WEB TECHNOLOGY

UNIT 4

Multiple Choice Questions:

1. C# is a?
   A. General-purpose programming language
   B. object-oriented programming language
   C. modern programming language
   D. All of the above

2. C# developed by?
   A. IBM
   B. Google
   C. Microsoft
   D. Facebook

3. CLI in C# Stands for?
   A. Common Language Infrastructure
   B. Code Language Infrastructure
   C. Computer Language Infrastructure
   D. C# Language Infrastructure

4. C# has strong resemblance with?
   A. **C**
   B. C++
   C. Java
   D. Python

5. Which of the following is correct about C#?

   A - It is component oriented.
   B - It can be compiled on a variety of computer platforms.
   C - It is a part of .Net Framework.
   D - All of the above

6. Which of the following converts a type to a small floating-point number in C#?

   A - ToInt64
   B - ToSbyte
   C - ToSingle
   D - ToInt32

7. Which of the following converts a type to an unsigned long type in C#?
   A - ToType
   B - ToUInt16
   C - ToUInt32
   D - ToString

8. Which of the following access specifier in C# allows a class to hide its member variables and member functions from other functions and objects?

   A - Public
   B - Private
   C - Protected
   D - Internal

9. Which of the following property of Array class in C# checks whether the Array is readonly?

   A - IsFixedSize
   B - IsReadOnly
   C - Length
   D - None of the above.

10. Which of the following is the default access specifier of a class member variable?

    A - Private
    B - Public
    C - Protected
    D - Internal

11. Which of the following preprocessor directive defines a sequence of characters as symbol in C#?

    A - define
    B - undef
    C - region
    D - endregion

12. Which of the following is true about System.ApplicationException class in C#?

    A - The System.ApplicationException class supports exceptions generated by application programs.
    B - Exceptions defined by the programmers should derive from this class.
    C - Both of the above.
    D - None of the above.

13. Which of the following variable types can be assigned a value directly in C#?

    A - Value types
    B - Reference types
    C - Pointer types
    D - All of the above.

14. Which of the following is correct about value type variables in C#?
    A - The value types directly contain data.
    B - int, char, and float, which stores numbers, alphabets, and floating point numbers, respectively are value types.
    C - When you declare an int type, the system allocates memory to store the value.
    D - All of the above.

15. Which of the following operator casts without raising an exception if the cast fails in C#?

    A - ?:
    B - is
    C - as
    D - *

16. Which of the following not true about C#?
    A. It is component oriented
    B. It is a unstructured language
    C. It is easy to learn
    D. It is a part of .Net Framework.

17. The first line of the program is?
    A. using System
    B. namespace
    C. using namespace
    D. None of the above

18. A namespace is a collection of classes.
    A. TRUE
    B. FALSE
    C. Can be true or false
    D. Can not say

19. In C#, Save the file using?
    A. .c extension
    B. .csharp extension
    C. .net extension
    D. .cs extension

20. Number of digits upto which precision value of float data type is valid?
    A. Upto 6 digit
    B. Upto 7 digit
    C. Upto 8 digit
    D. Upto 9 digit

21. Select the two types of threads mentioned in the concept of multithreading?
    a. Foreground
    b. Background
    c. Only foreground
    d. Both foreground and background

22. Choose the wrong statement about properties used in C#.Net?

A. Each property consists of accessor as getting and set.
B. A property cannot be either read or write-only.
C. Properties can be used to store and retrieve values to and from the data members of a class.
D. Properties are like actual methods that work like data members.

23. Which of the following keywords is used to refer base class constructor to subclass constructor?
A. this
B. static
C. base
D. extend

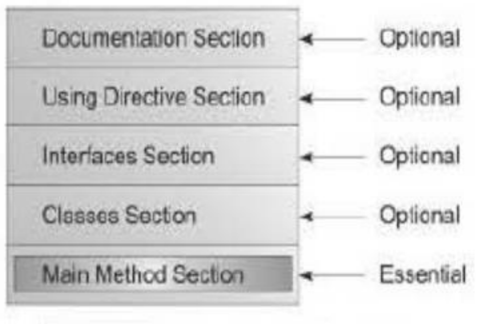
24. Struct's data members are ___ by default.
A. Protected
B. Public
C. Private
D. Default

25. The data members of a class by default are?

A. protected, public
B. private, public
C. private
D. public

Long Answer Questions:

1. Explain the structure of C# program.

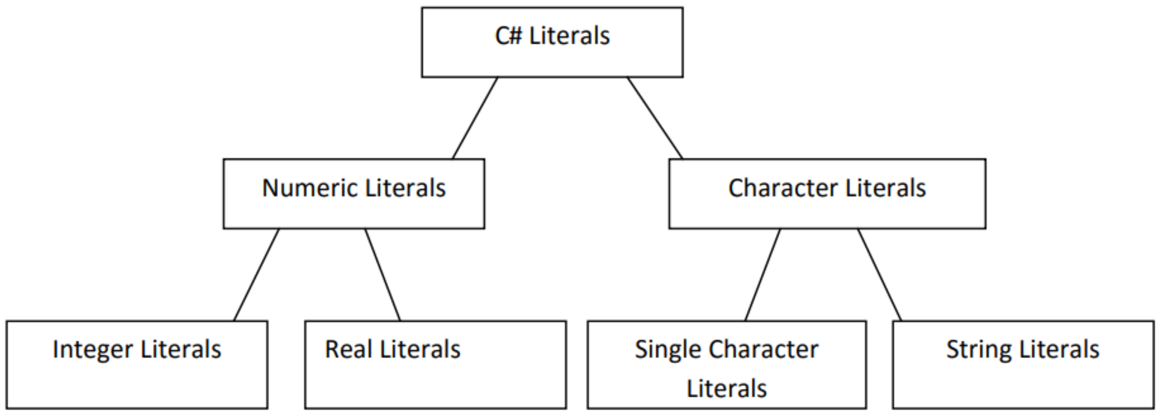


• . The documentation section consists of a set of comments giving the name of the program, the author, date and other details, which the programmer (or other users) may like to use at a later stage.
• The using directive section will include all those namespaces that contain classes required by the application. using directives tell the compiler to look in the namespace specified for these unresolved classes.
• An interface is similar to a class but contains only abstract members. Interfaces are used when we want to implement the concept of multiple inheritance in a program.
• A C# program may contain multiple class definitions. Classes are the primary and essential elements of a C# program. These classes are used to map the objects of real□world problems. The number of classes depends on the complexity of the problem.
• Since every C# application program requires a Main method as its starting point, the class containing the Main is the essential part of the program. A simple C# program may contain only this part. The Main method creates objects of various classes and establishes communications between them.
• On reaching the end of Main, the program terminates and the control passes back to the operating system.

2. How does C# differ from Java?
Like C#, Java was also derived from C++ and therefore they have similar roots. Moreover, C# was developed by Microsoft as an alternative to Java for web programming. C# has borrowed many good features from Java, which has already become a popular Internet language. However, there exist a number of differences between C# and Java:
    - Although C# uses .NET runtime that is similar to Java runtime, the C# compiler produces an executable code.
    - C# has more primitive data types.
    - Unlike Java, all C# data types are objects.
    - Arrays are declared differently in C#.
    - Java uses static final to declare a class constant while C# uses const.
    - C# supports the struct type and Java does not.
    - Java does not provide for operator overloading.
    - In Java, class members are virtual by default and a method having the same name in a derived class overrides the base class member. In C#, the base member is required to have the virtual keyword and the derived member is required to use the override keyword.

• In Java, parameters are always passed by value. C# allows parameters to be passed by reference by using the ref keyword.
• C# adds internal, a new accessibility modifier. Members with internal accessibility can be accessed from other classes within the same project, but not from outside the project

3. Explain about various literals supported by C#?



Literals are value constants assigned to variables in a program. C# supports types of literals
An **Integer literal** refers to a sequence of digits. There are two types of integers, namely, decimal integers and hexadecimal integers.
Real literals are numbers containing fractions. Such numbers are called real (or floating point)
numbers.
Boolean Literals
There are two Boolean literal values
• true
• false
They are used as values of relational expressions.
Single Character Literals
A single-character literal (or simply character constant) contains a single character enclosed within a pair of single quote marks.
String literals
A string literal is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces.
Backslash Character literal
C# supports some special backslash character constants that are used in output methods. For example, the symbol \n stands for a new-line character. A list of such backs lash character literals is given below

4. Explain about type conversion in C# with example.
The process of converting data of one type to another is called type conversion.
In C#, type conversions take place in two ways
• Implicit conversions
• Explicit conversions
Implicit conversions - Implicit conversion is a process of performing automatic conversion

between primitive data types without the knowledge of the user, and without any loss of data during conversion.

