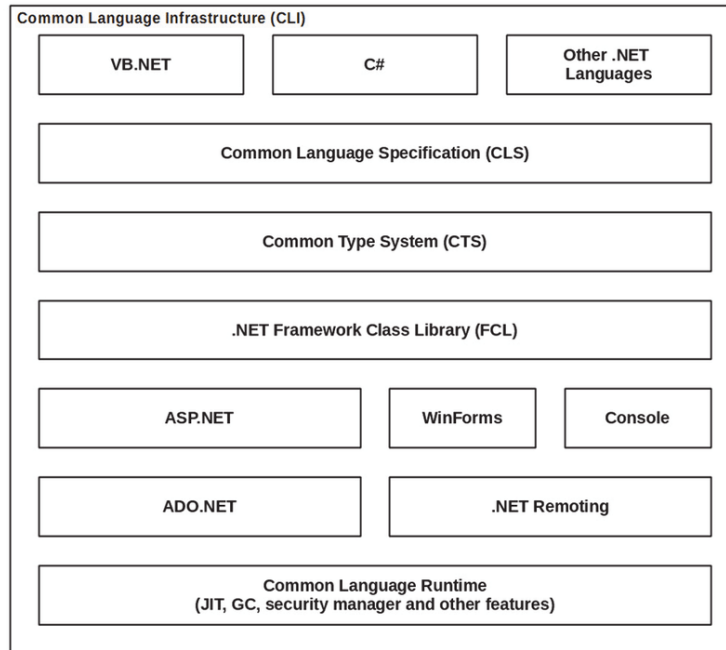


UNIT-I

1. .NET Framework Architecture.

Ans: The .NET Framework is a software development platform developed by Microsoft. It provides a controlled programming environment where software can be developed, installed, and executed on Windows-based operating systems.



Components of Framework:

- **Common Language Infrastructure (CLI):** The CLI in .NET is a specification that describes the executable code and runtime environment that form the core of the .NET Framework.
- **Application Layer:** VB.NET, C# etc. are different languages and technologies that you can use to build your applications in the Application Layer.
- **Common Language Specification (CLS):** The CLS in .NET is a set of basic language features that .NET languages need to agree upon to ensure interoperability. In other words, it's a contract or a set of rules that all .NET languages follow, which allows them to communicate with each other seamlessly. For example, if you write a library in C#, and you stick to the CLS rules, then a VB.NET program can use your library without any issues, and vice versa. This is because both C# and VB.NET adhere to the CLS, ensuring they can understand each other's code.
- **Common Type System (CTS):** The CTS in .NET is a standard that defines the rules for declaring, using, and managing types in the .NET environment, across all .NET languages. In simple terms, it's like a universal set of data types. Whether you're using C#, VB.NET, or any other .NET language, they all understand these common types. For example, when you declare an integer in C#, it's the same as an integer in VB.NET. This ensures that objects written in different .NET languages can interact with each other seamlessly.
- **Framework Class Library (FCL):** The .NET FCL is a large collection of pre-written classes, interfaces, and value types that you can use in your .NET applications. Think of it as a toolbox full of different tools that you can use to build your application. These tools include classes for file input/output, database interaction, XML manipulation, web development, threading, and many more. For example, if you need to read data from a file, you don't need to write that code from scratch. You can use the *StreamReader* class from the FCL.
- **Windows Forms:** This is a framework for building Windows desktop applications. It provides a large set of controls (like buttons, text boxes, etc.) that you can drag and drop onto your forms, and then write code for their events.

- **Console Applications:** These are simple applications that take input and display output at the command line.
- **ADO.NET:** This is a set of classes in the .NET Framework that provides access to data sources such as SQL Server, Oracle, XML, etc.
- **.NET Remoting:** This is a mechanism for communication between objects which are not in the same process.
- **Common Language Runtime (CLR):** The CLR in .NET is the engine that handles the execution of .NET applications. The CLR provides several important services like:
 - **Object Orientation:** This is a programming style that involves organizing data and functions into structures called "objects". It helps make code more understandable, flexible, and reusable.
 - **Type Safety:** This is a feature of a programming language that prevents or warns against type errors, such as trying to perform an operation on a data type that it's not meant for.
 - **Memory Management:** This is how a program handles and organizes the memory it uses. In some languages, programmers must do this manually, but in others (like .NET languages), it's mostly handled automatically.
 - **Platform Support:** This refers to the different operating systems and hardware that a piece of software can run on. For example, a platform-supporting language like C# can run on Windows, Linux, and macOS.

