**.NET FRAMEWORK**  
1**. What is .NET Framework?**  
The .NET Framework has two main components: the common language runtime and the .NET Framework class library.  
You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that ensure security and robustness.  
The class library, is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.   
**2. What is CLR, CTS, CLS?**  
The .NET Framework provides a runtime environment called the Common Language Runtime or CLR (similar to the Java Virtual Machine or JVM in Java), which handles the execution of code and provides useful services for the implementation of the program. CLR takes care of code management at program execution and provides various beneficial services such as memory management, thread management, security management, code verification, compilation, and other system services. The managed code that targets CLR benefits from useful features such as cross-language integration, cross-language exception handling, versioning, enhanced security, deployment support, and debugging.   
Common Type System (CTS) describes how types are declared, used and managed in the runtime and facilitates cross-language integration, type safety, and high performance code execution.   
The CLS is simply a specification that defines the rules to support language integration in such a way that programs written in any language, yet can interoperate with one another, taking full advantage of inheritance, polymorphism, exceptions, and other features. These rules and the specification are documented in the ECMA proposed standard document, "Partition I Architecture", http://msdn.microsoft.com/net/ecma/   
3. **What are the new features of Framework 1.1 ?**   
1. Native Support for Developing Mobile [Web Applications](javascript:void(0))   
2. Enable Execution of Windows Forms Assemblies Originating from the Internet  
Assemblies originating from the Internet zone—for example, Microsoft Windows® Forms controls embedded in an Internet-based Web page or Windows Forms assemblies hosted on an Internet Web server and loaded either through the Web browser or programmatically using the System.Reflection.Assembly.LoadFrom() method—now receive sufficient permission to execute in a semi-trusted manner. Default security policy has been changed so that assemblies assigned by the common language runtime (CLR) to the Internet zone code group now receive the constrained permissions associated with the Internet permission set. In the .NET Framework 1.0 Service Pack 1 and Service Pack 2, such applications received the permissions associated with the Nothing permission set and could not execute.   
3. Enable Code Access Security for ASP.NET Applications  
Systems administrators can now use code access security to further lock down the permissions granted to ASP.NET Web applications and Web services. Although the operating system account under which an application runs imposes security restrictions on the application, the code access security system of the CLR can enforce additional restrictions on selected application resources based on policies specified by systems administrators. You can use this feature in a shared server environment (such as an Internet service provider (ISP) hosting multiple Web applications on one server) to isolate separate applications from one another, as well as with stand-alone servers where you want applications to run with the minimum necessary privileges.   
4. Native Support for Communicating with ODBC and Oracle Databases   
5. Unified Programming Model for Smart Client Application Development  
The Microsoft .NET Compact Framework brings the CLR, Windows Forms controls, and other .NET Framework features to small devices. The .NET Compact Framework supports a large subset of the .NET Framework class library optimized for small devices.   
6. Support for IPv6  
The .NET Framework 1.1 supports the emerging update to the Internet Protocol, commonly referred to as IP version 6, or simply IPv6. This protocol is designed to significantly increase the address space used to identify communication endpoints in the Internet to accommodate its ongoing growth.   
http://msdn.microsoft.com/netframework/technologyinfo/Overview/whatsnew.aspx   
4**. Is .NET a runtime service or a development platform?**  
Ans: It's both and actually a lot more. Microsoft .NET includes a new way of delivering software and services to businesses and consumers. A part of Microsoft.NET is the .NET Frameworks. The .NET frameworks SDK consists of two parts: the .NET common language runtime and the .NET class library. In addition, the SDK also includes command-line compilers for C#, C++, JScript, and VB. You use these compilers to build applications and components. These components require the runtime to execute so this is a development platform.   
5**. What is MSIL, IL?**  
When compiling to managed code, the compiler translates your source code into Microsoft intermediate language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Microsoft intermediate language (MSIL) is a language used as the output of a number of compilers and as the input to a just-in-time (JIT) compiler. The common language runtime includes a JIT compiler for converting MSIL to native code.   
6**. Can I write IL programs directly?**  
Yes. Peter Drayton posted this simple example to the DOTNET mailing list:  
.assembly MyAssembly {}  
.class MyApp {  
.method static void Main() {  
.entrypoint  
ldstr "Hello, IL!"  
call void System.Console::WriteLine(class System.Object)  
ret  
}  
}  
Just put this into a file called hello.il, and then run ilasm hello.il. An exe assembly will be generated.  
Can I do things in IL that I can't do in C#?  
Yes. A couple of simple examples are that you can throw exceptions that are not derived from System.Exception, and you can have non-zero-based arrays.   
7**. What is JIT (just in time)? how it works?**  
Before Microsoft intermediate language (MSIL) can be executed, it must be converted by a .NET Framework just-in-time (JIT) compiler to native code, which is CPU-specific code that runs on the same computer architecture as the JIT compiler.   
Rather than using time and memory to convert all the MSIL in a portable executable (PE) file to native code, it converts the MSIL as it is needed during execution and stores the resulting native code so that it is accessible for subsequent calls.  
The runtime supplies another mode of compilation called install-time code generation. The install-time code generation mode converts MSIL to native code just as the regular JIT compiler does, but it converts larger units of code at a time, storing the resulting native code for use when the assembly is subsequently loaded and executed.  
As part of compiling MSIL to native code, code must pass a verification process unless an administrator has established a security policy that allows code to bypass verification. Verification examines MSIL and metadata to find out whether the code can be determined to be type safe, which means that it is known to access only the memory locations it is authorized to access.   
8. **What is strong name?**  
A name that consists of an assembly's identity—its simple text name, version number, and culture information (if provided)—strengthened by a public key and a digital signature generated over the assembly.   
**9. What is portable executable (PE)?**  
The file format defining the structure that all executable files (EXE) and Dynamic Link Libraries (DLL) must use to allow them to be loaded and executed by Windows. PE is derived from the Microsoft Common Object File Format (COFF). The EXE and DLL files created using the .NET Framework obey the PE/COFF formats and also add additional header and data sections to the files that are only used by the CLR. The specification for the PE/COFF file formats is available at http://www.microsoft.com/whdc/hwdev/hardware/pecoffdown.mspx   
**10. What is Event - Delegate?** clear syntax for writing a event delegate  
The event keyword lets you specify a delegate that will be called upon the occurrence of some "event" in your code. The delegate can have one or more associated methods that will be called when your code indicates that the event has occurred. An event in one program can be made available to other programs that target the .NET Framework Common Language Runtime.  
// keyword\_delegate.cs  
// delegate declaration  
delegate void MyDelegate(int i);  
class Program  
 {  
 public static void Main()  
 {  
 TakesADelegate(new MyDelegate(DelegateFunction));  
 }  
 public static void TakesADelegate(MyDelegate SomeFunction)  
 {  
 SomeFunction(21);  
 }  
 public static void DelegateFunction(int i)  
 {  
 System.Console.WriteLine("Called by delegate with number: {0}.", i);  
 }  
}

