Phase-2

Name : Kotadiya Amisha K.

College : Government Engineering College, Rajkot

Branch : Computer Engineering

Understanding

Module-8

- **Operators and Expressions**
  - Incremental, ternary etc.

1) Operators

Operators are symbols that are used to perform operations on operands. Operands may be variables and/or constants.

1. Basic Assignment Operator

Basic assignment operator (=) is used to assign values to variables.

For example:-

```
// Basic Assignment Operator

using System;

namespace Operator
{
   class AssignmentOperator
   {
      public static void Main(string[] args)
      {
         int firstNumber, secondNumber;
         // Assigning a constant to variable
         firstNumber = 10;
         Console.WriteLine("First Number = {0}", firstNumber);

         // Assigning a variable to another variable
         secondNumber = firstNumber;
         Console.WriteLine("Second Number = {0}", secondNumber);
      }
   }
}
```

2. Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

For example,

```
// Basic Arithmetic Operator

using System;

namespace Operator
{
   class ArithmeticOperator
   {
      public static void Main(string[] args)
      {
         double firstNumber = 14.40, secondNumber = 4.60, result;
         int number1 = 26, number2 = 4, rem;

         // Addition operator
         result = firstNumber + secondNumber;
         Console.WriteLine("{0} + {1} = {2}", firstNumber, secondNumber,
         result);

         // Subtraction operator
         result = firstNumber - secondNumber;
         Console.WriteLine("{0} - {1} = {2}", firstNumber, secondNumber,
         result);

         // Multiplication operator
         result = firstNumber * secondNumber;
         Console.WriteLine("{0} * {1} = {2}", firstNumber, secondNumber,
         result);

         // Division operator
         result = firstNumber / secondNumber;
         Console.WriteLine("{0} / {1} = {2}", firstNumber, secondNumber,
         result);

         // Modulo operator
         rem = number1 % number2;
         Console.WriteLine("{0} % {1} = {2}", number1, number2, rem);
      }
   }
}
```

3. Relational Operators

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.

Relational operators are used in decision making and loops.

For example,

```
// Relational Operator

using System;

namespace Operator
{
   class RelationalOperator
   {
      public static void Main(string[] args)
      {
         bool result;
         int firstNumber = 10, secondNumber = 20;

         // Equal to operator
         result = (firstNumber==secondNumber);
         Console.WriteLine("{0} == {1} returns {2}",firstNumber, secondNumber,
          result);

         // Greater than Operator
         result = (firstNumber > secondNumber);
         Console.WriteLine("{0} > {1} returns {2}",firstNumber, secondNumber,
         result);

         // Less than Operator
         result = (firstNumber < secondNumber);
         Console.WriteLine("{0} < {1} returns {2}",firstNumber, secondNumber,
         result);

         // Greater than or equal to
         result = (firstNumber >= secondNumber);
         Console.WriteLine("{0} >= {1} returns {2}",firstNumber, secondNumber,
         result);

         // Less than or equal to
         result = (firstNumber <= secondNumber);
         Console.WriteLine("{0} <= {1} returns {2}",firstNumber, secondNumber,
         result);
```

```
        // Not equal to
        result = (firstNumber != secondNumber);
        Console.WriteLine("{0} != {1} returns {2}",firstNumber, secondNumber,
        result);
      }
   }
}
```

4. Logical Operators

Logical operators are used to perform logical operation such as `and`, `or`. Logical operators operates on boolean expressions (`true` and `false`) and returns boolean values. Logical operators are used in decision making and loops.

Here is how the result is evaluated for logical `AND` and `OR` operators.

If one of the operand is true, the `OR` operator will evaluate it to `true`.

If one of the operand is false, the `AND` operator will evaluate it to `false`.

For Example :-

```
// Basic Logical Operator
using System;

namespace Operator
{
   class LogicalOperator
   {
      public static void Main(string[] args)
      {
         bool result;
         int firstNumber = 10, secondNumber = 20;

         // OR operator (||)
         result = (firstNumber == secondNumber) || (firstNumber > 5);
         Console.WriteLine(result);

         // AND operator (&&)
         result = (firstNumber == secondNumber) && (firstNumber > 5);
         Console.WriteLine(result);
      }
   }
}
```

5. Unary Operators

Unlike other operators, the unary operators operates on a single operand.

The increment (++) and decrement (--) operators can be used as prefix and postfix. If used as prefix, the change in value of variable is seen on the same line and if used as postfix, the change in value of variable is seen on the next line.

We can see the effect of using ++ as prefix and postfix. When ++ is used after the operand, the value is first evaluated and then it is incremented by 1. Hence the statement

int number=10;

Console.WriteLine((number++));

prints 10 instead of 11. After the value is printed, the value of number is incremented by 1.

The process is opposite when ++ is used as prefix. The value is incremented before printing. Hence the statement

Console.WriteLine((++number));

prints 12.

The case is same for decrement operator (--).

For Example:-

```
// Unary Operator

using System;

namespace Operator
{
   class UnaryOperator
   {
      public static void Main(string[] args)
      {
         int number = 10, result;
         bool flag = true;

         // Unary Plus
         result = +number;
         Console.WriteLine("+number = " + result);

         // Unary Minus
         result = -number;
         Console.WriteLine("-number = " + result);

         // Increment
         result = ++number;
         Console.WriteLine("++number = " + result);

         // Decrement
         result = --number;
         Console.WriteLine("--number = " + result);

         // Logical Negation (Not)
         Console.WriteLine("!flag = " + (!flag));
      }
   }
}
```

6. Ternary Operator

The ternary operator ? : operates on three operands. It is a shorthand for if-then-else statement. Ternary operator can be used as follows:

variable = Condition? Expression1 : Expression2;

The ternary operator works as follows:

If the expression stated by Condition is true, the result of Expression1 is assigned to variable.