2. A brief history of C#.

Ans: C# is a modern, object-oriented programming language developed by Microsoft. Here's a brief history:

- **2000:** Microsoft started developing C# as part of its .NET initiative. It was designed by Anders Hejlsberg, who had previously created Turbo Pascal and was a major contributor to Delphi.
- **2002:** Microsoft released the first version of C#, C# 1.0, along with Visual Studio .NET.
- **2005:** C# 2.0 was released, introducing significant enhancements like generics, anonymous methods, and nullable types.
- **2007:** C# 3.0 was released, introducing features like LINQ (Language Integrated Query), lambda expressions, and extension methods.
- **2010:** C# 4.0 was released, introducing dynamic binding, named and optional arguments, and more.
- **2012:** C# 5.0 was released, introducing async programming and caller information attributes.
- **2015:** C# 6.0 was released, introducing string interpolation, expression-bodied members, and more.
- **2017:** C# 7.0 was released, introducing tuples, pattern matching, and more.
- **2019:** C# 8.0 was released, introducing nullable reference types, switch expressions, and more.
- **2020:** C# 9.0 was released, introducing record types, init only setters, and more.
- **2021:** C# 10.0 was released, introducing record structs, global using directives, and more.

3. A First C# Program.

Ans: Here's the first program in C#:

Program.cs:

```
using System;
```

```
namespace MyFirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            // Basic calculation
            int x = 12 * 30;

            // Printing the result
            Console.WriteLine("Result of 12 * 30 = " + x);

            // Using a reusable method
            Console.WriteLine("FeetToInches(30) = " + FeetToInches(30));
        }
    }
}
```

```

        Console.WriteLine("FeetToInches(100) = " + FeetToInches(100));

        // Calling a method without input
        SayHello();

        // Compilation and running
        Console.WriteLine("Compiled and Run Successfully!");
    }

    // Method to convert feet to inches
    static int FeetToInches(int feet)
    {
        int inches = feet * 12;
        return inches;
    }

    // Method to say hello
    static void SayHello()
    {
        Console.WriteLine("Hello, world!");
    }
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Result of 12 * 30 = 360
FeetToInches(30) = 360
FeetToInches(100) = 1200
Hello, world!
Compiled and Run Successfully!

```

4. Syntax.

Ans: Here is a simple program for syntax:

Program.cs:

using System;

namespace SyntaxExample

{

class Program

{

static void Main(string[] args)

{

// Calculate and print the result

int x = 12 * 30;

Console.WriteLine(x);

// Demonstrate the use of identifiers and keywords

int @using = 123; // Using a reserved keyword as an identifier

Console.WriteLine(@using);

// Demonstrate contextual keywords

int ascending = 5; // Contextual keyword used as an identifier

Console.WriteLine(ascending);

// Demonstrate literals

```

int num1 = 12;
int num2 = 30;
Console.WriteLine($"Literals used: {num1} and {num2}");

// Demonstrate punctuators and operators
int sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
Console.WriteLine($"Sum: {sum}");

// Single-line comment
int y = 3; // Comment about assigning 3 to y
Console.WriteLine($"Value of y: {y}");

/* Multi-line comment
   demonstrating assignment */
int z = 5; /* This is a comment that
   spans two lines */
Console.WriteLine($"Value of z: {z}");
}
}
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
360
123
5
Literals used: 12 and 30
Sum: 55
Value of y: 3
Value of z: 5

```

5. Type Basics.

Ans: Here is a simple program for Type basics:

Program.cs:

using System;

```

// Define a custom struct for Point representing a value type
public struct Point
{
    public int X, Y; // Define fields for X and Y coordinates
}

