CMR Institute of Technology, Bangalore		
Department(s): Information Science & Engineering		
Semester: 05	Section(s): A &B	Lectures/week: 04
Subject: Introduction to Dot Net Framework for Application Development.		Code: 17CS564
Course Instructor(s): Mrs. Savitha Hiremath		
VTU- Question paper Solution- Dec 2019/ Jan 2020		

Module 1		
1. a	Explain namespaces with Programming example Namespaces are used to organize the classes. It helps to control the scope of methods and classes in larger .Net programming projects. It provides a way to keep one set of names(like class names) different from other sets of names. The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace. It is also referred as named group of classes having common features. The members of a namespace can be namespaces, interfaces, structures, and delegates. `namespace MyNamespace` `{` `class MyClass` `{` `public void MyMethod()` `{` `System.Console.WriteLine("Creating my namespace");` `}` `}` `}`	5M
b.	Define variable. Explain the details of the variable like declaration, initialization, accepting the values and also rules for it. Give simple examples to it. Ans: A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable. Declaration of variable: we declared a variable called "message" as shown below. `namespace CSharpTutorials` `{` `class Program`	7M

```
    {
        static void Main(string[] args)
        {
            string message = "Hello World!!";

            Console.WriteLine(message);
        }
    }
}
```

Initialization::
Multiple variables of the same data type can be declared and initialized in a single line separated by commas.

Example: Multiple Declarations

int i, j, k, l = 0;

int amount, num;

Rules to Declare C# Variables

Before we declare and define variables in c# programming language, we need to follow particular rules.

- You can define a variable name with the combination of alphabets, numbers, and underscore.
- A variable name must always start with either alphabet or underscore but not with numbers.
- While defining the variable, no white space is allowed within the variable name.
- You should not use any reserved keywords such as int, float, char, etc. for a variable name.
- In c#, once the variable is declared with a particular data type, it cannot be re-declared with a new type and we shouldn't assign a value that is not compatible with the declared type.

c.	Explain the Method with Syntax. Write a C# program for method overloading and also give explanation for overloading.	8M

Ans: Method Overloading is the common way of implementing polymorphism. It is the ability to redefine a function in more than one form. A user can implement function overloading by defining two or more functions in a class sharing the same name. C# can distinguish the methods with different method signatures. i.e. the methods can have the same name but with different parameters list (i.e. the number of the parameters, order of the parameters, and data types of the parameters) within the same class.

- Overloaded methods are differentiated based on the number and type of the parameters passed as arguments to the methods.
- You can not define more than one method with the same name, Order and the type of the arguments. It would be compiler error.
- The compiler does not consider the return type while differentiating the overloaded method. But you cannot declare two methods with the same signature and different return type. It will throw a compile-time error. If both methods have the same parameter types, but different return type, then it is not possible.
- Why do we need Method Overloading ??
- If we need to do the same kind of the operation in different ways i.e. for different inputs. In the example described below, we are doing the addition operation for different inputs. It is hard to find many different meaningful names for single action.

Different ways of doing overloading methods-

Method overloading can be done by changing:

1. The number of parameters in two methods.
2. The data types of the parameters of methods.
3. The Order of the parameters of methods.

Example:

```
/ C# program to demonstrate the function
// overloading by changing the Number
// of parameters
using System;
```

```
class GFG {

    // adding two integer values.
    public int Add(int a, int b)
    {
        int sum = a + b;
        return sum;
    }

    // adding three integer values.
    public int Add(int a, int b, int c)
    {
        int sum = a + b + c;
        return sum;
    }

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

        // Creating Object
        GFG ob = new GFG();

        int sum1 = ob.Add(1, 2);
        Console.WriteLine("sum of the two "
                          + "integer value : " + sum1);

        int sum2 = ob.Add(1, 2, 3);
        Console.WriteLine("sum of the three "
                          + "integer value : " + sum2);
    }
}
```

Output:

```
sum of the two integer value : 3

sum of the three integer value : 6
```

By changing the Data types of the parameters

```
// C# program to demonstrate the function
// overloading by changing the Data types
//  of the parameters
using System;
class GFG {

    // adding three integer values.
    public int Add(int a, int b, int c)
    {
        int sum = a + b + c;
        return sum;
    }

    // adding three double values.
    public double Add(double a,
                      double b, double c)
```

```
    {
        double sum = a + b + c;
        return sum;
    }

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

        // Creating Object
        GFG ob = new GFG();

        int sum2 = ob.Add(1, 2, 3);
        Console.WriteLine("sum of the three "
                          + "integer value : " + sum2);
        double sum3 = ob.Add(1.0, 2.0, 3.0);
        Console.WriteLine("sum of the three "
                          + "double value : " + sum3);
    }
}
```

Output:

```
sum of the three integer value : 6

sum of the three double value : 6
```

By changing the Order of the parameters

```
// C# program to demonstrate the function
// overloading by changing the
// Order of the parameters
using System;
class GFG {

    // Method
    public void Identity(String name, int id)
    {

        Console.WriteLine("Name : " + name + ", "
                          + "Id : " + id);
    }

    // Method
    public void Identity(int id, String name)
    {

        Console.WriteLine("Name : " + name + ", "
                          + "Id : " + id);
    }

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

        // Creating Object
        GFG obj = new GFG();
```

```
        obj.Identity("Akku", 1);
        obj.Identity("Abby", 2);
    }
}
```

