Module 2

8. Operators and Expression.

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C# has rich set of built-in operators and provides the following type of operators:

Arithmetic operators: they are used to perform simple mathematical operations. Generally there 2 types of arithmetic operators:

• Unary operators – that operates on only one operand

• Binary operators – that operates on two operands

Increment operator ++ (In arithmetic operators):

The unary increment operator ++ increments its operand by 1. The operand must be a variable, a property access, or an indexer access.

The increment operator is supported in two forms: the postfix increment operator, x++, and the prefix increment operator, ++x.

Post-incrementer operator: The result of x++ is the value of x before the operation
Eg: int i = 4;
Console.WriteLine(i++);
(output: 4)

Pre-incrementer operator: The result of ++x is the value of x after the operation
Eg: int i = 2;
Console.WriteLine(++i);
(output: 3)

Decrementer operator - - (in arithmetic operators):

The unary decrement operator -- decrements its operand by 1. The operand must be a variable, a property access, or an indexer access.

The decrement operator is supported in two forms: the postfix decrement operator, x--, and the prefix decrement operator, --x.

Post-decremnter operator: The result of x-- is the value of x before the operation
Eg: int i = 3;
Console.WriteLine(i--); // output: 3

Pre-decremnetor operator: The result of --x is the value of x after the operation

Eg: int i = 2;
Console.WriteLine(--i); // output: 1

Binary arithmetic operators:

Operator	Description	Example
+	Adds two opwrands	A+B=10
-	Substracts second operand from the first	A-B=20
*	Multiplies both operands	A*B=40
/	Divides numerator by de-numerator	B/A=4
%	Modulus operator and remainder of after an integer division	B%A=1

Comparison operators: These operators provides comparison between different operands.

Operator	Description	Example
==	Checks if the values of two operands are equal or not,if yes then condition becomes true.	(A==B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A!=B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A>B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A<B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A>=B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A<=B) is true.

Boolean logical operators:
The following operators perform logical operations with bool operands:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A&&B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A||B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A&&B) is true.

Assignment operators:

These operators assign value to an operand. It also contains some shorthand operators i.e which has = and an other operator together.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment	C >>= 2 is same as C =

	operator	C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C &2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C |2

Ternary operator:

C# includes a decision-making operator ?: which is called the conditional operator or ternary operator. It is the short form of the if else conditions.

Syntax: condition ? statement 1 : statement 2

In ternary operator first of all the condition is checked if the condition is true statement 1 will executed and if it is wrong statement 2 will be executed.

Eg: int x = 1, y=5;

int a = x>y? 1:2;

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.

Eg: int a, b, c, sum;

sum = a + b + c;

here a+b+c is an expression.

Now, I have tried to prepare a program that contains expressions and most type of operators :

Code:

```
using System;

namespace OperatorsandExpression
{
    class Program
    {
        static void Main(string[] args)
        {
            //incremental operator
            Console.WriteLine("Incremental operator:");
            int i = 3;
            Console.WriteLine("(initially i=3)i++ = {0}", i++);
            Console.WriteLine("After i++, i = {0}", i);
            //Binary Arithmetic operators
            Console.WriteLine("\nBinary arithmetic operator:");
            int a = 5, b = 8, c;
            Console.WriteLine("Operations on a = 5 and b = 8");
            c = a + b;
```

```csharp
        Console.WriteLine("a + b = {0}", c);
        c = b - a;
        Console.WriteLine("b - a = {0}", c);
        c = a * b;
        Console.WriteLine("a*b = {0}", c);
        //logical operators
        Console.WriteLine("\nLogical operator:");
        bool p = true & false;
        Console.WriteLine("for expression bool p = true & false, p={0}",
        p);
        bool q = true | true;
        Console.WriteLine("for expression q = true | true, q={0}", q);
        //comparison operators
        Console.WriteLine("\nComparison operator:");
        int x = 8, y = 10;
        Console.WriteLine("For x = 8 and y = 10");
        if (x > y)
            Console.WriteLine("x > y");
        else if (x < y)
            Console.WriteLine("x < y");
        else
            Console.WriteLine("x = y");
        //Assignment operators
        Console.WriteLine("\nAssignment operator:");
        int assign = 9;
        Console.WriteLine("assign = {0}", assign);
        assign += 4;
        Console.WriteLine("(after performing assign+=4) assign = {0}",
        assign);
        //Ternary operator
        Console.WriteLine("\nTernary operator:");
        int t1 = 5, t2 = 128;
        var m = t1 > t2 ? "t1>t2" : "t1<t2";
        Console.WriteLine(" m = {0}", m);
        //Expressions example
        Console.WriteLine("\nExpression:");
        int k = 1, l = 7, v= 3;
        int j = k * l - v; // expression
        Console.WriteLine("Here k*l-v is an expression");
        Console.Read();
    }
  }
}
```

Output:

```
Incremental operator:
(initially i=3)i++ = 3
After i++, i = 4

Binary arithmetic operator:
Operations on a = 5 and b = 8
a + b = 13
b - a = 3
a*b = 40

Logical operator:
for expression bool p = true & false, p=False
for expression q = true | true, q=True

Comparison operator:
For x = 8 and y = 10
x < y

Assignment operator:
assign = 9
(after performing assign+=4) assign = 13

Ternary operator:
 m = t1<t2

Expression:
Here k*l-v is an expression
```

9. Loop Iteration.

For Loop:

Syntax :

for (initialization; condition; iterator)

{

// body of for loop

}

initialization statement is executed at first and only once. Here, the variable is usually declared and initialized. Then, the **condition** is evaluated. The condition is a boolean expression, i.e. it returns either true or false. If the condition is evaluated to true: The statements inside the for loop are executed. Then, the iterator statement is executed which usually changes the value of the initialized variable. Again the condition is evaluated. The process continues until the condition is evaluated to false. If the condition is evaluated to false, the for loop terminates.