**25. What is Code Access Security (CAS)?**  
CAS is the part of the .NET security model that determines whether or not a piece of code is allowed to run, and what resources it can use when it is running. For example, it is CAS that will prevent a .NET web applet from formatting your hard disk.   
How does CAS work?   
The CAS security policy revolves around two key concepts - code groups and permissions. Each .NET assembly is a member of a particular code group, and each code group is granted the permissions specified in a named permission set.   
For example, using the default security policy, a control downloaded from a web site belongs to the 'Zone - Internet' code group, which adheres to the permissions defined by the 'Internet' named permission set. (Naturally the 'Internet' named permission set represents a very restrictive range of permissions.)   
Who defines the CAS code groups?   
Microsoft defines some default ones, but you can modify these and even create your own. To see the code groups defined on your system, run 'caspol -lg' from the command-line. On my syystem it looks like this:

Level = Machine  
 Code Groups:  
  
1. All code: Nothing  
1.1. Zone - MyComputer: FullTrust  
 1.1.1. Honor SkipVerification requests: SkipVerification  
 1.2. Zone - Intranet: LocalIntranet  
 1.3. Zone - Internet: Internet  
 1.4. Zone - Untrusted: Nothing  
 1.5. Zone - Trusted: Internet  
 1.6. StrongName - 0024000004800000940000000602000000240000525341310004000003  
000000CFCB3291AA715FE99D40D49040336F9056D7886FED46775BC7BB5430BA4444FEF8348EBD06  
F962F39776AE4DC3B7B04A7FE6F49F25F740423EBF2C0B89698D8D08AC48D69CED0FC8F83B465E0807AC11EC1DCC7D054E807A43336DDE408A5393A48556123272CEEEE72F1660B71927D38561AABF5CAC1DF1734633C602F8F2D5: Everything  
Note the hierarchy of code groups - the top of the hierarchy is the most general ('All code'), which is then sub-divided into several groups, each of which in turn can be sub-divided. Also note that (somewhat counter-intuitively) a sub-group can be associated with a more permissive permission set than its parent.   
How do I define my own code group?   
Use caspol. For example, suppose you trust code from www.mydomain.com and you want it have full access to your system, but you want to keep the default restrictions for all other internet sites. To achieve this, you would add a new code group as a sub-group of the 'Zone - Internet' group, like this:   
caspol -ag 1.3 -site www.mydomain.com FullTrust  
Now if you run caspol -lg you will see that the new group has been added as group 1.3.1:   
...  
1.3. Zone - Internet: Internet  
1.3.1. Site - www.mydomain.com: FullTrust  
...  
Note that the numeric label (1.3.1) is just a caspol invention to make the code groups easy to manipulate from the command-line. The underlying runtime never sees it.  
How do I change the permission set for a code group?   
Use caspol. If you are the machine administrator, you can operate at the 'machine' level - which means not only that the changes you make become the default for the machine, but also that users cannot change the permissions to be more permissive. If you are a normal (non-admin) user you can still modify the permissions, but only to make them more restrictive. For example, to allow intranet code to do what it likes you might do this:   
caspol -cg 1.2 FullTrust  
Note that because this is more permissive than the default policy (on a standard system), you should only do this at the machine level - doing it at the user level will have no effect.   
Can I create my own permission set?  
Yes. Use caspol -ap, specifying an XML file containing the permissions in the permission set. To save you some time, here is a sample file corresponding to the 'Everything' permission set - just edit to suit your needs. When you have edited the sample, add it to the range of available permission sets like this:  
caspol -ap samplepermset.xml  
Then, to apply the permission set to a code group, do something like this:   
caspol -cg 1.3 SamplePermSet (By default, 1.3 is the 'Internet' code group)  
I'm having some trouble with CAS. How can I diagnose my problem?   
Caspol has a couple of options that might help. First, you can ask caspol to tell you what code group an assembly belongs to, using caspol -rsg. Similarly, you can ask what permissions are being applied to a particular assembly using caspol -rsp.   
  
