C# [CSharp] Programming.

C# vs .Net

* C# is a programming Language.

* .Net is a framework for building applications on Windows, the dotnet framework is Not Limited to C#. There are many Languages in it.

.Net :-

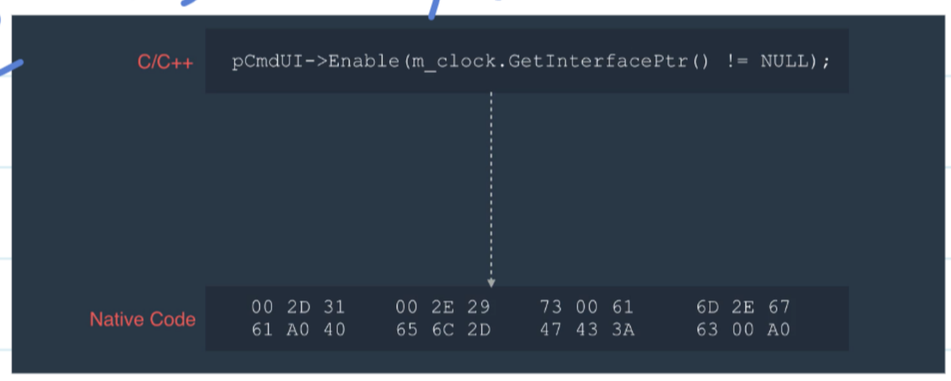
.Net consists of two components.

① CLR [Common Language Runtime]
② Class Library for developing applications.

CLR [Common Language Runtime]

Before going to CLR, we need to know a bit of history of C#.

Before C#, we have two Languages in C family ① C Language ② C++
In the both cases, the compiler will turn the code by converting it into its Native Code of that machine in which it was wrote.



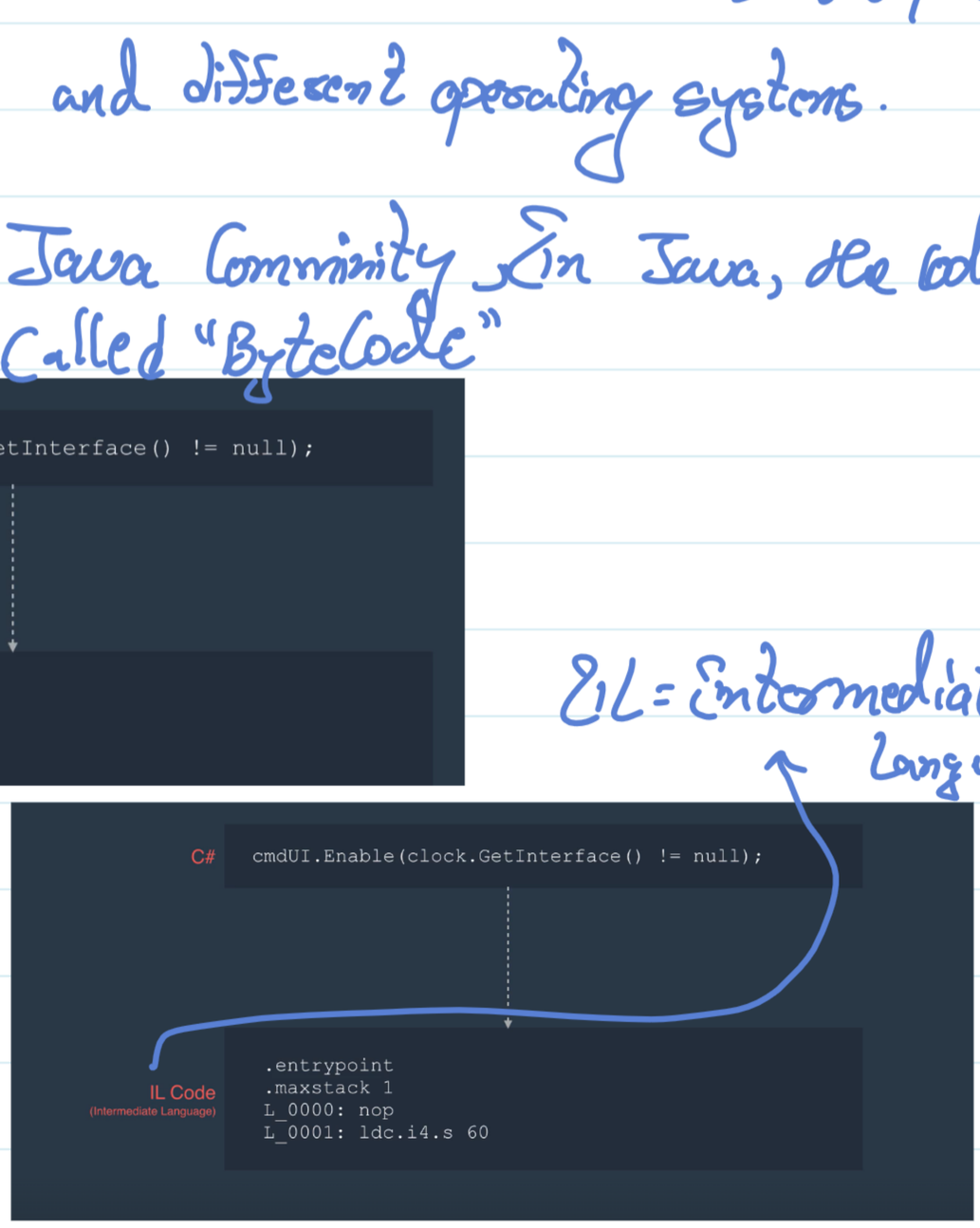
→ It means if we are writing on windows → it will write the native code, only for that particular machine, which has some processor.

Now, we have different hardwares and different operating systems.

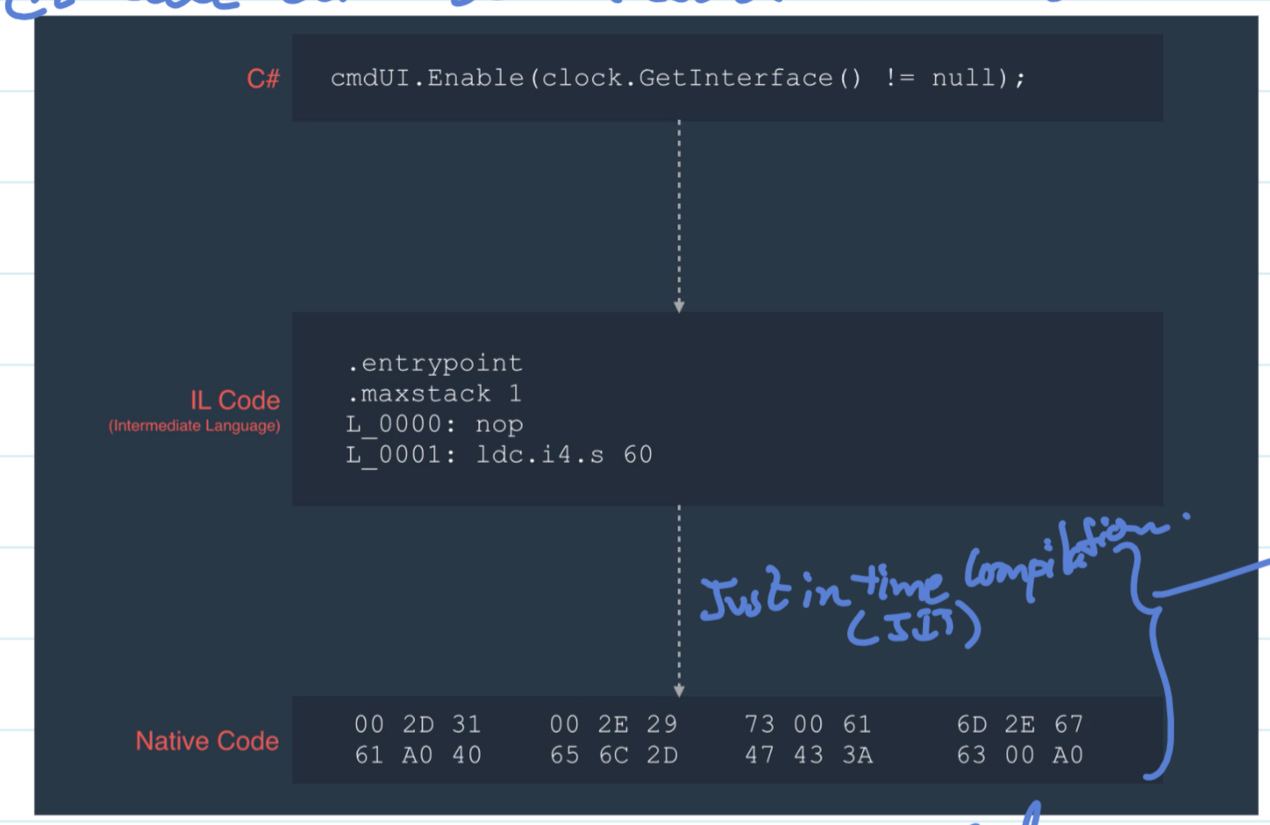
So Microsoft, took an concept of Java Community (In Java, the code will convert to an intermediate Code Called "ByteCode"

