CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1 – Sep. 2020

Sub:	Programming Using C# and .Net							**Sub Code:**	**18MC A51**
Date:	12/09/2020	Duration:	90 min's	Max Marks:	50	**Sem**	5th	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE CO	OBE RBT
1A	Explain the different components of .NET Framework 4.0	[5]	CO1	L1

Components of .NET Framework 4.0:

The .NET Framework provides all the necessary components to develop and run an application. The components of .NET Framework 4.0 architecture are as follows:

- Common Language Runtime (CLR)
- Common Type System (CTS)
- Metadata and Assemblies
- .NET Framework class library
- Windows Forms
- ASP.NET and ASP.NET AJAX
- ADO.NET
- Windows Workflow Foundation
- Windows Presentation Foundatio
- Windows Communication Found
- Windows CardSpace
- LINQ

Let's now discuss about each of them in detail.

CLR[Common Language Runtime]:

"CLR *is an* ***Execution Engine for .NET Framework applications*****".**

CLR is a heart of the.NET Framework. It provides a run-time environment to run the code and various services to develop the application easily.

The services provided by CLR are –

- Memory Management
- Exception Handling
- Debugging
- Security
- Thread execution
- Code execution
- Language Integration
- Code s
- Verific
- Compi

The following figure shows the **process** of compilation and execution of the code by the JIT Compiler:

i. After verifying, a **JIT** [*Just-In-Time*] compiler extracts the metadata from the file to translate that verified IL code into ***CPU-specific code*** or ***native code***. These type of IL Code is called as **managed code**.

ii. The source code which is directly compiles to the machine code and runs on the machine where it has been compiled such a code called as **unmanaged code**. It does not have any services of CLR.

iii. Automatic garbage collection, exception handling, and memory management are also the responsibility of the CLR.

Managed Code: Managed code is the code that is executed directly by the CLR. The application that are created using managed code automatically have CLR services, such as type checking, security, and automatic garbage collection.

The process of executing a piece of managed code is as follows:

- Selecting a language compiler
- Compiling the code to IL[This intermediate language is called managed code]
- Compiling IL to native code Executing the code

Unmanaged Code: Unmanaged Code directly compiles to the machine code and runs on the machine where it has been compiled. It does not have services, such as security or memory management, which are provided by the runtime. If your code is not security-prone, it can be directly interpreted by any user, which can prove harmful.

Automatic Memory Management: CLR calls various predefined functions of .NET framework to allocate and de-allocate memory of .NET objects. So that, developers need not to write code to explicitly allocate and de-allocate memory.

CTS [Common Type Specifications]:

The CTS defines the rules for declaring, using, and managing types at runtime. It is an integral part of the runtime for supporting cross-language communication.

The common type system performs the following functions:

- Enables cross-language integration, type safety, and high-performance code execution.
- Provides an object-oriented model for implementation of many programming languages.
- Defines rules that every language must follow which runs under .NET framework like C#, VB.NET, F# etc. can interact with each other.

	The CTS can be classified into two data types, are iv. Value Types v. Reference Type			
1b	What are the benefits of .NET framework **Benefits of .NET Framework** Consistent Programming Model:- Provides a consistent object oriented programming model across different languages to create programs for performing different tasks such as connecting to and retrieving data from databases , and reading and writing into files. Cross- Platform support:- Specifies that any windows platform that supports CLR can execute .NET application that is a .NET application enables interoperability between multiple windows operating systems. Language Interoperability:- Enables code written in different languages to interact with each other. This allows reusability of code and improves the efficiency of development process. Automatic Management of Resources :- While developing application we need not manually free up the application resources such as files, memory, network and database connections. The framework provides a feature called CLR that automatically tracks the resource usage and helps you in performing the task of manual resource management. Ease of development:- The framework installs applications or components that do not affect the existing applications. In most cases, to install an application we need to copy the application along with its components on the target computer. In .NET applications are deployed in the form of assemblies. Registry entries are not required to store information about components and applications. In addition assemblies store information about different versions of a single component used by an application. This resolves the version problem .	[5]	CO1	L1
2a	What is an Assembly? Explain each component in the assembly An **assembly** is a file that is automatically generated by the compiler upon successful compilation of every . NET application. It can be either a Dynamic Link Library or an executable file. It is generated only once for an application and upon each subsequent compilation the **assembly** gets updated **Assemblies can stored in two types:** **Static assemblies:** Static assemblies include interfaces, classes and resources. These assemblies are stored in PE (Portable executable) files on a disk. **Dynamic assemblies:** Dynamic assemblies run directly from the memory without being saved to disk before execution. However, after execution you can save the dynamic assemblies on the disk. **Global Assembly Cache:**	[10]	CO1	L2