Code:

```csharp
using System;

namespace ForLoop
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i <= 5; i++)
            {
                Console.WriteLine(i);
            }
            Console.Read();
```

```
      }
    }
}
```

Output:

Foreach Loop:

The foreach loop iterates through each item of collections/array, hence called foreach loop.

Syntax:

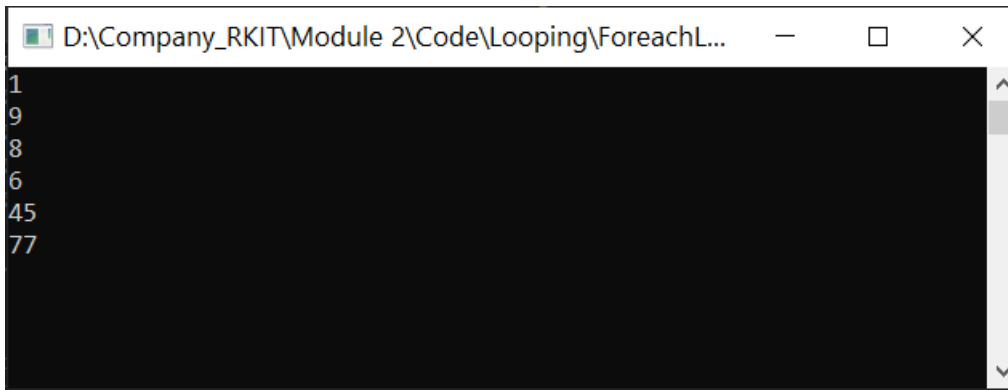foreach (element in iterable-item)
{
// body of foreach loop
}

The in keyword selects an item from the iterable-item on each iteration and store it in the variable element. On first iteration, the first item of iterable-item is stored in element. On second iteration, the second element is selected and so on. The number of times the foreach loop will execute is equal to the number of elements in the array or collection.

Code:

```
using System;

namespace ForeachLoop
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] IntArray = { 1,9,8,6,45,77};
            foreach (int i in IntArray)
            {
                Console.WriteLine(i);
            }
            Console.Read();
        }
    }
}
```

}

While Loop:

Syntax:

while (test-expression)

{

// body of while

}

C# while loop consists of a test-expression. If the test-expression is evaluated to true, statements inside the while loop are executed. after execution, the test-expression is evaluated again. If the test-expression is evaluated to false, the while loop terminates.
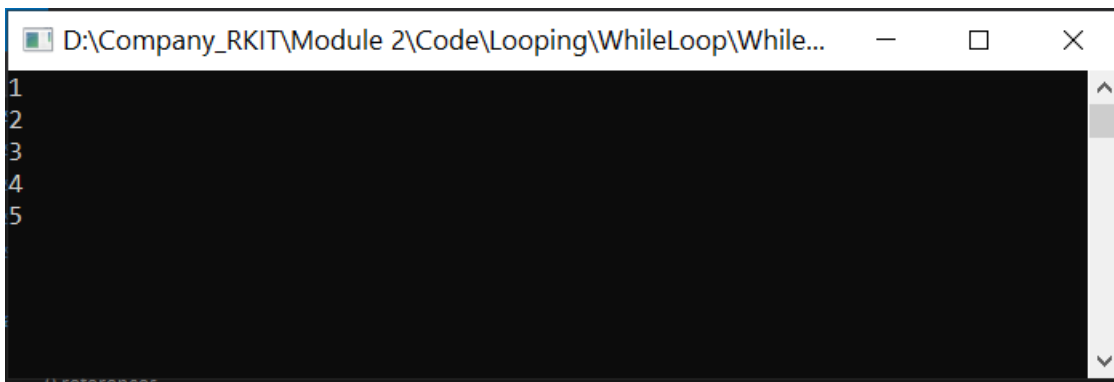
Code:

```csharp
using System;

namespace WhileLoop
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 1;
            while (i <= 5)
            {
                Console.WriteLine(i);
                i++;
            }
            Console.Read();

        }
    }
}
```

Output:

```
1
2
3
4
5
```

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.