```

// Define a custom class for Panda representing a reference type

```

public class Panda
{
    public string Name; // Define a field for Panda's name
    public static int Population; // Define a static field for Panda population

    // Constructor to initialize Panda with a name
    public Panda(string n)
    {
        Name = n; // Assign the instance field
        Population++; // Increment the static Population field
    }
}

```

```
// Main entry point of the program
class Program
{
    static void Main()
    {
        // Demonstrate predefined types and operations
        int x = 12 * 30;
        Console.WriteLine("Result of operation: " + x);

        // Demonstrate custom types and constructors
        Point p1 = new Point(); // Instantiate a Point struct
        p1.X = 10;
        p1.Y = 20;
        Console.WriteLine("Point p1 coordinates: (" + p1.X + ", " + p1.Y + ")");

        // Demonstrate reference types and static members
        Panda panda1 = new Panda("Pan Dee"); // Create a Panda instance
        Panda panda2 = new Panda("Pan Dah"); // Create another Panda instance
        Console.WriteLine("Panda 1 name: " + panda1.Name);
        Console.WriteLine("Panda 2 name: " + panda2.Name);
        Console.WriteLine("Total Pandas: " + Panda.Population);

        // Demonstrate value types and conversions
        int integerValue = 12345;
        long longValue = integerValue; // Implicit conversion to long
        short shortValue = (short)integerValue; // Explicit conversion to short
        Console.WriteLine("Long value: " + longValue);
        Console.WriteLine("Short value: " + shortValue);
    }
}
```

Output:

```
PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Result of operation: 360
Point p1 coordinates: (10, 20)
Panda 1 name: Pan Dee
Panda 2 name: Pan Dah
Total Pandas: 2
Long value: 12345
Short value: 12345
```

6. Numeric Types, Boolean type, and operators.

Ans: Here is a simple program for Numeric Types, Boolean type, and operators:

Program.cs:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        // Numeric Types
        byte myByte = 1; // Size: 1 byte
        sbyte mySByte = -1; // Size: 1 byte
        short myShort = 2; // Size: 2 bytes
        ushort myUShort = 3; // Size: 2 bytes
        int myInt = 10; // Size: 4 bytes
        uint myUInt = 11; // Size: 4 bytes
    }
}
```

```

long myLong = 12; // Size: 8 bytes
ulong myULong = 13; // Size: 8 bytes
float myFloat = 3.14f; // Size: 4 bytes
double myDouble = 3.14; // Size: 8 bytes
decimal myDecimal = 3.14m; // Size: 16 bytes

// Boolean Type
bool myBool = true; // Size: 1 byte

// Char and String
char myChar = 'A'; // Size: 2 bytes
string myString = "Hello, World!"; // Size: Variable

// Operators
int a = 5;
int b = 3;
int sum = a + b; // Addition
int difference = a - b; // Subtraction
int product = a * b; // Multiplication
int quotient = a / b; // Division
int remainder = a % b; // Modulus
int increment = a++; // Increment
int decrement = b--; // Decrement
bool isEqual = a == b; // Equality
bool isNotEqual = a != b; // Inequality
bool isGreater = a > b; // Greater than
bool isLess = a < b; // Less than
bool isGreaterOrEqual = a >= b; // Greater than or equal to
bool isLessOrEqual = a <= b; // Less than or equal to
bool logicalAnd = (a > 0) && (b > 0); // Logical AND
bool logicalOr = (a > 0) || (b > 0); // Logical OR
bool logicalNot = !(a > 0); // Logical NOT

Console.WriteLine(myByte);
Console.WriteLine(mySByte);
Console.WriteLine(myShort);
Console.WriteLine(myUShort);
Console.WriteLine(myInt);
Console.WriteLine(myUInt);
Console.WriteLine(myLong);
Console.WriteLine(myULong);
Console.WriteLine(myFloat);
Console.WriteLine(myDouble);
Console.WriteLine(myDecimal);
Console.WriteLine(myBool);
Console.WriteLine(myChar);
Console.WriteLine(myString);
Console.WriteLine(sum);
Console.WriteLine(difference);
Console.WriteLine(product);
Console.WriteLine(quotient);
Console.WriteLine(remainder);
Console.WriteLine(increment);
Console.WriteLine(decrement);
Console.WriteLine(isEqual);
Console.WriteLine(isNotEqual);