I can't be bothered with all this CAS stuff. Can I turn it off?   
Yes, as long as you are an administrator. Just run:   
caspol -s off  
http://www.codeproject.com/dotnet/UB\_CAS\_NET.asp   
**40. Which namespace is the base class for .net Class library?**  
Ans: system.object   
**41. What are object pooling and connection pooling and difference? Where do we set the Min and Max Pool size for connection pooling?**  
Object pooling is a COM+ service that enables you to reduce the overhead of creating each object from scratch. When an object is activated, it is pulled from the pool. When the object is deactivated, it is placed back into the pool to await the next request. You can configure object pooling by applying the ObjectPoolingAttribute attribute to a class that derives from the System.EnterpriseServices.ServicedComponent class.   
Object pooling lets you control the number of connections you use, as opposed to connection pooling, where you control the maximum number reached.  
Following are important differences between object pooling and connection pooling:   
• Creation. When using connection pooling, creation is on the same thread, so if there is nothing in the pool, a connection is created on your behalf. With object pooling, the pool might decide to create a new object. However, if you have already reached your maximum, it instead gives you the next available object. This is crucial behavior when it takes a long time to create an object, but you do not use it for very long.   
• Enforcement of minimums and maximums. This is not done in connection pooling. The maximum value in object pooling is very important when trying to scale your application. You might need to multiplex thousands of requests to just a few objects. (TPC/C benchmarks rely on this.)   
COM+ object pooling is identical to what is used in .NET Framework managed SQL Client connection pooling. For example, creation is on a different thread and minimums and maximums are enforced.   
42**. What is Application Domain?**  
The primary purpose of the AppDomain is to isolate an application from other applications. Win32 processes provide isolation by having distinct memory address spaces. This is effective, but it is expensive and doesn't scale well. The .NET runtime enforces AppDomain isolation by keeping control over the use of memory - all memory in the AppDomain is managed by the .NET runtime, so the runtime can ensure that AppDomains do not access each other's memory.  
Objects in different application domains communicate either by transporting copies of objects across application domain boundaries, or by using a proxy to exchange messages.  
MarshalByRefObject is the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy. Objects that do not inherit from MarshalByRefObject are implicitly marshal by value. When a remote application references a marshal by value object, a copy of the object is passed across application domain boundaries.  
How does an AppDomain get created?   
AppDomains are usually created by hosts. Examples of hosts are the Windows Shell, ASP.NET and IE. When you run a .NET application from the command-line, the host is the Shell. The Shell creates a new AppDomain for every application.  
AppDomains can also be explicitly created by .NET applications. Here is a C# sample which creates an AppDomain, creates an instance of an object inside it, and then executes one of the object's methods. Note that you must name the executable 'appdomaintest.exe' for this code to work as-is.  
using System;  
 using System.Runtime.Remoting;  
  
 public class CAppDomainInfo : MarshalByRefObject  
 {  
 public string GetAppDomainInfo()

{  
 return "AppDomain = " + AppDomain.CurrentDomain.FriendlyName;  
 }  
 }  
 public class App  
 {  
 public static int Main()  
 {  
 AppDomain ad = AppDomain.CreateDomain( "Andy's new domain", null, null );  
 ObjectHandle oh = ad.CreateInstance( "appdomaintest", "CAppDomainInfo" );  
 CAppDomainInfo adInfo = (CAppDomainInfo)(oh.Unwrap());  
 string info = adInfo.GetAppDomainInfo();  
 Console.WriteLine( "AppDomain info: " + info );  
 return 0;  
 }  
}  
**64. What is serialization in .NET? What are the ways to control serialization?**  
Serialization is the process of converting an object into a stream of bytes. Deserialization is the opposite process of creating an object from a stream of bytes. Serialization/Deserialization is mostly used to transport objects (e.g. during remoting), or to persist objects (e.g. to a file or database).Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.   
• Binary serialization preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application. For example, you can share an object between different applications by serializing it to the clipboard. You can serialize an object to a stream, disk, memory, over the network, and so forth. Remoting uses serialization to pass objects "by value" from one computer or application domain to another.   
• XML serialization serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web. SOAP is an open standard, which makes it an attractive choice.   
There are two separate mechanisms provided by the .NET class library - XmlSerializer and SoapFormatter/BinaryFormatter. Microsoft uses XmlSerializer for Web Services, and uses SoapFormatter/BinaryFormatter for remoting. Both are available for use in your own code.  
Why do I get errors when I try to serialize a Hashtable?  
XmlSerializer will refuse to serialize instances of any class that implements IDictionary, e.g. Hashtable. SoapFormatter and BinaryFormatter do not have this restriction.   
**65. What is exception handling?**  
When an exception occurs, the system searches for the nearest catch clause that can handle the exception, as determined by the run-time type of the exception. First, the current method is searched for a lexically enclosing try statement, and the associated catch clauses of the try statement are considered in order. If that fails, the method that called the current method is searched for a lexically enclosing try statement that encloses the point of the call to the current method. This search continues until a catch clause is found that can handle the current exception, by naming an exception class that is of the same class, or a base class, of the run-time type of the exception being thrown. A catch clause that doesn't name an exception class can handle any exception.  
Once a matching catch clause is found, the system prepares to transfer control to the first statement of the catch clause. Before execution of the catch clause begins, the system first executes, in order, any finally clauses that were associated with try statements more nested that than the one that caught the exception.   
Exceptions that occur during destructor execution are worth special mention. If an exception occurs during destructor execution, and that exception is not caught, then the execution of that destructor is terminated and the destructor of the base class (if any) is called. If there is no base class (as in the case of the object type) or if there is no base class destructor, then the exception is discarded.   
**66. What is Assembly?**Assemblies are the building blocks of .NET Framework applications; they form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.  
Assemblies are a fundamental part of programming with the .NET Framework. An assembly performs the following functions:   
• It contains code that the common language runtime executes. Microsoft intermediate language (MSIL) code in a portable executable (PE) file will not be executed if it does not have an associated assembly manifest. Note that each assembly can have only one entry point (that is, DllMain, WinMain, or Main).   
• It forms a security boundary. An assembly is the unit at which permissions are requested and granted.   
• It forms a type boundary. Every type's identity includes the name of the assembly in which it resides. A type called MyType loaded in the scope of one assembly is not the same as a type called MyType loaded in the scope of another assembly.   
• It forms a reference scope boundary. The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests. It specifies the types and resources that are exposed outside the assembly. The manifest also enumerates other assemblies on which it depends.   
• It forms a version boundary. The assembly is the smallest versionable unit in the common language runtime; all types and resources in the same assembly are versioned as a unit. The assembly's manifest describes the version dependencies you specify for any dependent assemblies.   
• It forms a deployment unit. When an application starts, only the assemblies that the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be kept simple and thin when first downloaded.   
• It is the unit at which side-by-side execution is supported.   
Assemblies can be static or dynamic. Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on). Static assemblies are stored on disk in PE files. You can also use the .NET Framework to create dynamic assemblies, which are run directly from memory and are not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.  
There are several ways to create assemblies. You can use development tools, such as Visual Studio .NET, that you have used in the past to create .dll or .exe files. You can use tools provided in the .NET Framework SDK to create assemblies with modules created in other development environments. You can also use common language runtime APIs, such as Reflection.Emit, to create dynamic assemblies.   
**67. What are the contents of assembly?**  
In general, a static assembly can consist of four elements:   
• The assembly manifest, which contains assembly metadata.   
• Type metadata.   
• Microsoft intermediate language (MSIL) code that implements the types.   
• A set of resources.   
**68. What are the different types of assemblies?**  
Private, Public/Shared, Satellite   
**69. What is the difference between a private assembly and a shared assembly?**   
0. Location and visibility: A private assembly is normally used by a single application, and is stored in the application's directory, or a sub-directory beneath. A shared assembly is normally stored in the global assembly cache, which is a repository of assemblies maintained by the .NET runtime. Shared assemblies are usually libraries of code which many applications will find useful, e.g. the .NET framework classes.  
  
