Q2 (a) what is a CLR? What are the responsibilities of CLR?

Answer
Full form of **CLR** is Common Language Runtime and it forms the heart of the .NET framework. All languages have runtime and it's the responsibility of the runtime to take care of the code execution of the program. For example VC++ has MSCRT40.dll, java has Java Virtual Machine etc. similarly .NET has CLR. Following are the responsibilities of CLR

Garbage Collection: CLR automatically manages memory thus eliminating memory leaks. When objects are not referred GC automatically releases those memories thus providing efficient memory management.

Code Access Security: CAS grants rights to program depending on the security configuration of the machine. Example the program has rights to edit or create a new file but the security configuration of the machine does not allow the program to delete a file. CAS will take care that the code runs under the environment of machines security configuration.

Code Verification: this ensures proper code execution and type safety while the code runs. It prevents the source code to perform illegal operation such as accessing invalid memory locations etc.

IL (Intermediate language)-to-native translators and optimizer's: CLR uses JIT and compiles the IL code to machine code and then executes.CLR also determines depending on platform what is optimized way of running the IL code.

Q2 (b) How does C# differ from java?

Answer Text Book 1, Section 1.7 of Page no. 8

Q3 (a) Write a C# program that will print prime numbers upto 100.

Answer

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

namespace Primenumber
{
   class Class1
   {
```

```
    static void Main(string[] args)
    {

        bool current = false;
        int j;

        Console.WriteLine("Prime numbers upto 100");

       // int num = Int32.Parse(Console.ReadLine());

        for (int i = 2; i <= 100; i++)   //2 is the first prime number.
        //I set i to 2 beacuse it has to print 2 firstly...
        {
            for (j = 2; j < i; j++)
            {
                if (i % j == 0)//Controls i is prime number or not...
                {
                    current = true;
                    break;//breaks for not controlling anymore...
                }
            }
            if (current == false)
                Console.Write("{0} ", j);//if i is prime number, print it...
            else
                current = false;
        }

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

Q3 (b) Explain arithmetic operators with an example in C#. Discuss the operator precedence and associatively.

Answer

Arithmetic operators: The operators **+, -, *, /,** and **%** are used to perform arithmetic calculations. So these operators are known as arithmetic operators. These can operate on any built-in numeric data type. We cannot use these operators on Boolean types.
+ Operator: used to perform addition or known as unary plus.
- Operator: used to perform subtraction or unary minus.
* Operator: used to perform multiplication

/ Operator: used to perform division & obtain quotient
% operator: module division which is used to obtain the remainder after division.
There are three types of arithmetic:

1. Integer arithmetic: When both the operands in a single arithmetic expression such as a + b are integers, the expression is called on integer expression, and the operation is called integer arithmetic. It always yield an integer value.

   e.g. If a and b are integers, then for a=14, b=4, we have

   a + b = 18

   a - b = 10

   a * b = 56

   a / b = 3 (decimal part truncated)

   a % b = 2 (remainder of integer division)

2. Real Arithmetic: An arithmetic operation involving only real operands in called real arithmetic. A real operand may assume values either in decimal or exponential notation.

   E.g.

   if a=20.5 , b=6.4

   a + b = 26.9

   a - b = 14.1

   a * b = 131.2

   a / b = 3.203125

   a % b = 1.3

3. Mixed Mode Arithmetic: When one of the operands is real and other is integer, the expression is called mixed- mode arithmetic expression. If either operand is of the real type, then the other operand is converted to real and real arithmetic is performed.

   E.g: 15 / 10.0 produces result 1.5

   Wheares

   15 / 10 produces result 1.

Operator precedence and Associativity:

Operator	Description	Associativity	Precedence
* / %	Multiplication Division modulus	Left to Right	1
+ -	Addition Subtraction	Left to Right	2

Q.4 (a) Difference between Array list and List. How we can insert and remove elements with the insert and remove methods in Array list.

Answer

ArrayList -

1) Namespace System.Collections contain ArrayList (This namespace is added by default when we we creat any aspx page in C#)

2) Create ArrayList:

ArrayList stringArrayList = new ArrayList();

Here we dont need to specify object type arraylist going to contain,store different type of objects/items/

3) In ArrayList each item is stored as an Object so while reteriving we can get object only.

4) It is like Array of Objects.

List<T> -

1) Namespace System.Collections.Generic List<T> (we need to add namespace if we want to use Generic Types)

2) Create List<T>:

List<string> stringList = new List<string>();

i.e.