IL = Intermediate Language Code

the same Concept, we have in C#



then the IL code will be converted to Native code to run the application.

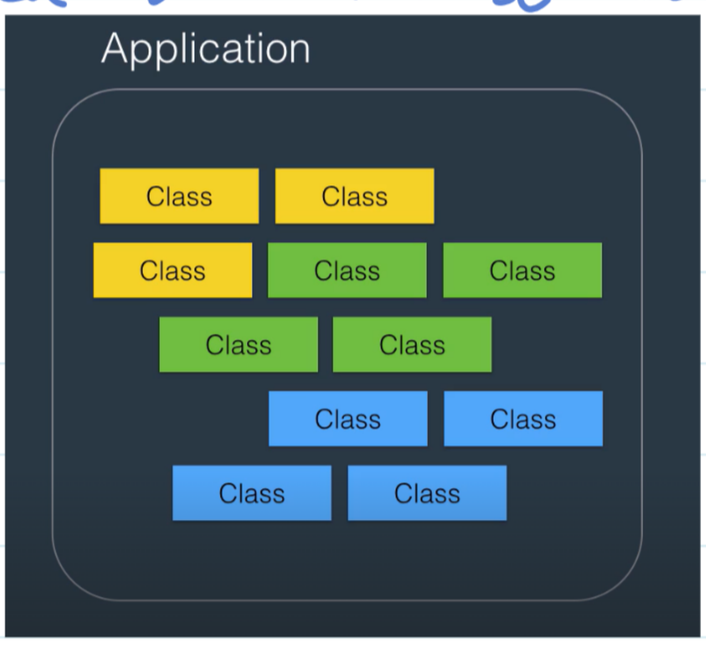


So, CLR is an application, sitting in the memory (Central) → It's job is to convert the IL code to machine code (or) Native code → this process is called "Just in time Compilation [JIT]"

So, in this case we can write a code in windox, & it will work in any machine as long as the other machines has CLR application in those system.

Architecture of .Net Applications.

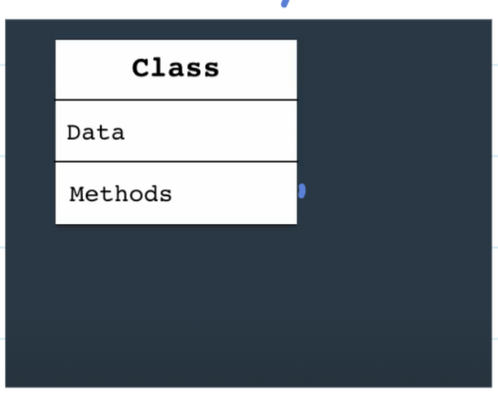
at an very high level, when we built an application. then the application will have building blocks called Class. → these Class Collebarate with each other. at run time

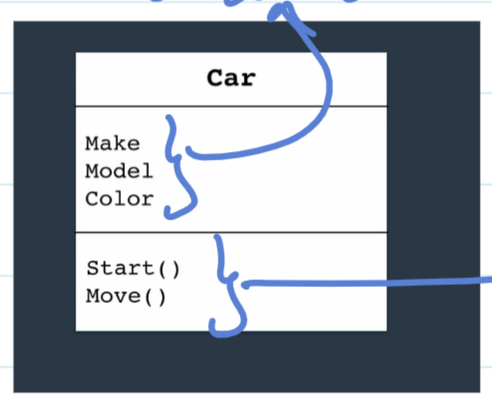


what is a class?

a class is an container that has some data called attributes and some function called methods. the functions (or) methods have the behaviors, they Exibute the Code. they do things for us.

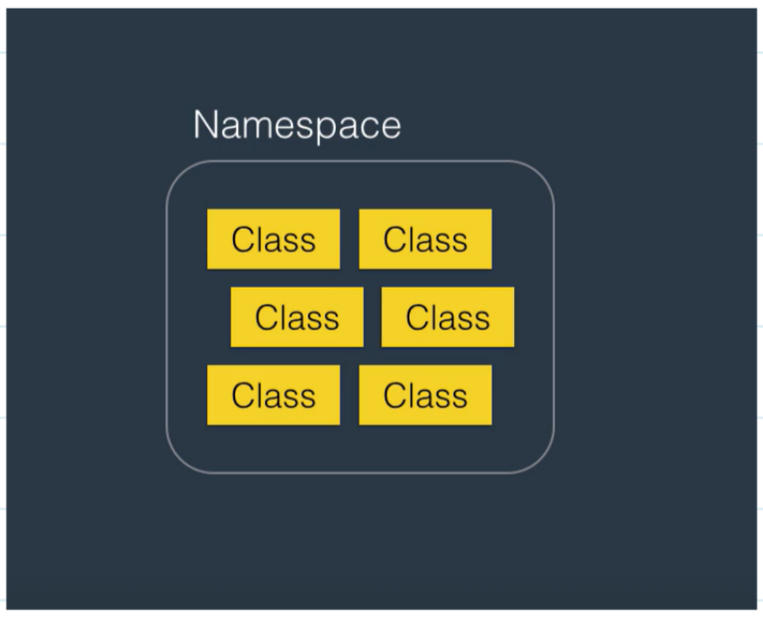
Data represents the state of the application → Example Car → the Car has some attributes.