Explicit conversions - Explicit conversion allows conversion from any numeric type to any other (for which no implicit conversion is available) by using two mechanisms:

1. Value casting or Typecasting
2. Boxing and Unboxing

Value casting, also known as Typecasting (or simply 'casting'), can be used to change the data of
one primitive type to another compatible primitive type. Suppose we have a long data type value,
and an expression is expecting an int data type value, then we could typecast a long value into int
value as follows.
long numberl = 23844L;
int number2 = (int) numberl; // casting long data type value to int data type value.

5. Explain the arithmetic operators available in C# with example.
   All the basic arithmetic operators are supported by C#. These operators can manipulate with any built in data types supported by C#. The various operators are tabulated as follows:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

Integer Arithmetic: When both the operands in the expression are integers, then the operation is known as integer arithmetic. For example if the values of a=14 and b=4 then a+b=18 , a-b=10, a*b=56 , a/b=3, a%b=2 During modulo division, the sign of the result is the sign of the first operand. E.g. -14%3=-2 , 14%-3=2

Real Arithmetic:

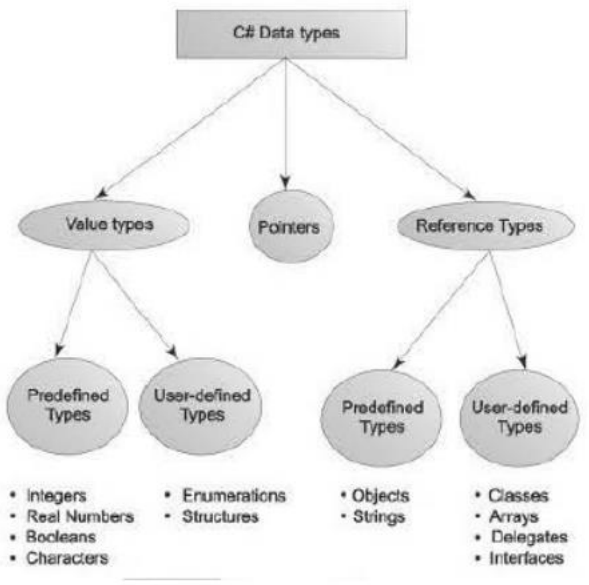
An arithmetic operation involving only real operands is called real arithmetic. A real
value can assume value either in decimal or exponential notation. % operator cannot be used with real operands
E.g. If a=2.4 and b=1.2 then
a+b=3.6 , a-b=1.2, a*b=2.88 a/b=2.0

Mixed mode Arithmetic:

When one of the operands is real and the other is an integer, the expression is called
mixed mode arithmetic expression. The result is always a real number
E.g. : 15/10.0=1.5 , 15/10=1.5

6. Explain C# type system.

   Every variable in C# is associated with a data type. Data types specify the size and type of values that can be stored. C# is a language rich in its data types. The variety available allows the
   programmer to select the type appropriate to the needs of the application.
   The types in C# are primarily divided into two categories:
   • Value types
   • Reference types



   Value types (which are of fixed length) are stored on the stack. and when a value of available is assigned to another variable, the value is actually copied. This means that two identical copies of the value are available in memory.
   Reference types (which are of variable length) are stored on the heap, and when an assignment between two reference variables occurs, only the reference is copied; the actual value remains in the same memory location. This means that there are two references to a single value. A third category of types called pointers is available for use only in unsafe code.

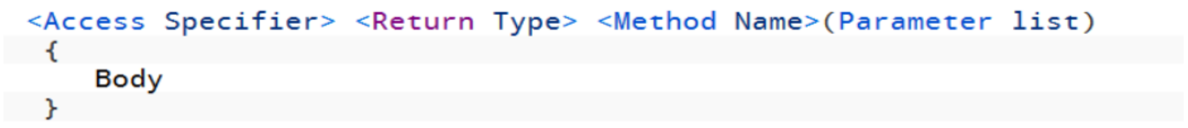
7. What is the difference between BOXING and UNBOXING?
   Boxing
   C# Type System contains three Types , they are Value Types , Reference Types and Pointer Types. C# allows us to convert a Value Type to a Reference Type, and back again to Value Types. The operation of Converting a Value Type to a Reference Type is called Boxing.
    1: int Val = 1;
    2: Object Obj = Val; //Boxing
   The first line we created a Value Type Val and assigned a value to Val. The second line, we created an instance of Object Obj and assign the value of Val to Obj. From the above operation (Object Obj = i) we saw converting a value of a Value Type into a value of a corresponding
   Reference Type . These types of operation is called Boxing.
   UnBoxing

Unboxing is the process of converting the object type back to the value type. Remember that, we can only unbox a variable that has previously been boxed.