List<type> nameOfList = new List<type>(); & type means object type which List<T> going to hold.

3) In List<T> , it can hold/contain only type of object which is mentioned while initialising the List<T>

Just like in above example stringList will only contain object of type string, the type supplied as generic parameter.

4) It is newly added in .Net 2.0 & onwards, fast processing no need of casting explicitly.

Insert and remove methods in ArrayList

```
using System;

  using System.Collections;


  class Program
  {
     static void Main()
     {
```

```
//
// Create an ArrayList with three strings.
//
ArrayList list = new ArrayList();
list.Add("Dot");
list.Add("Net");
list.Add("Perls");
//
// Remove middle element in ArrayList.
//
list.RemoveAt(1); // It becomes [Dot, Perls]
//
// Insert word at beginning of ArrayList.
//
list.Insert(0, "Carrot"); // It becomes [Carrot, Dot, Perls]
//
// Remove first two words from ArrayList.
//
list.RemoveRange(0, 2);
//
// Display the result ArrayList.
//
foreach (string value in list)
{
   Console.WriteLine(value); // <-- "Perls"
}
```

```
    }
  }
```

Output

Perls

Q 4(b) what is the method parameter? What are its types? Give examples of passing the parameter by value and by reference.

Answer

Method Parameter: A method invocation creates a copy specific to that invocation of the formal parameters and local variables of that method. The actual argument list of the invocation assigns values or variable references to the newly created formal parameters within the body of a method, formal parameters can be used like any other variables.
The invocation involves not only passing the values into the method but also getting back the result from the method. There are four types of parameters:

1. Value parameters
2. Reference parameters
3. Output parameters
4. 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 method methods by reference.
Output parameters: are used to pass results back from a method.
Parameter arrays: are used a method definite on to enable it to receive variable no. of arguments when called.
Pass By Value: A parameter declared with no modifier is passed by value and is called a value parameter is passed by value and is called a value parameter. The values of value parameter can be changed within the method. The value of actual parameter that is passed by value is not changed by any changes made to 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.

E.g:

```
Using system;
    Class PassByValue
    {
     Static void change (int m)
     {
       m=m+10; // value of m is changed
}
Public static void main()
{
Int x=100;
Change(x);
Console.writeline("x= " +x);
}
}
```

Output:
X=100

Pass by reference: It is usd when we would like to change the values of variables in the calling method.

Eg:

```
Class passbyref
{
  Static void swap (ref int x, ref int y)
    {
      Int temp = x;
      x = y;
      y = temp;
     }
Public static void main ( )
{
      Int m = 100; int = n = 200;
      Console writeline ("m ="+m);
      Console writeline ("n ="+n);
      Swap (ref m, ref n);
      Console writeline ("m ="+m);
      Console writeline ("n ="+n);
   }
}
```

Q5(a) Write a program in C# that stores a list of countries in a string array, sorts the array into reverse alphabetical order and then print the list.

Answer Page no. 206 of Text Book

Q.5 b What do you mean by structures? Explain nested structure using suitable example. Also explain significant differences between classes and structures.

Answer

Structure: A struct provides a unique way of packing together data of different types. It is a convenient tool for handling a group of logically related data items. It creates a template that may be used to define its data properties. Once the structure type has been defined, we can create variables of that type using declarations that are similar to the built-in type declaration.

The simple form of a struct definition is;

```
Struct    struct _name
       {
          Data member1;
          Datamember2;
              ……
              ……
        }
```

Eg:

```
    Struct student
   {
        Public string name;
        Public int roll number;
        Public double totalmarks;
     }
```

Nested structure:

The declaration of structs nested inside other structs is known as nested structures.

Declaration:

```
                Struct struct _name
                {
                  Data member 1;
                  Datamember 2;
                  ……
                Public struct  struct _name1
                    {
                       Data member 3;
                       ….
                       }
```

We can also wee struct variables as members of another struct. This implies nesting of references to structs.

```
Struct m
 {
    Public int x;
     }
  Struct N
 {
   Public mm; // object of m
   Public int y;
   }
 ……
…….
 N n;
n.m.x = 100;
n.y = 200;
```

Difference between classes and structures:

Category	Classes	Structures
1.Data type	Reference type and therefore stored on the heap.	Value type and therefore stored on the stack. Behave like simple data type.
2.Inheritance	Support inheritance	Do not support inheritance.
3.Default values	Default value of a class type is null	Default value is the value produced by 'zeroing out' the fields of the struct
4.Field initialization	Permit	Do not permit
5.constructurs	Permit declaration of paramterless constructors	Do not permit declaration of parameterless constructors
6.destructors	Supported	Not supported
7.Assignment	Copies the reference	Copies the value.