If it is false, the result of Expression2 is assigned to variable.

For Example:-

```
// Ternary Operator

using System;

namespace Operator
{
   class TernaryOperator
   {
      public static void Main(string[] args)
      {
         int number = 10;
         string result;

         // Ternary operator (?:)
         result = (number % 2 == 0)? "Even Number" : "Odd Number";
         Console.WriteLine("{0} is {1}", number, result);
      }
   }
}
```

7. Bitwise and Bit Shift Operators

Bitwise and bit shift operators are used to perform bit manipulation operations.

For Example:-

```
// Basic Bit And Bitwise Operator
using System;
namespace Operator
{
   class BitOperator
   {
      public static void Main(string[] args)
      {
         int firstNumber = 5;
         int secondNumber = 20;
         int result;
         // Bitwise Complement
         result = ~firstNumber;
         Console.WriteLine("~{0} = {1}", firstNumber, result);

         // Bitwise AND
         result = firstNumber & secondNumber;
         Console.WriteLine("{0} & {1} = {2}", firstNumber,secondNumber,
         result);

         // Bitwise OR
         result = firstNumber | secondNumber;
         Console.WriteLine("{0} | {1} = {2}", firstNumber,secondNumber,
         result);

         // Bitwise Exclusive OR
         result = firstNumber ^ secondNumber;
         Console.WriteLine("{0} ^ {1} = {2}", firstNumber,secondNumber,
         result);

         // Bitwise Left Shift
         result = firstNumber << 2;
         Console.WriteLine("{0} << 2 = {1}", firstNumber, result);

         // Bitwise Right Shift
         result = firstNumber >> 2;
         Console.WriteLine("{0} >> 2 = {1}", firstNumber, result);
      }
   }
}
```

8. Compound Assignment Operators

For Example:-

```
// Compound Assignment Operator

using System;
namespace Operator
{
   class CompoundAssignmentOperator
   {
      public static void Main(string[] args)
      {
         int number1, number2, number3, number4, number5, number6,
         number7, number8, number9, number10;

         number1=number2=number3=number4=number5=number6=number7=
         number8=number9=number10=10;

         // Addition Assignment
         number1 += 5;
         Console.WriteLine(number1);

         // Subtraction Assignment
         number2 -= 3;
         Console.WriteLine(number2);

         // Multiplication Assignment
         number3 *= 2;
         Console.WriteLine(number3);

         // Division Assignment
         number4 /= 3;
         Console.WriteLine(number4);

         // Modulo Assignment
         number5 %= 3;
         Console.WriteLine(number5);

         // Bitwise AND Assignment
         number6 &= 10;
         Console.WriteLine(number6);

         // Bitwise OR Assignment
         number7 |= 14;
         Console.WriteLine(number7);
```

```
        // Bitwise XOR Assignment
        number8 ^= 12;
        Console.WriteLine(number8);

        // Left Shift Assignment
        number9 <<= 2;
        Console.WriteLine(number9);

        // Right Shift Assignment
        number10 >>= 3;
        Console.WriteLine(number10);
    }
  }
}
```

C# Expressions

An expression in C# is a combination of operands (variables, literals, method calls) and operators that can be evaluated to a single value. To be precise, an expression must have at least one operand but may not have any operator.

Let's look at the example below:

```
double temperature;
temperature = 42.05;
```

Here, 42.05 is an expression. Also, temperature = 42.05 is an expression too.

```
int a, b, c, sum;
sum = a + b + c;
Here, a + b + c is an expression.
if (age>=18 && age<58)
   Console.WriteLine("Eligible to work");
```

Here, (age>=18 && age<58) is an expression that returns a boolean value. "Eligible to work" is also an expression.

Module-9

- **Loop Iteration**
  - for, foreach, while, do..while
  - break, continue

1) for, foreach, while, do..while

C# for loop

The for keyword is used to create for loop in C#. The syntax for for loop is:

```
for (initialization; condition; iterator)
{
   // body of for loop
}
```

How for loop works?

1.C# for loop has three statements: initialization, condition and iterator.

2.The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.

3.Then, the condition is evaluated. The condition is a boolean expression, i.e. it returns either true or false.

4.If the condition is evaluated to true:

a.The statements inside the for loop are executed.

b.Then, the iterator statement is executed which usually changes the value of the initialized variable.

c.Again the condition is evaluated.

d.The process continues until the condition is evaluated to false.

5.If the condition is evaluated to false, the for loop terminates.

For Exapmle:-

```
// C# for Loop
using System;
namespace Loop
{
   class ForLoop1
   {
      public static void Main(string[] args)
      {
         // For loop
         for (int number=1; number<=5; number++)
         {
            Console.WriteLine("C# For Loop: Iteration {0}", number);
         }
      }
   }
}
```

C# 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.

Before moving forward with foreach loop, visit:

Syntax of foreach loop

```
foreach (element in iterable-item)
{
   // body of foreach loop
}
```

How foreach loop works?

1.The in keyword used along with foreach loop is used to iterate over the iterable-item. The in keyword selects an item from the iterable-item on each iteration and store it in the variable element.

2.On first iteration, the first item of iterable-item is stored in element. On second iteration, the second element is selected and so on.

3.The number of times the foreach loop will execute is equal to the number of elements in the array or collection.