Output:

```
Name : Akku, Id : 1

Name : Abby, Id : 2
```

2. a

Write a C# program to find the factorial of a no. using while and for Loop. 6M

Program to find Factorial of a no. using For loop.

```
using System;
namespace LogicalPrograms
{
class Program
{
static void Main(string[] args)
{
Console.Write("Enter a Number : ");
int number = int.Parse(Console.ReadLine());

int factorial = 1;
for (int i = 1; i <= number; i++)
{
factorial = factorial * i;
  }
    Console.Write($"Factorial of {number} is: {factorial}");

    Console.ReadLine();
  }
 }
}
```

while loop to calculate the factorial of a number.

```
using System;
namespace LogicalPrograms
{
class Program
{
static void Main(string[] args)
```

	{ **Console.Write("Enter a Number : ");** **int number = int.Parse(Console.ReadLine());** **long factorial = 1;** **while (number != 1)** **{** **factorial = factorial * number;** **number = number - 1;** **}** **Console.Write($"Factorial is: {factorial}");** **Console.ReadLine();** **}** **}** **}**	
b.	Explain the conditional operators and write the C# program for the same. It is ternary operator which is a shorthand version of if-else statement. It has three operands and hence the name ternary. It will return one of two values depending on the value of a Boolean expression. Syntax : condition ? first_expression : second_expression; Explanation : condition: It must be evaluate to true or false. If the condition is true first_expression is evaluated and becomes the result. If the condition is false, second_expression is evaluated and becomes the result. Example : // C# program to demonstrate the working // of Conditional Operator using System; namespace Conditional { class GFG { // Main Function static void Main(string[] args) { int x = 5, y = 10, result; // To find which value is greater // Using Conditional Operator	6M

```
        result = x > y ? x : y;

        // To display the result
        Console.WriteLine("Result: " + result);

        // To find which value is greater
        // Using Conditional Operator
        result = x < y ? x : y;

        // To display the result
        Console.WriteLine("Result: " + result);
    }
}
}
```

Output :

```
Result: 10

Result: 5
```

c. Describe Try, Catch, Finally and throw keywords with programming example — 8M

Exception handling is based on the following keywords and its usage −

- **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.

To handle exceptions, you need to set them like the following syntax in C# −

```
try {
   // statements causing exception
} catch( ExceptionName e1 ) {
   // error handling code
} catch( ExceptionName e2 ) {
   // error handling code
} catch( ExceptionName eN ) {
   // error handling code
} finally {
   // statements to be executed
```

```
}
```

Example Program:

```
using System;

namespace ErrorHandlingApplication {
   class DivNumbers {
      int result;

      DivNumbers() {
         result = 0;
      }
      public void division(int num1, int num2) {
         try {
            result = num1 / num2;
         } catch (DivideByZeroException e) {
            Console.WriteLine("Exception caught: {0}",
e);
         } finally {
            Console.WriteLine("Result: {0}", result);
         }
      }
      static void Main(string[] args) {
         DivNumbers d = new DivNumbers();
         d.division(25, 0);
         Console.ReadKey();
      }
   }
}
```

Module 2

3. a	Define Constructor. Explain Constructor Overloading with programming example. Ans: Constructor is a special method that runs automatically when you create an instance of a class. It has same name as a class and it can take parameters. But cannot return a value. Every class will have a constructor and compiler automatically generates a default constructor. We can overload constructors in different ways as follows: • By using different type of arguments • By using different number of arguments • By using different order of arguments ***By changing the Data types of the parameters***	6M

	public ADD (int a, float b); public ADD (string a, int b); Here the name of the class is ADD. In first constructor there are two parameters, first one is int and another one is float and in second constructor, also there is two parameters, first one is string type and another one is int type. Here the constructors have the same name but the types of the parameters are different, similar to the concept of method overloading. ***By changing the number of the parameters*** In this case, we will use two or more constructors having the different number of parameters. The data types of arguments can be the same but the number of parameters will be different. **Example:** public ADD (int a, int b); public ADD (int a, int b, int c); **By changing the Order of the parameters** public student(double a, int x, string s) public student(string s, int x, double a) Here, the two constructor hold the same types of parameters, that is, each constructor has one double type, one int type and one string type parameter, but the positions of the parameters are different. The compiler will invoke the constructor according to their argument order.	
b.	Describe Static Class, Static Method and data with an example. Ans: C# Static Class A C# static class is a class that can't be instantiated. The sole purpose of the class is to provide blueprints of its inherited classes. A static class is created using the "static" keyword in C#. A static class can contain static members only. You can't create an object for the static class. **Advantages of Static Classes**	6M

1. If you declare any member as a non-static member, you will get an error.
2. When you try to create an instance to the static class, it again generates a compile time error, because the static members can be accessed directly with its class name.
3. The static keyword is used before the class keyword in a class definition to declare a static class.
4. A static class members are accessed by the class name followed by the member name.