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

First the body of do-while loop is executed and then the test expression is checked. If test expression is evaluated to true only then the body of do-while loop will be executed the next time. When test-expression returns false the loop terminates.
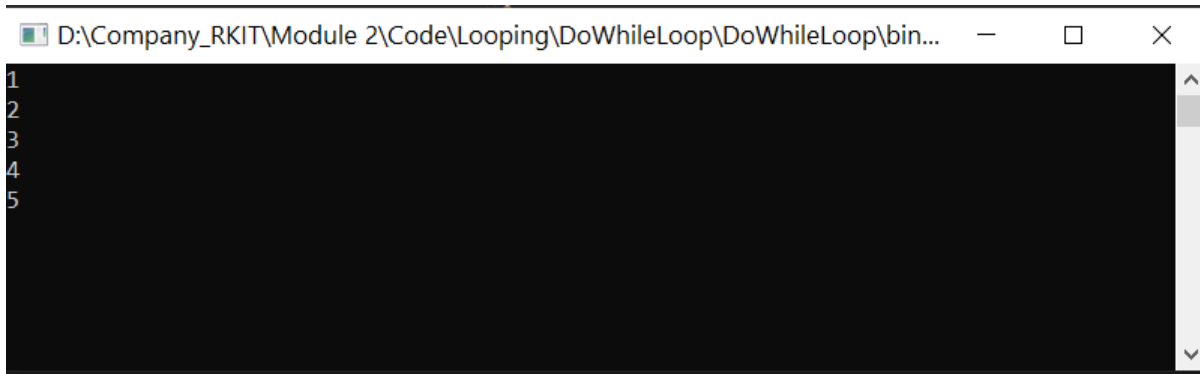
Code:

```csharp
using System;

namespace DoWhileLoop
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 1;
            do
            {
                Console.WriteLine(i);
                i++;
            }
            while (i < 6);
            Console.Read();

        }
    }
}
```
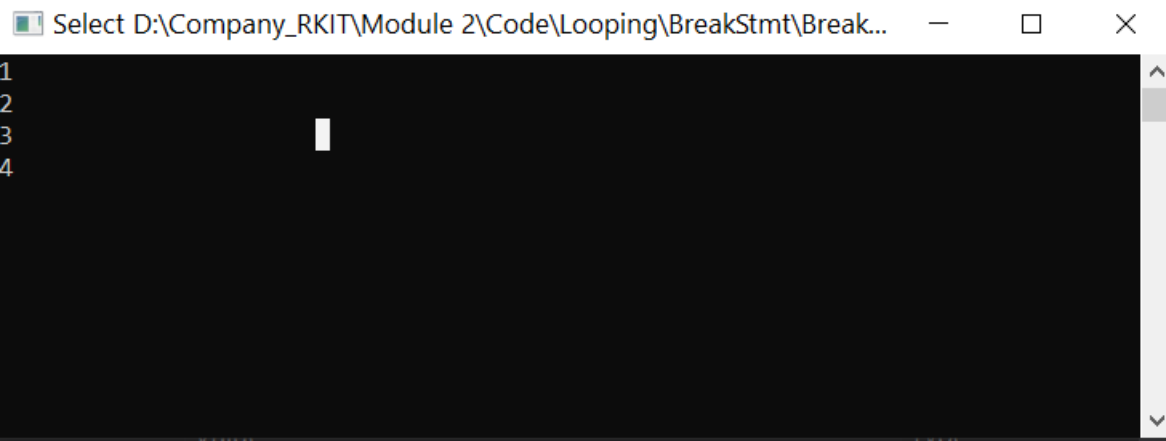
Output:

```
1
2
3
4
5
```

Break:

The break statement can be used to jump out of a **loop**.

Code:

```csharp
using System;

namespace BreakStmt
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i < 10; i++)
            {
                if (i == 5)
                {
                    break;
                }
                Console.WriteLine(i);

            }
            Console.Read();
        }
    }
}
```
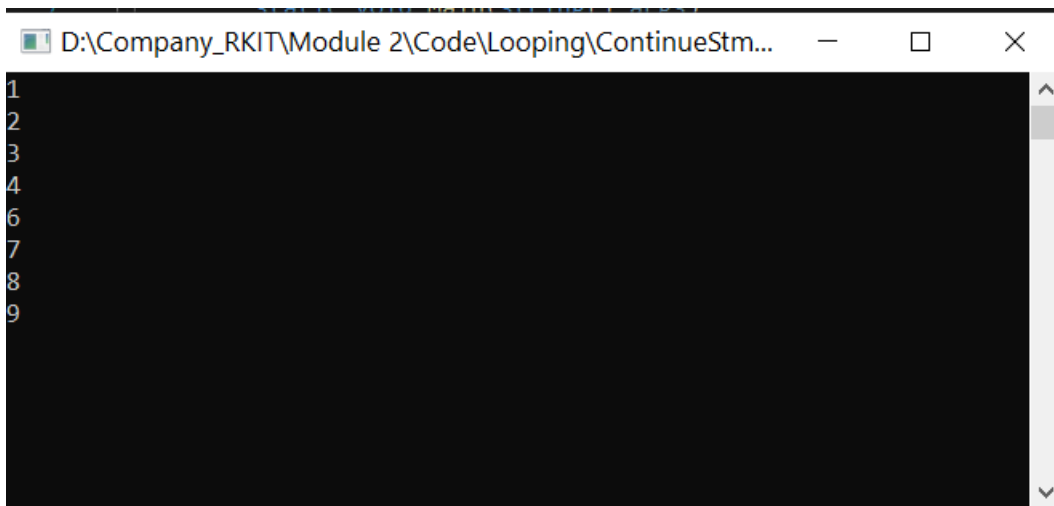
Output:

```
1
2
3
4
```

Continue:

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Code:

```csharp
using System;

namespace ContinueStmt
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i < 10; i++)
            {
                if (i == 5)
                {
                    continue;
                }
                Console.WriteLine(i);
            }
            Console.Read();
        }
    }
}
```

Output:

```
1
2
3
4
6
7
8
9
```

10.   Understanding Arrays.

➢ Define and usage of array

An array is a group of like-typed variables that are referred to by a common name. And each data item is called an element of the array. The data types of the elements may be any valid data type like char, int, float, etc. and the elements are stored in a contiguous location. Length of the array specifies the number of elements present in the array. In C# the allocation of memory for the arrays is done dynamically.