The Global Assembly Cache (GAC) is ***a folder in Windows directory*** to store the .NET assemblies that are specifically designated to be shared by all applications executed on a system.

- The assemblies must be sharable by registering them in the GAC, only when needed; otherwise, they must be kept private.
- Each assembly is accessed globally without any conflict by identifying its name, version, architecture, culture and public key.

You can deploy an assembly in GAC by using any one of the following:

- An installer that is designed to work with the GAC
- The GAC tool known as **Gacutil.exe**
- The Windows Explorer to drag assemblies into the cache.

Strong Name Assembly:

A Strong Name contains the assembly's identity, that is, the information about the assembly's name, version number, architecture, culture and public key.

- Using Microsoft Visual Studio .NET and other tools, you can provide a strong name to an assembly.
- By providing strong names to the assembly, you can ensure that assembly is globally unique.

Private and Shared Assembly:

A single application uses an assembly, then it is called as **a private assembly.**

Example: If you have created a DLL assembly containing information about your business logic, then the DLL can be used by your client application only. Therefore, to run the application, the DLL must be included in the same folder in which the client application has been installed. This makes the assembly private to your application.

Assemblies that are placed in the Global Assembly cache so that they can be used by multiple applications, then it is called as a **shared assembly**.

Example: Suppose the DLL needs to be reused in different applications. In this scenario, instead of downloading a copy of the DLL to each and every client application, the DLL can be placed in the global assembly cache by using the **Gacutil.exe** tool, from where the application can be accessed by any client application.

Side-by-Side Execution Assembly:

The process of executing **multiple versions** of an *application* or an *assembly* is known as **side-by-side execution**. Support for side-by-side storage and execution of

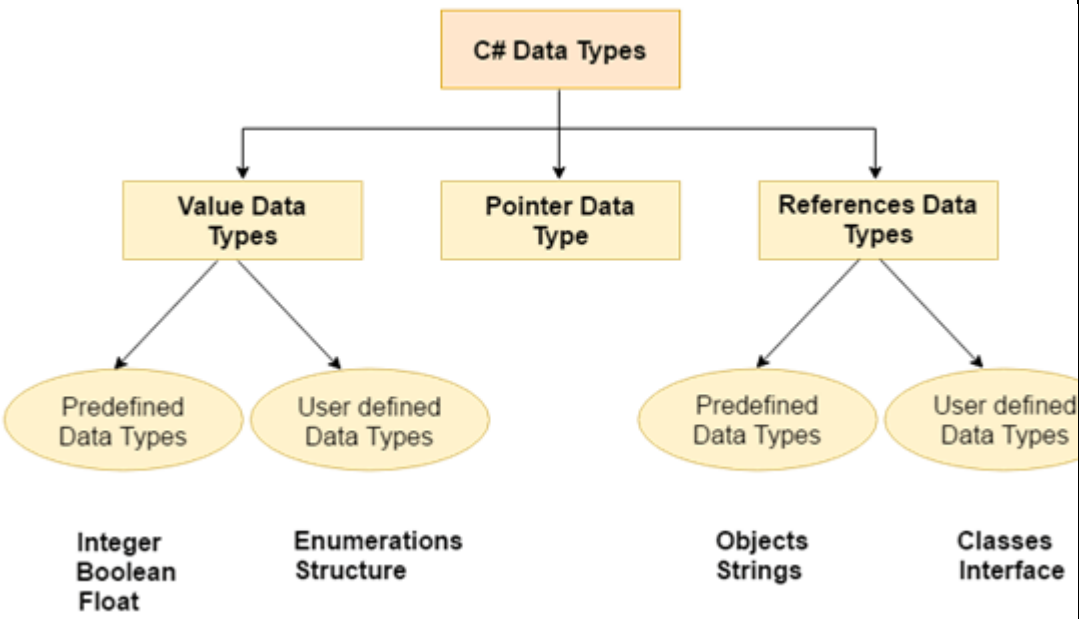
different versions of the same assembly is an integral part of creating a strong name for an assembly.