For Example:-

```
// Printing array using foreach loop
using System;
namespace Loop
{
   class ForEachLoop1
   {
      public static void Main(string[] args)
      {
         char[] myArray = {'H','e','l','l','o'};

         // ForEach Loop
         foreach(char ch in myArray)
         {
            Console.WriteLine(ch);
         }
      }
   }
}
```

C# while loop

The while keyword is used to create while loop in C#. The syntax for while loop is:

```
while (test-expression)
{
   // body of while
}
```

How while loop works?

1.C# while loop consists of a test-expression.

2.If the test-expression is evaluated to true,

a.statements inside the while loop are executed.

b.after execution, the test-expression is evaluated again.

3.If the test-expression is evaluated to false, the while loop terminates.

For Exapmle:-

```
// while loop to compute sum of first 5 natural numbers

using System;

namespace Loop
{
   class WhileLoop
   {
      public static void Main(string[] args)
      {
         int number=1, sum=0;

         // sum of five number using while Loop
         while (number<=5)
         {
            sum += number;
            number++;
         }
         Console.WriteLine("Sum = {0}", sum);
      }
   }
}
```

C# do...while loop

The do and while keyword is used to create a do...while loop. It is similar to a while loop, however there is a major difference between them.

In while loop, the condition is checked before the body is executed. It is the exact opposite in do...while loop, i.e. condition is checked after the body is executed.

This is why, the body of do...while loop will execute at least once irrespective to the test-expression.

The syntax for do...while loop is:

```
do
{
   // body of do while loop
} while (test-expression);
```

How do...while loop works?

1.The body of do...while loop is executed at first.

2.Then the test-expression is evaluated.

3.If the test-expression is true, the body of loop is executed.

4.When the test-expression is false, do...while loop terminates.

For Exapmle:-

```
// Multiplication table using dowhile llop

using System;

namespace Loop
{
   class DoWhileLoop
   {
      public static void Main(string[] args)
      {
         int number = 1, tableNumber = 5, product;

         // DoWhile Loop
         do
         {
            product = tableNumber * number;
            Console.WriteLine("{0} * {1} = {2}", tableNumber, number, product);
            number++;
         } while (number <= 10);
      }
   }
}
```

2) break, continue

C# break Statement

In C#, we use the break statement to terminate the loop.

As we know, loops iterate over a block of code until the test expression is false. However, sometimes we may need to terminate the loop immediately without checking the test expression. In such cases, the break statement is used. The syntax of break statement is,

```
break;
```

For Example:-

```
// C# break statement with for loop

using System;

namespace BreakStatement
{
  class Program
  {
    static void Main(string[] args)
    {
      for (int number = 1; number <= 4; ++number)
      {

        // terminates the loop
        if (number == 3)
        {
          break;
        }

        Console.WriteLine(number);
      }

      Console.ReadKey();
    }
  }
}
```

C# continue Statement

In C#, we use the continue statement to skip a current iteration of a loop.

When our program encounters the continue statement, the program control moves to the end of the loop and executes the test condition (update statement in case of for loop).

The syntax for continue is:

```
Continue;
```

For Example:-

```
// C# continue with for loop

using System;

namespace ContinueStatement
{
  class Program
  {
    static void Main(string[] args)
    {

      // continue with for loop
      for (int number = 1; number <= 5; ++number){

        if (number == 3)
        {
          continue;
        }

        Console.WriteLine(number);
      }
    }
  }
}
```

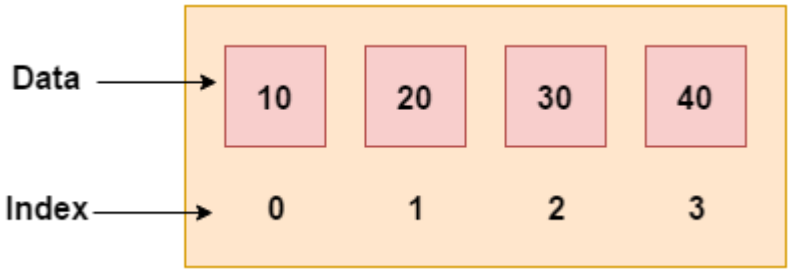
Module-10

- **Understanding Arrays**
  - Define and use of array

1) Define and use of array

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

- o Code Optimization (less code)
- o Random Access
- o Easy to traverse data
- o Easy to manipulate data
- o Easy to sort data etc.

C# Array Concepts

There are some concepts of arrays in C# programming:

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array
4. Implicitly Typed Array
5. Using foreach with Array
6. Passing Arrays as Arguments

1. Single Dimensional Array

Declaring Arrays

datatype[] arrayName;

where,

- datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.
- arrayName specifies the name of the array.

For example,

```
double[] balance;
```

Initializing an Array

Declaring an array does not initialize the array in the memory. When the array variable is initialized, you can assign values to the array.

Array is a reference type, so you need to use the new keyword to create an instance of the array. For example,

```
double[] balance = new double[10];
```

Assigning Values to an Array

```
//You can assign values to individual array elements, by using the index number, like
double[] balance = new double[10];
balance[0] = 4500.0;

//You can assign values to the array at the time of declaration, as shown −
double[] balance = { 2340.0, 4523.69, 3421.0};

//You can also create and initialize an array, as shown −
int[] marks = new int[5]  { 99,  98, 92, 97, 95};

//You may also omit the size of the array, as shown −
int[] marks = new int[]  { 99,  98, 92, 97, 95};

//You can copy an array variable into another target array variable.
//In such case, both the target and source point to the same memory location −
int[] marks = new int[]  { 99,  98, 92, 97, 95};
int[] score = marks;
```

2. Multidimensional Array

C# allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array.