Single-dimensional arrays:
You create a single-dimensional array using the new operator specifying the array element type and the number of elements.

Syntax:
type [ ] < Name_Array > = new < datatype > [size];

It can be declared and initialized in different ways
Eg: int[] intArray1 = new int[5];
int[] intArray2 = new int[5]{1, 2, 3, 4, 5};
int[] intArray3 = {1, 2, 3, 4, 5};

Index starts from 0 so we can assign value by directly giving its index
Eg: intArray[4] = 3;

To find out how many elements an array has, use the Length property:
Eg:
int[] car = {1, 2, 3};
Console.WriteLine(car.Length);

There are many array methods available, for example Sort(), which sorts an array alphabetically or in an ascending order.

Eg: int[] myNumbers = {5, 1, 8, 9};

Array.Sort(myNumbers);

foreach (int i in myNumbers)

{

Console.WriteLine(i);

}

Other useful array methods, such as Min, Max, and Sum, can be found in the System.Linq namespace

Code:

```csharp
using System;

namespace SingleDimensionArray
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Demonstration of 1D Array");
            int[] array1 = { 1, 57, 73, 26, 28, 50 };
            Array.Sort(array1);
            for (int i = 0; i < array1.Length; i++)
            {
                Console.WriteLine(array1[i]);
            }
            Console.Read();

        }
    }
}
```
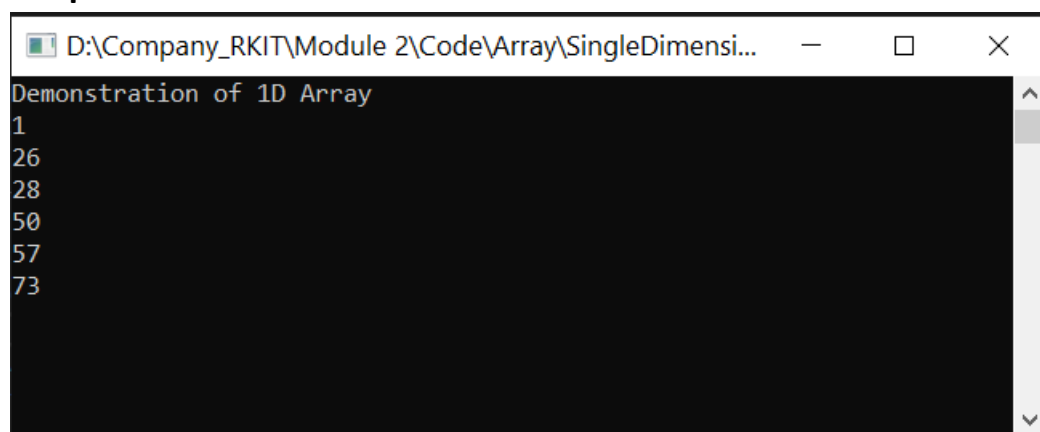
Output:

```
D:\Company_RKIT\Module 2\Code\Array\SingleDimensi...    —    ☐    ✕
Demonstration of 1D Array
1
26
28
50
57
73
```

Multi-dimensional array:

The multi-dimensional array contains more than one row to store the values. It is also known as a Rectangular Array in C# because it's each row length is same.

Initialization and declaration of array can be done as:
// creates a two-dimensional array of 8 rows and 2 columns.
int[,] intarray = new int[8, 2];
//creates an array of three dimensions, 4, 2, and 3
int[, ,] intarray1 = new int[4, 2, 3];