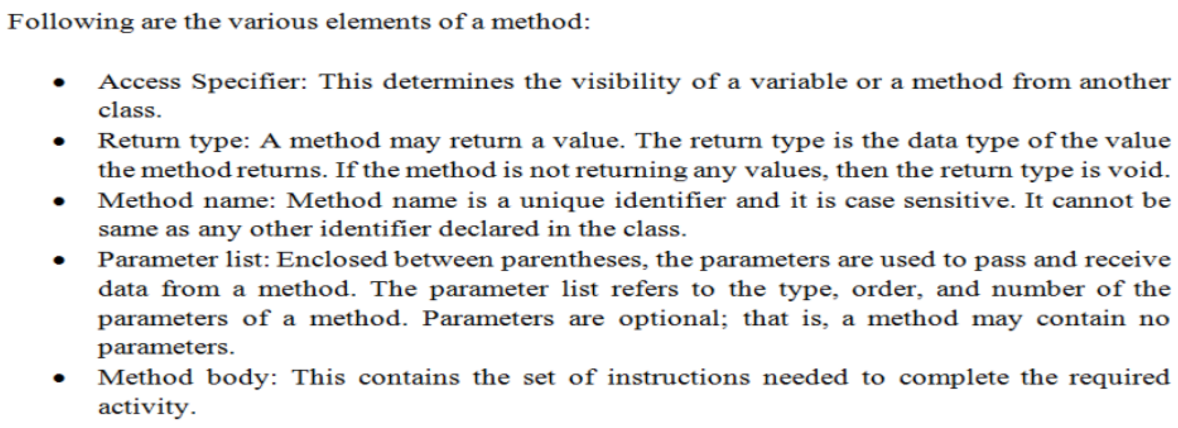
```
1: int Val = 1;
2: Object Obj = Val; //Boxing
3: int i = (int)Obj; //Unboxing
```

The first two line shows how to Box a Value Type. The next line (int i = (int) Obj) shows extracts the Value Type from the Object. That is converting a value of a Reference Type into a value of a Value Type. This operation is called Unboxing.
Boxing and unboxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed, also the cast required for unboxing is also expensive computationally.

8. Explain with example , how methods are declared in C# .
Method is the building block of object-oriented programming. It combines related code together and makes program easier. In C# method declaration, you can declare method by following way:



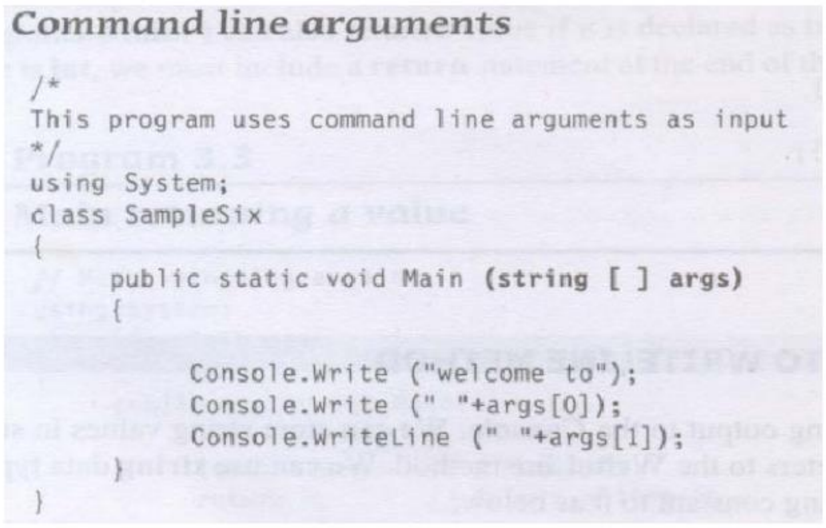
```
<Access Specifier> <Return Type> <Method Name>(Parameter list)
 {
    Body
 }
```



Following are the various elements of a method:

- Access Specifier: This determines the visibility of a variable or a method from another class.
- Return type: A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- Method name: Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- Parameter list: Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- Method body: This contains the set of instructions needed to complete the required activity.