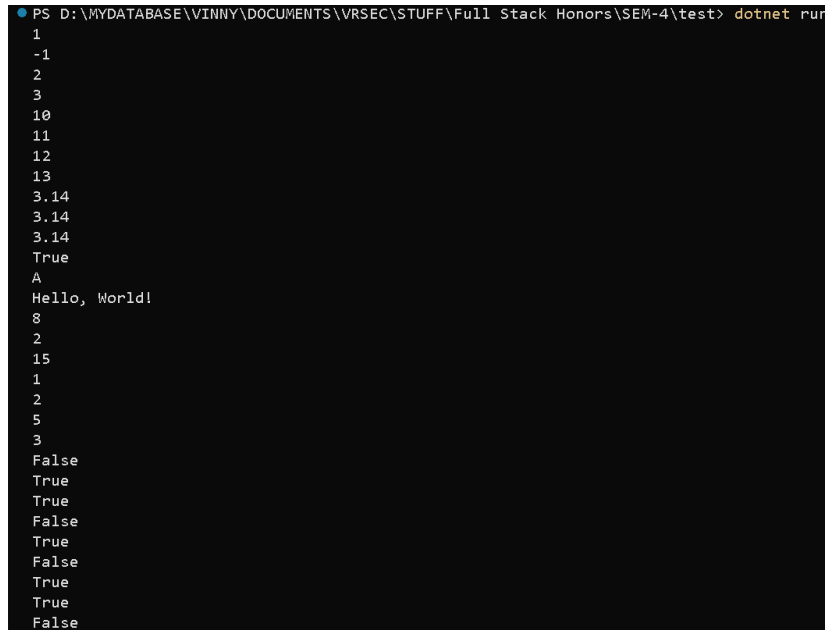
```

```

        Console.WriteLine(isGreater);
        Console.WriteLine(isLess);
        Console.WriteLine(isGreaterOrEqual);
        Console.WriteLine(isLessOrEqual);
        Console.WriteLine(logicalAnd);
        Console.WriteLine(logicalOr);
        Console.WriteLine(logicalNot);
    }
}

```

Output:



```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
1
-1
2
3
10
11
12
13
3.14
3.14
3.14
True
A
Hello, World!
8
2
15
1
2
5
3
False
True
True
False
True
False
True
True
False

```

7. Strings and Characters.

Ans: Here is a simple program for Strings and Characters:

Program.cs:

```

using System;

class Program
{
    static void Main()
    {
        // Char type and literals
        char c = 'A';
        // char newLine = '\n';
        // char backSlash = '\\';
        // char copyrightSymbol = '\u00A9';
        // char omegaSymbol = '\u03A9';

        // Char conversions
        int charToInt = (int)c;
        double charToDouble = (double)c;

        // String type and literals
        string a = "Heat";
        string b = "test";
        string tabString = "Here's a tab:\t";
        string escaped = "First Line\r\nSecond Line";
        string verbatim = @"First Line

```

```

Second Line";
    string xml = @"<customer id=""123""></customer>";

    // String concatenation
    string concatenatedString = "a" + "b";

    // String interpolation
    int x = 4;
    string interpolatedString = $"A square has {x} sides";
    string hexString = $"255 in hex is {byte.MaxValue:X2}";
    bool boolValue = true;
    string conditionalString = $"The answer in binary is {(boolValue ? 1 : 0)}";
    string multilineInterpolation = $"this interpolation spans {
        x} lines";

    // String comparisons
    bool areEqual = a == b;
    int comparisonResult = a.CompareTo(b);

    // Constant interpolated strings (C# 10)
    const string greeting = "Hello";
    const string message = $"{greeting}, world";