Code:

```csharp
using System;

namespace MultiDimensionArray
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Demonstration of 2D array");
            int[,] array1 = new int[4, 2] { { 51, 20 }, { 43, 46 }, { 85, 61 }, { 87,
80 } };
            for (int i = 0; i < 4; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    Console.WriteLine(array1[i, j]);
                }
            }
            Console.Read();
        }
    }
}
```
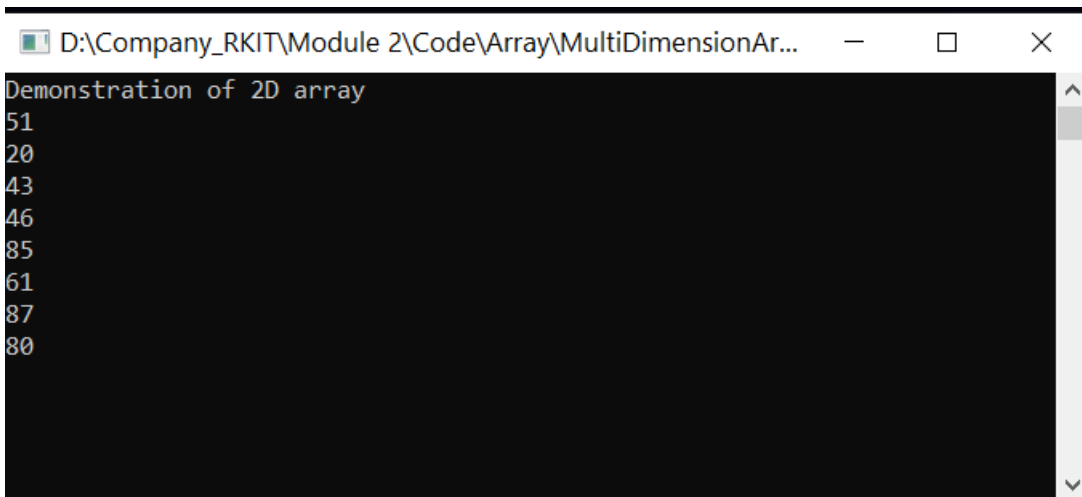
Output:


```
D:\Company_RKIT\Module 2\Code\Array\MultiDimensionAr...   —   □   ×
Demonstration of 2D array
51
20
43
46
85
61
87
80
```
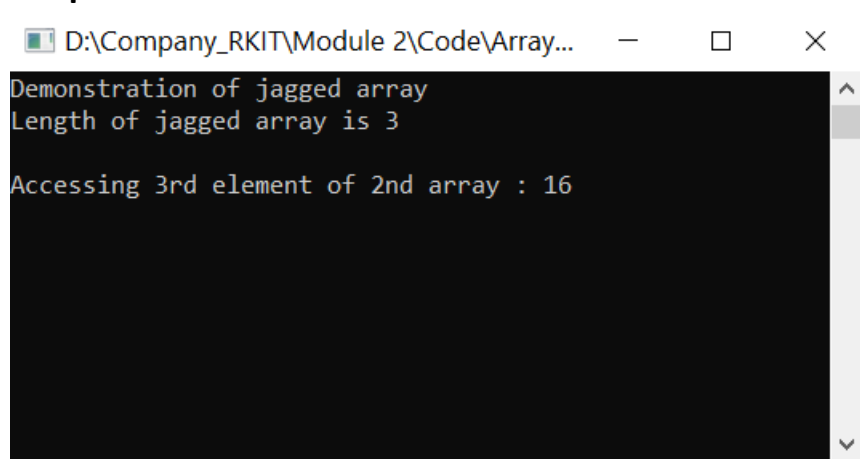
Jagged Array:

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. An example of how to declare, initialize and use jagged arrays is shown below:

Eg: int[][] jaggedArray = new int[3][];
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[2];

Code:

```
using System;

namespace JaggedArray
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Demonstration of jagged array");
      int[][] array1 = new int[3][];
      array1[0] = new int[4] { 54, 78, 92, 56 };
      array1[1] = new int[5] { 15, 80, 16, 20, 27 };
      array1[2] = new int[3] { 38, 90, 12 };
      Console.WriteLine("Length of jagged array is {0}",
      array1.Length);
      Console.WriteLine("\nAccessing 3rd element of 2nd array : {0}",
      array1[1][2]);
      Console.Read();

    }
  }
}
```

Output:

```
D:\Company_RKIT\Module 2\Code\Array...   —   □   ×
Demonstration of jagged array
Length of jagged array is 3

Accessing 3rd element of 2nd array : 16
```

11.   Defining and Calling Methods.
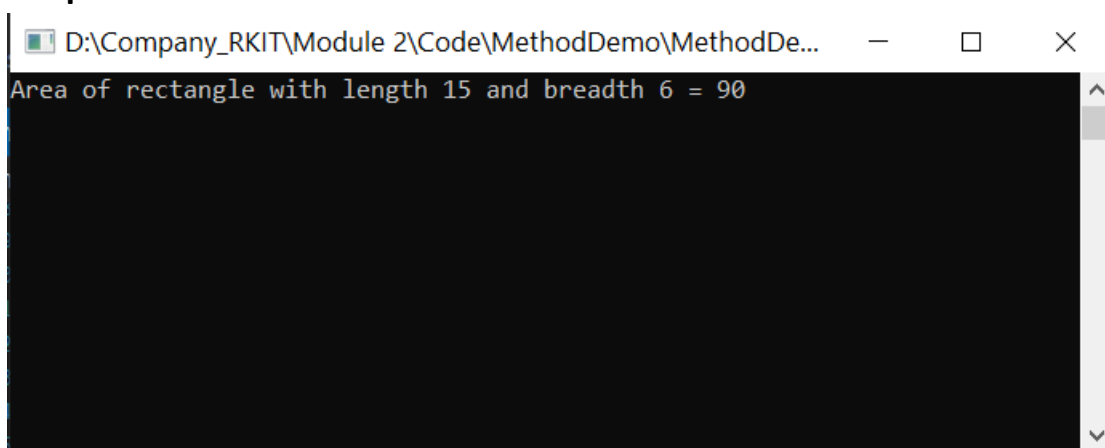➢ **Define method and use**

A method is a group of statements that together perform a task. When you define a method, you basically declare the elements of its structure. You can call a method using the name of the method.

Code:

```csharp
using System;

namespace MethodDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            int l = 15;
            int b = 6;
            int area = r.Area(l, b);
            Console.WriteLine("Area of rectangle with length {0} and breadth {1} = {2} ", l,b,area);
            Console.Read();
        }
    }
    class Rectangle
    {
        public int Area(int length, int breadth)
        {
            int ans = length * breadth;
            return ans;
        }
    }
}
```

Output:

D:\Company_RKIT\Module 2\Code\MethodDemo\MethodDe...

