**Basics of C#.NET Programming**

**(Part no. 3)**

# **Variables**

Variables are concerned with the storage of data. Think of variables in computer memory as boxes sitting on a shelf. You can put things in boxes and take them out again, or you can just look inside a box to see if anything is there. The same goes for variables, you place data in them and can take it out or look at it, as required. Data in a computer is also the same thing (a series of 0s and 1s), variables come in different flavors, known as data types which we will also cover in this tutorial. To use variables, you have to declare them. This means that you have to assign them a name and a type. After you have declared variables, you can use them as storage units for the type of data that you declared them to hold. Let's talk about some of the declaration types which are only two.

## **Explicit Variable declaration**

In the explicit declaration of the variables, we assign the type to our variable name all by ourselves instead of relying on the computer to do our job. Like

int age;

float division;

string name;

Yes, this is how it's done in C#. Whereas, int, float, string are the data types, and age, division, name are the names of the variables. Let me show you magic in our next variable declaration.

## **Implicit Variable declaration**

In the implicit declaration of the variables, we don’t have to worry about carefully declaring the type to our variables because in this one we use just one keyword ‘var’ and the compiler detects what kind of variable it is going to be when we initialize values into the program, like

var age;

var division;

var name;

Amazing right? You might be thinking of C# is also a dynamic language just like Python, but you are wrong C# is not that dynamic and system low-level programming language than Python is. In the above example keyword, ‘var’ will implicitly declare the values of the age, division, and name variables.

One more important thing here, if you try to use a variable that hasn't been declared, your code won't compile, but thanks to .NET smart compilers, as it tells you exactly what the problem is, so this isn't a disastrous error. Trying to use a variable without assigning it a value also causes an error, but again, the compiler detects this.

That was too much time with the variables lets move to our next topic.

# **Data Types & its Types**

We will start to form the very simple types and then proceeding towards some complex and bigger types. But before that let's solve a pandora box about the numeric types.

## **Mechanics of Numeric Types**

Numeric Types have some mechanics of storing numbers as a series of 0s and 1s in the memory of a computer. For example, for integer values, you simply take several bits (individual digits that can be 0 or 1) and represent your number in binary format. A variable storing N bits enables you to represent any number between 0 and (2N − 1). Any numbers above this value are too big to fit into this variable.

For example, suppose you have a variable that can store two bits. The data stored between integers and the bits representing those integers is therefore as follows:

0 = 00

1 = 01

2 = 10

3 = 11

To store more numbers you need more bits like three bits to enable you to store the numbers from 0 to 7. The inevitable result of this process or system is that you would need an infinite number of bits to be able to store every imaginable number, which isn't going to fit in your limited PC and you might have to buy a lot of hard drive space 😥.

## **Simple Types**

Simple types include types such as numbers and Boolean (true or false) values that make up the fundamental building blocks for your applications. Unlike complex types, simple types cannot have children or attributes which we will discuss in our future tutorials. Some of the simple types are

* **char:** Single Unicode character, stored as an integer between 0 and 65535 of values.
* **bool:** Boolean value, true or false.
* **String:** A sequence of characters.

## **Integer Types**

Let's talk about some integer types in the following

* **sbyte:** Integer values between -128 and 127
* **byte:** Integer values between 0 and 255
* **short:** Integer values between -32768 and 32767
* **ushort:** Integer values between 0 to 65535
* **int:** Integer values between −2147483648 and 2147483647
* **uint:** Integer values between 0 and 4294967295
* **long:** Integer values between −9223372036854775808 and 9223372036854775807
* **ulong:** Integer values between 0 and 18446744073709551615

The u characters before some variable names are shorthand for unsigned, meaning that you can't store negative numbers in variables of those types, as shown in the above.

## **Floating-point Types**

You also need to store floating-point values, those that aren't whole numbers. You can use three floating-point variable types **float, double, and decimal**. The first two store floating points in the form **6m × 2e**, where the allowed values for m and e differ for each type. Decimal also uses the alternative form **6m × 10e**. Their values are calculated by the compiler with the formulas stated before.

## **String Interpolation**

A new feature in C# is called String Interpolation. It is very similar to the simple method of writing text to the console that you will see in the following, but now you are specifying your variables. It's too soon to dive into the details of this line of code but believe me, it’s worth it. Let's consider the following two examples.

Console.WriteLine(“{0} {1}", myString, myInteger);

Console.WriteLine($"{myString} {myInteger}");

As you can see in the first one you have to be more careful and you have to put in more brainpower than in the second one. That was too much of the Data Types and about its further types so let's dive into using the expressions which come into being with the combination of operators, literal values, and variables.