1. Versioning: The runtime enforces versioning constraints only on shared assemblies, not on private assemblies.   
**70. What are Satellite Assemblies? How you will create this? How will you get the different language strings?**   
Satellite assemblies are often used to deploy language-specific resources for an application. These language-specific assemblies work in side-by-side execution because the application has a separate product ID for each language and installs satellite assemblies in a language-specific subdirectory for each language. When uninstalling, the application removes only the satellite assemblies associated with a given language and .NET Framework version. No core .NET Framework files are removed unless the last language for that .NET Framework version is being removed.  
(For example, English and Japanese editions of the .NET Framework version 1.1 share the same core files. The Japanese .NET Framework version 1.1 adds satellite assemblies with localized resources in a \ja subdirectory. An application that supports the .NET Framework version 1.1, regardless of its language, always uses the same core runtime files.)  
http://www.ondotnet.com/lpt/a/2637   
\*\*   
**71. How will u load dynamic assembly? How will create assemblies at run time?**  
\*\*   
**72. What is Assembly manifest?** what all details the assembly manifest will contain?  
Every assembly, whether static or dynamic, contains a collection of data that describes how the elements in the assembly relate to each other. The assembly manifest contains this assembly metadata. An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information.  
It contains Assembly name, Version number, Culture, Strong name information, List of all files in the assembly, Type reference information, Information on referenced assemblies.   
73**. Difference between assembly manifest & metadata?**  
assembly manifest - An integral part of every assembly that renders the assembly self-describing. The assembly manifest contains the assembly's metadata. The manifest establishes the assembly identity, specifies the files that make up the assembly implementation, specifies the types and resources that make up the assembly, itemizes the compile-time dependencies on other assemblies, and specifies the set of permissions required for the assembly to run properly. This information is used at run time to resolve references, enforce version binding policy, and validate the integrity of loaded assemblies. The self-describing nature of assemblies also helps makes zero-impact install and XCOPY deployment feasible.  
metadata - Information that describes every element managed by the common language runtime: an assembly, loadable file, type, method, and so on. This can include information required for debugging and garbage collection, as well as security attributes, marshaling data, extended class and member definitions, version binding, and other information required by the runtime.