Declaring Arrays

You can declare a 2-dimensional array of strings as −

string [,] names;

or, a 3-dimensional array of int variables as −

int [, ,] m;

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

```
int [,] a = new int [3,4]
{
  {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
  {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
  {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts. That is, row index and column index of the array.

For example,

```
int val = a[2,3];
```

For Example:-

```
// Demonstrates Multi Dimensional Array

using System;

namespace ArrayExample
{
  class MultiDimensionalArray
  {
    static void Main(string[] args)
    {
      /* an array with 5 rows and 2 columns*/
      int[,] myArray = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
      int i, j;

      /* output each array element's value */
      for (i = 0; i < 5; i++)
      {
        for (j = 0; j < 2; j++)
        {
          Console.WriteLine("a[{0},{1}] = {2}", i, j, myArray[i,j]);
        }
      }

      Console.ReadKey();
    }
  }
}
```

3. Jugged Array

A jagged array is an array whose elements are arrays, possibly of different sizes. A jagged array is sometimes called an "array of arrays." The following examples show how to declare, initialize, and access jagged arrays.

Note: The elements of a jagged array must be initialized before they are used.

Declaring Arrays

A Jagged array is an array of arrays. You can declare a jagged array named scores of type int as-

int [][] scores;

Declaring an array, does not create the array in memory. To create the above array −

```
int[][] scores = new int[5][];
for (int i = 0; i < scores.Length; i++)
{
      scores[i] = new int[4];
}
```

Initialize a jagged array

```
// Declare the array of two elements.
      int[][] arr = new int[2][];
// Initialize the elements.
      arr[0] = new int[5] { 1, 3, 5, 7, 9 };
      arr[1] = new int[4] { 2, 4, 6, 8 };
```

Where, scores is an array of two arrays of integers - arr[0] is an array of 5 integers and arr[1] is an array of 4 integers.

```
int[][] jaggedArray3 =
{
   new int[] { 1, 3, 5, 7, 9 },
   new int[] { 0, 2, 4, 6 },
   new int[] { 11, 22 }
};
```

For Example:-

```
// Demonstrates Jagged Array

using System;

class JaggedArray
{
   static void Main()
   {
      // Declare the array of two elements.
      int[][] myArray = new int[2][];

      // Initialize the elements.
      myArray[0] = new int[7] { 1, 3, 5, 7, 9 ,5,7};
      myArray[1] = new int[4] { 2, 4, 6, 8 };

      // Display the array elements.
      for (int i = 0; i < myArray.Length; i++)
      {
         Console.Write("Element({0}): ", i);

         for (int j = 0; j < myArray[i].Length; j++)
         {
            Console.Write("{0}{1}", myArray[i][j], j == (myArray[i].Length - 1) ? "" : " ");
         }

         Console.WriteLine();
      }

      // Keep the console window open in debug mode.
      Console.WriteLine("Press any key to exit.");
      Console.ReadKey();
   }
}
```

4.Implicit Typed Array

You can create an implicitly-typed array in which the type of the array instance is inferred from the elements specified in the array initializer. The rules for any implicitly-typed variable also apply to implicitly-typed arrays. For more information, see Implicitly Typed Local Variables.

Use:-

Implicitly-typed arrays are usually used in query expressions together with anonymous types and object and collection initializers.

For Example:-

```
// Demonstrate implicitly typed array
using System;
class ImplicitlyTypedArray
{
   static void Main()
   {
      var a = new[] { 1, 10, 100, 1000 }; // int[]
      var b = new[] { "hello", null, "world" }; // string[]
      // single-dimension jagged array
      var c = new[]
      {
         new[]{1,2,3,4},
         new[]{5,6,7,8}
      };
      // jagged array of strings
      var d = new[]
      {
         new[]{"Luca", "Mads", "Luke", "Dinesh"},
         new[]{"Karen", "Suma", "Frances"}
      };
      //Implicitly-typed Arrays in Object Initializers
      var contacts = new[]
      {
         new {
            Name = " Eugene Zabokritski",
            PhoneNumbers = new[] { "206-555-0108", "425-555-0001" }
         },
         new {
            Name = " Hanying Feng",
            PhoneNumbers = new[] { "650-555-0199" }
         }
      };
   }
}
```

5. Using foreach with Array

The foreach statement provides a simple, clean way to iterate through the elements of an array.

For single-dimensional arrays, the foreach statement processes elements in increasing index order, starting with index 0 and ending with index Length - 1:

For Example:-

```
// demonstrates foreach with SingleDimensionalArray and MultiDimensionalArray

using system;

class ForeachWithArray
{
   public static void main(String Args[])
   {
      // foreach with Single DimensionalArray
      int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
      foreach (int i in numbers)
      {
         Console.Write("{0} ", i);
      }

      //foreach with Multi DimensionalArray
      int[,] numbers2D = new int[3, 2] { { 9, 99 }, { 3, 33 }, { 5, 55 } };

      foreach (int i in numbers2D)
      {
         System.Console.Write("{0}{1} ", i);
      }
   }
}
```

6. Passing Arrays as Arguments

Passing single-dimensional arrays as arguments

Arrays can be passed as arguments to method parameters. Because arrays are reference types, the method can change the value of the elements.

Passing single-dimensional arrays as arguments

You can pass an initialized single-dimensional array to a method. For example, the following statement sends an array to a print method.

For Example:-