Area of rectangle with length 15 and breadth 6 = 90

➢ **different type of parameters in method (Value type, Ref. type, optional)**

Named parameters: Using named parameters, you can specify the value of the parameter according to their names not their order in the method.

Reference Parameters: The ref is a keyword in C# which is used for passing the value types by reference. In ref parameters, it is necessary that the parameters should initialize before

it pass to ref. We can say that if any changes made in this argument in the method will reflect in that variable when the control return to the calling method.

Out parameters: The out is a keyword in C# which is used for the passing the arguments to methods as a reference type.

Optional parameters or Default parameter: As the name suggests optional parameters are not compulsory parameters, they are optional. It helps to exclude arguments for some parameters.

Dynamic parameters: The parameters pass dynamically means the compiler does not check the type of the dynamic type variable at compile-time, instead of this, the compiler gets the type at the run time. The dynamic type variable is created using a dynamic keyword.

Params: It is useful when the programmer doesn't have any prior knowledge about the number of parameters to be used. By using params you are allowed to pass any variable number of arguments. Only one params keyword is allowed and no additional Params will be allowed in function declaration after a params keyword. The length of params will be zero if no arguments will be passed.

Code:

```csharp
using System;

namespace MethodParameter
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Method with named parameter");
            string fname = "Bansi";
            string lname = "Bhimani";
            string name = fullName(fname, lname);
            Console.WriteLine("Full name is {0}", name);
            Console.WriteLine("\nMethod with ref parameter");
            string refMethodParemeter = "RKIT Company";
            check(ref refMethodParemeter);
            Console.WriteLine("\nMethod with out parameter");
            int number;
            doubleNum(out number);
            Console.WriteLine("Value of num : {0}", number);
            Console.WriteLine("\nMethod with default parameters");
            studentInfo(fname, lname);
            Console.WriteLine("\nMethod with dynamic parameters");
            dynamicValueMethod(50);
            Console.WriteLine("\nMethod with params ");
```

```csharp
        int[] array = { 10, 20, 30, 40, 50 };
        int a = paramMethod(array);
        Console.WriteLine("value returned by paramMethod is {0}", a);
        Console.Read();
    }
    //named parameters
    static string fullName(string s1, string s2)
    {
        return s2 + s1;
    }
    //ref parameters
    static void check(ref string s1)
    {
        if (s1 == "RKIT Company")
            Console.WriteLine("RKIT Company is matched");
        else
            Console.WriteLine("RKIT Comapny is not matched");
    }
    //out parameters
    static void doubleNum(out int num)
    {
        num = 40;
        num *= 2;
    }
    //default parameters
    static void studentInfo(string first, string last, string dept =
    "IT")
    {
        Console.WriteLine("firstname : {0}, lastname : {1}, dept : {2} ", first, last, dept);
    }

    //dynamic parameters
    static void dynamicValueMethod(dynamic val)
    {
        val = val * 5;
        Console.WriteLine("value is {0}", val);
    }
    //params
    static int paramMethod(params int[] number)
    {
        int j = 0;
        foreach (int i in number)
        {
            j = j + i;
        }
        return j;
    }

}

}


    Output:
```

```
Method with named parameter
Full name is BhimaniBansi

Method with ref parameter
RKIT Company is matched

Method with out parameter
Value of num : 80

Method with default parameters
firstname : Bansi, lastname : Bhimani, dept : IT

Method with dynamic parameters
value is 250

Method with params
value returned by paramMethod is 150
```

12. Working with strings.

➤ String class study

In C#, a string is a sequence of Unicode characters or array of characters. 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 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.

Code:

```csharp
using System;

namespace StringDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            char[] charArray = { 'R', 'K', 'I', 'T' };
            string str1 = new string(charArray);
            Console.WriteLine(str1);
            string str2 = new string('B', 3);
            Console.WriteLine(str2);
            Console.Read();
        }
}
```
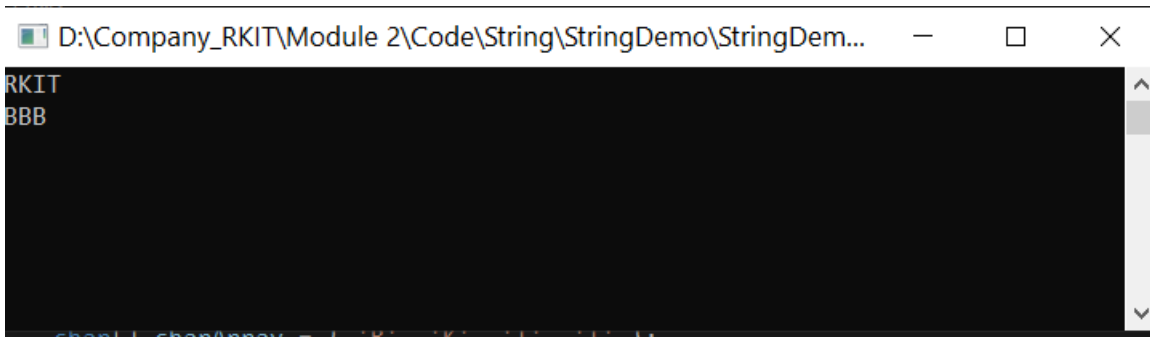
```
        }
    }
```

Output:

➤ Use of various string methods

There are many methods of Strings class which are used to manipulate strings and can be efficiently used for serving the required purpose.