As the number of classes grows in an application, we need to organize these class. So, we use "NameSpaces"

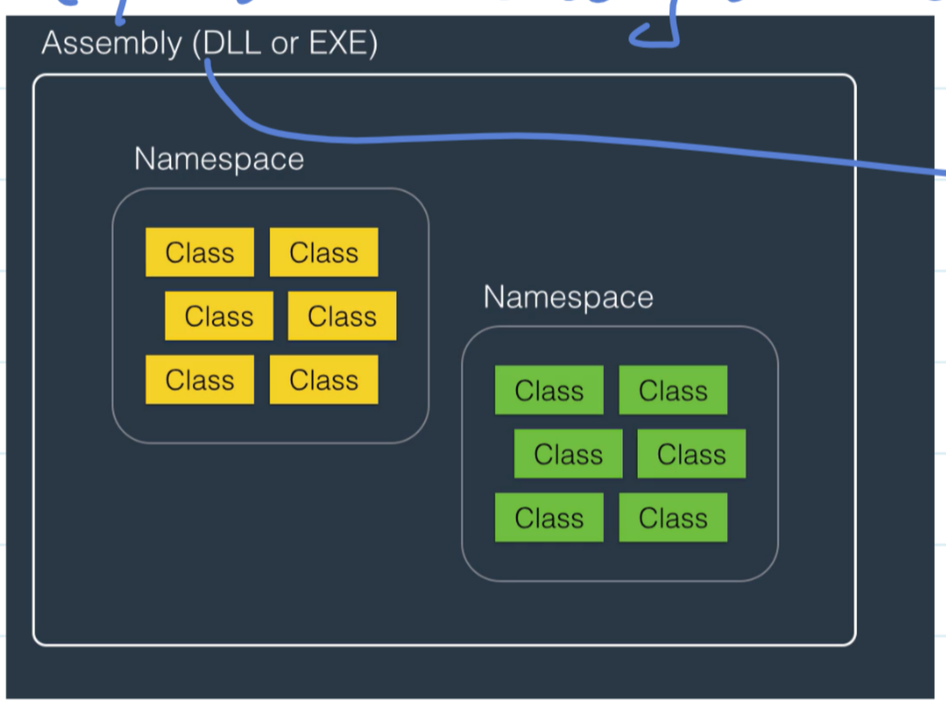
NameSpaces - It is an containes for related classes. for Example in .Net framework, we have NameSpaces with related classes.



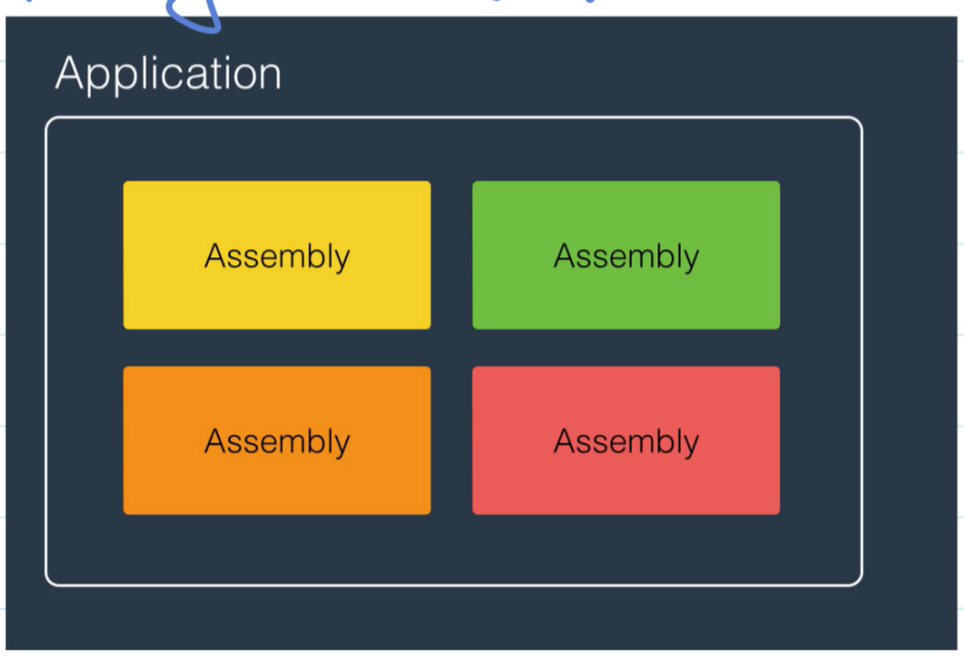
we have namespaces for data, images, Graphs, Security ....

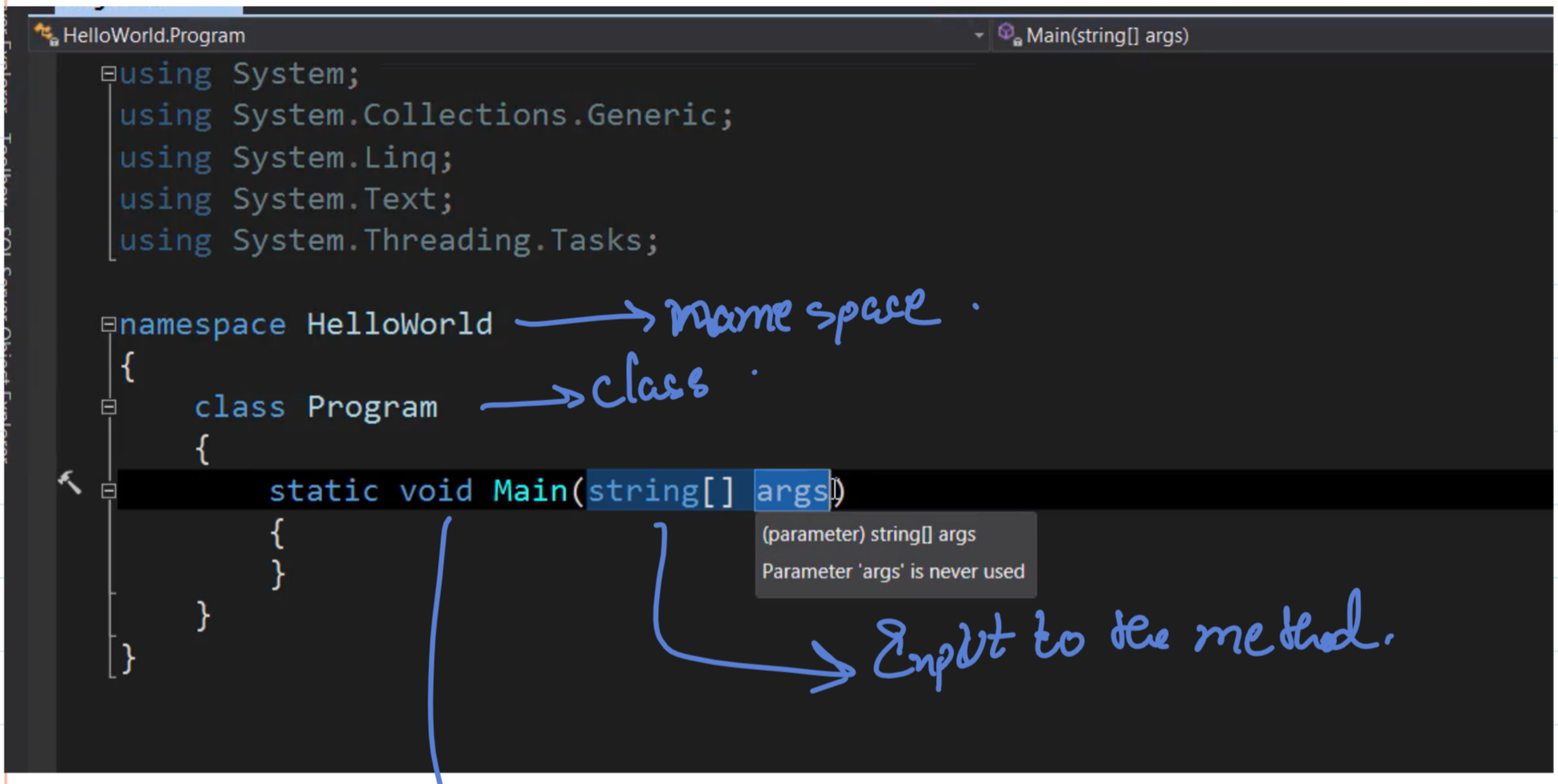
So, we have Many Name spaces in the application, so we need to partition the Namespaces - that is why we use assembly → it is an containes of different Namespaces.

DLL = Dynamically Linked library.



So, when we compile an application, the compiler will build one or more Assembly depending on How you partition our code.