**74. What is Global Assembly Cache (GAC) and what is the purpose of it? (How to make an assembly to public?** Steps) How more than one version of an assembly can keep in same place?  
Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer. You should share assemblies by installing them into the global assembly cache only when you need to.  
Steps  
- Create a strong name using sn.exe tool  
eg: sn -k keyPair.snk  
- with in AssemblyInfo.cs add the generated file name   
eg: [assembly: AssemblyKeyFile("abc.snk")]  
- recompile project, then install it to GAC by either  
drag & drop it to assembly folder (C:\WINDOWS\assembly OR C:\WINNT\assembly) (shfusion.dll tool)  
or  
gacutil -i abc.dll   
75. If I have more than one version of one assemblies, then how'll I use old version (how/where to specify version number?)in my application?  
\*\*   
**76. How to find methods of a assembly file (not using ILDASM)**  
Reflection   
**77. What is Garbage Collection in .Net? Garbage collection process?**  
The process of transitively tracing through all pointers to actively used objects in order to locate all objects that can be referenced, and then arranging to reuse any heap memory that was not found during this trace. The common language runtime garbage collector also compacts the memory that is in use to reduce the working space needed for the heap.   
**78. What is Reflection in .NET? Namespace? How will you load an assembly which is not referenced by current assembly?**   
All .NET compilers produce metadata about the types defined in the modules they produce. This metadata is packaged along with the module (modules in turn are packaged together in assemblies), and can be accessed by a mechanism called reflection. The System.Reflection namespace contains classes that can be used to interrogate the types for a module/assembly.  
Using reflection to access .NET metadata is very similar to using ITypeLib/ITypeInfo to access type library data in COM, and it is used for similar purposes - e.g. determining data type sizes for marshaling data across context/process/machine boundaries.  
Reflection can also be used to dynamically invoke methods (see System.Type.InvokeMember), or even create types dynamically at run-time (see System.Reflection.Emit.TypeBuilder).   
**79. What is Custom attribute? How to create? If I'm having custom attribute in an assembly, how to say that name in the code?**   
A: The primary steps to properly design custom attribute classes are as follows:   
. Applying the AttributeUsage Attribute ([AttributeUsage(AttributeTargets.All, Inherited = false, AllowMultiple = true)])   
a. Declaring the attribute. (class public class MyAttribute : System.Attribute { // . . . })   
b. Declaring constructors (public MyAttribute(bool myvalue) { this.myvalue = myvalue; })   
c. Declaring properties   
public bool MyProperty  
{  
get {return this.myvalue;}  
set {this.myvalue = value;}  
}   
The following example demonstrates the basic way of using reflection to get access to custom attributes.   
class MainClass   
{  
public static void Main()  
{  
System.Reflection.MemberInfo info = typeof(MyClass);  
object[] attributes = info.GetCustomAttributes();  
for (int i = 0; i < attributes.Length; i ++)   
{  
System.Console.WriteLine(attributes[i]);  
}  
}  
}   
**80. What is the managed and unmanaged code in .net?**  
The .NET Framework provides a run-time environment called the Common Language Runtime, which manages the execution of code and provides services that make the development process easier. Compilers and tools expose the runtime's functionality and enable you to write code that benefits from this managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code; it benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.   
**81. How do you create threading in .NET? What is the namespace for that?**  
\*\*  
System.Threading.Thread   
**82. Serialize and MarshalByRef?**   
**83. using directive vs using statement**  
You create an instance in a using statement to ensure that Dispose is called on the object when the using statement is exited. A using statement can be exited either when the end of the using statement is reached or if, for example, an exception is thrown and control leaves the statement block before the end of the statement.  
The using directive has two uses:   
• Create an alias for a namespace (a using alias).   
• Permit the use of types in a namespace, such that, you do not have to qualify the use of a type in that namespace (a using directive).   
**84. Describe the Managed Execution Process?**  
The managed execution process includes the following steps:   
0. Choosing a compiler.   
To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.   
1. Compiling your code to Microsoft intermediate language (MSIL).   
Compiling translates your source code into MSIL and generates the required metadata.   
2. Compiling MSIL to native code.   
At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.   
3. Executing your code.   
The common language runtime provides the infrastructure that enables execution to take place as well as a variety of services that can be used during execution.   
**85. What is Active Directory? What is the namespace used to access the Microsoft Active Directories? What are ADSI Directories?**  
Active Directory Service Interfaces (ADSI) is a programmatic interface for Microsoft Windows Active Directory. It enables your applications to interact with diverse directories on a network, using a single interface. Visual Studio .NET and the .NET Framework make it easy to add ADSI functionality with the DirectoryEntry and DirectorySearcher components.  
Using ADSI, you can create applications that perform common administrative tasks, such as backing up databases, accessing printers, and administering user accounts. ADSI makes it possible for you to:   
• Log on once to work with diverse directories. The DirectoryEntry component class provides username and password properties that can be entered at runtime and communicated to the Active Directory object you are binding to.   
• Use a single application programming interface (API) to perform tasks on multiple directory systems by offering the user a variety of protocols to use. The DirectoryServices namespace provides the classes to perform most administrative functions.   
• Perform "rich querying" on directory systems. ADSI technology allows for searching for an object by specifying two query dialects: SQL and LDAP.   
• Access and use a single, hierarchical structure for administering and maintaining diverse and complicated network configurations by accessing an Active Directory tree.   
• Integrate directory information with databases such as SQL Server. The DirectoryEntry path may be used as an ADO.NET connection string provided that it is using the LDAP provider.   
using System.DirectoryServices;   
**86. How Garbage Collector (GC) Works?**  
The methods in this class influence when an object is garbage collected and when resources allocated by an object are released. Properties in this class provide information about the total amount of memory available in the system and the age category, or generation, of memory allocated to an object. Periodically, the garbage collector performs garbage collection to reclaim memory allocated to objects for which there are no valid references. Garbage collection happens automatically when a request for memory cannot be satisfied using available free memory. Alternatively, an application can force garbage collection using the Collect method.  
Garbage collection consists of the following steps:   
0. The garbage collector searches for managed objects that are referenced in managed code.   
1. The garbage collector attempts to finalize objects that are not referenced.   
2. The garbage collector frees objects that are not referenced and reclaims their memory.   
**87. Why do we need to call CG.SupressFinalize?**  
Requests that the system not call the finalizer method for the specified object.   