• Clone() - The clone method in C# is used to duplicate a string type object. It returns a clone of the same data as the object type.

• Concate() - A concat method in C# helps combine or concatenate several strings. It returns a combined string.

• Contains() - Contain method in C# is used to determine if a particular substring is present inside a given string or not. Contains method returns a Boolean value, hence if the given substring is present inside the string then it will return "true" and if it is absent then it will return "false".

• Equals()-The Equals method in C# is used to validate if the two given strings are the same or not. If both the strings contain the same value then this method will return true and if they contain different value then this method will return false.

• Indexof() - The IndexOf method in C# is used to find the index of a specific character inside a string. This method provides an index in the form of an integer. It counts the index value starting from zero.

• Insert() - The Insert method in C# is used for inserting a string at a specific index point. As we learned in our earlier, the index method starts with zero. This method inserts the string inside another string and returns a new modified string as the result.

• Replace() - The Replace method in C# is used to replace a certain set of concurrent characters from a given string. It returns a string with characters replaced from the original string. Replace method has two overloads, it can be used to replace both strings as well as characters.

• SubString() - The SubString method in C# is used to get a part of the string from a given string. By using this method, the program can specify a starting index and can get the substring till the end.

- Trim() - The Trim method in C# is used to remove all the whitespace characters at the start and end of a string. It can be used whenever a user needs to remove extra whitespace at the start or end of a given string.

Code:

```csharp
using System;

namespace StringMethod
{
    class Program
    {
        static void Main(string[] args)
        {
            //clone method
            Console.WriteLine("Demonstration of clone method");
            String a = "Hello";
            String b = (String)a.Clone();
            Console.WriteLine(b);

            //Concat method
            Console.WriteLine("\nDemonstration of Concate method");
            String c = "World";
            Console.WriteLine(string.Concat(a, c));

            //Contains method
            Console.WriteLine("\nDemonstration of Contains method");
            String d = "Hello World";
            bool e = d.Contains(c);
            if (e)
                Console.WriteLine("d contains c");
            else
                Console.WriteLine("d does not contains c");

            //Copy method
            Console.WriteLine("\nDemonstration of Copy method");
            String f = string.Copy(d);
            Console.WriteLine(d);

            //Equals method
            Console.WriteLine("\nDemonstration of Equals method");
            Console.WriteLine(a.Equals(c));

            //IndexOf method
            Console.WriteLine("\nDemonstration of IndexOf method");
            int g = a.IndexOf('e');
            Console.WriteLine(g);

            //Insert method
            Console.WriteLine("\nDemonstration of Insert method");
            String h = a.Insert(2, c);
            Console.WriteLine(h);

            //Replace method
```
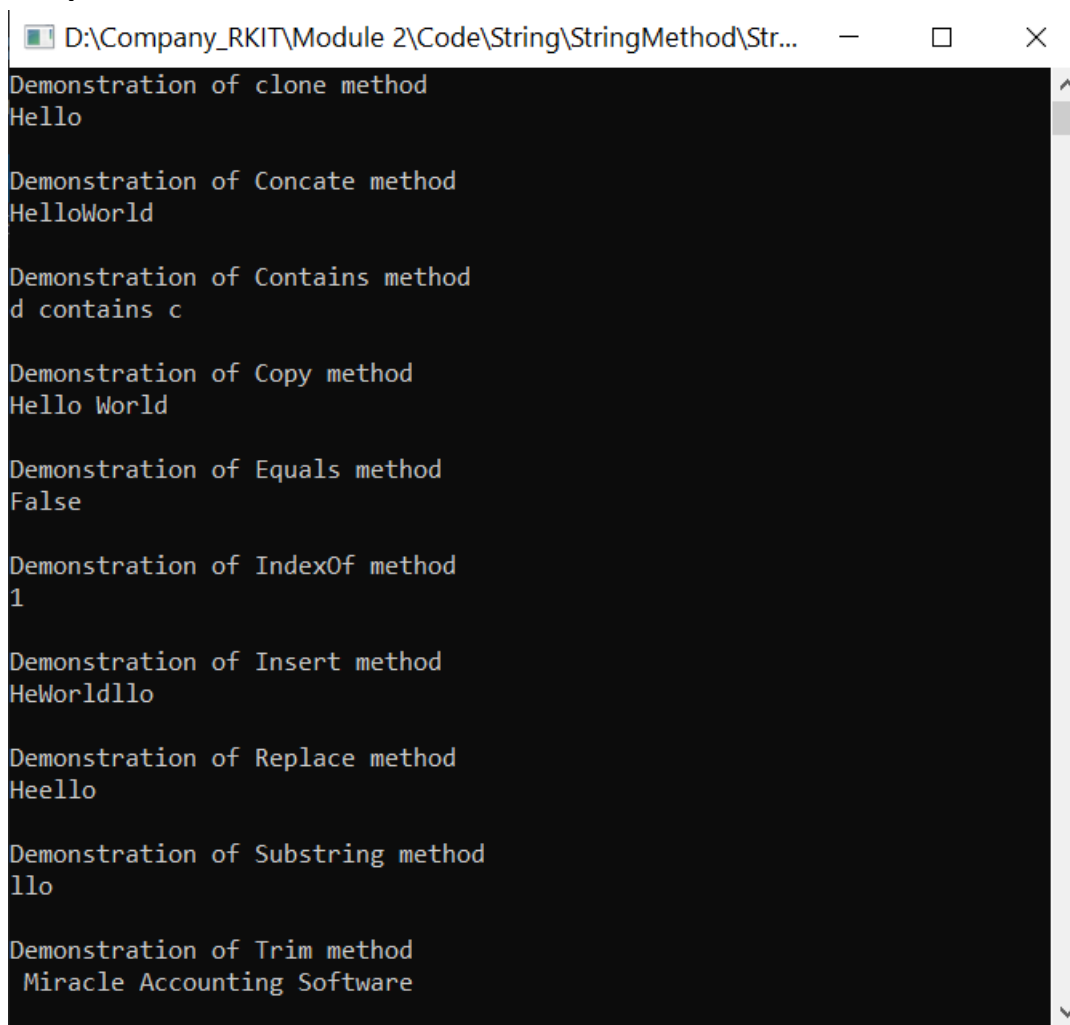
```csharp
        Console.WriteLine("\nDemonstration of Replace method");
        string i = a.Replace("He", "Hee");
        Console.WriteLine(i);

        //Substring method
        Console.WriteLine("\nDemonstration of Substring method");
        string j = a.Substring(2);
        Console.WriteLine(j);

        //Trim method
        Console.WriteLine("\nDemonstration of Trim method");
        string k = " Miracle Accounting Software ";
        Console.WriteLine(k);
        Console.Read();

    }
  }
}
```