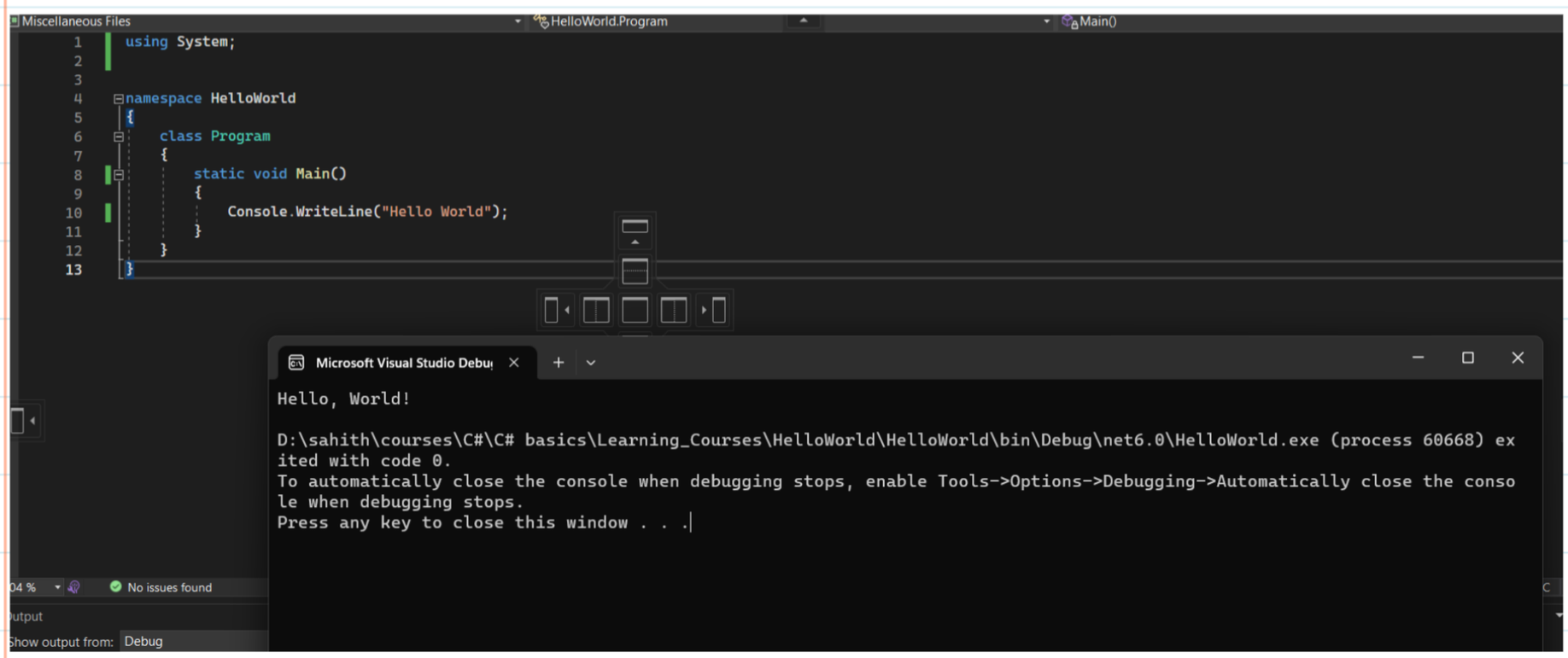


we have a class called Console. → which is used to read data (or) write data.
It has many methods, we can access with dot Notation.

Program:-
to Print "Hello world"

```
using System;

namespace Helloworld
{
    class Program.
    {
        static void Main( )
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

output.
Hello world.



Questions:-

1 / 3

What is a namespace?

- A type that contains code for the program.
- A container for classes. ✓
- A deployable unit of application.

2 / 3

What is JIT Compilation?

- The compilation of IL code to native machine code at run-time. ✓
- The compilation of C# code to native code.
- The compilation of C# code to IL code.

3 / 3

What is an Assembly?

- A piece of code that is executed by CLR.
- Container of one or more classes.
- A single unit of deployment of .NET applications. ✓

Summary

So in this section, you learned the basics of C#.

C# vs .NET

C# is a programming language, while .NET is a framework. It consists of a run-time environment (CLR) and a class library that we use for building applications.

CLR

When you compile an application, C# compiler compiles your code to IL (Intermediate Language) code. IL code is platform agnostics, which makes it possible to a take a C# program on a different computer with different hardware architecture and operating system and run it. For this to happen, we need CLR. When you run a C# application, CLR compiles the IL code into the native machine code for the computer on which it is running. This process is called Just-in-time Compilation (JIT).

Architecture of .NET Applications

In terms of architecture, an application written with C# consists of building blocks called classes. A class is a container for data (attributes) and methods (functions). Attributes represent the state of the application. Methods include code. They have logic. That's where we implement our algorithms and write code.

A namespace is a container for related classes. So as your application grows in size, you may want to group the related classes into various namespaces for better maintainability.

As the number of classes and namespaces even grow further, you may want to physically separate related namespaces into separate assemblies. An assembly is a file (DLL or EXE) that contains one or more namespaces and classes. An EXE file represents a program that can be executed. A DLL is a file that includes code that can be re-used across different programs.

In the next section, you'll learn about basics of the C# language, including variables, constants, type conversion and operators.

Primitive Types and Expressions :-

Variables and Constants :-

Variable :- A name given to a storage Location in Memory.
Constant :- a immutable value.
↳ which means it cannot be changed.
Ex :- Pi = 3.14.

Declaring Variables / constants :-
int number ; → semicolon.
↳ type of the variable. → name of the variable.
→ C++ is an case sensitive.

int Number = 1;
Const float Pi = 3.14f; → value.
↳ const ↳ Datatype → name(or) identifier

rules for identifiers :-
1. Cannot start with a number Ex :- 1Route ✗, OneRoute ✓
2. Cannot include a whitespace ; Ex :- first Name ✗, FirstName ✓.
3. Cannot be an reserved Keyword. Ex : int ✗, @int ✓
4. Always use meaning Names Ex :- fn ✗, firstName ✓

In terms of Naming Conventions, these are three popular Naming Conventions.
1. Camel Case :- firstName
2. Pascal Case :- FirstName.
3. Hungarian Notation :- strFirstName.
↳ datatype → we don't use in C++

So, we can use, Camel Case (or) Pascal Case.
Ex :- int number; → for Variables.
Const int MaxZoom = 5; for constants.

Primitive Types

	C# Type	.NET Type	Bytes	Range
Integral Numbers	**byte**	Byte	1	0 to 255
	short	Int16	2	-32,768 to 32,767
	int	Int32	4	-2.1B to 2.1B
	long	Int64	8	…
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	…
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}
Character	**char**	Char	2	Unicode Characters
Boolean	**bool**	Boolean	1	True / False

Real Numbers

	C# Type	.NET Type	Bytes	Range
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	…
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}