# **Operators**

C# contains several operators for this purpose. By combining operators with variables and literal values (together referred to as operands when used with operators), you can create expressions, which are the basic building blocks of computation.

The operator's available range from the simple to the highly complex, some of which you might never encounter outside of mathematical applications. The simple ones include all the basic mathematical operations, such as the + operator to add two operands. There are also logical operators specifically for dealing with Boolean values, and assignment operators such as =. Operators have three main categories.

* **Unary:** Acts on a single operand
* **Binary:** Acts on two operands
* **Ternary:** Acts on three operands

Most operators fall into the binary category, with a few unary ones, and a single ternary one called the conditional operator which is also a logical one.

Operators are usually divided into the four main types which are Mathematical Operator, String Catenation Operator, Increment and Decrement Operator, and Assignment Operators. Each of these types has its functionality, use, and purpose. Let's discuss them one by one.

## **Mathematical Operators**

Following are some simple mathematical operators mostly used in every program.

* **+: var1 = var2 + var3;** (used as binary and var1 is assigned the value that is the sum of var2 and var3) also **var1 =+ var2;** (used as unary and var1 is assigned the value of var2)
* **-: var1 = var2 - var3;** (used as binary and var1 is assigned the value that is the value of var3 subtracted from the value of var2) also **var1 =+ var2;** (var1 is assigned the value of var2 multiplied by -1)
* **\*: var1 = var2 \* var3;** (used as binary and var1 is assigned the value that is the product of var2 and var3)
* **/: var1 = var2 / var3;** (used as binary and var1 is assigned the value that is the result of dividing var2 by var3)
* **%: var1 = var2 % var3;** (used as binary and var1 is assigned the value that is the remainder when var2 is divided by var3)

## **Increment and Decrement Operators**

Lets discuss some increment and decrement operators

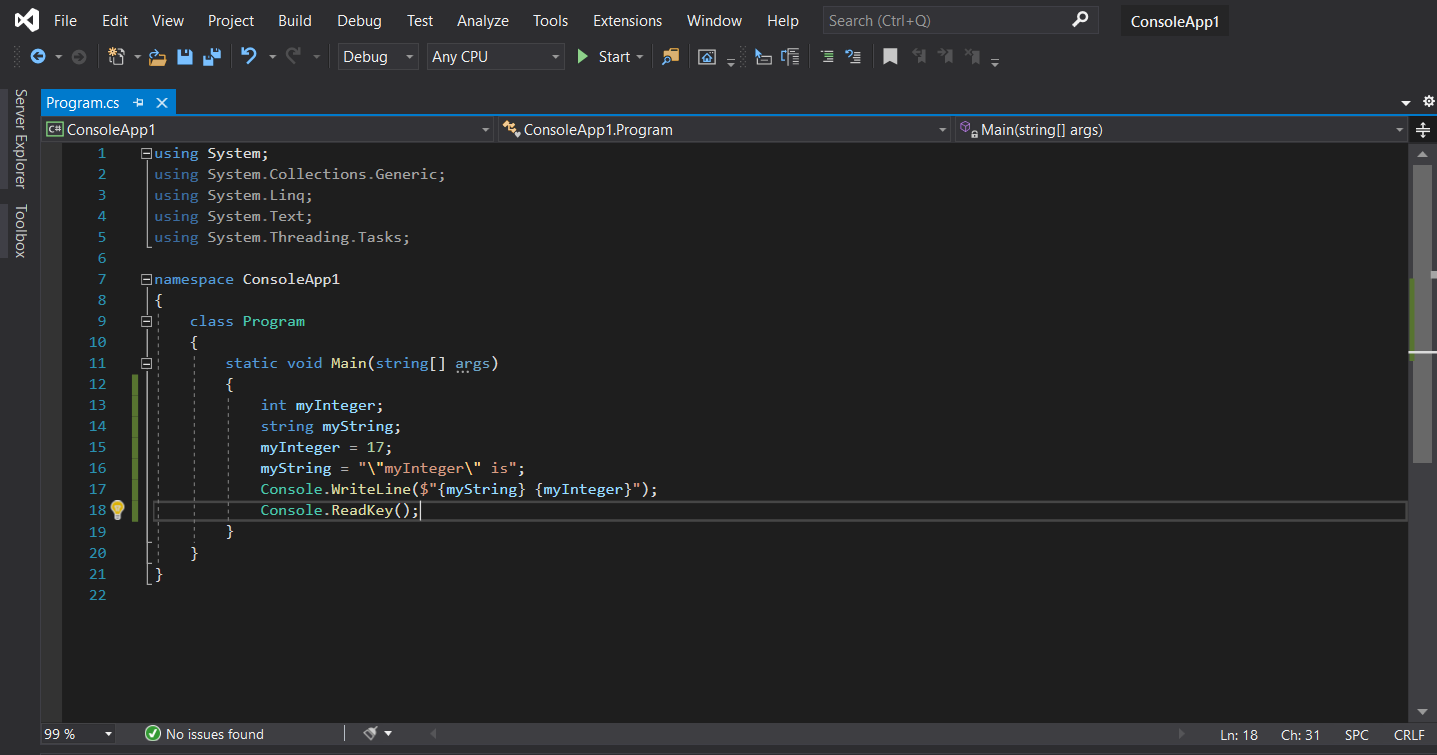
* **++: var1 = ++var2;** (used as unary and var1 is assigned the value of var2 + 1. var2 is incremented by 1)
* **--: var1 = --var2;** (used as unary and var1 is assigned the value of var2 - 1. var2 is decremented by 1)
* **++: var1 = var2++;** (used as unary and var1 is assigned the value of var2. var2 is incremented by 1)
* **--: var1 = var2--;** (used as unary and var1 is assigned the value of var2. var2 is decremented by 1)