Output:


```
D:\Company_RKIT\Module 2\Code\String\StringMethod\Str...    —    □    ×

Demonstration of clone method
Hello

Demonstration of Concate method
HelloWorld

Demonstration of Contains method
d contains c

Demonstration of Copy method
Hello World

Demonstration of Equals method
False

Demonstration of IndexOf method
1

Demonstration of Insert method
HeWorldllo

Demonstration of Replace method
Heello

Demonstration of Substring method
llo

Demonstration of Trim method
 Miracle Accounting Software
```

13.   Date-time.

➢ Date-time class study

C# DateTime is a structure of value Type like int, double etc. It is available in System namespace and present in mscorlib.dll assembly.

DateTime constructor - 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. It contains around 11 overload methods.

DateTime object contains two static read-only fields called as MaxValue and Minvalue.

There are numeroys date and time methods which will be demonstrated in the following program.

Code:

```csharp
using System;

namespace DateTimeDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            //constructors
            DateTime date1 = new DateTime(2021, 12, 24);
            Console.WriteLine(date1.ToString());
            DateTime date2 = new DateTime(2020, 12, 24, 10, 30, 50);
            Console.WriteLine(date1.ToString());
            Console.WriteLine();

            //DateTime Fields
            Console.WriteLine(DateTime.MinValue);
            Console.WriteLine(DateTime.MaxValue);
            Console.WriteLine();

            //DateTime Methods
            Console.WriteLine("DateTime Methods");
            Console.WriteLine();

            //Add methods that adds duration
            Console.WriteLine("Add Method");
            System.TimeSpan duration = new System.TimeSpan(30, 0, 0, 0);
            System.DateTime newDate1 = DateTime.Now.Add(duration);
            Console.WriteLine(newDate1);
            Console.WriteLine();

            // Parsing Method
            string dateString = "Wed Dec 29, 2021";
            DateTime dateTime12 = DateTime.Parse(dateString);
            Console.WriteLine(dateTime12);
            Console.WriteLine();

            // Date Difference Method
            System.DateTime date3 = new System.DateTime(2021, 3, 10, 2, 15, 10);
            System.DateTime date4 = new System.DateTime(2021, 7, 15, 6, 30, 20);
```

```csharp
        System.TimeSpan diff1 = date4.Subtract(date2);
        Console.WriteLine(diff1.ToString());
        Console.WriteLine();

        //Universal time
        DateTime dateObj = new DateTime(2021, 12, 20, 10, 20, 30);
        Console.WriteLine("universal time : " + dateObj.ToUniversalTime());

        //DateTime properties
        Console.WriteLine("\nDateTime Properties are used as follows");
        DateTime myDate = new DateTime(2021, 12, 24, 10, 30, 45);
        int year = myDate.Year;
        Console.WriteLine("year - {0}", year);
        int month = myDate.Month;
        Console.WriteLine("month - {0}", month);
        int day = myDate.Day;
        Console.WriteLine("day - {0}", day);
        int hour = myDate.Hour;
        Console.WriteLine("Hour - {0}", hour);
        int minute = myDate.Minute;
        Console.WriteLine("minute - {0}", minute);
        int second = myDate.Second;
        Console.WriteLine("second - {0}", second);
        int weekDay = (int)myDate.DayOfWeek;
        Console.WriteLine("day of week - {0}", weekDay);
        Console.Read();

    }
  }
}
```

Output:

```
D:\Company_RKIT\Module 2\Code\DateTimeDemo\DateTimeDemo\bin\Debug\DateTimeDemo.exe        —    □    ×

24-12-2021 00:00:00
24-12-2021 00:00:00

01-01-0001 00:00:00
31-12-9999 23:59:59

DateTime Methods

Add Method
11-11-2021 15:08:25

29-12-2021 00:00:00

202.19:59:30

universal time : 20-12-2021 04:50:30

DateTime Properties are used as follows
year - 2021
month - 12
day - 24
Hour - 10
minute - 30
second - 45
day of week - 5
```