Q6 (a) what do you understand by inheritance and its types? Explain difference between abstract class and abstract method.

Answer

Inheritance: The mechanism of designing or constructing one class from another or deriving a new class from an old one is called inheritance (or derivation). The old class is referred to as the base class and the new class is called the derived class or subclass.

It may be achieved in two different forms:

1. Classical form
2. Containment form

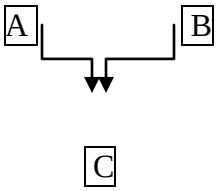
1. Classical inheritance: It represents a kind of relationship between two classes. Let us consider two classes A and B. we can create a class hierarchy such that B is derived from A. It has is-a relationship.

simple inheritance

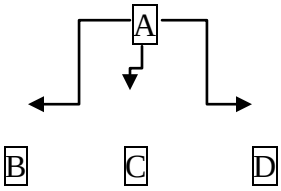
The classical inheritance may be implemented in different combinations, they include:

- Single inheritance: having only one base class and one subclass.

- Multiple inheritance: having several base classes.



- Hierarchical inheritance: one base class and many subclasses



- Multilevel inheritance: derived from a derived class.

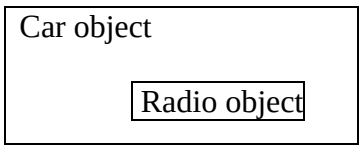
A Super class

B Intermediate super class

C Subclass

2. Containment inheritance: In this the object a is contained in the object b. this relationship between a and b is referred to as 'has-a' relationship. The outer class B which contains the inner class A is termed the 'parent' class and contained class A is termed a 'child' class.

S



that

i.e. a car has-a radio.

Abstract Class: The abstract is a modifier and when used to be declare a class indicates the class cannot be instantiated. Only its derived classes (that are not marked abstract) can be instantiated.

Eg:

```
Abstract class base
    {
      …….
      }
  Class derived : base
```

```
{      …
    }
  …
  …
 Base b1; // cannot be instantiated
Derived d1;
```

Abstract class cannot be instantiated directly. It can have abstract members and w cannot create objects of base type but we can derive its subclasses which can be instantiated.

Abstract methods:

When an instance method declaration includes the modifier abstract, the method is said to be an abstract method. It does not have method body.

Eg:

Public abstract void draw (int x, int y);

It can be declare only in abstract classes. It cannot take either static or virtual modifies. Its implementation must be provided in a non-abstract derived classes by overriding the method.

Q 6(b) what is sealed class? Why we need it? And explain sealed method also.

Answer

Sealed classes: Sometimes, we, may like to prevent a class being further subclassed for securily reasons. A class that cannot be subclassed is called a saled class. It is achieved by using the modifier sealed as:

```
Sealed class Aclass
{
  Datamember1;
 ……
  }
Sealed class bclass : Someclass
 {  ……
    …….
     }
```

Any attempt to inherit these classes will cause an error and the compiler will not allow it. Declaring a class sealed prevents any unwanted extensions to the class. It also allows the compiler to perform some optimization when a methods of a sealed class is invoked. It cannot also be an abstract class.

Sealed method: A sealed method is used to override an inheritance virtual method with the some signature i.e. the sealed modifier is always used in combination with the override modifier.

Eg:

```
Class A
  {
     Public virtual void fun ( )
         {  ….
            }
   }
Class B: A
{
    Public sealed override void fun ( )
      {  …
         }

  }
```

Q7 (a) Why do we need interfaces in C#. How can we implement interface in C#?

Answer

As c# could not overlook the importance of multiple inheritances. A large no. of real-life application requires the use of multiple inheritances. So it provides an approach known an interface to support the concept of multiple inheritances i.e. we can implement more than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

- All the members of an interface are implicitly public and abstract
- Its members cannot b declared static.
- It cannot contain constant fields, constructors and destructors.
- An interface can inherit multiple interchanges

Implementing interface:

Interfaces are used as ‘superclass’ whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface.

```
Class classname   :   interfacename
{
  Body of classname
   }
```

Here class classname 'implements' the interface interfacename.

General form:

```
Class classname: superclass interface1, interface2……
 {
   Body of classname
    }
```

When a class inherits from a superclass, the name of each interface to be implemented must appear after the superclass name. Example:

```
Class A: B, I1, I2 …
  {
      Body
   }
```

Where B is a base class and I1 I2 are interfaces.

