UNDERSTANDMENTS & CONCLUSIONS

NANDINI BHATTACHARYA

MODULE – 2

8. Operators and Expressions

- **Arithmetic operators** that perform arithmetic operations with numeric operands.
  +, -, *, /, % (reminder), ++ (Unary increment ++ operator increases its operand by 1), --(Unary decrement -- operator decreases its operand by 1), +(Unary plus: Returns the value of operand),-(Unary minus: Computes the numeric negation of its operand.)

- **Comparison operators** that compare numeric operands
  <, >, <=, >=

- **Equality operators** that check if their operands are equal or not
  ==, !=

- **Boolean Logical operators** that perform logical operations with bool operands
  !, &&, ||

- **Misc operators** perform miscellaneous tasks like giving type of operand, size of operand etc.
  sizeof(), typeof(), ?:(Ternary), etc

- **Interpolated string expressions** that provide convenient syntax to create formatted strings:

  ```
  var r = 2.3;
  var message = $"The area of a circle with radius {r} is {Math.PI * r * r:F3}.";
  Console.WriteLine(message);
  // Output:
  // The area of a circle with radius 2.3 is 16.619.
  ```

- **Lambda expressions** that allow you to create anonymous functions:

  ```
  int[] numbers = { 2, 3, 4, 5 };
  var maximumSquare = numbers.Max(x => x * x);
  Console.WriteLine(maximumSquare);
  // Output:
  // 25
  ```

- **Query expressions** that allow you to use query capabilities directly in C#:

  ```
  var scores = new[] { 90, 97, 78, 68, 85 };
  IEnumerable<int> highScoresQuery =
     from score in scores
     where score > 80
     orderby score descending
     select score;
  Console.WriteLine(string.Join(" ", highScoresQuery));
  // Output:
  // 97 90 85
  ```

9. Loop Iteration

9.1.1 For Loop

- The **for** keyword indicates a loop in C#. The **for** loop executes a block of statements repeatedly until the specified condition returns false.

- The **for** loop contains the following three optional sections, separated by a semicolon:

  Initializer: The initializer section is used to initialize a variable that will be local to a for loop and cannot be accessed outside loop. e.g., ++i or i++, and await expression.

  Condition: The condition is a boolean expression that will return either true or false. If an expression evaluates to true, then it will execute the loop again; otherwise, the loop is exited.

  Iterator: The iterator defines the incremental or decremental of the loop variable.

  ```
  For (initializer; condition; iterator)
  {
   //code
   }
  ```

9.1.2 Foreach Loop

- The **foreach** loop is used to iterate over the elements of the collection.
- The collection may be an array or a list.
- It executes for each element present in the array

  ```
  foreach(data_type var_name in collection_variable)
  {
   // code
  }
  ```

9.1.3. while Loop

- The **while loop** executes a statement or a block of statements while a specified Boolean expression evaluates to true.
- Because that expression is evaluated before each execution of the loop, a while loop executes zero or more times.
- This differs from a do loop, which executes one or more times.

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

9.1.4. do...while Loop

- **Do while** loop execute the code before the condition check.
- It means first code will be executing and at the end it will check the condition.
- It will execute the block until condition become false.

  ```
  do {
   //code
  } While(condition);
  ```

9.2.1 break

- The break statement is used to terminate the loop or statement.
- If the break statement presents in the nested loop, then it terminates only those loops which contains break statement.

9.2.2 continue

- This keyword is used to skip over the execution of iteration on a certain condition.
- After that, it transfers the control onto the next iteration of the loop.

10. Understanding Arrays

- An array is the data structure that stores a fixed number of literal values (elements) of the same data type.
- Array elements are stored contiguously in the memory.
- The data types of the elements may be any valid data type like char, int, float, etc.
- The variables in the array are ordered and each has an index beginning from 0
- An array can be of three types: single-dimensional, multidimensional, and jagged array. Here you will learn about the single-dimensional array.
- If you are adding array elements at the time of declaration, then size is optional.