```
float number = 1.2f;

decimal number = 1.2m;
```

to make as float.

Char is initialized in (' ') [single quotes]
String " " [" "] [double quotes]

overflowing :-

byte numbers = 255; → the Largest va.
number = number + 1; // 0. → we have Exceeded the size of byte.
so, we get output "0"

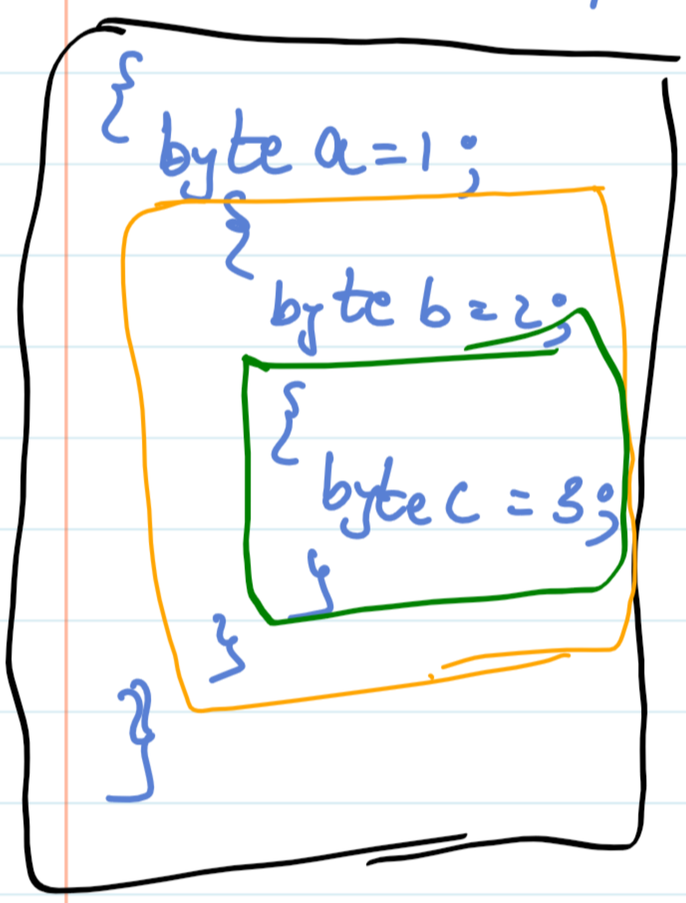
If you want to stop the overflow, we need to use "checked" keyword.

```
checked
{
  byte numbers = 255;
  number = number + 1;
}
```

} with this overflow will not happen at runtime. instead an Exception will be throoced. & Program will be crashed, if we don't handle it the Exception properly.

Scope :-

* where a Variable / constant has meaning and accessable.

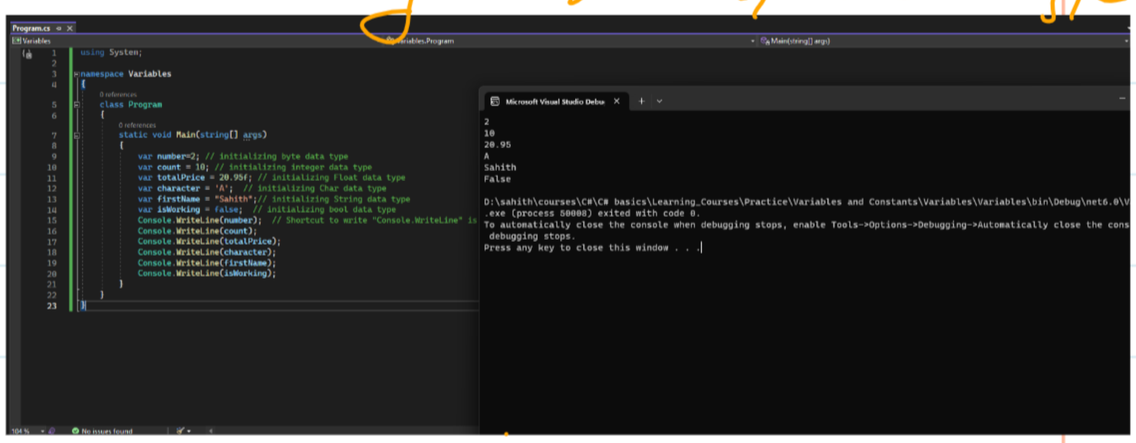


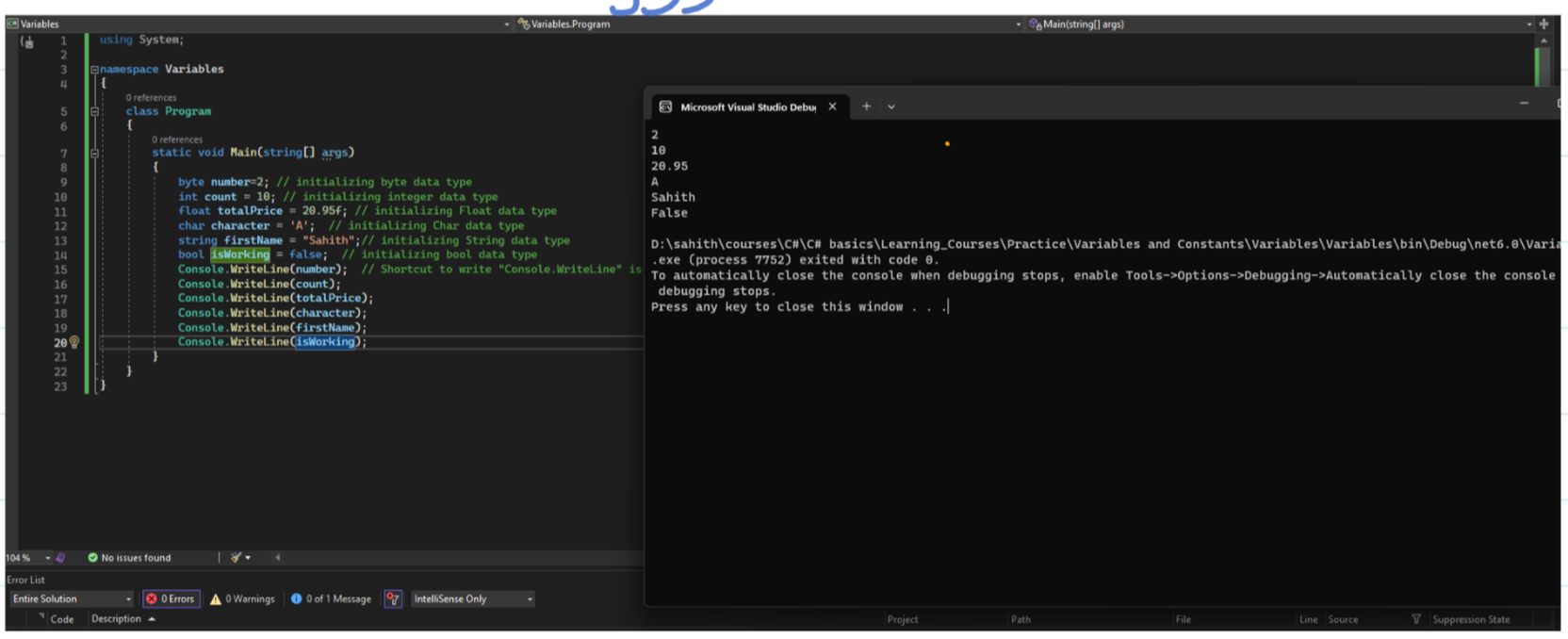
} Here we have 3 blocks of code.

Write an C# program to initialize and print all the data types :-

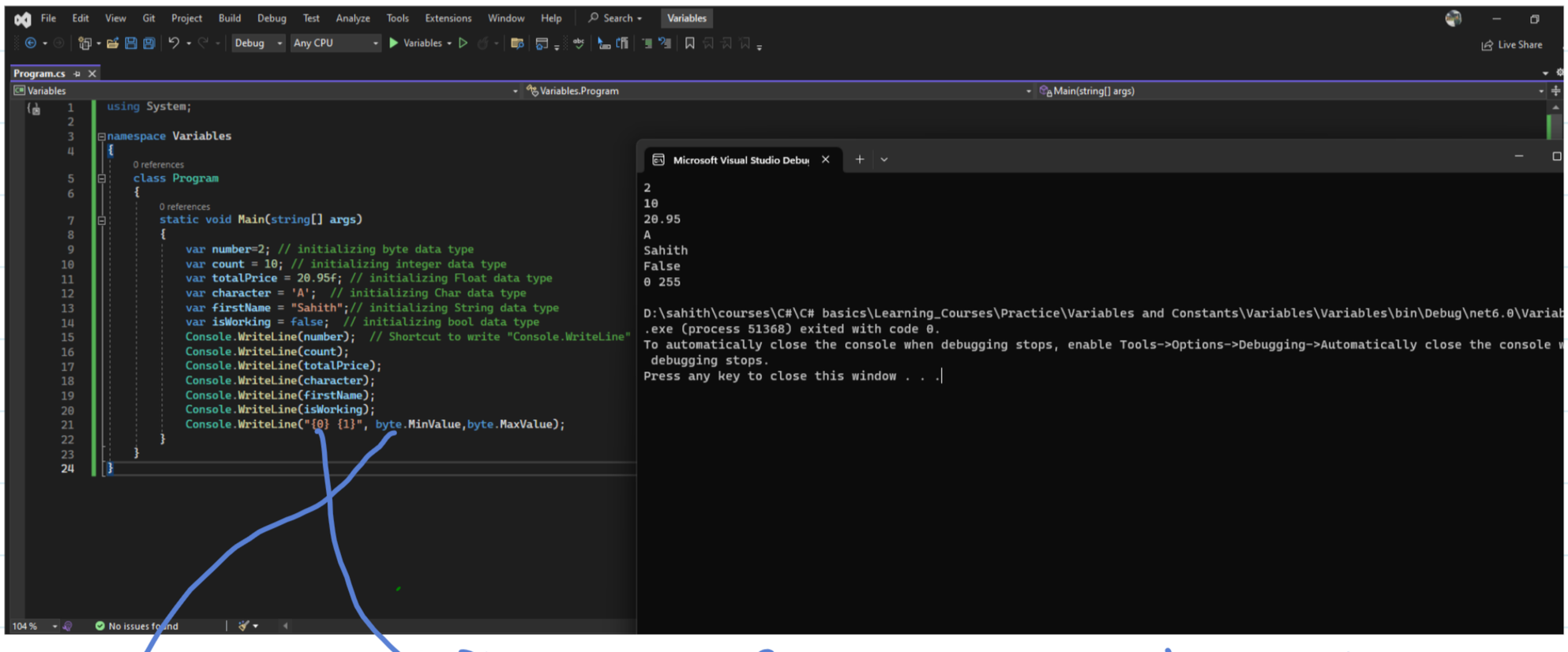
```
using System;                                   → name of the project
namespace Variable                              → class name
{
    Class Program                               → method name
    {
        static void Main(string[] arg)
        {
            byte number=2;
            int count=10;
            float totalPrice=20.95f;
            Char character='A';
            string firstName="Sahith"
            bool isWorking = false;
            Console.writeLine(number);
            Console.writeline(count);
            Console.writeline(totalPrice);
            Console.writeline(character);
            Console.writeline(firstName);
            Console.writeline(isWorking);
        }
    }
}
```

In C#, we have keyword "var", we no need to write byte, int, float ---
we can just write var, the compiler automatically takes it respective datatype.





Shortcut :- Press "Control + x" to delet the particular line.



we can do it to float, int...

It is a format for string. {0} = 1st argument i.e minvalue.
{1} = 2nd argument i.e maxvalue.

Type Conversion:-

we have ① Implicit type Conversion.
② Explicit type Conversion (Casting)
③ Conversion between non-compatible types.

Example for implicit type Conversion:-