public static void SuppressFinalize(  
object obj  
); The method removes obj from the set of objects that require finalization. The obj parameter is required to be the caller of this method.  
Objects that implement the IDisposable interface can call this method from the IDisposable.Dispose method to prevent the garbage collector f rom calling Object.Finalize on an object that does not require it.   
**88. What is nmake tool?**  
The Nmake tool (Nmake.exe) is a 32-bit tool that you use to build projects based on commands contained in a .mak file.  
usage : nmake -a all   
**89. What are Namespaces?**  
The namespace keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally-unique types. Even if you do not explicitly declare one, a default namespace is created. This unnamed namespace, sometimes called the global namespace, is present in every file. Any identifier in the global namespace is available for use in a named namespace. Namespaces implicitly have public access and this is not modifiable.   
**90. What is the difference between CONST and READONLY?**  
Both are meant for constant values. A const field can only be initialized at the declaration of the field. A readonly field can be initialized either at the declaration or in a constructor. Therefore, readonly fields can have different values depending on the constructor used.   
readonly int b;  
public X()  
{  
b=1;  
}  
public X(string s)  
{  
b=5;  
}  
public X(string s, int i)  
{  
b=i;  
}  
Also, while a const field is a compile-time constant, the readonly field can be used for runtime constants, as in the following example:  
public static readonly uint l1 = (uint) DateTime.Now.Ticks; (this can't be possible with const)   
**91. What is the difference between ref & out parameters?**  
An argument passed to a ref parameter must first be initialized. Compare this to an out parameter, whose argument does not have to be explicitly initialized before being passed to an out parameter.   
**92. What is the difference between Array and LinkedList?**   
**93. What is the difference between Array and Arraylist?**  
As elements are added to an ArrayList, the capacity is automatically increased as required through reallocation. The capacity can be decreased by calling TrimToSize or by setting the Capacity property explicitly.   
**94. What is Jagged Arrays?**  
A jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array-of-arrays."   
**95. What are indexers?**  
Indexers are similar to properties, except that the get and set accessors of indexers take parameters, while property accessors do not.   
**96. What is Asynchronous call and how it can be implemented using delegates?**   
**97. How to create events for a control? What is custom events? How to create it?**   
**98. If you want to write your own dot net language, what steps you will u take care?**   
**99. Describe the difference between inline and code behind - which is best in a loosely coupled solution?   
100. how dot net compiled code will become platform independent?   
101. without modifying source code if we compile again, will it be generated MSIL again?   
102. C++ & C# differences**  
OOPS  
1. What are the OOPS concepts?  
1) Encapsulation: It is the mechanism that binds together code and data in manipulates, and keeps both safe from outside interference and misuse. In short it isolates a particular code and data from all other codes and data. A well-defined interface controls the access to that particular code and data.  
2) Inheritance: It is the process by which one object acquires the properties of another object. This supports the hierarchical classification. Without the use of hierarchies, each object would need to define all its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. A new sub-class inherits all of the attributes of all of its ancestors.  
3) Polymorphism: It is a feature that allows one interface to be used for general class of actions. The specific action is determined by the exact nature of the situation. In general polymorphism means "one interface, multiple methods", This means that it is possible to design a generic interface to a group of related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler's job to select the specific action (that is, method) as it applies to each situation.   
2. What is the difference between a Struct and a Class?   
o The struct type is suitable for representing lightweight objects such as Point, Rectangle, and Color. Although it is possible to represent a point as a class, a struct is more efficient in some scenarios. For example, if you declare an array of 1000 Point objects, you will allocate additional memory for referencing each object. In this case, the struct is less expensive.   
o When you create a struct object using the new operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the new operator. If you do not use new, the fields will remain unassigned and the object cannot be used until all of the fields are initialized.   
o It is an error to declare a default (parameterless) constructor for a struct. A default constructor is always provided to initialize the struct members to their default values.   
o It is an error to initialize an instance field in a struct.   
o There is no inheritance for structs as there is for classes. A struct cannot inherit from another struct or class, and it cannot be the base of a class. Structs, however, inherit from the base class Object. A struct can implement interfaces, and it does that exactly as classes do.   
o A struct is a value type, while a class is a reference type.   
3. Value type & reference types difference? Example from .NET. Integer & struct are value types or reference types in .NET?  
Most programming languages provide built-in data types, such as integers and floating-point numbers, that are copied when they are passed as arguments (that is, they are passed by value). In the .NET Framework, these are called value types. The runtime supports two kinds of value types:   
• Built-in value types   
The .NET Framework defines built-in value types, such as System.Int32 and System.Boolean, which correspond and are identical to primitive data types used by programming languages.   
• User-defined value types   
Your language will provide ways to define your own value types, which derive from System.ValueType. If you want to define a type representing a value that is small, such as a complex number (using two floating-point numbers), you might choose to define it as a value type because you can pass the value type efficiently by value. If the type you are defining would be more efficiently passed by reference, you should define it as a class instead.   
Variables of reference types, referred to as objects, store references to the actual data. This following are the reference types:   
• class   
• interface   
• delegate   
This following are the built-in reference types:   
• object   
• string   
4. What is Inheritance, Multiple Inheritance, Shared and Repeatable Inheritance?  
\*\*   
5. What is Method overloading?  
Method overloading occurs when a class contains two methods with the same name, but different signatures.   
6. What is Method Overriding? How to override a function in C#?  
Use the override modifier to modify a method, a property, an indexer, or an event. An override method provides a new implementation of a member inherited from a base class. The method overridden by an override declaration is known as the overridden base method. The overridden base method must have the same signature as the override method.  
You cannot override a non-virtual or static method. The overridden base method must be virtual, abstract, or override.   
7. Can we call a base class method without creating instance?  
Its possible If its a static method.  
Its possible by inheriting from that class also.  
Its possible from derived classes using base keyword.   
8. You have one base class virtual function how will call that function from derived class?  
Ans:  
  