- Strong naming of .NET assembly is used to provide unique assembly identity by using the sn.exe command utility.
- The strong-named assembly's version number is a part of its identity, the runtime can store multiple versions of the same assembly in the GAC.
- Load these assemblies at runtime.

3a What are different types in C# with example. [5] CO1 L2

C# DataTypes

Types	Data Types
Value Data Type	short, int, char, float, doubl
Reference Data Type	String, Class, Object and Inte
Pointer Data Type	Pointers



Value Data Type

The value data types are integer-based and floating-point based. C# language supports both signed and unsigned

literals.

There are 2 types of value data type in C# language.

1) Predefined Data Types - such as Integer, Boolean, Float, etc.

2) User defined Data Types - such as Structure, Enumerations, etc.

The memory size of data types may change according to 32 or 64 bit operating system.

Let's see the value data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	2 byte	-128 to 127
signed char	2 byte	-128 to 127
unsigned char	2 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to -2,147,483,64
signed int	4 byte	-2,147,483,648 to -2,147,483,64

unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	8 byte	0 - 18,446,744,073,709,551,615
float	4 byte	1.5 * 10-45 - 3.4 * 1038, 7-dig
double	8 byte	5.0 * 10-324 - 1.7 * 10308, 15- precision
decimal	16 byte	at least -7.9 * 10?28 - 7.9 * 1 least 28-digit precision

Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables. In other words they refer to the memory location.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

There are 2 types of reference data type in C# language.

1) Predefined Types - such as Objects, String.

2) User defined Types - such as Classes, Interface.

Object Type: The Object Type is the ultimate base class for all data types in C# Common Type System (CTS). Object is an alias for System.Object class. The object types can be assigned values of any other types, value types,

reference types, predefined or user-defined types. However, before assigning values, it needs type conversion.

When a value type is converted to object type, it is called **boxing** and on the other hand, when an object type is converted to a value type, it is called **unboxing**.

```
Object ob1;

ob1 = 100; // This is boxing
```