```
// pass single dimensional array as an argument

using System;

class PassArrayasArgument
{
   static void DisplayArray(string[] myArray)
   {
      Console.WriteLine(string.Join(" ", myArray));
   }
   // Change the array by reversing its elements.
   static void ChangeArray(string[] myArray)
   {
      Array.Reverse(myArray);
   }
   static void ChangeArrayElements(string[] myArray)
   {
      // Change the value of the first three array elements.
      myArray[0] = "Mon";
      myArray[1] = "Wed";
      myArray[2] = "Fri";
   }

   static void Main()
   {
      // Declare and initialize an array.
      string[] weekDays = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
      // Display the array elements.
      DisplayArray(weekDays);
      Console.WriteLine();
```

```
        // Reverse the array.
        ChangeArray(weekDays);
        // Display the array again to verify that it stays reversed.
        Console.WriteLine("Array weekDays after the call to ChangeArray:");
        DisplayArray(weekDays);
        Console.WriteLine();

        // Assign new values to individual array elements.
        ChangeArrayElements(weekDays);
        // Display the array again to verify that it has changed.
        Console.WriteLine("Array weekDays after the call to ChangeArrayElements:");
        DisplayArray(weekDays);
    }
}

/* output:
        Sun Mon Tue Wed Thu Fri Sat

        Array weekDays after the call to ChangeArray:
        Sat Fri Thu Wed Tue Mon Sun

        Array weekDays after the call to ChangeArrayElements:
        Mon Wed Fri Wed Tue Mon Sun
*/
```

Module-11

- **Defining and Calling Methods**
  - Define method and use
  - Different type of parameters in method(Value type, Ref, type, optional)

1) Define method and use

A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.

To use a method, you need to −

•Define the method

•Call the method

Defining Methods in C#

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows −

<Access Specifier> <Return Type> <Method Name>(Parameter List)

{

  Method Body

}

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

Passing Parameters to a Method

When method with parameters is called, you need to pass the parameters to the method. There are three ways that parameters can be passed to a method –

1.Value parameters

2.Reference parameters

3.Optional parameters

1.Value parameters

This is the default mechanism for passing parameters to a method. In this mechanism, when a method is called, a new storage location is created for each value parameter

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

For Example:-

```
// example of Value Parameters

using System;
namespace Method1
{
  class ValueParameters
  {
    static void swap(int x, int y)
    {
      int temp;
      temp = x; /* save the value of x */
      x = y;    /* put y into x */
      y = temp; /* put temp into y */

      Console.WriteLine("value of x : {0}", x);
      Console.WriteLine("value of y : {0}", y);
    }
    static void Main(string[] args)
    {
      /* local variable definition */
      int number1 = 15;
      int number2 = 20;

      Console.WriteLine("Before swap, value of number1 : {0}", number1);
      Console.WriteLine("Before swap, value of number2 : {0}", number2);

      /* calling a function to swap the values */
      swap(number1, number2);

      Console.WriteLine("After swap, value of number1 : {0}", number1);
      Console.WriteLine("After swap, value of number2 : {0}", number2);

      Console.ReadLine();
    }
  }
}
```

2.Reference parameters

A reference parameter is a reference to a memory location of a variable. When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method.

You can declare the reference parameters using the ref keyword.

This method copies the reference to the memory location of an argument into the formal parameter. This means that changes made to the parameter affect the argument.

For Example:-

```
// example of Reference Parameters

using System;
namespace Method2
{
  class ReferenceParameters
  {
    static void swap(ref int x, ref int y)
    {
      int temp;
      temp = x; /* save the value of x */
      x = y;    /* put y into x */
      y = temp; /* put temp into y */
    }
    static void Main(string[] args)
    {
      /* local variable definition */
      int number1 = 100;
      int number2 = 200;

      Console.WriteLine("Before swap, value of number1 : {0}", number1);
      Console.WriteLine("Before swap, value of number2 : {0}", number2);

      /* calling a function to swap the values */
      swap(ref number1, ref number2);

      Console.WriteLine("After swap, value of number1 : {0}", number1);
      Console.WriteLine("After swap, value of number2 : {0}", number2);

      Console.ReadLine();
    }
  }
}
```

3.Optional parameters

A return statement can be used for returning only one value from a function. However, using output parameters, you can return two values from a function. Output parameters are similar to reference parameters, except that they transfer data out of the method rather than into it.

This method helps in returning more than one value.

For Example:-

```
// example of Optional Parameters

using System;

namespace Method3
{
  class OptionalParameters
  {
    static void getValues(out int x, out int y )
    {
       Console.WriteLine("Enter the first value: ");
       x = Convert.ToInt32(Console.ReadLine());

       Console.WriteLine("Enter the second value: ");
       y = Convert.ToInt32(Console.ReadLine());
    }

    static void Main(string[] args)
    {
      /* local variable definition */
      int number1, number2;

      /* calling a function to get the values */
      getValues(out number1, out number2);

      Console.WriteLine("After method call, value of number1 : {0}", number1);
      Console.WriteLine("After method call, value of number2 : {0}", number2);
      Console.ReadLine();
    }
  }
}
```

Module-12

- **Working with strings**
  - String class study
  - Use of various string methods

1) String class study

In C#, a string is a sequence of Unicode characters or array of characters.
The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the text. So the string is the representation of the text. A string is represented by a class System.String.

The String class is defined in the .NET base class library. In other words, a String object is a sequential collection of System.Char objects which represent a string. The maximum size of the String object in memory can be 2GB or about 1 billion characters.

Characteristics of String Class:

- The System.String class is immutable, i.e once created its state cannot be altered.
- With the help of length property, it provides the total number of characters present in the given string.
- String objects can include a null character which counts as the part of the string's length.
- It provides the position of the characters in the given string.
- It allows empty strings. Empty strings are the valid instance of String objects that contain zero characters.
- A string that has been declared but has not been assigned a value is null. Attempting to call methods on that string throws a NullReferenceException.
- It also supports searching strings, comparison of string, testing of equality, modifying the string, normalization of string, copying of strings, etc.
- It also provides several ways to create strings like using a constructor, using concatenation, etc.

2) Use of various string methods

String methods:-

1.Clone()

The C# Clone() method is used to clone a string object. It returns another copy of same data. The return type of Clone() method is object.

Syntax:-

public object Clone()

For Example:-