 class a  
{  
 public virtual int m()  
 {  
 return 1;  
 }  
}  
 class b:a  
 {  
 public int j()  
 {  
 return m();  
 }  
}

**22. In which cases you use override and new base?**  
Use the new modifier to explicitly hide a member inherited from a base class. To hide an inherited member, declare it in the derived class using the same name, and modify it with the new modifier.   
C# Language features   
**23. What are Sealed Classes in C#?**  
The sealed modifier is used to prevent derivation from a class. A compile-time error occurs if a sealed class is specified as the base class of another class. (A sealed class cannot also be an abstract class)   
**24. What is Polymorphism? How does VB.NET/C# achieve polymorphism?**  
\*\*   
class Token  
{  
public string Display()  
{  
//Implementation goes here  
return "base";  
 }  
 }  
class IdentifierToken:Token  
 {  
 public new string Display() //What is the use of new keyword  
{  
 //Implementation goes here  
return "derive";  
 }  
 }  
 static void Method(Token t)  
 {  
Console.Write(t.Display());  
 }  
public static void Main()  
 {  
IdentifierToken Variable=new IdentifierToken();  
 Method(Variable); //Which Class Method is called here  
Console.ReadLine();  
 }  
**51. For the above code What is the "new" keyword and Which Class Method is called here**  
A: it will call base class Display method   
class Token  
 {  
 public virtual string Display()  
 {  
 //Implementation goes here  
 return "base";  
 }  
 }  
 class IdentifierToken:Token  
 {  
 public override string Display() //What is the use of new keyword  
{  
 //Implementation goes here  
 return "derive";  
 }  
 }  
 static void Method(Token t)  
 {  
 Console.Write(t.Display());  
}  
public static void Main()  
{  
IdentifierToken Variable=new IdentifierToken();  
 Method(Variable); //Which Class Method is called here  
 Console.ReadLine();  
 }  
 A: Derive   
**80. In which Scenario you will go for Interface or Abstract Class?**  
Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation. They are implemented by classes, and defined as separate entities from classes. Even though class inheritance allows your classes to inherit implementation from a base class, it also forces you to make most of your design decisions when the class is first published.  
Abstract classes are useful when creating components because they allow you specify an invariant level of functionality in some methods, but leave the implementation of other methods until a specific implementation of that class is needed. They also version well, because if additional functionality is needed in derived classes, it can be added to the base class without breaking code.   
Interfaces vs. Abstract Classes  
Feature Interface Abstract class  
Multiple inheritance A class may implement several interfaces. A class may extend only one abstract class.   
Default implementation An interface cannot provide any code at all, much less default code. An abstract class can provide complete code, default code, and/or just stubs that have to be overridden.  
Constants Static final constants only, can use them without qualification in classes that implement the interface. On the other paw, these unqualified names pollute the namespace. You can use them and it is not obvious where they are coming from since the qualification is optional. Both instance and static constants are possible. Both static and instance intialiser code are also possible to compute the constants.  
Third party convenience An interface implementation may be added to any existing third party class. A third party class must be rewritten to extend only from the abstract class.  
is-a vs -able or can-do Interfaces are often used to describe the peripheral abilities of a class, not its central identity, e.g. an Automobile class might implement the Recyclable interface, which could apply to many otherwise totally unrelated objects. An abstract class defines the core identity of its descendants. If you defined a Dog abstract class then Damamation descendants are Dogs, they are not merely dogable. Implemented interfaces enumerate the general things a class can do, not the things a class is.   
Plug-in You can write a new replacement module for an interface that contains not one stick of code in common with the existing implementations. When you implement the interface, you start from scratch without any default implementation. You have to obtain your tools from other classes; nothing comes with the interface other than a few constants. This gives you freedom to implement a radically different internal design. You must use the abstract class as-is for the code base, with all its attendant baggage, good or bad. The abstract class author has imposed structure on you. Depending on the cleverness of the author of the abstract class, this may be good or bad. Another issue that's important is what I call "heterogeneous vs. homogeneous." If implementors/subclasses are homogeneous, tend towards an abstract base class. If they are heterogeneous, use an interface. (Now all I have to do is come up with a good definition of hetero/homogeneous in this context.) If the various objects are all of-a-kind, and share a common state and behavior, then tend towards a common base class. If all they share is a set of method signatures, then tend towards an interface.  
Homogeneity If all the various implementations share is the method signatures, then an interface works best. If the various implementations are all of a kind and share a common status and behavior, usually an abstract class works best.  
Maintenance If your client code talks only in terms of an interface, you can easily change the concrete implementation behind it, using a factory method. Just like an interface, if your client code talks only in terms of an abstract class, you can easily change the concrete implementation behind it, using a factory method.  
Speed Slow, requires extra indirection to find the corresponding method in the actual class. Modern JVMs are discovering ways to reduce this speed penalty. Fast  
Terseness The constant declarations in an interface are all presumed public static final, so you may leave that part out. You can't call any methods to compute the initial values of your constants. You need not declare individual methods of an interface abstract. They are all presumed so. You can put shared code into an abstract class, where you cannot into an interface. If interfaces want to share code, you will have to write other bubblegum to arrange that. You may use methods to compute the initial values of your constants and variables, both instance and static. You must declare all the individual methods of an abstract class abstract.  
Adding functionality If you add a new method to an interface, you must track down all implementations of that interface in the universe and provide them with a concrete implementation of that method. If you add a new method to an abstract class, you have the option of providing a default implementation of it. Then all existing code will continue to work without change.  
 see the code  
interface ICommon  
 { int getCommon();  
 }  
interface ICommonImplements1:ICommon  
 {  
 }  
 interface ICommonImplements2:ICommon  
 {  
 }  
public class a:ICommonImplements1,ICommonImplements2 { }

**93. How to implement getCommon method in class a? Are you seeing any problem in the implementation?**  
Ans:   
public class a:ICommonImplements1,ICommonImplements2  
{  
public int getCommon()  
{  
return 1;  
}  
}  
94. interface IWeather  
95. {  
96. void display();  
97. }  
98. public class A:IWeather  
99. {  
100. public void display()  
101. {  
102. MessageBox.Show("A");  
103. }  
104. }  
105. public class B:A  
106. {  
107. }  
108. public class C:B,IWeather  
109. {  
110. public void display()  
111. {  
112. MessageBox.Show("C");  
113. }  
114. }  
115. When I instantiate C.display(), will it work?   
116. interface IPrint  
117. {  
118. string Display();  
119. }  
120. interface IWrite  
121. {  
122. string Display();  
123. }  
124. class PrintDoc:IPrint,IWrite  
125. {  
126. //Here is implementation  
127. }