    // Output
    Console.WriteLine($"Character: {c}");
    Console.WriteLine($"Char to Int: {charToInt}");
    Console.WriteLine($"Char to Double: {charToDouble}");
    Console.WriteLine(tabString);
    Console.WriteLine(escaped);
    Console.WriteLine(verbatim);
    Console.WriteLine(xml);
    Console.WriteLine($"Concatenated String: {concatenatedString}");
    Console.WriteLine(interpolatedString);
    Console.WriteLine(hexString);
    Console.WriteLine(conditionalString);
    Console.WriteLine(multilineInterpolation);
    Console.WriteLine($"Are strings a and b equal? {areEqual}");
    Console.WriteLine($"Comparison result of strings a and b:
{comparisonResult}");
    Console.WriteLine(message);
}
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Character: A
Char to Int: 65
Char to Double: 65
Here's a tab:
First Line
Second Line
First Line
Second Line
<customer id="123"></customer>
Concatenated String: ab
A square has 4 sides
255 in hex is FF
The answer in binary is 1
this interpolation spans 4 lines
Are strings a and b equal? False
Comparison result of strings a and b: -1
Hello, world

```


8. Arrays.

Ans: Here is a simple program for arrays:

Program.cs:

using System;

class Program

{

static void Main()

{

// Single-dimensional array

char[] vowels = new char[5];

vowels[0] = 'a';

vowels[1] = 'e';

vowels[2] = 'i';

vowels[3] = 'o';

vowels[4] = 'u';

// Accessing elements

Console.WriteLine(vowels[1]); // Output: e

// Iterating through the array

Console.Write("All vowels: ");

for (int i = 0; i < vowels.Length; i++)

{

Console.Write(vowels[i]);

}

Console.WriteLine();

// Array initialization expression

char[] vowelsInit = new char[] { 'a', 'e', 'i', 'o', 'u' };

// Or simply:

// char[] vowelsInit = { 'a', 'e', 'i', 'o', 'u' };

// Value types versus reference types

Point[] points = new Point[1000];

for (int i = 0; i < points.Length; i++)

{

points[i] = new Point(); // Explicitly instantiate each Point

points[i].X = i;

points[i].Y = i;

}

// Indices and ranges

char[] lastTwo = vowels[^2..]; // 'o', 'u'

// Multidimensional arrays - rectangular

Console.WriteLine("Rectangular array:");

int[,] matrix = new int[3, 3];

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

matrix[i, j] = i * 3 + j;

Console.Write(matrix[i, j] + " ");

}

Console.WriteLine();

```

    }

    // Jagged arrays
    Console.WriteLine("Jagged array:");
    int[][] jaggedMatrix = new int[3][];
    for (int i = 0; i < jaggedMatrix.Length; i++)
    {
        jaggedMatrix[i] = new int[3];
        for (int j = 0; j < jaggedMatrix[i].Length; j++)
        {
            jaggedMatrix[i][j] = i * 3 + j;
            Console.Write(jaggedMatrix[i][j] + " ");
        }
        Console.WriteLine();
    }

    // Simplified array initialization expressions
    char[] simplifiedVowels = { 'a', 'e', 'i', 'o', 'u' };
    int[,] simplifiedRectMatrix =
    {
        {0, 1, 2},
        {3, 4, 5},
        {6, 7, 8}
    };
    int[][] simplifiedJaggedMatrix =
    {
        new int[] {0, 1, 2},
        new int[] {3, 4, 5},
        new int[] {6, 7, 8, 9}
    };

    // Bounds checking
    int[] arr = new int[3];
    try
    {
        arr[3] = 1; // IndexOutOfRangeException thrown
    }
    catch (IndexOutOfRangeException e)
    {
        Console.WriteLine(e.Message);
    }
}

// Example struct for value type demonstration
public struct Point
{
    public int X, Y;
}
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
e
All vowels: aeiou
Rectangular array:
0 1 2
3 4 5
6 7 8
Jagged array:
0 1 2
3 4 5
6 7 8
Index was outside the bounds of the array.

```

9. Variables and Parameters.

Ans: Here is a simple program for Variables and Parameters:

Program.cs:

```

using System;
using System.Text;

```

```

class Program
{
    static void Main(string[] args)
    {
        // Arrays
        char[] vowels = { 'a', 'e', 'i', 'o', 'u' };
        Console.WriteLine("Vowels:");
        foreach (var vowel in vowels)
        {
            Console.Write(vowel + " ");
        }
        Console.WriteLine();

        // Optional parameters
        DisplayNumber(); // Output: 10
        DisplayNumber(5); // Output: 5
        DisplayNumber(y: 7); // Output: 7