Syntax of static class

```
static class classname
{
   //static data members
   //static methods
}
```

Static Class Demo

```
namespace StaticConstructorsDemo
{
    class MyCollege
    {
        //static fields
        public static string CollegeName;
        public static string Address;

        //static constructor
            static MyCollege()
            {
                CollegeName = "ABC College of Technology";
                Address = "Hyderabad";
            }
         }
         class Program
         {
            static void Main(string[] args)
            {
                Console.WriteLine(MyCollege.CollegeName);
                Console.WriteLine(MyCollege.Address);
                Console.Read();
            }
```

```
24.         }
25.     }
```

Static Members

There are two types of C# static class members, static and non-static.

Non-static members

This is the default type for all the members. If you do not use the "static" keyword for the declaration of a field / property or a method, then it can be called a "Non-static member". The main feature of a non-static member is it will be bound with the object only.

Non-static Fields / Properties
The memory is allocated when the object is created.

Non-static Methods
These methods can implement operations on non-static fields and properties

Static Members

If you use the "static" keyword for the declaration of a field / property or a method, then it is called a "Static member". The main feature of a non-static member is that it will not be bound with any object. It is individually accessible with the class name. In other words, the static members are accessible directly, without even creating one object also.

Static Fields / Properties
The memory will be allocated individually, without any relation with the object.

Static Methods
These methods can implement operations on static fields and properties only; and can‘t access the non-static members.

Example for Static method:

```
1. using System;
2. using System.Linq;
3. using System.Text;
4. using System.Collections.Generic;
5. namespace StaticConstructorsDemo
6. {
```

```
7.      class Student
8.      {
9.          //non-statcic data members
10.             public string StudentName;
11.             public string Course;
12.             public void SetStudentDetails(string St
    uName, string Cou)
13.             {
14.                 StudentName = StuName;
15.                 Course = Cou;
16.             }
17.             public void DisplayStudentDetails()
18.             {
19.                 Console.WriteLine(StudentName + " -
     " + Course);
20.             }
21.
22.             //static data members
23.             public static string CollegeName = "ABC
     College of Technology";
24.             public static string CollegeAddress = "
    Hyderabad";
25.
26.             //static methods
27.             public static void DisplayCollegeDetail
    s()
28.             {
29.                 Console.WriteLine(CollegeName);
30.                 Console.WriteLine(CollegeAddress);

31.             }
32.         }
33.         class Program
34.         {
35.             static void Main(string[] args)
36.             {
37.                 //access static members
38.                 Student.DisplayCollegeDetails();
39.
40.                 //acess non-static members
41.                 Console.WriteLine();
```

```
                Student s1= new Student();
                Student s2 = new Student();
                s1.SetStudentDetails("Sarath","MCA"
  );
                s1.SetStudentDetails("Syam","MBA");

                s1.DisplayStudentDetails();
                s2.DisplayStudentDetails();
                Console.Read();
            }
        }
    }
```

c. Explain value Type and reference type and boxing and unboxing with programming example. 8M

Ans:

BASIS FOR COMPARISON	BOXING	**UNBOXING**
Basic	Object type refers to the value type.	process of retrieving value from the boxed object.
Storage	The value stored on the stack is copied to the object stored on heap memory.	The object's value stored on the heap memory is copied to the value type stored on stack.
Conversion	Implicit conversion.	Explicit conversion.
Example	int n = 24; object ob = n;	int m = (int) ob;

4. a	Briefly example “ref” and “out” keywords with example. Ans: Ans: The keywords ref and out are used to pass arguments within a method or function. Both indicate that an argument/parameter is passed by reference. By default parameters are passed to a method by value. By using these keywords (ref and out) we can pass a parameter by reference. Ref Keyword The ref keyword passes arguments by reference. It means any changes made to this argument in the method will be reflected in that variable when control returns to the calling method. Out Keyword The out keyword passes arguments by reference. This is very similar to the ref keyword. Parameter to the method is reference type. static void increment(int par) { par++; } Static void Main() { Int arg = 42; Increment(arg); Console.WriteLine(arg); } • Prefix parameter with ref key • Send reference to the actual parameter, not the copy of it. static void increment(ref int par) { par++; } Static void Main()	5M

	{ int arg = 42; Increment(ref arg); Console.WriteLine(arg); // prints 43 } Example for Out parameter; static void increment(out int par) { par=42; } Static void Main() { int arg ; Increment(out arg); Console.WriteLine(arg); // prints 42 }	
b.	Define enumerators with syntax. Write c# program that display month name and its numeric value using enum. Ans: ***Enumeration (or enum)*** is a value data type in C#. It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain. For example, the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit. Other examples include natural enumerated types (like the planets, days of the week, colors, directions, etc.). The main objective of enum is to define our own data types(Enumerated Data Types). Enumeration is declared using ***enum*** keyword directly inside a namespace, class, or structure. **Syntax:** `enum Enum_variable` `{` `string_1...;` `string_2...;` `.` `.`	7M

```
}
```

In above syntax, Enum_variable is the name of the enumerator, and string_1 is attached with value 0, string_2 is attached value 1 and so on. Because by default, the first member of an enum has the value 0 and the value of each successive enum member is increased by 1. We can change this default value.

