C# Control Statements
In C# programming, the if statement is used to test the condition. There are various types of if statements in C#.
if statement
if-else statement
nested if statement
if-else-if ladder

C# IF Statement
The C# if statement tests the condition. It is executed if condition is true.
Syntax:
```
if(condition){
//code to be executed
}
```

```
using System;
public class IfExample
{ public static void Main(string[] args)
{ int num = 10;
if (num % 2 == 0)
{ Console.WriteLine("It is even number");
} } }
```

C# IF-else Statement
The C# if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.
Syntax:
```
if(condition)
{ //code if condition is true }
else{ //code if condition is false }
```

Example
```
using System;
public class IfExample
{ public static void Main(string[] args)
{ int num = 11;
if (num % 2 == 0)
{ Console.WriteLine("It is even number");
}
else
{ Console.WriteLine("It is odd number");
} } }
```

C# IF-else-if ladder Statement
The C# if-else-if ladder statement executes one condition from multiple statements. Syntax:
```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
```

```
//code to be executed if condition3 is true

}
... else{
//code to be executed if all the conditions are false
}
```

```
using System;
public class IfExample
{
public static void Main(string[] args)
{
Console.WriteLine("Enter a number to check grade:");
int num = Convert.ToInt32(Console.ReadLine());
if (num <0 || num >100)
{
Console.WriteLine("wrong number");
}
else if(num >= 0 && num < 50){
Console.WriteLine("Fail");
}
else if (num >= 50 && num < 60)
{
Console.WriteLine("D Grade");
}
else if (num >= 60 && num < 70)
{
Console.WriteLine("C Grade");
}
else if (num >= 70 && num < 80)
{
Console.WriteLine("B Grade");
}
else if (num >= 80 && num < 90)
{
Console.WriteLine("A Grade");
}
else if (num >= 90 && num <= 100)
{
Console.WriteLine("A+ Grade");
}
}
}
```

C# switch –
A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case. The C# switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C#.

Syntax:

```
switch(expression){
case value1:
//code to be executed;
break;
case value2:
//code to be executed;
break;
...... default:
//code to be executed if all cases are not matched;
break;
}
```

C# Switch Example

```
using System;
public class SwitchExample
{
public static void Main(string[] args)
{
Console.WriteLine("Enter a number:");
int num = Convert.ToInt32(Console.ReadLine());
switch (num)
{
case 10: Console.WriteLine("It is 10"); break;
case 20: Console.WriteLine("It is 20"); break;
case 30: Console.WriteLine("It is 30"); break;
default: Console.WriteLine("Not 10, 20 or 30"); break;
} } }
```

Fallthrough in switch case->
C# For Loop
The C# for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is
recommended to use for loop than while or do-while loops. The C# for loop is same as C/C++.
We can initialize variable, check condition and increment/decrement value.
Syntax:

```
for(initialization; condition; incr/decr){
//code to be executed
}
```

Example

```
using System;
public class ForExample

{
public static void Main(string[] args)
{
for(int i=1;i<=10;i++){
Console.WriteLine(i);
} } }
```

C# While Loop
In C#, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.
Syntax:
```
while(condition){
//code to be executed }
```
Example
```
using System;
public class WhileExample
{
public static void Main(string[] args)
{
int i=1;
while(i<=10)
{
Console.WriteLine(i);
i++;
} } }
```

C# Do-While Loop
The C# do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop. The C# do-while loop is executed at least once because condition is checked after loop body.
Syntax:
```
do{
//code to be executed
}while(condition);
```

Example
```
using System;
public class DoWhileExample
{
public static void Main(string[] args)
{
int i = 1;
do{
Console.WriteLine(i);
i++;

} while (i <= 10) ;
}
}
```

C# Arrays –
Like other programming languages, array in C# is a group of similar types of elements that have contiguous memory location. In C#, array is an object of base type System.Array. In C#, array index starts from 0. We can store only fixed set of elements in C# array.
Advantages of C# Array

(1)Code Optimization (less code)(2) Random Access(3) Easy to traverse data(4) Easy to manipulate data
C# Array Types

There are 3 types of arrays in C# programming:

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array

C# Single Dimensional Array
To create single dimensional array, you need to use square brackets [] after the type.int[] arr = new int[5];
//creating array

```
using System;
public class ArrayExample
{
public static void Main(string[] args)
{
int[] arr = new int[5];//creating array
arr[0] = 10;//initializing array
arr[2] = 20;
arr[4] = 30;
//traversing array
for (int i = 0; i < arr.Length; i++)
{
Console.WriteLine(arr[i]);
} } }
```