        // Ref locals
        int[] numbers = { 0, 1, 2, 3, 4 };
        ref int numRef = ref numbers[2];
        numRef *= 10;
        Console.WriteLine("Array after modifying ref local:");
        foreach (var number in numbers)
        {
            Console.Write(number + " ");
        }
        Console.WriteLine();

        // Ref returns
        ref string xRef = ref GetX();
        xRef = "New Value";
        Console.WriteLine("Value of x after ref return modification: " + xRef);

        // Implicitly typed local variables (var)
        var message = "hello";
        var sb = new StringBuilder();
        var pi = (float)Math.PI;
    }
}

```

```

        Console.WriteLine("Implicitly typed variables: " + message + ", " +
sb.ToString() + ", " + pi);

        // Target-typed new expressions
        StringBuilder sb1 = new();
        StringBuilder sb2 = new("Test");
        Console.WriteLine("Target-typed new expressions: " + sb1.ToString() + ", "
+ sb2.ToString());
    }

    // Optional parameters
    static void DisplayNumber(int x = 10, int y = 0)
    {
        Console.WriteLine("Value of x: " + x);
    }

    // Ref returns
    static string x = "Old Value";
    static ref string GetX() => ref x;
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Vowels:
a e i o u
Value of x: 10
Value of x: 5
Value of x: 10
Array after modifying ref local:
0 1 20 3 4
Value of x after ref return modification: New Value
Implicitly typed variables: hello, , 3.1415927
Target-typed new expressions: , Test

```

10. Null Operators.

Ans: Here is a simple program for null operators:

Program.cs:

```

using System;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        // Null-Coalescing Operator
        string s1 = null;
        string s2 = s1 ?? "nothing";
        Console.WriteLine("Null-Coalescing Operator: " + s2); // Output: "nothing"

        // Null-Coalescing Assignment Operator
        string myVariable = null;
        string someDefault = "defaultValue";
        myVariable ??= someDefault;
        Console.WriteLine("Null-Coalescing Assignment Operator: " + myVariable); //
Output: "defaultValue"

        // Null-Conditional Operator
        StringBuilder sb = null;

```

```

        string s = sb?.ToString();
        Console.WriteLine("Null-Conditional Operator: " + s); // Output: null

        // Combining null-conditional operator with null-coalescing operator
        string result = sb?.ToString() ?? "nothing";
        Console.WriteLine("Combining Operators: " + result); // Output: "nothing"
    }
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Null-Coalescing Operator: nothing
Null-Coalescing Assignment Operator: defaultValue
Null-Conditional Operator:
Combining Operators: nothing

```

11. Statements.

Ans: Here is a simple program for statements:

Program.cs:

using System;

class Program

```

{
    static void Main(string[] args)
    {
        // Declaration Statements
        string someWord = "rosebud";
        int someNumber = 42;
        bool rich = true, famous = false;
        const double c = 2.99792458E08;
        // Constant declaration cannot be changed
        // c += 10; // Compile-time Error

        // Local variables
        int x;
        {
            int declarable;
            // int x; // Error - x already defined
        }
        // Console.Write(y); // Error - y is out of scope

        Console.WriteLine(someWord);
        Console.WriteLine(someNumber);
        Console.WriteLine(rich);
        Console.WriteLine(famous);
        Console.WriteLine(c);

        // Expression Statements
        int xValue, yValue;
        System.Text.StringBuilder sb;
        xValue = 1 + 2; // Assignment expression
        xValue++; // Increment expression
        yValue = Math.Max(xValue, 5); // Assignment expression
        Console.WriteLine(yValue); // Method call expression
        sb = new System.Text.StringBuilder(); // Assignment expression
        new System.Text.StringBuilder(); // Object instantiation expression
    }
}

```

```

// Selection Statements: if statement
if (5 < 2 * 3)
{
    Console.WriteLine("true");
    Console.WriteLine("Let's move on!");
}

// if-else statement
if (2 + 2 == 5)
    Console.WriteLine("Does not compute");
else
    Console.WriteLine("False");