```
byte b=1;
int i=b;
```

byte consists of 1byte of memory } the binary representations.
but integer takes 4 bytes.

00000001

00000000 00000000 00000000 00000001

2nd Example for implicit type Conversion:-

```
int i=1;
float f=i;
```

In this case Not data will get Loss.

3rd Example

```
int i=1;
byte b=i;
```

→ 1 can be converted to byte, But if we have 300 a.... Hence need Explicitly converted.

// won't compile. } Integers is 4 bytes, when we convert 4 bytes to 1byte, there are many chances of data Loss (i.e 3byte)

If we want to forcefully convert the integers to byte. we can use Explicit type.

Ex:- `int i=300;`
`byte b=(byte)i;`

{ we know that data will be Lost, But also if we want to convert, we can use this Explicit type }

↳ This process is called "Casting"

Ex:- `float f=1.0f;`
`int i=(int)f;`

Some, datatypes are not compatable. Like.

Ex:- `String s="1";`
`int i=(int)s;` // won't compile.

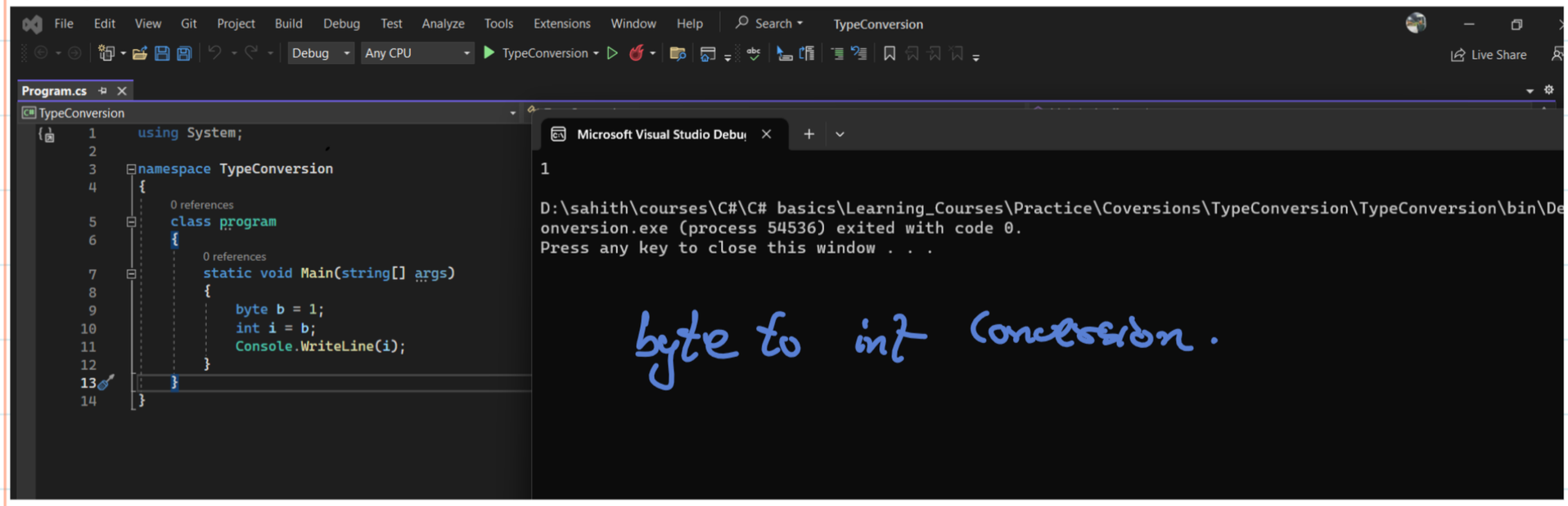
In this, we need an different mechanism, to convert string to int.

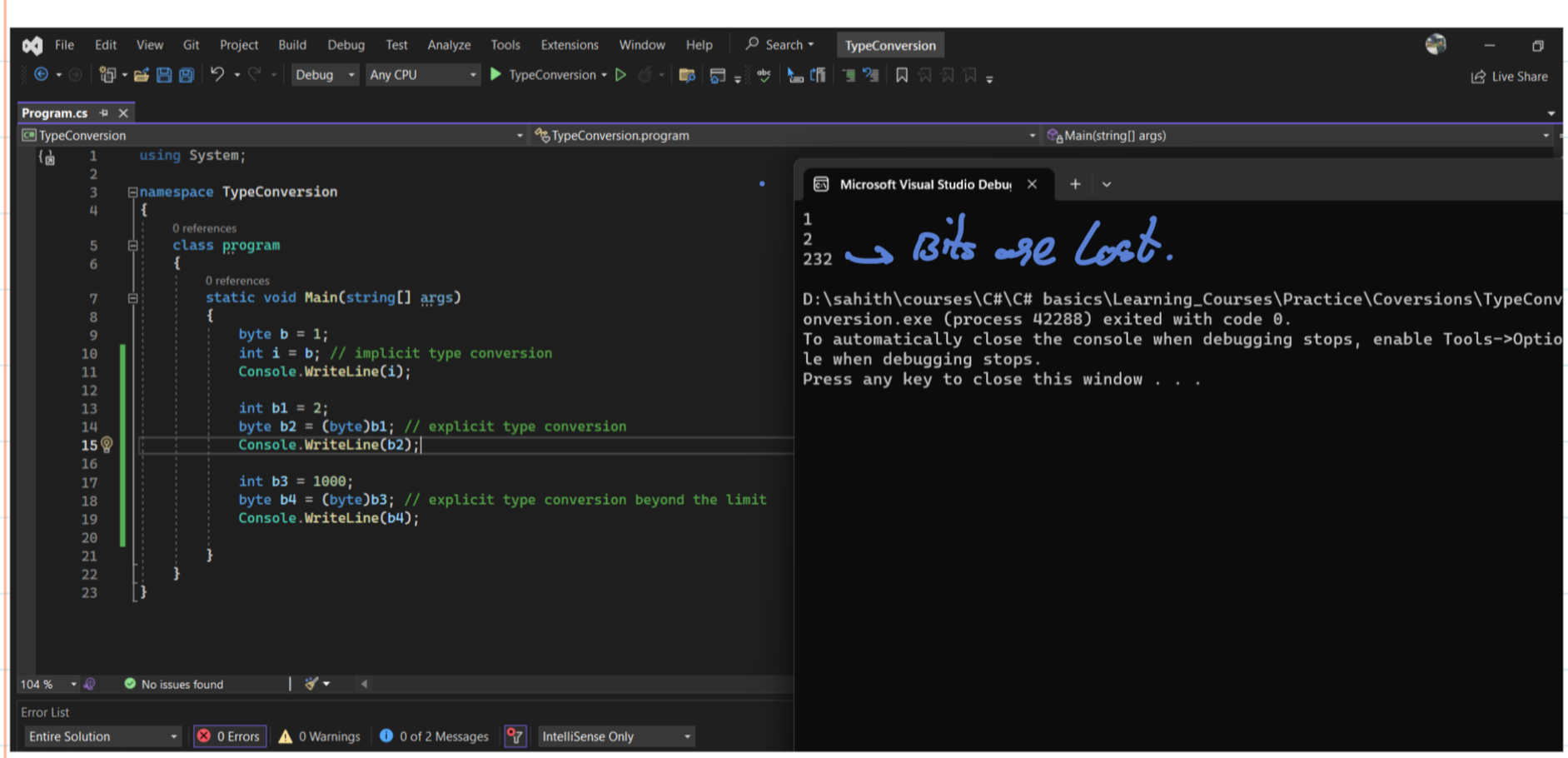
Ex:- `String s="1";`
`int i=Convert.ToInt32(s);` // we can use Convert class, which is in .Net framework
`int j=int.Parse(s);` // (or) we can use Parse method.

the methods, we have in Convert class:-

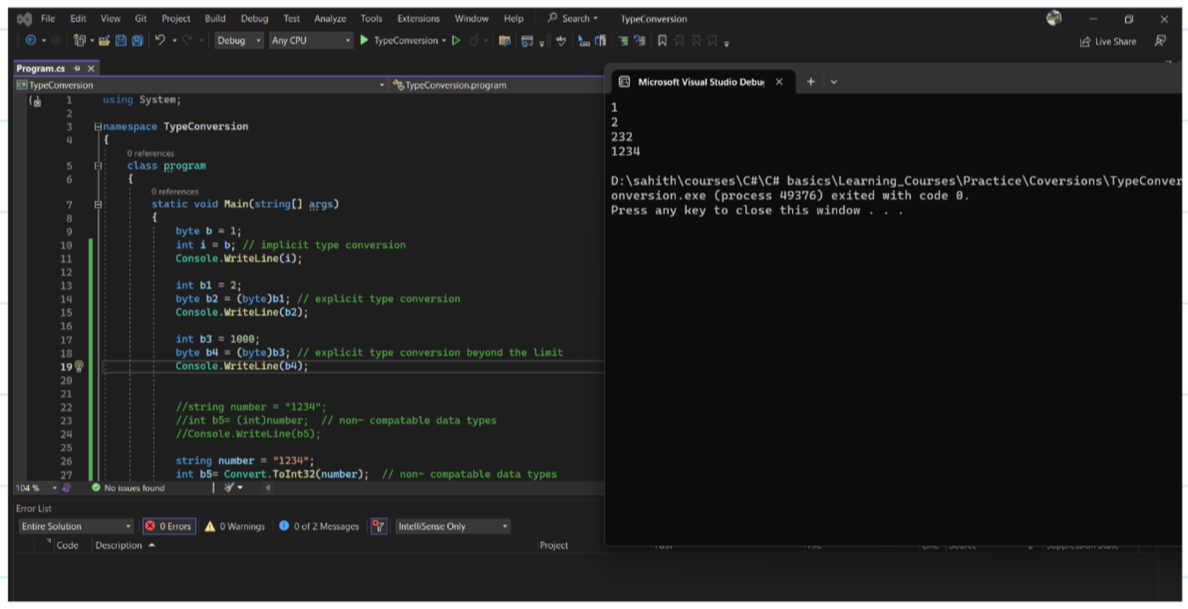
* ToByte()
* ToInt16()
* ToInt32()
* ToInt64()

Program implicite type conversion:-





using System;




```
namespace TypeConversion
{
   class program
   {
      static void Main(string[] args)
      {
         byte b = 1;
         int i = b; // implicit type conversion
         Console.WriteLine(i);

         int b1 = 2;
         byte b2 = (byte)b1; // explicit type conversion
         Console.WriteLine(b2);

         int b3 = 1000;
         byte b4 = (byte)b3; // explicit type conversion beyond the limit
         Console.WriteLine(b4);


         //string number = "1234";
         //int b5= (int)number;  // non- compatable data types
         //Console.WriteLine(b5);

         string number = "1234";
         int b5= Convert.ToInt32(number);  // non- compatable data types
         Console.WriteLine(b5);


