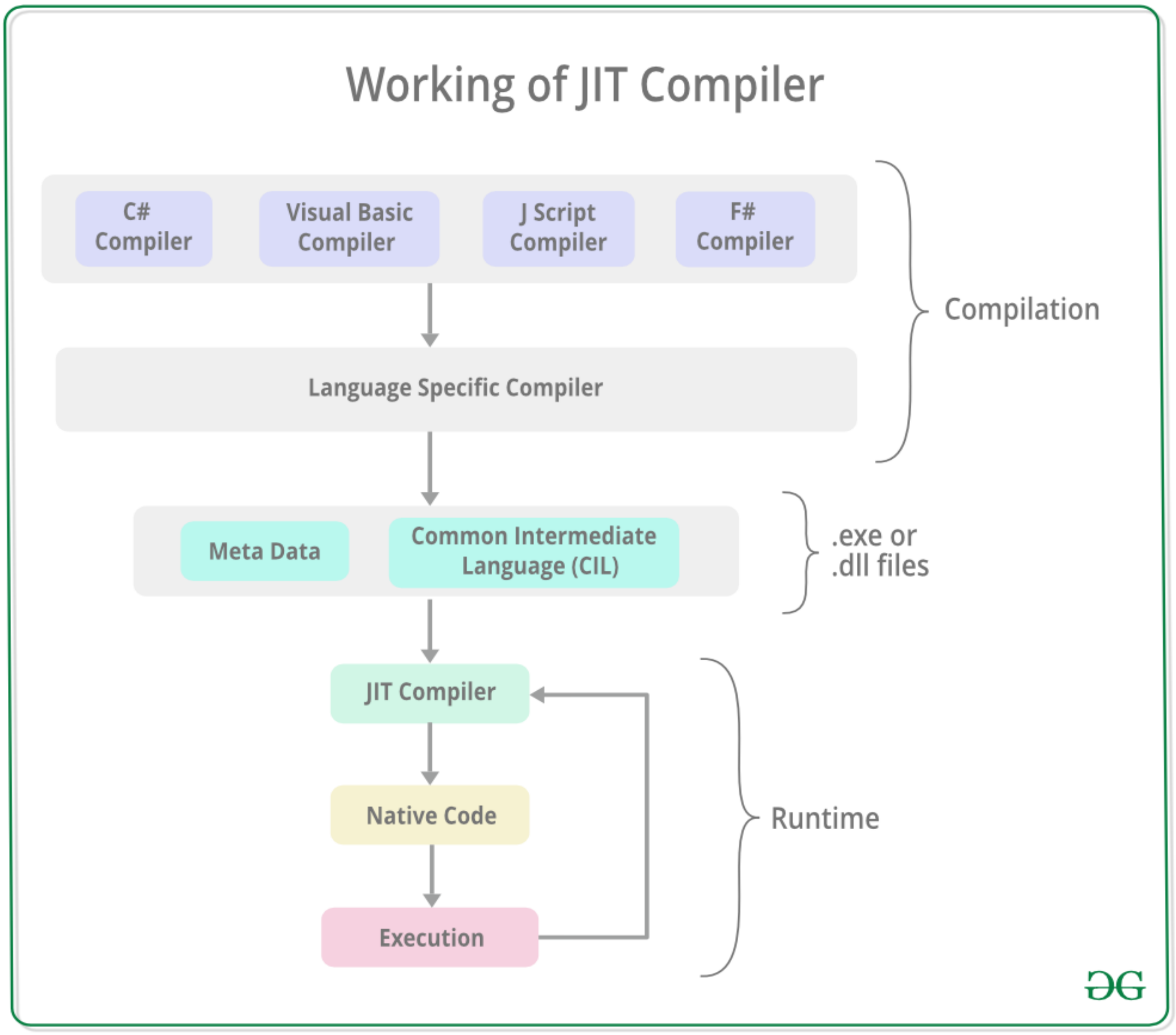
Assignment .Net Framework Intro Morning

1. Demonstrate the process of conversion of Source code into the native machine code in .Net framework with the help of a flowchart.

Ans 1.



2. Explain in detail the CTS and how the .net framework implements CTS.

Ans 2.

i) In Microsoft's .NET Framework, the Common Type System (CTS) is a standard that specifies how type definitions and specific values of types are represented in computer memory.

ii) It is intended to allow programs written in different programming languages to easily share information.