These operators always result in a change to the value stored in their operand:

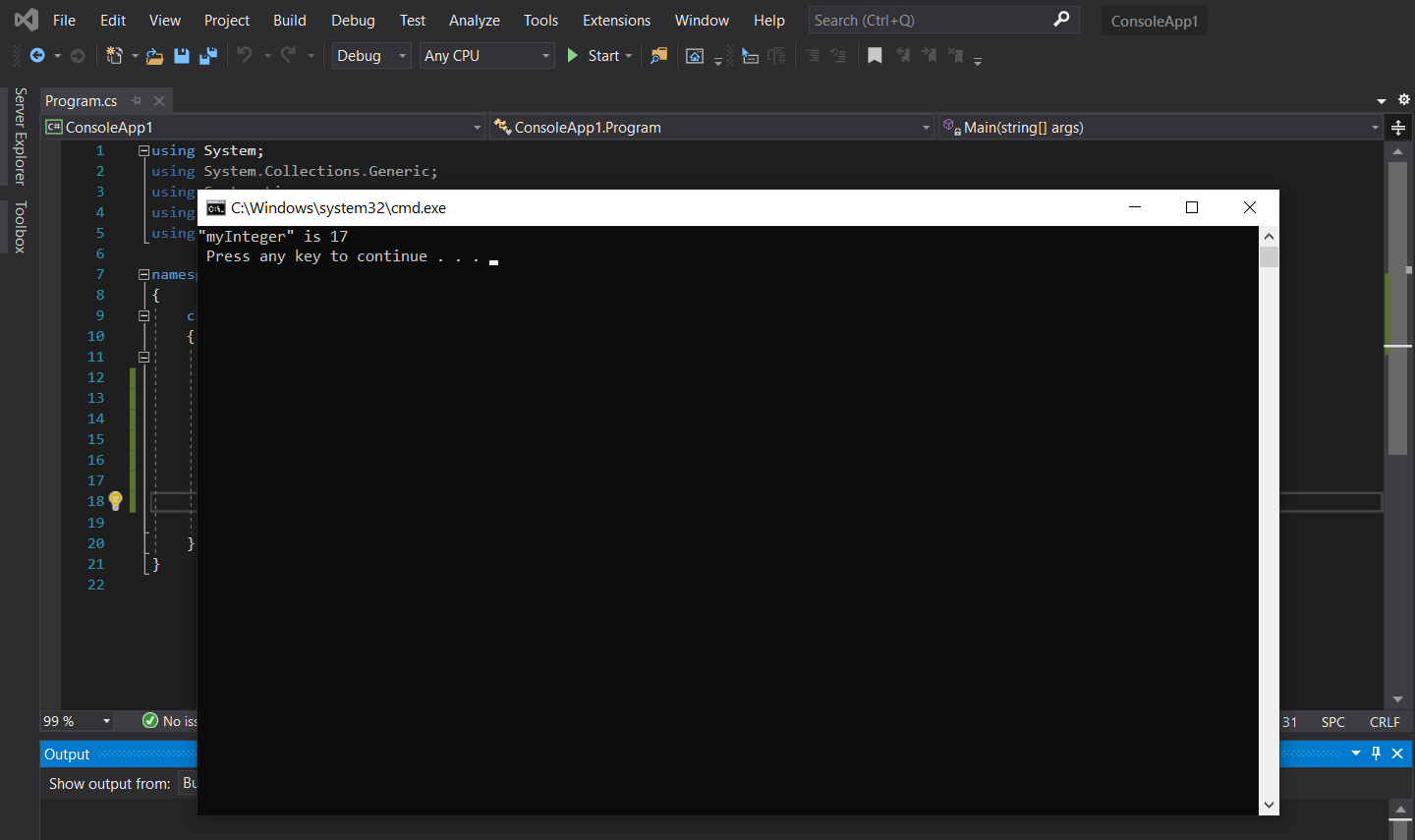
* **++** always results in its operand being incremented by one.
* **--** always results in its operand is decremented by one.