9. What are command line arguments? Explain with example
Command line arguments are used to provide input at the time of execution. They are the parameters supplied to the Main method at the time invoking it for execution.



Command line arguments

```
/*
This program uses command line arguments as input
*/
using System;
class SampleSix
{
    public static void Main (string [ ] args)
    {

        Console.Write ("welcome to");
        Console.Write (" "+args[0]);
        Console.WriteLine (" "+args[1]);
    }
}
```

Main is declared with a parameter args. The parameter args is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements. We can access the array elements by using a subscript like args[0], args[1] and so on.

For example, consider the command line:

Samp1eSi x C sharp

This command line contains two arguments which are assigned to the array args as follows:

C ------ args [0]

sharp -------- args[1]

Difference between the Write and WriteLine ( ) method is that Write () does not cause a line break and therefore the next output statement is printed on the same line.

10. What are Method parameters? Explain the different types.

The invocation involves not only passing the values into the method but also getting back the results from the method. For managing the process of passing values and getting back the results,

C# employs four kinds of parameters.

- Value parameters
- Reference parameters
- Output parameters
- Parameter arrays

Value parameters are used for passing parameters into methods by value. On the other hand reference parameters are used to pass parameters into methods by reference. Output parameters, as the name implies, are used to pass results back from a method. Parameter arrays are used in a method definition to enable it to receive variable number of arguments when called.

Value Parameters By default, method parameters are passed by value. When a method is invoked, the values of actual parameters are assigned to the corresponding formal parameters. The values of the value parameters can be changed within the method. The value of the actual parameter that is passed by value to a method is not changed by any changes made to the corresponding formal parameter within the body of the method. This is because the methods refer to only copies of those variables when they are passed by value.

Reference Parameters

Default behaviour of methods in C# is pass by value. We can, however, force the value parameters to be passed by reference. To do this, we use the ref keyword. A parameter declared with the ref modifier is a reference parameter.

Example:

```
void Modify (ref int x)
```

Here, x is declared as a reference parameter.

Unlike a value parameter, a reference parameter does not create a new storage location. Instead, it presents the same storage location as the actual parameter used in the method invocation.

Example:

```
void Modify ( ref int x )
{
 x+=10;
}
int m = 5;
Modify ( ref m ); // pass by reference
```

Reference parameters are used in situations where we would like to change the values of variables in the calling method. When we pass arguments by reference, the 'formal' arguments in the called method become aliases to the 'actual" arguments in the calling method. This means that when the method is working with its own arguments, it is actually working with the original data.

The Output Parameters

Output parameters are used to pass results back to the calling method. This is achieved by declaring the parameters with an out keyword. Similar to a reference parameter, an output parameter does not create a new storage location. Instead, it becomes an alias to the parameter in the calling method. When a formal parameter is declared as out, the corresponding actual parameter in the calling method must also be declared as out.

Parameter Arrays In C#, we can define methods that can handle variable number of arguments using what are known as parameter arrays. Parameter arrays are declared using the keyword params. Example:

```
void Functionl (Params int [ ] x )
{
--------
}
```

Here, x has been declared as a parameter array. Note that parameter arrays must be one□dimensional arrays. A parameter may be a part of a formal parameter list and in such cases, it must be the last parameter.

The method Functionl defined above can be invoked in two ways:

• Using int type an-ay as a value parameter. Example: Functionl(a);
 Here, a is an array of type int

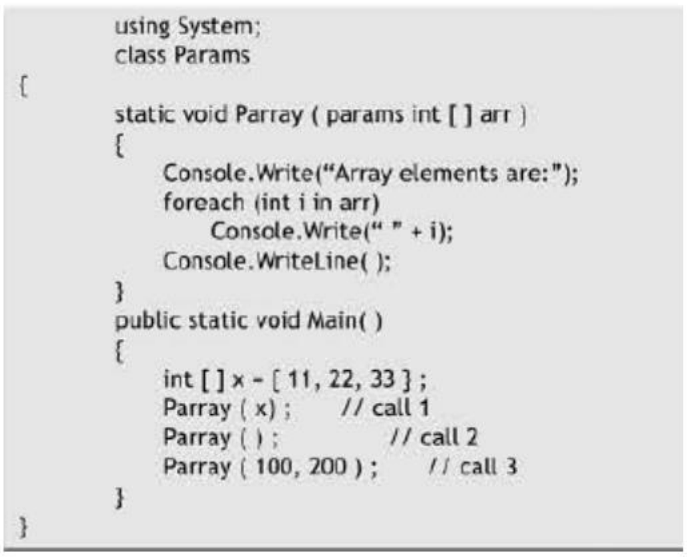
• Using zero or more int type arguments for the parameter array. Example: Function (10, 20);

The second invocation creates an int type array with two elements 10 and 20 and passes the newly created array as the actual argument to the method.