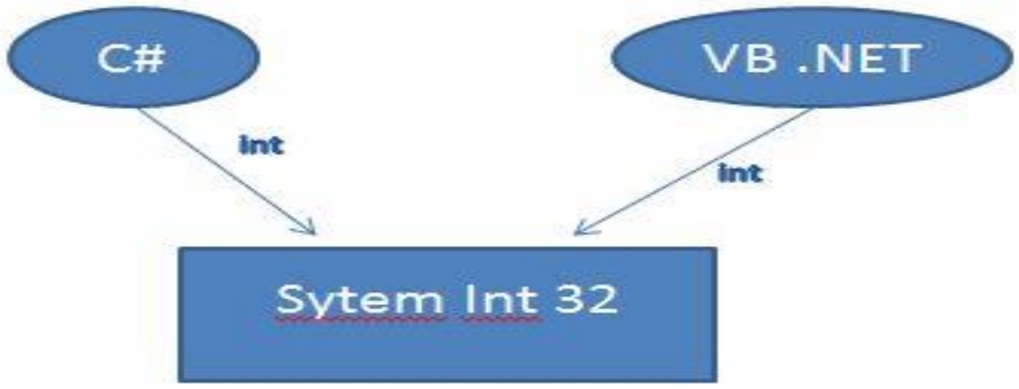
iii) As used in programming languages, a type can be described as a definition of a set of values (for example, "all integers between 0 and 10"), and the allowable operations on those values (for example, addition and subtraction).

Functions of the Common Type System

i) To establish a framework that helps enable cross-language integration, type safety, and high performance code execution.
ii) To provide an object-oriented model that supports the complete implementation of many programming languages.
iii) To define rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
iv) The CTS also defines the rules that ensures that the data types of objects written in various languages are able to interact with each other.
v) The CTS also specifies the rules for type visibility and access to the members of a type, i.e. the CTS establishes the rules by which assemblies form scope for a type, and the Common Language Runtime enforces the visibility rules.
vi) The CTS defines the rules governing type inheritance, virtual methods and object lifetime.
vii) Languages supported by .NET can implement all or some common data types…

How CTS converts the data type to a common data type

To implement or see how CTS is converting the data type to a common data type, for example, when we declare an int type data type in C# and VB.Net then they are converted to int32. In other words, now both will have a common data type that provides flexible communication between these two languages.



3. Name at least 3 runtime services provided by CLR and explain their role in .net framework.

Ans 3.

3 Runtime services provided by CLR and their roles are mentioned below:

1. Type Safety : Is referred as the strongly typed feature. It allows to access memory only in authorized ways makes sure that it is within the bounds. It ensures that code cannot perform operations that are invalid for an object.

2. Managed Code execution : It is the process followed by the CLR from loading MSIL, converting it into machine code, memory management. Managed code execution also handles JIT compilation, handles exceptions, and confirms type safety and security issues.

3. Side-by-side execution : Provides an opportunity to execute multiple versions of an application or component on one computer. It allows the execution of multiple versions of CLR and multiple versions of application executing at the same time on the same computer. It also allows to control what version of a component would an application bind to and what version of CLR would an application choose for execution.

4. What are the differences between Library vs DLL vs .Exe? Explain.

Ans 4.

Differences Between Library and DLL:

When developing software, we are often asked whether we want to use LIB or DLLs in containing functions for the application. LIB is a static library where functions and procedures can be placed and called as the application is being compiled. A DLL or Dynamic Link Library does the same function but is dynamic in a sense that the application can call these libraries during run-time and not during the compilation. This presents a few significant advantages compared to using LIB.
For starters, you would have a single file that is significantly bigger as it contains all of the code while you would have multiple smaller files when using DLL. Compiling your functions and procedures would also allow you more reusability as once you are happy with the functions on the DLL because you can keep it as is with each version of the application and not have to mess with it. You can also use the same DLL if you want to create another application that uses the same functions and procedures. You can directly link to the DLL rather than copy the code from the source as you would need to do with LIB.
A problem with DLL is when you change the content of the DLL. This can lead to versioning problems where an application uses the incorrect version of the DLL causing problems. You need to keep track of your DLLs in order to avoid these problems. You would not have this problem with LIB as you would only get one large file.

When developing the software and choosing DLL, you would still have a LIB file in your project. But unlike when using LIB, this file does not contain the code of the functions and procedures but only stubs that the program needs to call the procedures from the DLL's.

Summary Library vs DLL:

1. A DLL is a library that contains functions that can be called by applications at run-time while LIB is a static library whose code needs to be called during the compilation
2. Using LIB would result in a single file that is considerable bigger while you end up with multiple smaller files with DLL's
3. DLL's are more reusable than LIBs when writing new versions or totally new applications
4. DLL files can be used by other applications while LIB files cannot
5. DLL's are prone to versioning problems while LIB is not
6. You would still have a LIB file when developing software with DLLs but it only contains stubs

Differences Between EXE and DLL:

The terms EXE and DLL are very common in programming. When coding, you can either export your final project to either a DLL or an EXE. The term EXE is a shortened version of the word executable as it identifies the file as a program. On the other hand, DLL stands for Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs.