```
// Example of clone method

using System;

   public class CloneMethod
   {
      public static void Main(string[] args)
      {
          string str1 = "Hello ";
          string str2 = (String)str1.Clone();
          Console.WriteLine("Original String1 : {0}",str1);
          Console.WriteLine("Clone string2    : {0}",str2);
      }
   }
```

2. Compare()

The C# Compare() method is used to compare first string with second string lexicographically. It returns an integer value.

If both strings are equal, it returns 0. If first string is greater than second string, it returns 1 else it returns -1.

Rule

s1==s2 returns 0

s1>s2 returns 1

s1<s2 returns -1

Syntax:

public static int Compare(String first, String second)

3.CompareOrdinal()

The C# CompareOrdinal() method compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.

If both strings are equal, it returns 0. If first string is greater than second string, it returns positive number in difference else it returns negative number.

Rule

s1==s2 returns 0

s1>s2 returns positive number in difference

s1<s2 returns negative number in difference

Synatx

public static int CompareOrdinal(String first, String second)

4.compareTo()

The C# CompareTo() method is used to compare String instance with a specified String object. It indicates whether this String instance precedes, follows, or appears in the same position in the sort order as the specified string or not.

Syntax

public int CompareTo(String str)

public int CompareTo(Object)

For Example:-

```
// Example of compare, compareTo, compareOrdinal method

using System;

   public class CompareMethod
   {
      public static void Main(string[] args)
      {
         string str1 = "hello";
         string str2 = "hello";
         string str3 = "csharp";
         string str4 = "mello";

         //Compare methods
         Console.WriteLine("Compare {0} and {1} : {2}",str1,str2,string.Compare(str1,str2));
         Console.WriteLine("Compare {0} and {1} : {2}",str2,str3,string.Compare(str2,str3));
         Console.WriteLine("Compare {0} and {1} : {2}",str3,str4,string.Compare(str3,str4));

         //s1==s2 returns 0
         //s1>s2 returns positive number in difference
         //s1<s2 returns negative number in difference
         Console.WriteLine("CompareOrdinal {0} and {1} : {2}",str1,str2,string.CompareOrdinal
(str1,str2));
         Console.WriteLine("CompareOrdinal {0} and {1} : {2}",str1,str3,string.CompareOrdinal
(str1,str3));
         Console.WriteLine("CompareOrdinal {0} and {1} : {2}",str1,str4,string.CompareOrdinal
(str1,str4));

         //CompareTo methods
         Console.WriteLine("Compare {0} and {1} : {2}",str1,str2,str1.CompareTo(str2));
         Console.WriteLine("Compare {0} and {1} : {2}",str2,str3,str2.CompareTo(str3));
         Console.WriteLine("Compare {0} and {1} : {2}",str1,str4,str1.CompareTo(str4));
      }
   }
```

5.Concat()

The C# Concat() method is used to concatenate multiple string objects. It returns concatenated string. There are many overloaded methods of Concat().

Syntax:

public static string Concat(String, String)

public static string Concat(Object)

For Example:-

```
// Example of concate method

using System;

public class ConcateMethod
{
   public static void Main(string[] args)
   {
      string str1 = "Hello ";
      string str2 = "C#";
      Console.WriteLine("Concate {0}and {1} : {2}",str1,str2,string.Concat(str1,str2));
   }
}
```

6.Contains()

The C# Contains() method is used to return a value indicating whether the specified substring occurs within this string or not. If the specified substring is found in this string, it returns true otherwise false.

Syntax

public bool Contains(String str)

For Example:-

```
// Example of contains method

using System;
   public class ContainsMethod
   {
      public static void Main(string[] args)
      {
          string str1 = "Hello ";
          string str2 = "He";
          string str3 = "Hi";
          Console.WriteLine("Contains {0} and {1} : {2}",str1,str2,str1.Contains(str2));
          Console.WriteLine("Contains {0} and {1} : {2}",str1,str3,str1.Contains(str3));
      }
   }
```

7. Copy()

The C# Copy() method is used to create a new instance of String with the same value as a specified String. It is a static method of String class. Its return type is string.

Syntax

public static string Copy(String str)

8.CopyTo()

The C# CopyTo() method is used to copy a specified number of characters from the specified position in the string. It copies the characters of this string into a char array.

Syntax

public void CopyTo(int index, char[] ch, int start, int end)

For Example:-

```
// Example of copy, copyTo method

using System;
   public class CopyMethod
   {
      public static void Main(string[] args)
      {
         // copy string
          string str1 = "Hello ";
          string str2 = string.Copy(str1);
          Console.WriteLine("Original string : {0}",str1);
          Console.WriteLine("Copy string  : {0}",str2);