11. What are Parameter Arrays? Explain with example.
    In C#, we can define methods that can handle variable number of arguments using what are known as parameter arrays. Parameter arrays are declared using the keyword params. Example:
    void Functionl (Params int [ ] x)
    {

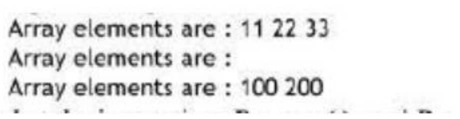
    }
    Here, x has been declared as a parameter array. Note that parameter arrays must be one□dimensional arrays. A parameter may be a part of a formal parameter list and in such cases, it must be the last parameter.
    The method Functionl defined above can be invoked in two ways:
    - Using int type an-ay as a value parameter. Example: Functionl(a);
      Here, a is an array of type int
    - Using zero or more int type arguments for the parameter array. Example: Function (10, 20);

    The second invocation creates an int type array with two elements 10 and 20 and passes the newly created array as the actual argument to the method.



```
          using System;
          class Params
{
          static void Parray ( params int [ ] arr )
          {
               Console.Write("Array elements are:");
               foreach (int i in arr)
                    Console.Write(" " + i);
               Console.WriteLine( );
          }
          public static void Main( )
          {
               int [ ] x - [ 11, 22, 33 ] ;
               Parray ( x) ;       // call 1
               Parray ( ) ;              // call 2
               Parray ( 100, 200 ) ;       // call 3
          }
}
```

Output



```
Array elements are : 11 22 33
Array elements are :
Array elements are : 100 200
```

12. Explain Overloaded Constructor in C#?
    It is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, C# matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism. We can extend the concept of method overloading to provide more than one constructor to a class.

To create an overloaded constructor method, all we have to do is to provide several different constructor definitions with different parameter lists. The difference may be in either the number or type of arguments. That is, each parameter list should be unique. Here is an example of creating an overloaded constructor

```
class Room
{
      public double length ;
      public double breadth ;

      public Room(double x, double y)           // constructor1
      {
            length = x ;
            breadth = y ;
      }

      public Room(double x)                     // constructor2
      {
            length = breadth = x ;
      }

      public int Area( )
      {
            return (length * breadth) ;
      }
}
```

Here we are overloading the constructor method Room( ). An object representing a rectangular room will be created as
Room rooml = new Room(25.0,15.0); //using constructor!
On the other hand, if the room is square, then we may create the corresponding object as
Room room2 = new Room(20.0); // using constructor2

13. What are the Differences between Interface and class.
An interface in C# is a reference type. It is basically a kind of class with some differences.
Major differences include:
• All the members of an interface are implicitly public and abstract.
• An interface cannot contain constant fields, constructors and destructors.
• Its members cannot be declared static.
• Since the methods in an interface are abstract, they do not include implementation code.
• An interface can inherit multiple interfaces.

14. What is C# interfaces?
C# does not support multiple inheritance. That is, classes in C# cannot have more than one superclass. For instance is not permitted in C#.
An interface in C# is a reference type.
Defining an Interface
An interface can contain one or more methods, properties, indexers and events but none of them are implemented in the interface itself. It is the responsibility of the class that implements the interface to define the code for implementation of these members.
The syntax for defining an interface is very similar to that used for defining a class. The general form of an interface definition is:

```
interface InterfaceName
{
     Member declarations;
}
```

Here, interface is the keyword and InterfaceName is a valid C# identifier. Remember declarations will contain only a list of members without implementation code.

Extending an interface

Like classes, interfaces can also be extended. That is, an interface can be sub interfaced from other interfaces. The new sub interface will inherit all the members of the super interface in the manner similar to subclasses. This is achieved as follows:

```
interface name2 : name1
{

     Members of name2
}
```

Implementing Interfaces

Interfaces are used as 'superclasses' whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done as follows:

```
class classname : interfacename
{
     body of classname
}
```

Here the class classname 'implements' the interface interfacename. A more general form of implementation may look like this:

```
class classname : superclass, interface1, interface2. . . .
{
     body of classname
}
```