C# Array Example:
Declaration and Initialization at same time –
There are 3 ways to initialize array at the time of declaration. int[] arr = new int[5]{ 10, 20, 30, 40, 50 };
We can omit the size of array. int[] arr = new int[]{ 10, 20, 30, 40, 50 };
We can omit the new operator also. int[] arr = { 10, 20, 30, 40, 50 };
Let's see the example of array where we are declaring and initializing array at the same time.

```
using System;
public class ArrayExample
{
public static void Main(string[] args)
{
int[] arr = { 10, 20, 30, 40, 50 };//Declaration and Initialization of array
//traversing array
for (int i = 0; i < arr.Length; i++)
{
Console.WriteLine(arr[i]);
}
```

OR

```
foreach (int i in arr)
{ Console.WriteLine(i); }
```

```
} }
```

C# Multidimensional Arrays
The multidimensional array is also known as rectangular arrays in C#. It can be two dimensional or three
dimensional. The data is stored in tabular form (row * column) which is also known as matrix.
To create multidimensional array, we need to use comma inside the square brackets. For example:

```
int[,] arr=new int[3,3];//declaration of 2D array

int[,,] arr=new int[3,3,3];//declaration of 3D array
```

C# Multidimensional Array Example
Let's see a simple example of multidimensional array in C# which declares, initializes and traverse two dimensional
array. using System;

```
public class MultiArrayExample
{
public static void Main(string[] args)
{
int[,] arr=new int[3,3];//declaration of 2D array
arr[0,1]=10;//initialization
arr[1,2]=20;
arr[2,0]=30;
//traversal
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
Console.Write(arr[i,j]+" ");
}
Console.WriteLine();//new line at each row
} } }
```

C# Multidimensional Array Example: Declaration and initialization at same time
There are 3 ways to initialize multidimensional array in C# while declaration.
int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
We can omit the array size. int[,] arr = new int[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
We can omit the new operator also. int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
Let's see a simple example of multidimensional array which initializes array at the time of declaration.

```
public class MultiArrayExample
{
public static void Main(string[] args)
{
int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };//declaration and initialization
//traversal
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
Console.Write(arr[i,j]+" ");
}
```

Console.WriteLine();//new line at each row
} } }

C# Jagged Arrays
In C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.
Declaration of Jagged array
Let's see an example to declare jagged array that has two elements. int[][] arr = new int[2][];
Initialization of Jagged array
Let's see an example to initialize jagged array. The size of elements can be different. arr[0] = new int[4];
arr[1] = new int[6];
Initialization and filling elements in Jagged array
Let's see an example to initialize and fill elements in jagged array. arr[0] = new int[4] { 11, 21, 56, 78 };
arr[1] = new int[6] { 42, 61, 37, 41, 59, 63 };

Assemblies
(1)The .NET assembly is the standard for components developed with the Microsoft.NET. Dot
(2)NET assemblies may or may not be executable, i.e., they might exist as the executable
(.exe) file or dynamic link library (DLL) file. (3)All the .NET assemblies contain the definition of types, versioning information for the
type, meta-data, and manifest. (4) Assembly is a logical collection of smallest unit in .Net Framework. (5)An assembly is a single deployable unit that contains all the information about the
implementation of classes, structures and interfaces. (6)Assemblies are a basic fundamental unit of application development and deployment
in the .Net Framework. it can be a .dll or an exe . (7)During the compile time Metadata is created with Microsoft Intermediate
Language (MSIL) and stored in a file called Assembly . (8)Assemblies and the metadata provide the CLR with the information required for
executing an application. (9)Assemblies are the smallest units to which the .NET Framework grants permissions. They provide security boundaries within the .NET Framework. You specify the
permission required by your application while building assemblies. When the assembly
is loaded into the runtime, the assembly sends a request to the runtime to grant the permission.
Functions of Assemblies:
An assembly in a .net frame work performs following function
1. Support Execution: An assembly contains the manifest code that is used to execute the MSIL code stored in a portable Executable (PE) file.
2. Provide unique Type Identification: An assembly contains the TYPE class where every type's identity includes the name of the assembly in which it resides.
3. Provide Security: An assembly contains the manifest code where permission are requested and granted.
4. Supports Deployment: assembly is a DLL or EXE file which makes dynamic registration of the component.
5. Track Scope: An assembly module metadata describes the type and resources that are exposed outside the assembly
6. Track Version: An assembly's manifest describes the version dependencies for any dependent assemblies.

7. Supports Side-by-Side Execution: it allows to run multiple versions of an assembly simultaneousaly.
8. Better Code Management and Maintenance

Types of assemblies:
(a). Private Assemblies:

- They are accessible by a single application.
- They reside within the application folder and are unique by name.
- They can be directly used by copying and pasting them to the bin folder.

(b). Shared Assemblies:

- They are shared between multiple applications to ensure reusability
- When you deploy an assembly which can be used by several applications, than this assembly is called shared assembly.
- Shared assemblies are stored in a special folder called Global Assembly Cache (GAC), which is accessible by all applications.
- Shared assemblies must have a strong name.
- A strong name consists of an assembly name, a version number, a culture, a public key and an optional digital signature.