- **Example 1:** Consider the below code for the enum. Here enum with name **month** is created and its data members are the name of months like jan, feb, mar, apr, may. Now let's try to print the default integer values of these enums. **An explicit cast is required to convert from enum type to an integral type.**

filter_none
edit
play_arrow
brightness_4

```
// C# program to illustrate the enums
// with their default values
using System;
namespace ConsoleApplication1 {

// making an enumerator 'month'
enum month
{

    // following are the data members
    jan,
    feb,
    mar,
    apr,
    may

}

class Program {

    // Main Method
    static void Main(string[] args)
    {

        // getting the integer values of data members..
        Console.WriteLine("The value of jan in month " +
                          "enum is " + (int)month.jan);
        Console.WriteLine("The value of feb in month " +
                          "enum is " + (int)month.feb);
```

```
            Console.WriteLine("The value of mar in month " +
                              "enum is " + (int)month.mar);
            Console.WriteLine("The value of apr in month " +
                              "enum is " + (int)month.apr);
            Console.WriteLine("The value of may in month " +
                              "enum is " + (int)month.may);
        }
    }
    }
```

Output:

```
The value of jan in month enum is 0

The value of feb in month enum is 1

The value of mar in month enum is 2

The value of apr in month enum is 3

The value of may in month enum is 4
```

c. Describe the structures and jagged arrays with example. 8M

Ans:

agged array is a **array of arrays** such that member arrays can be of different sizes. In other words, the length of each array index can differ. The elements of Jagged Array are reference types and initialized to null by default. Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.

Declaration

In Jagged arrays, user has to provide the number of rows only. If the user is also going to provide the number of columns, then this array will be no more Jagged Array.

Syntax:

```
data_type[][] name_of_array = new data_type[rows][]
```

Example:

```
int[][] jagged_arr = new int[4][]
```

In the above example, a single-dimensional array is declared that has 4 elements(rows), each of which is a 1-D array of integers.

Initialization

	The elements of Jagged Array **must be initialized** before its use. You can separately initialize each array element. There are many ways to initialize the Jagged array's element. **Example 1:** Providing the size of each array elements separately. Here each of the elements is a 1-D array of integers where: • The first row or element is an array of 2 integers. • The second row or element is an array of 4 integers. • The third row or element is an array of 6 integers. • The fourth row or element is an array of 7 integers. `jagged_arr[0] = new int[2];` `jagged_arr[1] = new int[4];` `jagged_arr[2] = new int[6];` `jagged_arr[3] = new int[7];`	
	Module 3	
5 a.	Explain params array with programming example. ANS: The "params" keyword in C# allows a method to accept a variable number of arguments. C# params works as an array of objects. By using params keyword in a method argument definition, we can pass a number of arguments. arams keyword in C# can be used to declare method that does not know the number of parameters. Parmas are also useful to write "clean code". Instead of using various overloaded methods to pass multiple values, we can simply create an array and pass it as an arugment or a comma separated list of values. Let's say, we have a class, Student. It has two methods, TotalMarks and AllSubjects. Each of these methods take a parameter of type params. As you can see from the below code, params is an array. 1. `public class Students` 2. `{` 3. `public static int TotalMarks(params int[] list)` 4. `{` 5. `int total = 0;` 6. `for (int i = 0; i < list.Length; i++)` 7. `total += list[i];` 8. `return total;` 9. `}` 10. 11. `public static string AllSubjects(params string[] subjects)`	6M

```
        {
            System.Text.StringBuilder builder = new
  System.Text.StringBuilder();
            for (int i = 0; i < subjects.Length; i+
  +)
            {
                builder.Append(subjects[i]);
                builder.Append(" ");
            }
            return builder.ToString();
        }
    }
```

b

Define Inheritance. Explain new methods, virtual methods and override methods with examples in Inheritance.

6M

Ans:

n C#, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which is defined in other class.

In C#, the class which inherits the members of another class is called **derived class** and the class whose members are inherited is called **base** class. The derived class is the specialized class for the base class.

Advantage of C# Inheritance

Code reusability: Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class

1. Virtual keyword: This modifier or keyword use within base class method. It is used to modify a method in base class for overridden that particular method in the derived class.
2. override: This modifier or keyword use with derived class method. It is used to modify a virtual or abstract method into derived class which presents in base class.

```
class base_class
{
    public virtual void gfg();
}

```

```
8.    class derived_class : base_class
9.    {
10.       public override void gfg();
11.   }
12.   class Main_Method
13.   {
14.    static void Main()
15.    {
16.       derived d_class = new derived_class();
17.       d.gfg();
18.
19.         base_class b = new derived_class();
20.           b.gfg();
21.    }
22.   }
```

Here first, *d* refers to the object of the class *derived_class* and it invokes *gfg()* of the class *derived_class* then, *b* refers to the reference of the class base and it hold the object of class derived and it invokes *gfg()* of the class derived. Here *gfg()* method takes permission from base class to overriding the method in derived class.

c	Explain the use of extension methods in c# with programming example. Ans: Extension methods are static methods, which are called as if they were instance methods on the extended type. With Extension methods, you can add methods to existing types without even creating a new derived type, recompiling, or modifying the original type. The following is the extension method we have created. `public static int myExtensionMethod(this string str) {` `return Int32.Parse(str);`	8M