         //It copies the characters of this string into a char array.
          string str3 = "Hello C#, How Are You?";
          char[] chArray = new char[15];
          str3.CopyTo(10,chArray,0,12);
          Console.WriteLine(chArray);
      }
   }
```

9.EndWith()

The C# EndsWith() method is used to check whether the specified string matches the end of this string or not. If the specified string is found at the end of this string, it returns true otherwise false.

Syntax

public bool EndsWith(String str)

10.Equals()

The C# Equals() method is used to check whether two specified String objects have the same value or not. If both strings have same value, it return true otherwise false.

In other words, it is used to compare two strings on the basis of content.

Syntax

public bool Equals(String str)

public static bool Equals(String, String)

11.GetType()

The C# GetType() method is used to get type of current object. It returns the instance of Type class which is used for reflection.

Syntax

public Type GetType()

12.IndexOf()

The C# IndexOf() is used to get index of the specified character present in the string. It returns index as an integer value.

Syntax

public int IndexOf(Char ch)

13.Insert()

The C# Insert() method is used to insert the specified string at specified index number. The index number starts from 0. After inserting the specified string, it returns a new modified string.

Syntax

public string Insert(Int32 first, String second)

14.Remove()

The C# Remove() method is used to get a new string after removing all the characters from specified beginIndex till given length. If length is not specified, it removes all the characters after beginIndex.

Syntax

public string Remove(Int32 beginIndex)

For Example:-

```
// Example of Endswith, euals, getType, indexOf, insert, remove method
using System;
public class StringMethods
{
   public static void Main(string[] args)
   {
      string str1 = "Hello";
      string str2 = "llo";
      string str3 = "C#";
      string str4 = "Hello";
      string str5 = "Hello c#";
      //check whether the specified string matches the end of this string or not.
      Console.WriteLine("\nEndsWith() method : ");
      Console.WriteLine("String {0} endwith {1} : {2}",str1,str2,str1.EndsWith(str2));
      Console.WriteLine("String {0} endwith {1} : {2}",str1,str3,str1.EndsWith(str3));

      //compare two strings on the basis of content.
      Console.WriteLine("\nEndsWith() method : ");
      Console.WriteLine("String {0} equals {1} : {2}",str1,str2,str1.Equals(str2));
      Console.WriteLine("String {0} equals {1} : {2}",str1,str4,str1.Equals(str4));

      //method is used to get type of current object.
      Console.WriteLine("\nGetType() method : ");
      Console.WriteLine("get type of {0} : {1} ",str1,str1.GetType());

      //get index of the specified character present in the string.
      Console.WriteLine("\nIndexOf() method : ");
      int index = str1.IndexOf('e');
      Console.WriteLine("string: {0}\nIndexOf e : {1}",str1,index);

      //insert the specified string at specified index number
      Console.WriteLine("\nInsert() method : ");
      string str6 = str5.Insert(5," -");
      Console.WriteLine(str6);

      /* get a new string after removing all the characters from specified beginIndex
      till given length. If length is not specified, it removes all the characters after
      beginIndex. */
      Console.WriteLine("\nRemove() method : ");
      str2 = str1.Remove(2);
      Console.WriteLine(str2);
   }
}
```

15. Replace()

The C# Replace() method is used to get a new string in which all occurrences of a specified Unicode character in this string are replaced with another specified Unicode character.

There are two methods of Replace() method. You can replace string also.

Syntax

public string Replace(Char first, Char second)

public string Replace(String firstString, String secondString)

16. StartsWith()

The C# StartsWith() method is used to check whether the beginning of this string instance matches the specified string.

Syntax

public bool StartsWith(String str)

17. Substring()

The C# SubString() method is used to get a substring from a String. The substring starts at a specified character position and continues to the end of the string.

Syntax

public string Substring(Int32 index)

public string Substring(Int32, Int32)

For Example:-

```
// Example of replace, startWith, subString method

using System;
   public class ReplaceMethod
   {
      public static void Main(string[] args)
      {
        //replace method
        Console.WriteLine("\n\n**** Replace() method ****");
        string str1 = "Hello F#";
        string str2 = str1.Replace('F','C');
        Console.WriteLine("string = {0} \nF replace with C : {1}",str1,str2);

        //to check whether the beginning of this string instance matches the specified string.
        Console.WriteLine("\n\n**** Startswith() method ****");
        bool bool1 = str1.StartsWith("h");
        bool bool2 = str1.StartsWith("H");
        Console.WriteLine("String = {0} \nStarts with h : {1}",str1,bool1);
        Console.WriteLine("String = {0} \nStarts with H : {1}",str1,bool2);

        //to get a substring from a String.
        //The substring starts at a specified character position and continues to the end of
       // the string.
        Console.WriteLine("\n\n**** substring() method ****");
        string str3 = "Hello C#";
        string str4 = str3.Substring(5);
        Console.WriteLine("substring = {0}",str4);
      }
   }
```

18. ToUpper()

The C# ToUpper() method is used to convert string into uppercase. It returns a string.

Syntax

public string ToUpper()

public string ToUpper(CultureInfo)

19. TrimStart()

The C# TrimStart() method id used to remove all leading occurrences of a set of characters specified in an array from the current String object.

Syntax

public string TrimStart(params Char[] ch)

20. TrimEnd()

The C# TrimEnd() method is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.

Syntax

public string TrimEnd(params Char[] ch)

For Example:-

```
// Example of trimStart, trimEnd, strUpper method

using System;

   public class TrimMethods
   {
      public static void Main(string[] args)
      {

       //TrimStart Method
       Console.WriteLine("\nTrimStart() method : ");
       string str1 = "Hello C#";
       char[] chStart = {'H'};
       string str2 = str1.TrimStart(chStart);
       Console.WriteLine(str2);

       //TrimEnd method
       Console.WriteLine("\nTrimEnd() method : ");
       char[] chEnd = {'#'};
       str2 = str1.TrimEnd(chEnd);
       Console.WriteLine(str2);

       //convert string into uppercase.
       Console.WriteLine("\nstrUpper() method : ");
       string strUpper = "Hello C#";
       string str3 = strUpper.ToUpper();
       Console.WriteLine(str3);

       }
   }
```

Module-13

- **Datetime class study**

1) Datetime class study

We used the DateTime when there is a need to work with the dates and times in C#.

We can format the date and time in different formats by the properties and methods of the DateTime.

We have different ways to create the DateTime object. A DateTime object has Time, Culture, Date, Localization, Milliseconds.

The DateTime struct represents a single moment in time. This is typically represented by a date and a time. Each of the following can be represented by an instance of DateTime:

15th June 1215

28th January 1986 1139 Hours

18th February 1930

24th October 1648

12th April 1945 3:35 PM

We can instantiate a new DateTime instance with a default value like so:

```
DateTime objDateTime = new DateTime(); //1 January 0001 0000 Hours (Midnight)
```

However, that's not very useful. What's more useful is to use overloads of DateTime's constructor to create actual dates.