Component of Assembly (Assembly consists of Manifest, Module and Type)

Assembly manifest –

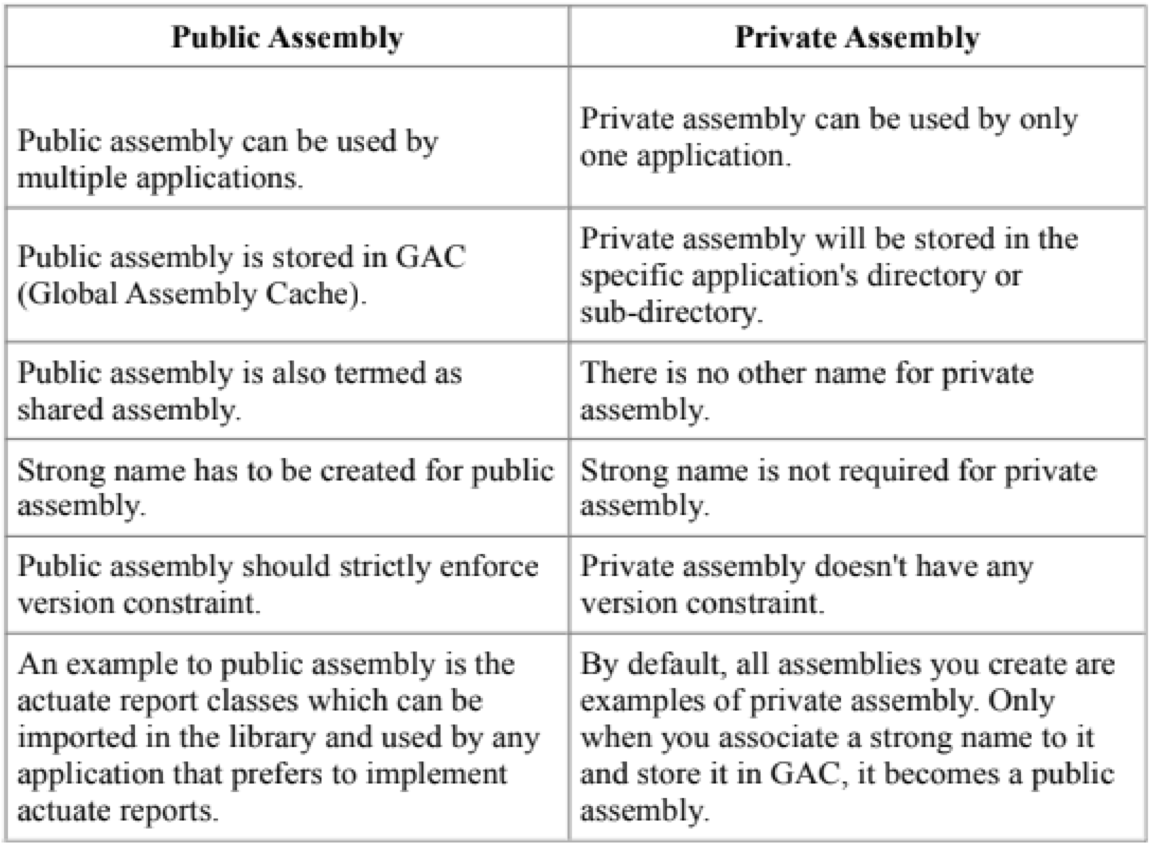
- An assembly is a fundamental building block of any .NET framework application.
- It contains the code that is executed by common language runtime.
- For example, when we build a simple C# application, Visual Studio creates an assembly in the form of a single portable executable (PE) file, specifically an EXE orDLL.
- Every assembly has a file called **'manifest' file** that stores all information about that assembly.
- This information's are known as **metadata.**
- The manifest file contains all the metadata needed to specify the assembly's version requirements, security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. Assembly Manifest describes the assembly.
- It contains(a) Name and Version number culture, strong name, list of all files Information of Assembly. (b) Files (or Resource) that is required by the assembly. (c) Set of permissions required for the assembly to run properly. (d) all meta data needed to define the scope of the assembly and

resolve references to resources and classes

Type Metadata –
It contains metadata, which describes all types (class, structure, enumeration, and so forth),their properties and methods

MSIL –
It contains Intermediate language code.

Resources –
It contains bitmaps, icons, audios and other types of resources.



Public Assembly	Private Assembly
Public assembly can be used by multiple applications.	Private assembly can be used by only one application.
Public assembly is stored in GAC (Global Assembly Cache).	Private assembly will be stored in the specific application's directory or sub-directory.
Public assembly is also termed as shared assembly.	There is no other name for private assembly.
Strong name has to be created for public assembly.	Strong name is not required for private assembly.
Public assembly should strictly enforce version constraint.	Private assembly doesn't have any version constraint.
An example to public assembly is the actuate report classes which can be imported in the library and used by any application that prefers to implement actuate reports.	By default, all assemblies you create are examples of private assembly. Only when you associate a strong name to it and store it in GAC, it becomes a public assembly.