10.1 Single-dimensional array

- In this array contains only one row for storing the values.
- All values of this array are stored contiguously starting from 0 to the array size.
- Example: int[] arrayint = new int[2];

10.2 Multidimensional array

- The multi-dimensional array contains more than one row to store the values.
- It can be a 2D-array or 3D-array or more.
- To storing and accessing the values of the array, one required the nested loop.
- Example: int[,] intarray = new int[4, 2];

10.3 Jagged array

- An array whose elements are arrays is known as Jagged arrays.
- It means "array of arrays ".
- The jagged array elements may be of different dimensions and sizes.
- Example: int[][] arr1 = { new int[] { 1, 3, 5, 7, 9 }, new int[] { 2, 4, 6, 8 } };

11. Defining and Calling Methods

11.1 Define method and use

- Methods are generally the block of codes or statements in a program that gives the user the ability to reuse the same code.
- So basically, a method is a collection of statements that perform some specific task and return the result to the caller. A method can also perform some specific task without returning anything.

```
<Access Specifier><Return Type><Method Name>(Parameter List)
{
//Method Body
}
```

- Method Calling is done when the user wants to execute the method.
- The method needs to be called for using its functionality.
- A method returns to the code that invoked it when:
  It completes all the statements in the method
  It reaches a return statement
  Throws an exception

11.2 Different type of parameters in method (Value type, Ref. type, optional)

- **Value type** – It is the normal C# value parameter which means value is directly passed.
- **Ref. Type** – References of a variable is passed as arguments which are assigned first. Argument passed consists of 'ref' keyword first. Any changes made in this argument in method will reflect on the variable in the calling method.
- **Optional or Default type** – These are the type of arguments which may be passed only if operations on them are required otherwise the function consists default values of these parameters. This parameter should only be passed after required parameters.
- **Named Parameters** – These are Parameters which are given the same name in arguments as well so the ordering doesn't matter and user doesn't have to remember the ordering, they can assign values according to argument names.
- **Out parameters** – It's just like ref parameter but the keyword here is 'out' and any argument passed need not to be initialized.

12. Working with strings

12.1. string class study

- A string is a series of characters that is used to represent text. It can be a character, a word or a long passage surrounded with the double quotes "".
- C# provides the String data type to store string literals.
- There two ways to declare a string variable in C#. Using **System.String** class and using **string** keyword. Both are the same and make no difference.

12.2 Use of various string methods

- **Copy(String)** - Creates a new instance of string with the same value as a specified string.
- **Compare()** - Used to compare the two string objects
- **Contains(String)** - Returns a value indicating whether a specified substring occurs within this string.
- **Equals()** - Determines whether two string objects have the same value.
- **indexOf()** - Return the position of specify char or string. Return -1 if char does not found.
- **Insert(int32, String)** - Insert the new string at the given position.
- **Trim()** - Returns a new string in which all leading and trailing occurrences of a set of specified characters from the current String object are removed.
- **TrimEnd(Char[])** - Removes all trailing occurrences of a set of characters specified in an array from the current String object.
- **TrimStart(Char[])** - Removes all leading occurrences of a set of characters specified in an array from the current String object.
- **ToString()** - Converts the value into string type.
- **ToUpper()** - Converts the value into uppercase.
- **ToLower()** - Converts the value into lowercase.
- **ToCharArray ()** - Copies the characters in this instance to a Unicode character array.
- **Substring()** - Retrieves a substring from this instance.
- **Split()** - Returns a string array that contains the substrings in this instance that are delimited by elements of a specified string or Unicode character array.

13. Datetime class study

- We use DateTime class to work with dates and time.
- DateTime helps developer to find out more information about Date and Time like Get month, day, year, week day.
- It also helps to find date difference, add number of days to a date, etc.
- It initializes a new instance of DateTime object.
- At the time of object creation we need to pass required parameters like year, month, day etc