Dynamic Type : You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

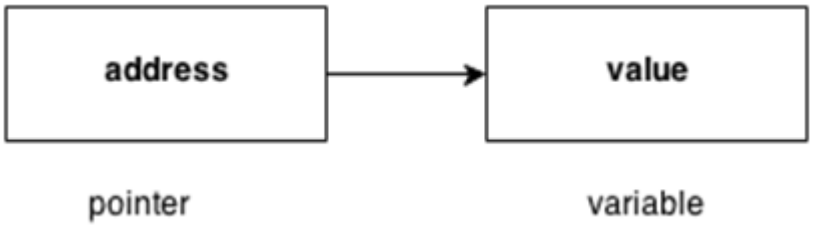
```
dynamic variablename = value;

dynamic d = 10;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

Pointer Data Type

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.



Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the a variable.
* (asterisk	Indirection	Access the val

	sign) operator address. Example: int *a; char *b;			
3b	Type conversion is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms − • **Implicit type conversion** − These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes. • **Explicit type conversion** − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.	[5]	CO1	L2

Sr.No.	Methods & Description
1	**ToBoolean** Converts a type to a Boolean value, where possible.
2	**ToByte** Converts a type to a byte.
3	**ToChar** Converts a type to a single Unicode character, where possible.
4	**ToDateTime** Converts a type (integer or string type) to date-time structures.
5	**ToDecimal** Converts a floating point or integer type to a decimal type.
6	**ToDouble**

		Converts a type to a double type.			
	7	**ToInt16** Converts a type to a 16-bit integer.			
	8	**ToInt32** Converts a type to a 32-bit integer.			
	9	**ToInt64** Converts a type to a 64-bit integer.			
	10	**ToSbyte** Converts a type to a signed byte type.			
	11	**ToSingle** Converts a type to a small floating point number.			
	12	**ToString** Converts a type to a string.			
	13	**ToType** Converts a type to a specified type.			
	14	**ToUInt16** Converts a type to an unsigned int type.			
	15	**ToUInt32** Converts a type to an unsigned long type.			
	16	**ToUInt64** Converts a type to an unsigned big integer.			
4a	**How method overloading is different from overriding . illustrate with an example** In **overriding**, a child class can implement the parent class method in a different way but the child class method has the same name and same method signature as parent whereas		[10]	CO1	L3

in **overloading** there are multiple methods in a class with the same name and different parameters.

Overloading

```
1. class Program
2.     {
3.         public int Add(int num1, int num2)
4.         {
5.             return (num1 + num2);
6.         }
7.         public int Add(int num1, int num2, int num3)
8.         {
9.             return (num1 + num2 + num3);
10.        }
11.        public float Add(float num1, float num2)
12.        {
13.            return (num1 + num2);
14.        }
15.        public string Add(string value1, string value2)
16.        {
17.            return (value1 + " " + value2);
18.        }
19.        static void Main(string[] args)
20.        {
21.            Program objProgram = new Program();
22.            Console.WriteLine("Add with two int parameter :" + objProg
   ram.Add(3, 2));
23.            Console.WriteLine("Add with three int parameter :" + objPr
   ogram.Add(3, 2, 8));
24.            Console.WriteLine("Add with two float parameter :" + objPr
   ogram.Add(3 f, 22 f));
25.            Console.WriteLine("Add with two string parameter :" + objP
   rogram.Add("hello", "world"));
26.            Console.ReadLine();
27.        }
28.    }
```

Overriding

Method Overriding is a type of polymorphism. It has several names like “Run Time Polymorphism” or “Dynamic Polymorphism” and sometime it is called “Late Binding”.

Method Overriding means having two methods with same name and same signatures [parameters], one should be in the base class and other method should be in a derived class [child class]. You can override the functionality of a base class method to create a same name method with same signature in a derived class. You can achieve method overriding using inheritance. Virtual and Override keywords are used to achieve method overriding.

```
1. class BaseClass
2.     {
3.         public virtual int Add(int num1, int num2)
```

```
4.          {
5.              return (num1 + num2);
6.          }
7.      }
8.      class ChildClass: BaseClass
9.      {
10.         public override int Add(int num1, int num2)
11.         {
12.             if (num1 <= 0 || num2 <= 0)
13.             {
14.                 Console.WriteLine("Values could not be less than zero
    or equals to zero");
15.                 Console.WriteLine("Enter First value : ");
16.                 num1 = Convert.ToInt32(Console.ReadLine());
17.                 Console.WriteLine("Enter First value : ");
18.                 num2 = Convert.ToInt32(Console.ReadLine());
19.             }
20.             return (num1 + num2);
21.         }
22.     }
23.     class Program
24.     {
25.         static void Main(string[] args)
26.         {
27.             BaseClass baseClassObj;
28.             baseClassObj = new BaseClass();
29.             Console.WriteLine("Base class method Add :" + baseClassObj
    .Add(-3, 8));
30.             baseClassObj = new ChildClass();
31.             Console.WriteLine("Child class method Add :" + baseClassOb
    j.Add(-2, 2));
32.             Console.ReadLine();
33.         }
34.     }
```

5a	Write a C# program to find Fibonacci series with and without recursion Without recursion	[10]	CO1	L3