Q7 (b) what is operator overloading? How to overload binary operator in C#? Give C# program to illustrate.

Answer

Operator overloading: The ability to provide the operators with a special meaning is known as operator overloading. It gives us syntactical convenience; it also helps us greatly to generate more readable and intuitive code in a no. of situations

General form:

```
Public static return_val operator op (argument list)
{
   Method body // task defined
 }
```

The key features of operator methods are:

- They must be defined as public and static.
- The return value is the type that we get when we use this operator.

- The arglist is the list of arguments passed.The no. of arguments will be one for the unary operators and two for binary operators
- In the case of unary operators, the argument must be the same type as that of the enclosing class or struct.
- In the case of binary operators, the first argument must be of same type as that of enclosing class or struct and second may be of any type.

Overloading binary operators:

The steps are:

- Create a class (or struct) that defines the data type that is to be used.
- Declare the operator method operator op( ) using public and static modifiers
- Define the body of operator method to implement the required operation.
  E.g.

```
Class complex
    {
        Double x;
        Double y;
Public complex ( )
{
  }
 Public complex (double real, double imag)
 {
    x = real;
    y = imag;
   }
Public static complex operator + (complex c1. Complex c2)
{
```

```
Complex c3= new complex ( );
C3.x = c1.x + c2.y;
C3.y = c1.y + c2.x;
Return (c3);
 }
Public void display ( )
  {
     Console. write (x);
     Console. Write ("+j"+y);
        Console. Writeline ( );
       }
    }
Class complex test
{
     Public static void main ( )
      {
          Complex a,b,c;
          a = new complex (2.5, 3.5);
          b = new complex (1.6, 2.7);
          c = a + b;
         console. write ("a=");
          a.  Display( );
          Console. write ("b=");
```

```
    b. Display ( );

    Console. write ("c=")
    c. display( );

   }
}
```

Q8 (a) How to declare a delegates? Explain multicast delegates and events.

Answer
Delegates: It is a class type object and is used to invoke a method that has been encapsulated into it at the time of its creation.

Delegate declaration:
General form: modifier delegate return type delegate-name (parameters); where, delegate is the keyword that signifies that the declaration that represents a class type derived from system Delegate.

- return-type indicates the return-type of delegate
- parameters identifies signature of delegate.
- deleagate -name is the name that will be used to instantiate delegate objects.
- modifier controls the accessibility of the delegate.It is optional.

Multicast delegates:

It is possible for certain delegates to hold and invoke multiple methods. Such delegates are called multicast delegates. Also called combinable delegate which must satisfy following condition:

- The return type of the delegate must be void
- None of the parameters of the delegate type can be declared as output parameters, using out keyword.

E.g.

```
using system;
    delegate void mdelegate( ); // delegate declaration
class dm
  {
      static pubic void display ( )
```

```
        {
              console. writeline ("new delhi");
             }
    static public void print ( )
  {
    console writeline ("new York");
     }
}
Class mtest
  {
     public static void main ( )
      {
           M delegate m1 = new m delegate (dm. display);
           M delegate m2 = new m delegate (dm print);
            M delegate m3 = m1+m2;
            M delegate m4 = m2 + m1;
            M delegate m5 = m3 – m2;
            M3 ( );
             M4 ( );
            }
      }
```

Events:

An event is a delegate type class member that is used by the object or class to provide a notification to other objects than an event has occurred .

Event declaration:

```
          modifier event type event-name;
E.g.
 using system;
     public delegate void Edelegate (string str);
     Class event class
        {
            public event Edelegate status; // event declaration
            pubic void trigger event ( )
             {
                if (status! = null)
               status ("Event triggered");
                }
         }
    Class EventTest
     {
```

```
        public static void main ( )
        {
            Event class ec = new Eventclass ( );
            Event test et = new Event test ( );
              ec. Status + = new E delegate (et. Event catch);
              ec. Trigger event ( );
          }
       public void Event catch (string str)
       {
              console. writeline (str);
          }
}
```

Q 8(b) Write a program to display the following output:

```
          *
        *  *  *
      *  *  *  *  *
    *  *  *  *  *  *  *
```

Answer

C# Program to print pyramid:

```
using System;

namespace MyApp
{
   class triangle_class
   {
      static void Main(string[] args)
      {
         int number = 12;

         for (int i = 1; i <= number; i++)
         {
            for (int j = 1; j <= number - i; j++)
               Console.Write(" ");
            for (int k = 1; k <= i; k++)
               Console.Write(" *");
            Console.WriteLine("");
         }
```

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

Q9 (a) what is Multi-threading? What is suspend and resume in threading? How a thread is created?