In the basest application package, you would find at least a single EXE file that may or may not be accompanied with one or more DLL files. An EXE file contains the entry point or the part in the code where the operating system is supposed to begin the execution of the application. DLL files do not have this entry point and cannot be executed on their own.

The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications as long as the coder knows the names and parameters of the functions and procedures in the DLL file. Because of this capability, DLL files are ideal for distributing device drivers. The DLL would facilitate the communication between the hardware and the application that wishes to use it. The application would not need to know the intricacies of accessing the hardware just as long as it is capable of calling the functions on the DLL.

Launching an EXE would mean creating a process for it to run on and a memory space. This is necessary in order for the program to run properly. Since a DLL is not launched by itself and is called by another application, it does not have its own memory space and process. It simply shares the process and memory space of the application that is calling it. Because of this, a DLL might have limited access to resources as it might be taken up by the application itself or by other DLLs.

Summary DLL vs .EXE:

1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library.
2.An EXE file can be run independently while a DLL is used by other applications.
3.An EXE file defines an entry point while a DLL does not.
4.A DLL file can be reused by other applications while an EXE cannot.
5.A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

5. How does CLR in .net ensure security and type safety? Explain.

Ans 5. Security:

The Security Infrastructure of the CLR Provides Evidence, Policy, Permissions, and Enforcement Services

The common language runtime of the .NET Framework has its own secure execution model that isn't bound by the limitations of the operating system it's running on. In addition, unlike the old principal-based security, the CLR enforces security policy based on where code is coming from rather than who the user is. This model, called code access security, makes sense in today's environment because so much code is installed over the Internet and even a trusted user doesn't know when that code is safe.

One of the advantages of virtualized execution environments like the common language runtime (CLR) is that new security models can be developed that transcend the underlying operating system's security model. To that end, the CLR implements its own secure execution model that is independent of the host platform. Beyond the benefits of bringing security to platforms that have never had it (for example, Windows® 98), this also is an opportunity to impose a more component-centric security model that takes into account the nature of dynamically composed systems. This component-centric model is known as code access security, which is the focus of this article.

Components and Security

Systems that are dynamically composed from components have unique security requirements. Because individual components of an application often come from disparate organizations, it is likely that different aspects of the application may warrant different degrees of trust. For example, components from trusted organizations may need access to private information or critical resources that normally would need to be protected from malicious code. Unfortunately, the classic principal-based security model of Windows NT® and Unix ignores where the code came from and only focuses on who is running the code. For 1980s-era programs built before components, this model made sense. However, for a component-centric world in which an application's code may come from all corners of the globe, this model is far too coarsely grained to be useful by itself. Hence, code access security.

The CLR implements a code access security model in which privileges are granted to code, not users. Upon loading a new assembly, the CLR gathers evidence about the code's origins. This evidence is associated with the in-memory representation of the assembly by the CLR and is used by the security system to determine what privileges to grant to the newly loaded code. This determination is made by running the evidence through a security policy. The security policy accepts evidence as input and produces a permission set as output. To avoid performance hits, security policy is typically not run until an explicit security demand is made.

Type Safety:

C# is primarily a *type-safe* language, meaning that types can interact only through protocols they define, thereby ensuring each type’s internal consistency. For instance, C# prevents you from interacting with a *string* type as though it were an *integer* type.

More specifically, C# supports *static typing*, meaning that the language enforces type safety at *compile time*. This is in addition to *dynamic* type safety, which the .NET CLR enforces at *runtime*.

Static typing eliminates a large class of errors before a program is even run. It shifts the burden away from runtime unit tests onto the compiler to verify that all the types in a program fit together correctly. This makes large programs much easier to manage, more predictable, and more robust. Furthermore, static typing allows tools such as IntelliSense in Visual Studio to help you write a program, since it knows for a given variable what type it is, and hence what methods you can call on that variable. C# is called a *strongly typed language* because its type rules (whether enforced statically or dynamically) are very strict. For instance, you cannot call a function that’s designed to accept an integer with a floating-point number, unless you first *explicitly* convert the floating-point number to an integer. This helps prevent mistakes.

Strong typing also plays a role in enabling C# code to run in a sandbox—an environment where every aspect of security is controlled by the host. In a sandbox, it is important that you cannot arbitrarily corrupt the state of an object by bypassing its type rules.