	} Let us see an example wherein we have used extension method. **Example** `using System;` `using System.Text;` `namespace Program {` `public static class Demo {` `public static int myExtensionMethod(this string str) {` `return Int32.Parse(str);` `}` `}` `class Program {` `static void Main(string[] args) {` `string str1 = "565";` `int n = str1.myExtensionMethod();` `Console.WriteLine("Result: {0}", n);` `Console.ReadLine();` `}` `}` `}` **Output** `Result: 565`	
6 a	Define Interface. Demonstrate the implementation of an Interface Ans: • An interface can only contain declarations but not implementations. • When we implement an interface, we must ensure that each method matches its corresponding interface method.	6M

	• Return type should match • Any parameters passed should match exactly. • In C#, you cannot use any access modifier for any member of an interface. • Example: interface landbound { int numberoflegs(); } class horse : landbound { public int numberoflegs() { return 4; } }	
b	Explain Abstract Class and Abstract Method with syntax and programming example: Ans: Abstract Class in C# A method that does not have a body is called an abstract method and the class that is declared by using the keyword abstract is called an abstract class. If a class contains an abstract method, then it must be declared as abstract. In this article, we will discuss abstract class and abstract methods in details using some real-time examples. What is an abstract class? A class that is declared by using the keyword abstract is called an abstract class. An abstract class is a partially implemented class used for developing some of the operations of an object which are common for all next level subclasses. So it contains both abstract methods, concrete methods including variables, properties, and indexers. It is always created as superclass next to interface in object inheritance hierarchy for implementing common operations from an interface. An abstract class may or may not have abstract methods. But if a class contains an abstract method then it must be declared as abstract. An abstract class cannot be instantiated directly. It's compulsory to create/derive a subclass from the abstract class in order to provide functionality to its abstract functions. What is the abstract method? A method that does not have a body is called an abstract method. It is declared with the modifier abstract. It contains only Declaration/signature and does not contain implementation/body/definition of the method. An abstract function should be terminated with a semicolon. Overriding of an abstract function is compulsory. Why should method have abstract keyword if it does not have a body? In a class, we are allowed only to define a class with the body. Since we are changing its default behavior (means removing its body) it must have the abstract keyword in its prototype.	6M

```
// C# program to show the
// working of abstract class
using System;

// abstract class 'GeeksForGeeks'
public abstract class GeeksForGeeks {

    // abstract method 'gfg()'
    public abstract void gfg();

}

// class 'GeeksForGeeks' inherit
// in child class 'Geek1'
public class Geek1 : GeeksForGeeks
{

    // abstract method 'gfg()'
    // declare here with
    // 'override' keyword
    public override void gfg()
    {
        Console.WriteLine("class Geek1");
    }
}

// class 'GeeksForGeeks' inherit in
// another child class 'Geek2'
public class Geek2 : GeeksForGeeks
{

    // same as the previous class
    public override void gfg()
    {
        Console.WriteLine("class Geek2");
    }
}

// Driver Class
public class main_method {

    // Main Method
    public static void Main()
    {

        // 'g' is object of class
        // 'GeeksForGeeks' class '
        // GeeksForGeeks' cannot
        // be instantiate
        GeeksForGeeks g;
```

```
        // instantiate class 'Geek1'
        g = new Geek1();

        // call 'gfg()' of class 'Geek1'
        g.gfg();

        // instantiate class 'Geek2'
        g = new Geek2();

        // call 'gfg()' of class 'Geek2'
        g.gfg();

    }
}
```

c | Explain Garbage collector along with working procedure and also explain the managing system resources by garbage collector. | 8M

ANS:

Automatic memory management is made possible by Garbage Collection in .NET Framework. When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

Garbage collection will always work on Managed Heap and internally it has an Engine which is known as the Optimization Engine.

Garbage Collection occurs if at least one of multiple conditions is satisfied. These conditions are given as follows:

- If the system has low physical memory, then garbage collection is necessary.
- If the memory allocated to various objects in the heap memory exceeds a pre-set threshold, then garbage collection occurs.
- If the GC.Collect method is called, then garbage collection occurs. However, this method is only called under unusual situations as normally garbage collector runs automatically.

There are mainly **3** phases in garbage collection. Details about these are given as follows:

1. Marking Phase: A list of all the live objects is created during the marking phase. This is done by following the references from all the root objects. All of the objects that are not on the list of live objects are potentially deleted from the heap memory.
2. Relocating Phase: The references of all the objects that were on the list of all the live objects are updated in the relocating phase so that they point to the new location where the objects will be relocated to in the compacting phase.
3. Compacting Phase: The heap gets compacted in the compacting phase as the space occupied by the dead objects is released and the live objects remaining are

moved. All the live objects that remain after the garbage collection are moved towards the older end of the heap memory in their original order.

Managing system resources by garbage collector.

- Garbage Collection succeeds in allocating objects efficiently on the heap memory using the generations of garbage collection.
- Manual freeing of memory is not needed as garbage collection automatically releases the memory space after it is no longer required.
- Garbage collection handles memory allocation safely so that no objects use the contents of another object mistakenly.
- The constructors of newly created objects do not have to initialize all the data fields as garbage collection clears the memory of objects that were previously released.

Module 4

7 a | Describe the Implementation of Encapsulation by using methods and properties in a class with programming example | 8M

ANS:

In c#, **Encapsulation** is a process of binding the data members and member functions into a single unit. In c#, the class is the real-time example for encapsulation because it will combine various types of data members and member functions into a single unit.

Generally, in c# the encapsulation is used to prevent alteration of code (data) accidentally from the outside of functions. In c#, by defining the class fields with properties we can protect the data from accidental corruption.