```
1.  class Program
2.      {
3.          static int FibonacciSeries(int n)
4.          {
5.              int firstnumber = 0, secondnumber = 1, result = 0;
6.
7.              if (n == 0) return 0; //To return the first Fibonacci numb
    er
8.              if (n == 1) return 1; //To return the second Fibonacci num
    ber
9.
10.
11.             for (int i = 2; i <= n; i++)
12.             {
```

```
                result = firstnumber + secondnumber;
                firstnumber = secondnumber;
                secondnumber = result;
            }

            return result;
        }

        static void Main(string[] args)
        {
            Console.Write("Enter the length of the Fibonacci Series: "
);
            int length = Convert.ToInt32(Console.ReadLine());

            for (int i = 0; i < length; i++)
            {
                Console.Write("{0} ", FibonacciSeries(i));
            }
            Console.ReadKey();
        }
    }
```

With recursion

```
 class Program
      {
            public static int FibonacciSeries(int n)
            {
                  if (n == 0) return 0; //To return the first Fibonacc
i number
                  if (n == 1) return 1; //To return the second Fibonac
ci number
                  return FibonacciSeries(n - 1) + FibonacciSeries(n -
 2);
            }
            public static void Main(string[] args)
            {
                  Console.Write("Enter the length of the Fibonacci Ser
ies: ");
                  int length = Convert.ToInt32(Console.ReadLine());
                  for (int i = 0; i < length; i++)
                  {
                        Console.Write("{0} ", FibonacciSeries(i));
                  }
                  Console.ReadKey();
            }
      }
```

6a	Write a c# program to print factorial of a number
	`using System;`
	`public class FactorialExample`
	`{`
		[10]	CO1	L3

```
  public static void Main(string[] args)
   {
    int i,fact=1,number;
    Console.Write("Enter any Number: ");
    number= int.Parse(Console.ReadLine());
    for(i=1;i<=number;i++){
     fact=fact*i;
    }
    Console.Write("Factorial of " +number+" is: "+fact);
  }
}
```

7a **Write a c# program to convert number in characters** [10] CO1 L3

```
using System;
 public class ConversionExample
 {
   public static void Main(string[] args)
    {
     int n,sum=0,r;
     Console.Write("Enter the Number= ");
     n= int.Parse(Console.ReadLine());
     while(n>0)
     {
      r=n%10;
      sum=sum*10+r;
      n=n/10;
     }
     n=sum;
     while(n>0)
     {
      r=n%10;
      switch(r)
      {
       case 1:
       Console.Write("one ");
       break;
       case 2:
       Console.Write("two ");
       break;
       case 3:
       Console.Write("three ");
       break;
       case 4:
       Console.Write("four ");
       break;
       case 5:
       Console.Write("five ");
       break;
       case 6:
```

	Console.Write("six ");
	break;
	case 7:
	Console.Write("seven ");
	break;
	case 8:
	Console.Write("eight ");
	break;
	case 9:
	Console.Write("nine ");
	break;
	case 0:
	Console.Write("zero ");
	break;
	default:
	Console.Write("tttt ");
	break;
	}//end of switch
	n=n/10;
	}//end of while loop
	}
	}			

8a	Write a C# program to convert decimal to binary
	using System;
	public class ConversionExample
	{
	public static void Main(**string**[] args)
	{
	int n, i;
	int[] a = **new int**[10];
	Console.Write("Enter the number to convert: ");
	n= **int**.Parse(Console.ReadLine());
	for(i=0; n>0; i++)
	{
	a[i]=n%2;
	n= n/2;
	}
	Console.Write("Binary of the given number= ");
	for(i=i-1 ;i>=0 ;i--)
	{
	Console.Write(a[i]);
	}
	}
	}
		[10]	CO1	L3

10		[10]	CO2	L3