interface are abstract and public.
 Note: Interfaces can contain properties and methods, but not fields.
 *
 * To implement the interface we use the same (:);
 * in the childe class we don't use the override keyword.
 */
// interface
interface IAnimal // it good practice to us I befor the name of interface.
{
    void animalSound();// by default it's public and abstract.
}

// first Childe implement the IAnimal:
class Pig : IAnimal
{
    public void animalSound()
    {
        // The bod of animalSound is provided here.
        Console.WriteLine("The Pig say: wee wee");
    }
}

// The second implement the IAnimal:
class Dog : IAnimal
{
    public void animalSound()
    {
        // The bod of animalSound is provided here.
        Console.WriteLine("The Dog say: bow bow");
    }
}

//Note Why we are using the interface:
// 1) to achive the security
// 2) C# don'nt support multiple inheritance so we can achinve this using interface.

// Multiple Inheritance:
interface IFirstInterface
{
    void myMethod(); // interface method
}

interface ISecondInterface
{
    void myOtherMethod(); // interface method
}

// Implement multiple interfaces
class DemoClass : IFirstInterface, ISecondInterface
{
    public void myMethod()
    {
        Console.WriteLine("Some text..");
    }
    public void myOtherMethod()
    {
            Console.WriteLine("Some other text...");
        }
    }

    class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine("The Pig Class:");
                Pig pig = new Pig();
                pig.animalSound();

                Console.WriteLine("\nThe Dog Class: ");
                Dog dog = new Dog();
                dog.animalSound();

                Console.WriteLine("\n\nMultiple inheritance:");
                DemoClass myObj = new DemoClass();
                myObj.myMethod();
                myObj.myOtherMethod();
            }
        }
}
```

C# Enum

```
using System;
namespace Mustafa_Hayat_Amarkhil
{
    // C# Enum:
    /*
     * An enum is a special "class" that represents a group of constants
     * (unchangeable/read-only variables).
     */
    //Enum is short for "enumerations", which means "specifically listed".
    enum Level// we can use this inside or outside the class.
    {
        High,// this will be 0,
        Medium,// this is 1,
        Low // this is 2 and so on
    }
    class Program
        {
            static void Main(string[] args)
            {
                Level level = Level.High;
                Console.WriteLine("Your Level is: " + level);

                // we can use the enum in switch like follow:

               Level myState = Level.Low;
               switch (myState)
               {
                   case Level.High:
                       Console.WriteLine("High");
```

```
                        break;
                    case Level.Low:
                        Console.WriteLine("Low");
                        break;
                    case Level.Medium:
                        Console.WriteLine("Medium");
                        break;
                    default:
                        Console.WriteLine("Please choese the level!");
                        break;
                }
                 // When and Where to use:
                 /* Use enums when you have values that you know aren't going to
                  * change, like month days, days, colors, deck of cards, etc. */
            }
        }
}
```

File Handling:

```
using System;// use fon Console.Write() and for Console.ReadLine();
using System.IO; // use for File Class.

namespace Mustafa_Hayat_Amarkhil
{
    // C# File Class:
    //The File class from the System.IO namespace, allows us to work with files:

    // the file calss has many method:
    /*
     * AppendText(): append the text.
     * Copy(): Copies a file.
     * Create(): Creates or Overwrites a file
     * Delete(): Delete a file.
     * Exists(): Tests if the file exists.
     * ReadAllText(): Read all the content in the file.
     * Replace(): Replace the file.
     * WriteAllText(): Create new file and write content if it already exists
     *                 it will be overwritten.
     */
    class Program
        {
            static void Main(string[] args)
            {
                // Wrtie to file and read it:
                File.WriteAllText("E:\\AA_ SOFTWARE ENGINEERING\\6SIXTH SEMESTER\\01-MY
SEMESTER\\C# Practical\\test.txt","This text will write in the text");
                string result = File.ReadAllText("E:\\AA_ SOFTWARE ENGINEERING\\6SIXTH
SEMESTER\\01-MY SEMESTER\\C# Practical\\test.txt");
                Console.WriteLine(result);
            }
        }
}
```

Exception Handling (try, catch, finally, throw)

```
using System;// use fon Console.Write() and for Console.ReadLine();
using System.Linq; // use for array.

namespace Mustafa_Hayat_Amarkhil
{
    // C# Exception:
    // The try and catch method.
    // to control the error whenever it's occured.
    class Program
        {

            static void Main(string[] args)
            {
                /*
                 * C# try and catch:
The try statement allows you to define a block of code to be tested for errors while it
is being executed.
The catch statement allows you to define a block of code to be executed, if an error
occurs in the try block.
The try and catch keywords come in pairs:
                 */
                try
                {
                    int[] number = { 3, 2, 1, 4, 2 };
                    // indexOutOfRange Exception is Occured.
                    Console.WriteLine("The data: " + number[6]);
                }
                catch (Exception e)
                {
                    // e.message is describe the error.
                    Console.WriteLine("Exception Occured: \n" + e.Message);
                }

                /*The finally statement lets you execute code, after
                 * try...catch, regardless of the result:
                 */
                finally
                {
                    Console.WriteLine("The 'try and catch' is fineshed");
                }

                /*
                 * The throw keyword
The throw statement allows you to create a custom error.
The throw statement is used together with an exception class.

There are many exception classes available in C#:
ArithmeticException, FileNotFoundException, IndexOutOfRangeException,
                 * TimeOutException,
                 */

                try
                {
                    checkAge(18);// here we control the use defined exceptions.
                }
                catch(Exception e)
                {
                    Console.WriteLine(e.Message);
                }

            }
            public static void checkAge(int age)
            {
                if (age < 20)
                {
                    throw new ArithmeticException("Access Denied - You must be at least
20 years old!");
                }
                else
                    Console.WriteLine("Your Age is Right");
            }
        }
}
```

End Of Console Applications