If we define class fields with properties, then the encapsulated class won't allow us to access the fields directly instead, we need to use getter and setter functions to read or write data based on our requirements.

Following is the example of defining an **encapsulation** class using properties with **get** and **set** accessors.

```
class User
    {
        private string location;
        private string name;
        public string Location
        {
            get
            {
                return location;
            }
            set
            {
                location = value;
            }
```

	`` } public string Name { get { return name; } set { name = value; } } } ``` If you observe the above code, we defined variables with private access modifiers and exposing those variables in a public way by using properties **get** and **set** accessors. In case, if you want to make any modifications to the defined variables, then we can make it by using properties with **get** and **set** accessors.	
b	List and explain the property restrictions in C# encapsulation ANS: • We can assign a value through a property only after class has been initialized. • Property can contain at most one get and one set accessors. • Property cannot contain other methods, fields or properties. • get and set accessors cannot take any parameters	5M
c	Define Indexer. Demonstrate the use of Indexers in C# with programming example ANS: • Indexer can be thought of as a smart Array. • Property is a Smart Field. • Property encapsulates a single value in a class. • Indexer Encapsulates a set of values. • Introduce indexer with this keyword. • Specify type of value returned by indexer. • Specify the type of value to use as index into the indexer between Square brackets. ``` using System; namespace IndexerApplication { class IndexedNames { private string[] namelist = new string[size]; static public int size = 10; ```	7M

```
      public IndexedNames() {
         for (int i = 0; i < size; i++) {
            namelist[i] = "N. A.";
         }
      }
      public string this[int index] {
         get {
            string tmp;

            if( index >= 0 && index <= size-1 )
              {
                tmp = namelist[index];
               }
              else
              {
                tmp = "";
            }

            return ( tmp );
              }
         set
              {
            if( index >= 0 && index <= size-1 ) {
               namelist[index] = value;
            }
         }
      }

      public int this[string name] {
          get
               {
            int index = 0;

            while(index < size) {
               if (namelist[index] == name) {
                return index;
               }
               index++;
            }
            return index;
         }
      }
```

```
        static void Main(string[] args) {
            IndexedNames names = new IndexedNames();
            names[0] = "Zara";
            names[1] = "Riz";
            names[2] = "Nuha";
            names[3] = "Asif";
            names[4] = "Davinder";
            names[5] = "Sunil";
            names[6] = "Rubic";

            //using the first indexer with int parameter
            for (int i = 0; i < IndexedNames.size; i++) {
               Console.WriteLine(names[i]);
            }

            //using the second indexer with the string
parameter
            Console.WriteLine(names["Nuha"]);

        }
    }
}
```

8 a

Define generic. Write a C# program for swapping of two numbers using generic method. 6M

ANS:

- Arrays are strong type-:
  - no need of boxing and unboxing.(Advantage)
  - Fixed Length(Disadvantage)
- Array list and hash tables:
  -Resizable(Advantage)
  -they take only objects, hence boxing and un boxing required (Disadvantage)
- Hence, we need strong typed and also flexible.
- Thus require Generics
- Generics helps us to create flexible strong typed collection.
- Generics separate logic from the datatypes.
- This increases reusability and better maintenance of code

Advantages:

- **Reusability:** You can use a single generic type definition for multiple purposes in the same code without any alterations. For example, you can create a generic method to add two numbers. This method can be used to add two integers as well as two floats without any modification in the code.

<table>
<tr><td></td><td>

Type Safety: Generic data types provide better type safety, especially in the case of collections. When using generics you need to define the type of objects to be passed to a collection. This helps the compiler to ensure that only those object types that are defined in the definition can be passed to the collection.
Performance: Generic types provide better performance as compared to normal system types because they reduce the need for boxing, unboxing, and typecasting of variables or objects.

</td><td></td></tr>
<tr><td>b</td><td>Define Binary tree. Build a binary class by using generics
A binary tree is defined as a tree where each node can have no more than two children. By limiting the number of children to 2, we can write efficient programs for inserting data, deleting data, and searching for data in a binary tree.</td><td>5M</td></tr>
<tr><td>c</td><td>Define collection class. List different collection classes and explain any one in detail.
ANS:
Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.
Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.
List of collection Classes:
<table>
<tr><th>CLASS NAME</th><th>DESCRIPTION</th></tr>
<tr><td>Dictionary<TKey,TValue></td><td>It stores key/value pairs and provides functionality similar to that found in the non-generic Hashtable class.</td></tr>
<tr><td>List<T></td><td>It is a dynamic array that provides functionality similar to that found in the non-generic ArrayList class.</td></tr>
<tr><td>Queue<T></td><td>A first-in, first-out list and provides functionality similar to that found in the non-generic</td></tr>
</table>
</td><td>9M</td></tr>
</table>

	Queue class.		
SortedList<TKey,TValue>	It is a sorted list of key/value pairs and provides functionality similar to that found in the non-generic SortedList class.		
Stack<T>	It is a first-in, last-out list and provides functionality similar to that found in the non-generic Stack class.		
HashSet<T>	It is an unordered collection of the unique elements. It prevent duplicates from being inserted in the collection.		
LinkedList<T>	It allows fast inserting and removing of elements. It implements a classic linked list.		