         try
         {
            string number1 = "1234";
            int b6 = Convert.ToByte(number);  // the code will crash so, we can use try
                                              // and catch method
            Console.WriteLine(b6);
         }
         catch(Exception)
         {
            Console.WriteLine("the number could not be converted to byte");
         }
      }
   }
}
```

C# Operators :-

1. Arithmetic Operators.
2. Comparision Operators.
3. Assignment Operators.
4. Logical Operators.
5. Bitwise Operators.

1) Arithmetic operators.

	Operator	Example
Add	**+**	a + b
Subtract	**-**	a - b
Multiply	*****	a * b
Divide	**/**	a / b
Remainder	**%**	a % b

	Operator	Example	Same as
Increment	**++**	a++	a = a + 1
Decrement	**--**	a--	a = a - 1

the increment and decrement can be done in two ways 1) Postfix
2) Prefix.

Example for Postfix:- `int a=1;` `int b=a++;` o/p = a=2, b=1

Example for Prefix:- `int a=1;` `int b=++a;` o/p :- a=2, b=2

2) Comparision Operators :-

	Operator	Example
Equal	**==**	a == b
Not Equal	**!=**	a != b
Greater than	**>**	a > b
Greater than or equal to	**>=**	a >= b
Less than	**<**	a < b
Less than or equal to	**<=**	a <= b

(3) Assignment Operators :-

	Operator	Example	Same as
Assignment	**=**	a = 1	
Addition assignment	**+=**	a += 3	a = a + 3
Subtraction assignment	**-=**	a -= 3	a = a - 3
Multiplication assignment	***=**	a *= 3	a = a x 3
Division assignment	**/=**	a /= 3	a = a / 3

(4) Logical Operators :-

	Operator	Example
And	**&&**	a && b
Or	**II**	a || b
Not	**!**	!a

(5) Bitwise operators :-

	Operator	Example
And	**&**	a & b
Or	**I**	a | b

What are Logical Operations?

Logical operations are part of Boolean algebra, which is often taught to students of computer science and electronic engineering at University. So, if you're not familiar with Boolean algebra, here is a brief introduction.

In Boolean algebra, the value of variables can only be **true** or **false**, also denoted 1 and 0 respectively. Unlike elementary algebra, where the main operations are addition, subtraction, etc, the main operations of Boolean algebra are **conjunction** (AND), **disjunction** (OR) and **negation** (NOT).

Logical AND

Let's assume we have two variables: **x** and **y**. In C#, the logical AND operator is indicated by &&. We can define a Boolean expression as follows:

```
z = x && y
```

In this expression, **z** is true if *both* **x** and **y** are true; otherwise, it'll be false.

What is a real-world example of this in programming? Imagine you're developing a loan application. The provider only offers loans to applicants who are over 18 and are citizen of the given country. In this example, we have two variables:

x = applicant being over 18
y = application being a citizen

z = is eligible to apply for loan = x && y

If *both* **x** and **y** are true, the applicant is eligible to apply for a loan.

Later, when we get to conditional statements, you can check to see if the above expression evaluates to true or false, and then, can change the flow of your application.

So, here is the rule of thumb with logical AND: if *both* **x** and **y** are true, **x && y** will be true; otherwise, it'll be false.

Logical OR

In C#, logical OR is indicated by two vertical lines (||). Considering the following expression:

```
z = x || y
```

z will be true, if *either* **x** or **y** is true.

What is a real-world example of this? Imagine you're building software for a recruiter. For a given job application, applicants can apply if they have a degree in computing, or more than 5 years of experience in the field. You can model this using a Boolean expression as follows:

x = applicant has a degree in computing
y = applicant has more than 5 years of experience

z = application is eligible = x || y

If either **x** or y is true, z will be true.

So, unlike the logical AND, where both variables must be true, with logical OR, if at least one of them is true, the result will be true.

Logical NOT

The NOT operator in C# is indicated by an exclamation mark (!) and it reverses the value of a given variable or expression:

```
y = !x
```

So, here, if **x** is true, **y** will be false, and if **x** is false, **y** will be true.

Clean Coding

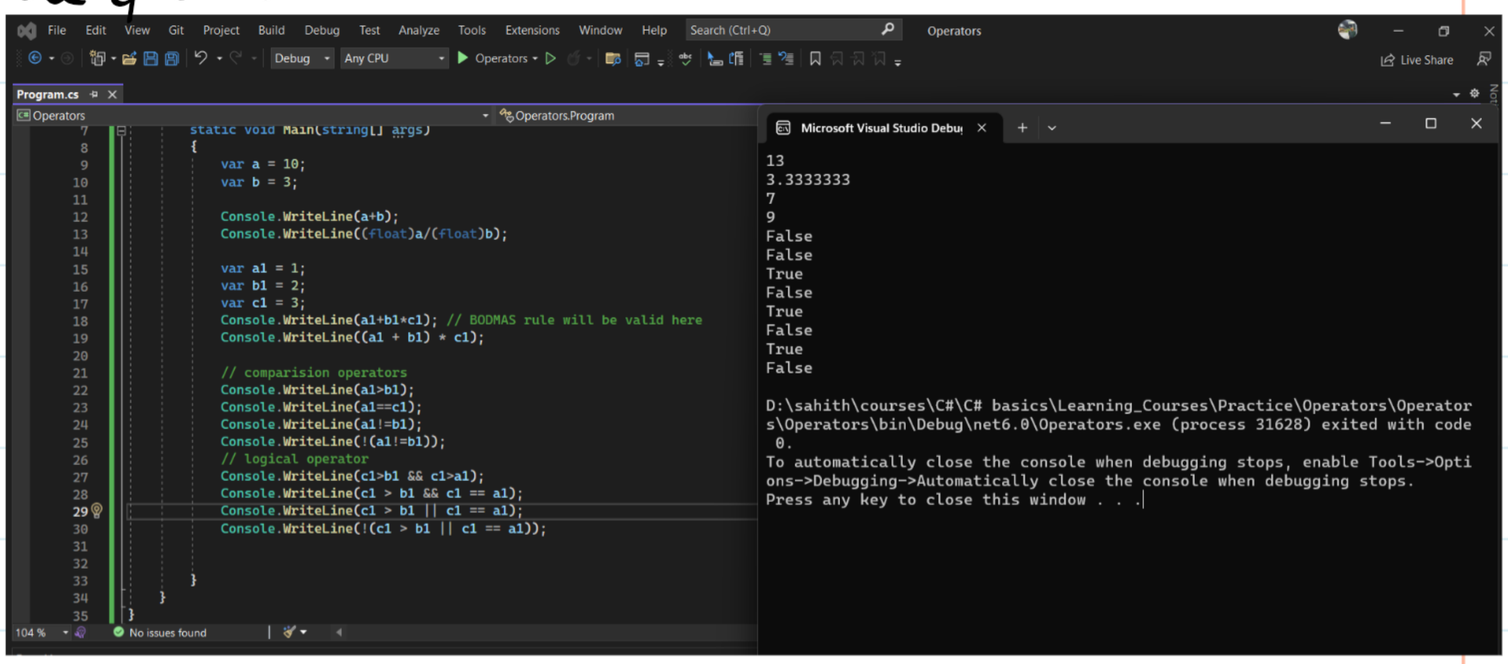
In all examples here, I used variables **x** and **y**, mainly to relate programming to Boolean algebra. But when it comes to coding, you should avoid using variable names such as x, y, z as they don't give a clue to other developers reading your code (or even yourself). Instead, use meaningful names. For instance, in the first example, you can replace **x**, **y** and **z** as follows:

```
x: isOver18
y: isCitizen
z: isEligible
```

Often, it's a good practice to prefix Boolean names with IS or HAS (if possible).

Program to perform all the operators :-

```
using System;

namespace Operators
{
    class Program
    {
        static void Main(string[] args)
        {
            var a = 10;
            var b = 3;

            Console.WriteLine(a+b);
            Console.WriteLine((float)a/(float)b);

            var a1 = 1;
            var b1 = 2;
            var c1 = 3;
            Console.WriteLine(a1+b1*c1); // BODMAS rule will be valid here
            Console.WriteLine((a1 + b1) * c1);

            // comparision operators
            Console.WriteLine(a1>b1);
            Console.WriteLine(a1==c1);
            Console.WriteLine(a1!=b1);
            Console.WriteLine(!(a1!=b1));
            // logical operator
            Console.WriteLine(c1>b1 && c1>a1);
            Console.WriteLine(c1 > b1 && c1 == a1);
            Console.WriteLine(c1 > b1 || c1 == a1);
            Console.WriteLine(!(c1 > b1 || c1 == a1));


        }
    }
}
```



Comments :-

Single-line Comment	`// Here is a single-line comment` `int a = 1;`

Multi-line Comments	`/*`
	`  Here is a multi-line`
	`  comment`
	`*/`
	`int a = 1;`

Non-Primitives :-

Classes :-