**how to implement the Display in the class printDoc (How to resolve the naming Conflict)**

A: no naming conflicts  
class PrintDoc:IPrint,IWrite  
{  
public string Display()  
{  
return "s";  
}  
}  
128. interface IList  
129. {  
130. int Count { get; set; }  
131. }  
132. interface ICounter  
133. {  
134. void Count(int i);  
135. }  
136. interface IListCounter: IList, ICounter {}  
137. class C  
138. {  
139. void Test(IListCounter x)  
140. {  
141. x.Count(1); // Error  
142. x.Count = 1; // Error  
143. ((IList)x).Count = 1; // Ok, invokes IList.Count.set  
144. ((ICounter)x).Count(1); // Ok, invokes ICounter.Count  
145. }  
146. }  
**147. Write one code example for compile time binding and one for run time binding? What is early/late binding?**  
An object is early bound when it is assigned to a variable declared to be of a specific object type. Early bound objects allow the compiler to allocate memory and perform other optimizations before an application executes.  
' Create a variable to hold a new object.  
Dim FS As FileStream  
' Assign a new object to the variable.  
FS = New FileStream("C:\tmp.txt", FileMode.Open)  
By contrast, an object is late bound when it is assigned to a variable declared to be of type Object. Objects of this type can hold references to any object, but lack many of the advantages of early-bound objects.  
Dim xlApp As Object  
xlApp = CreateObject("Excel.Application")   
**148. Can you explain what inheritance is and an example of when you might use it?   
149. How can you write a class to restrict that only one object of this class can be created (Singleton class)?   
150. What are the access-specifiers available in c#?**  
Private, Protected, Public, Internal, Protected Internal.   
**151. Explain about Protected and protected internal, “internal” access-specifier?**  
protected - Access is limited to the containing class or types derived from the containing class.   
internal - Access is limited to the current assembly.  
protected internal - Access is limited to the current assembly or types derived from the containing class.

**CONSTRUCTORS**  
**1. Difference between type constructor and instance constructor? What is static constructor, when it will be fired? And what is its use?**  
(Class constructor method is also known as type constructor or type initializer)  
Instance constructor is executed when a new instance of type is created and the class constructor is executed after the type is loaded and before any one of the type members is accessed. (It will get executed only 1st time, when we call any static methods/fields in the same class.) Class constructors are used for static field initialization. Only one class constructor per type is permitted, and it cannot use the vararg (variable argument) calling convention.  
A static constructor is used to initialize a class. It is called automatically to initialize the class before the first instance is created or any static members are referenced.   
**2. What is Private Constructor? and it’s use? Can you create instance of a class which has Private Constructor?**  
A: When a class declares only private instance constructors, it is not possible for classes outside the program to derive from the class or to directly create instances of it. (Except Nested classes)  
Make a constructor private if:  
- You want it to be available only to the class itself. For example, you might have a special constructor used only in the implementation of your class' Clone method.  
- You do not want instances of your component to be created. For example, you may have a class containing nothing but Shared utility functions, and no instance data. Creating instances of the class would waste memory.   
**3. I have 3 overloaded constructors in my class. In order to avoid making instance of the class do I need to make all constructors to private?**  
(yes)   
**4. Overloaded constructor will call default constructor internally?**  
(no)   
**5. What are virtual destructors?**

**6. Destructor and finalize**  
Generally in C++ the destructor is called when objects gets destroyed. And one can explicitly call the destructors in C++. And also the objects are destroyed in reverse order that they are created in. So in C++ you have control over the destructors.  
In C# you can never call them, the reason is one cannot destroy an object. So who has the control over the destructor (in C#)? it's the .Net frameworks Garbage Collector (GC). GC destroys the objects only when necessary. Some situations of necessity are memory is exhausted or user explicitly calls System.GC.Collect() method.  
Points to remember:  
1. Destructors are invoked automatically, and cannot be invoked explicitly.  
2. Destructors cannot be overloaded. Thus, a class can have, at most, one destructor.  
3. Destructors are not inherited. Thus, a class has no destructors other than the one, which may be declared in it.  
4. Destructors cannot be used with structs. They are only used with classes.  
5. An instance becomes eligible for destruction when it is no longer possible for any code to use the instance.   
6. Execution of the destructor for the instance may occur at any time after the instance becomes eligible for destruction.  
7. When an instance is destructed, the destructors in its inheritance chain are called, in order, from most derived to least derived.  
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconfinalizemethodscdestructors.asp   
**7. What is the difference between Finalize and Dispose (Garbage collection)**  
Class instances often encapsulate control over resources that are not managed by the runtime, such as window handles (HWND), database connections, and so on. Therefore, you should provide both an explicit and an implicit way to free those resources. Provide implicit control by implementing the protected Finalize Method on an object (destructor syntax in C# and the Managed Extensions for C++). The garbage collector calls this method at some point after there are no longer any valid references to the object.  
In some cases, you might want to provide programmers using an object with the ability to explicitly release these external resources before the garbage collector frees the object. If an external resource is scarce or expensive, better performance can be achieved if the programmer explicitly releases resources when they are no longer being used. To provide explicit control, implement the Dispose method provided by the IDisposable Interface. The consumer of the object

**8. where you will find polymorphism in .NET?**

1. Virtual override

2.Interfaces

3. reflection(late binding of methods)