Example List Collections:

```
// C# program to illustrate the concept
// of generic collection using List<T>
using System;
using System.Collections.Generic;

class Geeks {

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

        // Creating a List of integers
        List<int> mylist = new List<int>();

        // adding items in mylist
        for (int j = 5; j < 10; j++) {
            mylist.Add(j * 3);
        }

        // Displaying items of mylist
        // by using foreach loop
        foreach(int items in mylist)
        {
```

```
            Console.WriteLine(items);
        }
    }
}
```

Output:

```
15

18

21

24

27
```

	Module 5	
9.a	Explain implementation of an enumerator by using iterator ANS: n the previous article we have discussed on Enumerable classes and enumerators and the difference between IEnumerator and IEnumerable Interface in C#. You can understand Enumerators **here**. C# language provides a construct for creating Enumerators and Enumerables in simpler way and that is Iterator.By using Iterators, compiler will create Enumerators and Enumerables for you. You can use the enumerators and enumerables generated by iterators wherever you would use manually coded enumerators orenumerables.Iterators require the Sy	6M

```
// C# program to illustrate the concept
// of iterator using list collection
using System;
using System.Collections.Generic;

 class GFG {

    public static IEnumerable<string> GetMyList()
    {
        // Creating and adding elements in list
        List<string> my_list = new List<string>() {
                        "Cat", "Goat", "Dog", "Cow" };



        // Iterating the elements of my_list
        foreach(var items in my_list)
        {
            // Returning the element after every iteration
            yield return items;
        }
    }
```

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

        // Storing the elements of GetMyList
        IEnumerable<string> my_slist = GetMyList();

        // Display the elements return from iteration
        foreach(var i in my_slist)
        {
            Console.WriteLine(i);
        }
    }
}
```

Output:

```
Cat

Goat

Dog

Cow
```

b

Define Delegate. Explain the use of Delegate in C# with an programming example

6M

A delegate is an object which refers to a method or you can say it is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++. It provides a way which tells which method is to be called when an event is triggered.

For example, if you click an *Button* on a form (Windows Form application), the program would call a specific method. In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.

Important Points About Delegates:

- Provides a good way to encapsulate the methods.
- Delegates are the library class in System namespace.
- These are the type-safe pointer of any method.
- Delegates are mainly used in implementing the call-back methods and events.
- Delegates can be chained together as two or more methods can be called on a single event.
- It doesn't care about the class of the object that it references.
- Delegates can also be used in "anonymous methods" invocation.
- Anonymous Methods(C# 2.0) and Lambda expressions(C# 3.0) are compiled to delegate types in certain contexts. Sometimes, these features together are known as anonymous functions.

Declaration of Delegates

Delegate type can be declared using the **delegate** keyword. Once a delegate is

declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

Syntax:

```
[modifier] delegate [return_type] [delegate_name]
([parameter_list]);
```

modifier: It is the required modifier which defines the access of delegate and it is optional to use.
delegate: It is the keyword which is used to define the delegate.
return_type: It is the type of value returned by the methods which the delegate will be going to call. It can be void. A method must have the same return type as the delegate.
delegate_name: It is the user-defined name or identifier for the delegate.
parameter_list: This contains the parameters which are required by the method when called through the delegate.

Example:

```
// C# program to illustrate the use of Delegates
using System;
namespace GeeksForGeeks {

// declare class "Geeks"
class Geeks {

// Declaring the delegates
// Here return type and parameter type should
// be same as the return type and parameter type
// of the two methods
// "addnum" and "subnum" are two delegate names
public delegate void addnum(int a, int b);
public delegate void subnum(int a, int b);

    // method "sum"
    public void sum(int a, int b)
    {
        Console.WriteLine("(100 + 40) = {0}", a + b);
    }

    // method "subtract"
    public void subtract(int a, int b)
    {
        Console.WriteLine("(100 - 60) = {0}", a - b);
    }

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

    // creating object "obj" of class "Geeks"
    Geeks obj = new Geeks();
```

```
    // creating object of delegate, name as "del_obj1"
    // for method "sum" and "del_obj2" for method "subtract" &
    // pass the parameter as the two methods by class object "obj"
    // instantiating the delegates
    addnum del_obj1 = new addnum(obj.sum);
    subnum del_obj2 = new subnum(obj.subtract);

    // pass the values to the methods by delegate object
    del_obj1(100, 40);
    del_obj2(100, 60);

    // These can be written as using
    // "Invoke" method
    // del_obj1.Invoke(100, 40);
    // del_obj2.Invoke(100, 60);
}
}
}
```

Output:

```
(100 + 40) = 140

(100 - 60) = 40
```

c

Explain declaring, subscribing, unsubsring and raising with respect to an event.

8M

ANS:

n event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the *event sender*. The event sender doesn't know which object or method will receive (handle) the events it raises. The event is typically a member of the event sender; for example, the Click event is a member of the Button class, and the PropertyChanged event is a member of the class that implements the INotifyPropertyChanged interface.

To define an event, you use the C# event or the Visual Basic Event keyword in the signature of your event class, and specify the type of delegate for the event. Delegates are described in the next section.

Typically, to raise an event, you add a method that is marked as `protected` and `virtual` (in C#) or `Protected` and `Overridable` (in Visual Basic). Name this method `On`*EventName*; for example, `OnDataReceived`.

	The method should take one parameter that specifies an event data object, which is an object of type EventArgs or a derived type. You provide this method to enable derived classes to override the logic for raising the event. A derived class should always call the `On`*EventName* method of the base class to ensure that registered delegates receive the event. The following example shows how to declare an event named `ThresholdReached`. The event is associated with the EventHandler delegate and raised in a method named `OnThresholdReached`.	
10 a	Define LINQ. Explain LINQ to selecting, filtering and ordering data with an example ANS: A query is an expression which is used to recover data from the data source. Generally, queries are expressed in some specialized language. Different types of languages are developed to access the different type of data sources like SQL for a relational database, XQuery for XML, etc. So, every time the developer needs to learn different type of languages for a different type of data sources, this situation leads to the developer to develop a language through which they can access any type of data source with the help of a single language. So this requirement is fulfilled by the LINQ query, using LINQ query you can access any type of data source like XML document, SQL database, ADO.NET dataset, etc. provided by the LINQ provider. In LINQ query, the query always returns the result as an object, which allows you to use the object-oriented approach on the result and not to worry about transforming different data format into an object. **rderBy in Method Syntax:** OrderBy operator in method syntax is overloaded in two different types: • **OrderBy<TSource, TKey>(IEnumerable<TSource>, Func<TSource, TKey>):** This method sort the items of the given sequence in ascending order according to the key. • **OrderBy<TSource, TKey>(IEnumerable<TSource>, Func<TSource, TKey>, IComparer<TKey>):** This method sort the items of the given sequence in ascending order by using a specified comparer. It present in both the Queryable and Enumerable class. And support method syntax in both C# and VB.NET languages. As shown in the below example: **Example:** filter_none	10M

edit
play_arrow
brightness_4

```
// C# program to print the salary of
// the employees in ascending
using System;
using System.Linq;
using System.Collections.Generic;

// Employee details
public class Employee {

    public int emp_id
    {
        get;
        set;
    }

    public string emp_name
    {
        get;
        set;
    }

    public string emp_gender
    {
        get;
        set;
    }

    public string emp_hire_date
    {
        get;
        set;
    }

    public int emp_salary
    {
        get;
        set;
    }
}

public class GFG {

    // Main method
    static public void Main()
    {
        List<Employee> emp = new List<Employee>() {

            new Employee() {emp_id = 209, emp_name = "Anjita", emp
                                        emp_hire_date = "12/3/2017",
```

```
            new Employee() {emp_id = 210, emp_name = "Soniya", emp_gender = "Female",
                                    emp_hire_date = "22/4/2018", emp_salary = 30000}

            new Employee() {emp_id = 211, emp_name = "Rohit", emp_gender = "Male",
                                  emp_hire_date = "3/5/2016", emp_salary = 40000},

            new Employee() {emp_id = 212, emp_name = "Supriya", emp_gender = "Female"
                                    emp_hire_date = "4/8/2017", emp_salary = 80000

            new Employee() {emp_id = 213, emp_name = "Anil", emp_gender = "Male",
                                 emp_hire_date = "12/1/2016", emp_salary = 60000},

            new Employee() {emp_id = 214, emp_name = "Anju", emp_gender = "Female",
                                  emp_hire_date = "17/6/2015", emp_salary = 50000},
        };

        // Print the salary of the employees
        // in ascending order using the
        // OrderBy operator
        var res = emp.OrderBy(a => a.emp_salary);

        Console.WriteLine("Salary of the employees: ");

        foreach(var val in res)
        {
            Console.WriteLine(val.emp_salary);
        }
    }
}
```

Output:

```
Salary of the employees:
20000
30000
40000
50000
60000
80000
```

b	Explain operator overloading constraints. Write a C# program for operator + overloading. ANS: The concept of overloading a function can also be applied to operators. Operator overloading gives the ability to use the same operator to do various operations. It provides additional capabilities to C# operators when they are applied to user-defined data types. It enables to make user-defined implementations of various operations	10M

where one or both of the operands are of a user-defined class. Only the predefined set of C# operators can be overloaded. To make operations on a user-defined data type is not as simple as the operations on a built-in data type. To use operators with user-defined data types, they need to be overloaded according to a programmer's requirement. An operator can be overloaded by defining a function to it. The function of the operator is declared by using the **operator keyword**.

edit
play_arrow
brightness_4

```
// C# program to illustrate the
// unary operator overloading
using System;
namespace Calculator {

class Calculator {

    public int number1 , number2;
    public Calculator(int num1 , int num2)
    {
        number1 = num1;
        number2 = num2;
    }

// Function to perform operation
// By changing sign of integers
public static Calculator operator -(Calculator c1)
{
    c1.number1 = -c1.number1;
    c1.number2 = -c1.number2;
    return c1;
}

// Function to print the numbers
public void Print()
{
    Console.WriteLine ("Number1 = " + number1);
    Console.WriteLine ("Number2 = " + number2);
}
}

class EntryPoint
{

    // Driver Code
    static void Main(String []args)
    {

        // using overloaded - operator
        // with the class object
```

```
        Calculator calc = new Calculator(15, -25);

        calc = -calc;

        // To display the result
        calc.Print();
    }
}
}
```

Output :

```
Number1 = -15

Number2 = 25
```