Module 1 Theory

1) A **variable** is nothing but a name given to a storage area.

Syntax for variable definition in C# is −

<data type> <variable list> ;

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variable_list may consist of one or more identifier names separated by commas.

We can initialize a variable at the time of definition as −

int i=100;

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is −

<variable name>=<value>;

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as −

<data type> <variable name>=<value> ;

2) **Type conversion** is converting one type of data to another type. It is also known as Type Casting. In C#, type casting has two forms −

- Implicit type conversion − These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.
- Explicit type conversion − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

3) **Boxing and Unboxing** allows to convert . NET data types from value type to reference type and vice versa. Converting a value type to a reference type is called called boxing and converting a reference type to a value type is called unboxing.

4) If the boolean expression evaluates to true, then the **if block** of code is executed, otherwise **else block** of code is executed.

An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if, else if, else statements --

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

5) 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 expression used in a switch statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, then it will raise a compile time error.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true.

6) Create a new project for **Class library**. Add the code in Class1.cs (created by default) . Build the class library project . Add reference of it's DLL file in our console app . Add "using ClassLibrary1;" statement at the top of our code in console app. Use it's classes in the console app as they were defined in the same file or namespace.

7) For **importing other namespace**, create a new project for Console App. Add the code in Program.cs (created by default) . Build the console app project . Add reference of it's EXE file in our console app . Add "using CA2=ConsoleApp2;" statement at the top of our code in console app. Alias is created so that we can specify full name of classes defined in ConsoleApp2 whenever we use that classes in our console app. Alias makes the task easier whenever namespace has long names. Moreover, using full name of classes helps to avoid collisions in name of classes.