Answer

Multithreading: The execution of a c# program starts with a single thread called the main thread that is automatically run by CLR and the operating system from the main thread; we can create other threads for performing desired tasks in the program. The process of developing a program for execution with multiple threads is called multithreaded programming and the process of execution is called multithreading.

Some instances when multithreading can prove beneficial are:
- When a program needs to perform two or more tasks simultaneously.
- By having distinct threads for each tasks, the program can achieve execution of the tasks in a faster and more efficient manner.
- When a program has to wait for some event or user input or data to be read/written from a file.

The classes and interfaces contained in the 'System. Threading' namespace allow us to perform multithreading. We can use the methods and properties contained in these classes to perform tasks such as synchronizing the activities of a thread and creating a thread.

The following are the important classes contained in system. Threading namespace:
-Thread
-Thread pool
-Monitor
-Mutex

Creation of a thread:

A thread can be created by using the constructor of the thread class. We need to pass the thread start delegate to the thread class constructor along with the name of the method from which the execution should start.
The thread start delegate is defined as:
public delegate void thread start ( ) ;
To create a new thread, we use the foil.

Thread thread_name= new Thread (new Threadstart (method_name) ;
Here , thread name represents the name of new thread and method_name is the name of the method from which the execution starts.
E.g.
Thread t1= new Thread (new Threadstart (First):
if the method from which execution needs to start is defined in a class other than the class in which the thread is created then we must use an object of that class to access the method.

Suspend and Resume in threading:
Suspend is to suspend a thread and resume is to resume a thread, which has been suspended earlier. It is similar to sleep and interrupt. Suspend allow you to block a thread until another thread calls thread resume. The difference between sleep and suspend is that the latter does not immediately place a thread in the wait state. The thread does not suspend until the .NET runtime determines that it is in a safe place to suspend it sleep will immediately place a thread in a wait state.

Q 9.b. Explain any two of the following:
(i) Thread Pool
(ii) Thread synchronization
(iii) Mutex class

Answer
Thread Pool:
It provides a pool of threads that helps us to perform tasks such as processing of asynchronous I/O and waiting on behalf of another method. i.e. perform tasks in the background. A thread pool may contain a no. of threads, each performing a specific task. If all the threads in a thread pool are occupied in performing their tasks then a new task, which need to be processed, waits in a queue until a thread becomes free.
The .NET framework provides a thread pool through the 'Thread Pool' class. We can either implement a custom thread pool in a c# program or use the thread pool provided through the thread pool class. It is easy to implement a thread pool through the thread pool class.
Some of the important methods of Thread Pool class are:

1. Equals - determine whether two thread pool are equal or not.
2. Get type - to obtain the type for the current thread pool.
3. Queue user work item - to allow a method to be queued for execution.
4. Set max threads – to specify the no. of request to the thread pool that can be concurrently active.

5. Set min threads – to specify the no. of idle threads that can be maintained by a thread pool for new requests.

b) Thread synchronization:

Synchronization of thread means coordinating the access to resources for different threads running for execution. To synchronies the threads, we can either block a thread fill the time another thread has completed its tasks or lock access to resource. The thread class provides methods such as 'spin wait' and 'suspend' that help block a thread for a specific time period.
In c#, we can use the lock keyword to lock access to a resource for a specific thread. When the program statement has been executed, the mutually exclusive lock is released. The syntax is:

```
Lock (obj) || obj is object being synchronized
{
    || set of statements to be synchrosied
    }
```

c) **Mutex Class**:
A mutex is a synchronization primitive that helps to perform interprocess synchronization in c# shared resources. When a thread acquires a mutex, another thread, which wants to obtain the mutex, is suspended until the first thread releases the mutex.
The mutex class provides the 'Handle and 'Safe Wait Handle' properties that can be used to retrieve the handle for the operating system.
Some important methods provided by the mutex class are:

1. Close: To release the resources held by an object of the Wait Handle Class.

2. Equals: TO determine whether two mutex are equal or not.

3. Open Existing: TO open an existing mutex

4. Release Mutex: To release a mutex once.

5. Set Access Control: TO set the access control security for a specified mutex.

Text Book

Programming in C# - A Primer, E. Balagurusamy, Second Edition, TMH, 2008