// Nested if-else statement
if (2 + 2 == 5)
    Console.WriteLine("Does not compute");
else if (2 + 2 == 4)
    Console.WriteLine("Computes");

// Switch statement
int cardNumber = 13;
switch (cardNumber)
{
    case 13:
        Console.WriteLine("King");
        break;
    case 12:
        Console.WriteLine("Queen");
        break;
    case 11:
        Console.WriteLine("Jack");
        break;
    default:
        Console.WriteLine(cardNumber);
        break;
}

// Iteration Statements: while loop
int i = 0;
while (i < 3)
{
    Console.WriteLine(i);
    i++;
}

// do-while loop
i = 0;
do
{
    Console.WriteLine(i);
    i++;
} while (i < 3);

// for loop
for (int j = 0; j < 3; j++)
{

```

```

        Console.WriteLine(j);
    }

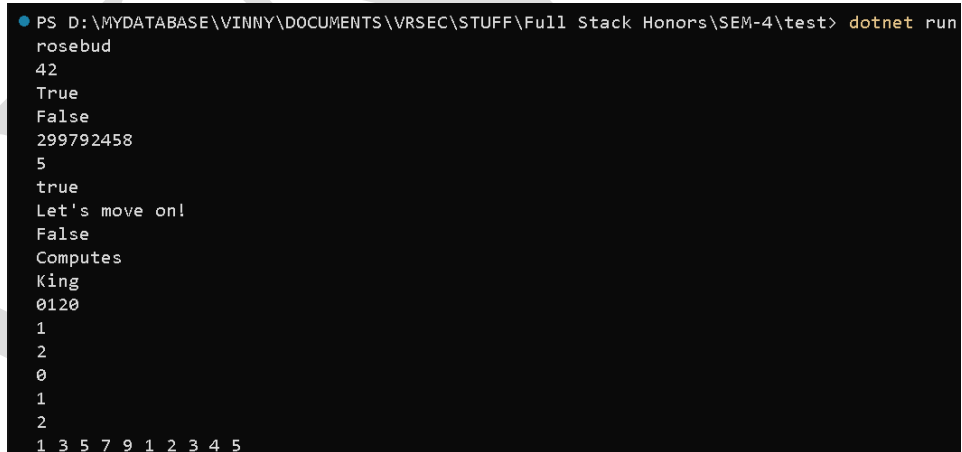
    // Jump Statements: break
    int y = 0;
    while (true)
    {
        if (y++ > 5)
            break;
    }

    // continue
    for (int k = 0; k < 10; k++)
    {
        if ((k % 2) == 0)
            continue;
        Console.Write(k + " ");
    }

    // goto
    int z = 1;
startLoop:
    if (z <= 5)
    {
        Console.Write(z + " ");
        z++;
        goto startLoop;
    }
}

```

Output:



```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
rosebud
42
True
False
299792458
5
true
Let's move on!
False
Computes
King
0120
1
2
0
1
2
1 3 5 7 9 1 2 3 4 5

```

12. Namespaces.

Ans: Here is a simple program for Namespaces:

Program.cs:

```

using System;
using Outer.Middle.Inner; // Importing namespace using directive

namespace Outer.Middle.Inner
{
    class Class1
    {

```

```

        public void Display()
        {
            Console.WriteLine("Inside Outer.Middle.Inner.Class1");
        }
    }
}

namespace Outer
{
    namespace Middle
    {
        namespace Inner
        {
            class Class2
            {
                public void Display()
                {
                    Console.WriteLine("Inside Outer.Middle.Inner.Class2");
                }
            }
        }
    }
}

namespace MyNamespace
{
    class Program
    {
        static void Main(string[] args)
        {
            // Using fully qualified name
            Outer.Middle.Inner.Class1 obj1 = new Outer.Middle.Inner.Class1();
            obj1.Display();

            // Using nested using directive
            Class2 obj2 = new Class2();
            obj2.Display();
        }
    }
}

```

Output:

```

PS D:\MYDATABASE\VINNY\DOCUMENTS\VRSEC\STUFF\Full Stack Honors\SEM-4\test> dotnet run
Inside Outer.Middle.Inner.Class1
Inside Outer.Middle.Inner.Class2

```