```
DateTime objDateTime1 = new DateTime(1215, 6, 15);
var objDateTime2 = new DateTime(1930, 2, 18);
var objDateTime3 = new DateTime(1945, 4, 12, 15, 35, 0);
```

Let's look closer at that last line. To create a new DateTime instance with a specified date, we pass these parameters in this order:

The year

The month (1-12)

The day (1-31)

The hour (0-23)

The minute (0-59)

The second (0-59)

For Example:-

```
// Display Date and Time

using System;

namespace DateTimeEx
{
   class DateTimeDisplay
   {
      static void Main(string[] args)
      {

         DateTime objDateTimeProperty = new DateTime(1974, 7, 10, 7, 10, 24);

         // Display All date and time
         Console.WriteLine("all Date and Time:{0}", objDateTimeProperty);

         // Display Day
         Console.WriteLine("\nDay:{0}",objDateTimeProperty.Day);

         // Display Month
         Console.WriteLine("Month:{0}", objDateTimeProperty.Month);

         // Display Year
         Console.WriteLine("Year:{0}", objDateTimeProperty.Year);

         // Display Hour
         Console.WriteLine("Hour:{0}", objDateTimeProperty.Hour);
```

```
        // Display Minute
        Console.WriteLine("Minute:{0}", objDateTimeProperty.Minute);

        // Display Second
        Console.WriteLine("Second:{0}", objDateTimeProperty.Second);

        // Display Millisecond
        Console.WriteLine("Millisecond:{0}", objDateTimeProperty.Millisecond);

        // Display Day of Week
        Console.WriteLine("Day of Week:{0}", objDateTimeProperty.DayOfWeek);

        // Display Day of Year
        Console.WriteLine("Day of Year: {0}", objDateTimeProperty.DayOfYear);

        // Display Time of Day
        Console.WriteLine("Time of Day:{0}", objDateTimeProperty.TimeOfDay);

        // Display Tick
        Console.WriteLine("Tick:{0}", objDateTimeProperty.Ticks);

        // Display Kind
        Console.WriteLine("Kind:{0}", objDateTimeProperty.Kind);

        // Display Short Date
        var display = DateTime.Now.ToShortDateString();
        Console.WriteLine(display); //15-Oct-21

        // Display Short Time
        var displayTime = DateTime.Now.ToShortTimeString();
        Console.WriteLine(displayTime); //11:04 AM

        // Display Long Date
        var longDisplay = DateTime.Now.ToLongDateString();
        Console.WriteLine(longDisplay); //Friday, 15 October, 2021

        // Display Long Time
        var longDisplayTime = DateTime.Now.ToLongTimeString();
        Console.WriteLine(longDisplayTime); //11:04:10 AM

    }
  }
}
```

Shorthand Methods

There are a few shorthand methods on the DateTime struct that we can use instead of specifying an entire format.

```
// Display Short Date
var display = DateTime.Now.ToShortDateString();
Console.WriteLine(display); //15-Oct-21

// Display Short Time
var displayTime = DateTime.Now.ToShortTimeString();
Console.WriteLine(displayTime); //11:04 AM

// Display Long Date
var longDisplay = DateTime.Now.ToLongDateString();
Console.WriteLine(longDisplay); //Friday, 15 October, 2021

// Display Long Time
var longDisplayTime = DateTime.Now.ToLongTimeString();
Console.WriteLine(longDisplayTime); //11:04:10 AM
```

TimeSpan

The TimeSpan struct represents a duration of time, whereas DateTime represents a single point in time. Instances of TimeSpan can be expressed in seconds, minutes, hours, or days, and can be either negative or positive.

We can create a default instance of TimeSpan using the parameterless constructor; the value of such an instance is TimeSpan.Zero.

```
TimeSpan newTimeSpan = new TimeSpan();
Console.WriteLine(newTimeSpan);
```

For Example:-

```
// TimeSpan Exapmle

using System;

namespace DateTimeEx
{
   class TimeSpanEx
   {
      static void Main(string[] args)
      {

         //6 days, 8 hours, 12 minutes, and 35 seconds
         TimeSpan objTimeSpan = new TimeSpan(6, 8, 12, 35);
         Console.WriteLine("\nTimeSapn(6,8,12,35) : {0}",objTimeSpan); //6.08:12:35

         //Calculations with DateTime and TimeSpan
         DateTime objStartDate = new DateTime(2020, 11, 10, 9, 35, 0);
         Console.WriteLine("\nStart Date and Time : {0}",objstartDate);

         DateTime objEndDate = new DateTime(2020, 11, 14, 15, 10, 20);
         Console.WriteLine("End Date and Time : {0}",objEndDate);

         TimeSpan duration = endDate - startDate;
         Console.WriteLine("\nCalculations with DateTime and TimeSpan : {0}",duration);
        //4.05:35:20

      }
   }
}
```