Let's move towards coding and the implementation of all the above that we have learned by now.

# **Coding a program in C#**

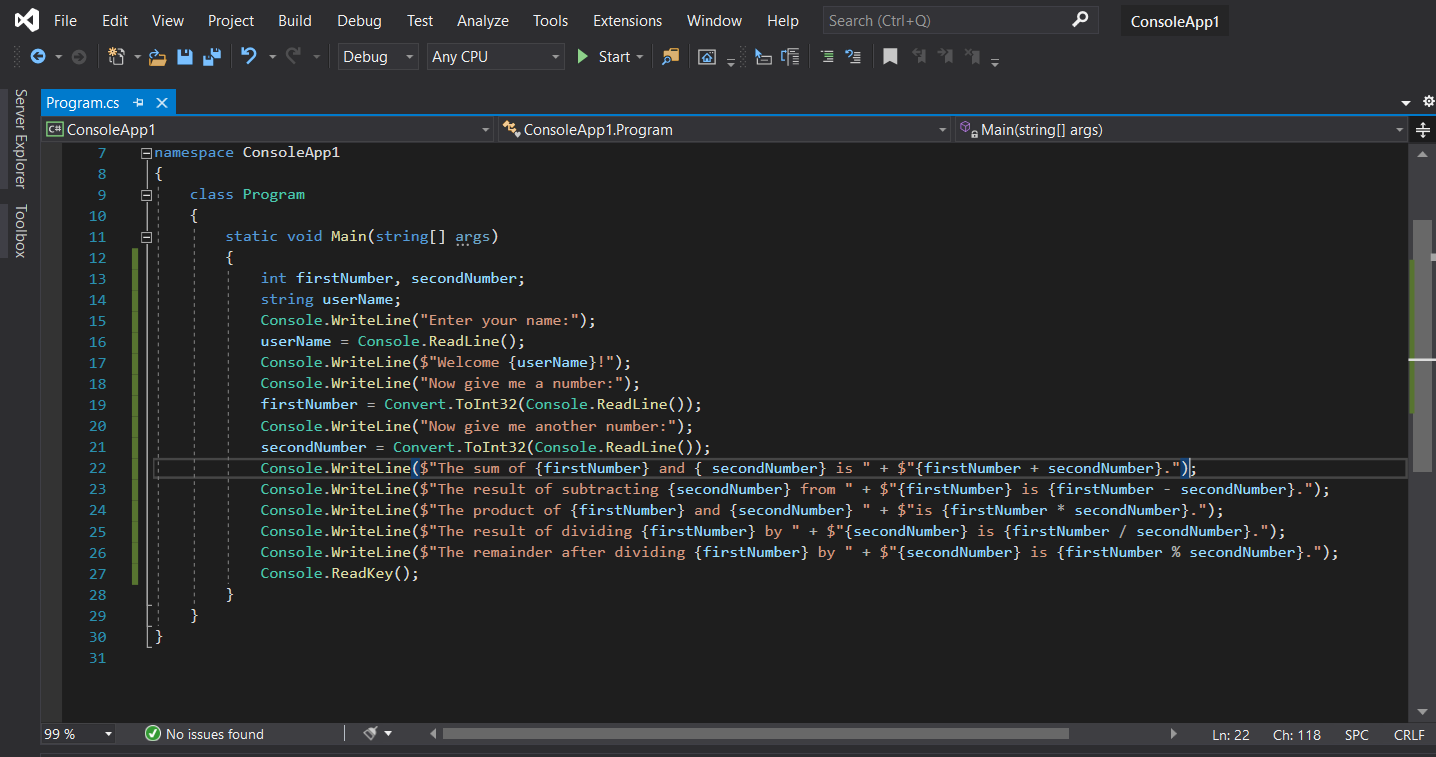
Let's start with the variables

Let's run by pressing ctrl+f5 from the keyboard and see what happens

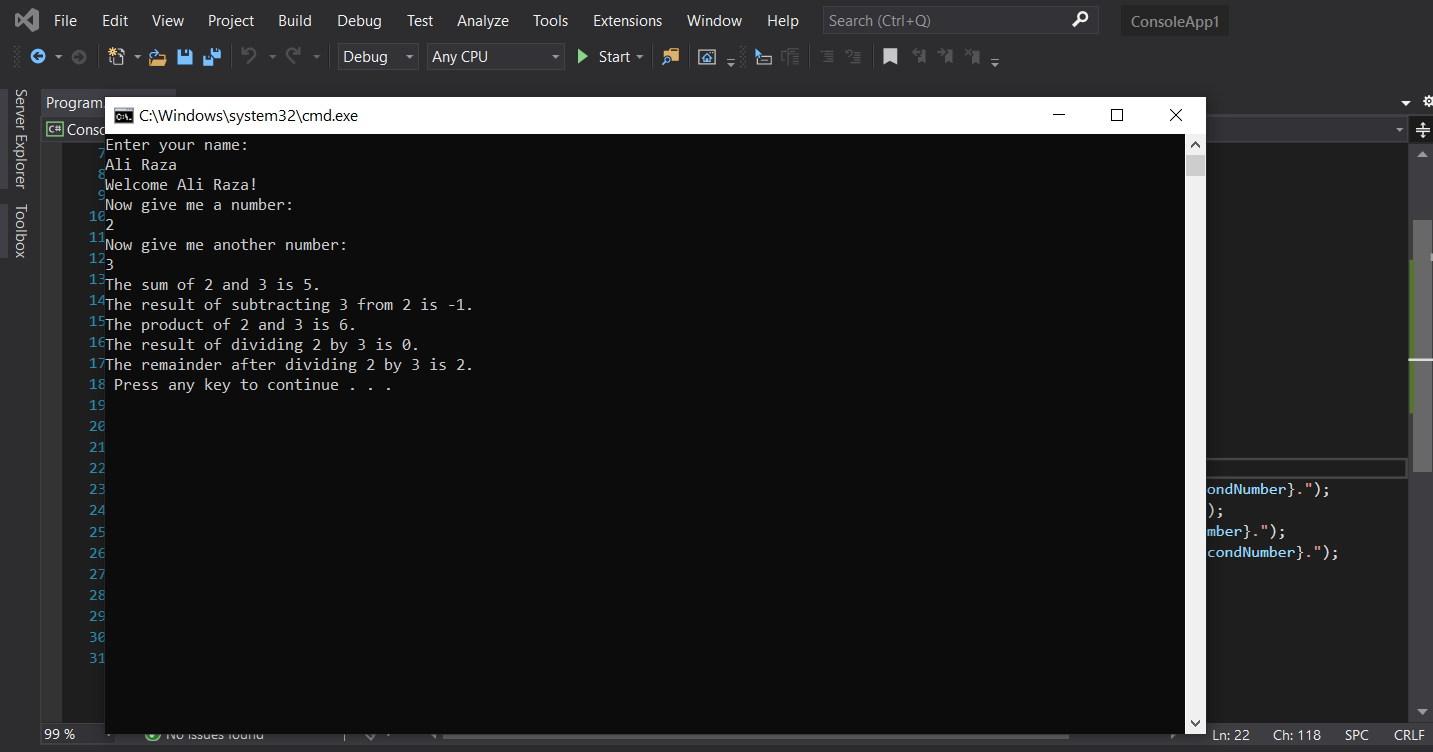


Vola! Our program is running successfully. Let's move on and make something more spectacular

Let's start coding



As you can see in the above diagram that we have coded the program now in which we will enter the name after that it will ask for a number and we will enter a number after that it will ask for the second number from us and we will enter the number of our own choice. Then we hit enter and it will show us four results, sum, subtraction, product, division, and remainder. Let's run the program and see what happens as in the following.



So here it is guys. We developed our very first program in which made our very first variables declared them and then we also learned to use assignment operators and used string interpolation that you will not have seen in most of the tutorials out there.

I hope you guys have enjoyed a lot and learned a lot today. 😇 Stay tuned for further more tutorials in the future.