MODULE – 2

1. Operators and Expressions:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

- **Arithmetic operators** are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.
- **Relational operators** are used to check the relationship between two operands. If the relationship is true the result will be true, otherwise it will result in false
- **Comparison operators** compare two numeric operands and returns true or false.
- The **equality operator** checks whether the two operands are equal or not.
- C# includes a decision-making operator ?: which is called the conditional operator or **ternary operator**.
- **Bitwise operator** works on bits and perform bit by bit operation.
- **Assignment Operators** are useful to assign a new value to the operand, and these operators will work with only one operand.

2. Loops:

➢ while loop:

- The while loop loops through a block of code as long as a specified condition is True:
- Syntax:

```
while(condition)
{
//code
}
```

➢ do while

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax:

```
do
{
//code
}while(condition)
```

➢ For loop

- When you know exactly how many times you want to loop through a block of code, for loop is used.
- Syntax:

```
for(statement 1: statement 2:statement 3)
{
//code
}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.

➢ Foreach loop:

- C# provides an easy to use and more readable alternative to for loop, the foreach loop when working with arrays and collections to iterate through the items of arrays/collections.
- The foreach loop iterates through each item, hence called foreach loop.
- Syntax:

```
foreach(element in iterable-
item)
{
//code
}
```

➢ Break/Continue:

- The break statement can also be used to jump out of a **loop**.
- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- You can also use break and continue in while loops.

3. Arrays:

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- In C#, all arrays are dynamically allocated.
- C# array is an object of base type **System.Array**.

- There are 3 types of array:

i. **Single dimensional array**: You create a single-dimensional array using the new operator specifying the array element type and the number of elements. To create single dimensional array, you need to use square brackets [] after the type.

ii. **Multidimensional array**: C# supports multidimensional arrays up to 32 dimensions. The multidimensional array can be
declared by adding commas in the square brackets. For example, [,] declares two-dimensional array, [, ,] declares three-dimensional array, [, , ,] declares four-dimensional array, and so on. So, in a multidimensional array, no of commas = No of Dimensions - 1.

iii. **Jagged array**: A jagged array is an array of array. Jagged arrays store arrays instead of literal values.A jagged array is initialized with two square brackets [][]. The first bracket specifies the size of an array, and the second bracket specifies the dimensions of the array which is going to be stored.

4. Strings:

- In C#, string is an object of **System.String** class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.

- In C#, string is keyword which is an alias for System.String class. That is why string and String are equivalent.
- The + operator can be used between strings to combine them. This is called **concatenation**

Method Name	Description
Clone()	It is used to return a reference to this instance of String.
Compare(String, String)	It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
CompareOrdinal(String, String)	It is used to compare two specified String objects by evaluating the numeric values of the corresponding Char objects in each string..
CompareTo(String)	It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
Concat(String, String)	It is used to concatenate two specified instances of String.
Contains(String)	It is used to return a value indicating whether a specified substring occurs within this string.
Copy(String)	It is used to create a new instance of String with the same value as a specified String.
CopyTo(Int32, Char[], Int32, Int32)	It is used to copy a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
EndsWith(String)	It is used to check that the end of this string instance matches the specified string.
Equals(String, String)	It is used to determine that two specified String objects have the same value.
Format(String, Object)	It is used to replace one or more format items in a specified string with the string representation of a specified object.
GetEnumerator()	It is used to retrieve an object that can iterate through the individual characters in this string.
GetHashCode()	It returns the hash code for this string.
GetType()	It is used to get the Type of the current instance.
GetTypeCode()	It is used to return the TypeCode for class String.
IndexOf(String)	It is used to report the zero-based index of the first occurrence of the specified string in this instance.
Insert(Int32, String)	It is used to return a new string in which a specified string is inserted at a specified index position.
LastIndexOf(Char)	It is used to report the zero-based index position of the last occurrence of a specified character within String.
LastIndexOfAny(Char[])	It is used to report the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.

Remove(Int32)	It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.
Replace(String, String)	It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
Split(Char[])	It is used to split a string into substrings that are based on the characters in an array.
StartsWith(String)	It is used to check whether the beginning of this string instance matches the specified string.
Substring(Int32)	It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
ToCharArray()	It is used to copy the characters in this instance to a Unicode character array.
ToLower()	It is used to convert String into lowercase.
ToUpper()	It is used to convert String into uppercase.
Trim()	It is used to remove all leading and trailing white-space characters from the current String object.
TrimEnd(Char[])	It Is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.
TrimStart(Char[])	It is used to remove all leading occurrences of a set of characters specified in an array from the current String object.

5. Methods:

- A method is a code block that contains a series of statements.
- The Main method is the entry point for every C# application.

METHOD SIGNATURE:

- Declaration is done in: class, interface, or struct.
- Specified by access level such as :public, private, abstract or sealed.

METHOD ACCESS:

- Calling a method on an object is like accessing a field.
- A program causes the statements to be executed by calling the method and specifying any required method arguments.

METHOD CALLING:

- For **calling a method by reference**, ref keyword is used.
- Passing a <u>value type</u> parameter to a method by reference means passing a reference of the variable to the method.
- So the changes made to the parameter inside the called method will affect the original data stored in the argument variable.
- In **call by value**, the changes made to the parameter inside of the called method will not have an effect on the original value. The Variable value is directly stored in the memory.

RETURN VALUES:

- Methods can return a value to the caller. If the return type is not void, the method can return the value by using the return keyword.
- A statement with the return keyword followed by a value that matches the return type will return that value to the method caller.

6. <u>Date time:</u>

- Date and Time in C# are two commonly used data types. Both Date and Time in C# are represented using C# DateTime class.
- Datetime constructor initializes a new instance of DateTime object. At the time of object creation we need to pass required parameters like year, month, day, etc.
- DateTime object contains two static read-only fields called as MaxValue and Minvalue.
- DateTime contains a variety of methods which help to manipulate DateTime Object. It helps to add number of days, hour, minute, seconds to a date, date difference, parsing from string to datetime object, get universal time and many more.
- It contains properties like Day, Month, Year, Hour, Minute, Second, DayOfWeek and others in a DateTime object.
- DateTime object facilitates to perform addition, subtraction, equality, greater than using operators like “+”, “-”, “==” etc.
- Different users need different kinds of format date. For instance some users need date like "mm/dd/yyyy", some need "dd-mm-yyyy".