**Basics of C#.NET Programming**

**(Part no. 4)**

# **Flow Control**

All of the C# code you've seen so far has had one thing in common. In each case, program execution has proceeded from one line to the next in top-to-bottom order, missing nothing. If all applications worked like this, then you would be very limited in what you could do. So to save ourselves from such a bummer, we will discuss the Flow control in the programs/applications. To control the program flow which is the execution of the lines of codes in C#, there are two main methods. The first one is through ‘Branching’ and the second one is through ‘looping’.

But here is an issue both of these above-mentioned techniques use Boolean logistics. In our previous tutorial we discussed the Boolean type, which is bool type, and not the Boolean logic so don’t get confused here because bool type is one of the data types. On the other hand, Boolean logic is all the same thing but in a slightly different context. Let's get started with Boolean logic and then we will lead ourselves into the Flow Control methods.

# **Boolean Logic**

Let's consider a situation in which you want to execute code based on whether a variable is less than 10. To do this, you need some indication of whether the statement “variable is less than 10” is true or false, for that, you need to know the Boolean result of a comparison. Boolean comparisons require the use of Boolean comparison operators (also known as relational operators), some of which are as follows:

## **Boolean Comparison Operators**

Some of the main and basic Boolean comparison operators are as follows

* == : var1 = var2 == var3; (var1 is assigned the value true if var2 is equal to var3, or false otherwise)
* != : var1 = var2 != var3; (var1 is assigned the value true if var 2 is not equal to var3, or false otherwise)
* < : var1 = var2 < var3; (var1 is assigned the value true if var2 is smaller than var3, or false otherwise)
* > : var1 = var2 > var3; (var1 is assigned the value true if var2 is greater than var3, or false otherwise)
* <= : var1 = var2 <= var3; (var1 is assigned the value true if var2 is smaller than or equal to var3, or false otherwise)
* >= : var1 = var2 >= var3; (var1 is assigned the value true if var2 is greater than or equal to var3, or false otherwise)

All the above-stated examples are for the binary operands, but they also work for unary and ternary operands too, if you get the point and concept both you can now use it very easily in your programs because your computer logics are mostly based on the true and false game which is another alphabetical representation of the 0’s and 1’s in the computer memory. Now let's talk about conditional and assignment operators.

## **Conditional Operators**

Let's talk about some conditional operators now. Yes, just like the conditional statements in mathematics as you might have seen in your mathematical subjects. There are some also used in computer programming such as

* && : var1 = var2 && var3; (var1 is assigned the value true if var2 and var3, both are true, or false otherwise (also known as the Logical AND))
* || : var 1 = var2 || var3; (var1 is assigned the value true if either var2 or var3 or, both are true, or false otherwise (also known as the Logical OR))

The above-mentioned Logical operators or Conditional operators are used in every domain, whether it is physics, mathematics, or hardware devices. You can find logical gates, arrays, and operators working everywhere out there. There is an interesting thing about these operators in C#, which is, the result of these operators is the same as & and |, but there is an important difference in the way this result is obtained, which can result in better performance. If the value of the first operand of the && operator is false, then there is no need to consider the value of the second operand, because the result will be false regardless. Similarly, the || operator returns true if its first operand is true, regardless of the value of the second operand. Lets now move on to our next Boolean operator

## **Boolean Assignment Operators**

Boolean comparisons can be combined with assignments by combining Boolean bitwise and assignment operators. These work in the same way as the mathematical assignment operators such as (+=, \*=, and so on). Some of these used in programming are

* &= : var1 &= var2; (var1 is assigned the value that is the result of var1 and var2)
* |= : var1 |= var2; (var1 is assigned the value that is the result of var1 or var2)
* ^= : var1 ^= var2; (var1 is assigned the value that is the result of either var1 or var2)

For example, the equation var1 ^= var2 is similar to var1 = var1 ^ var2 where var1 = true and var2 = false.

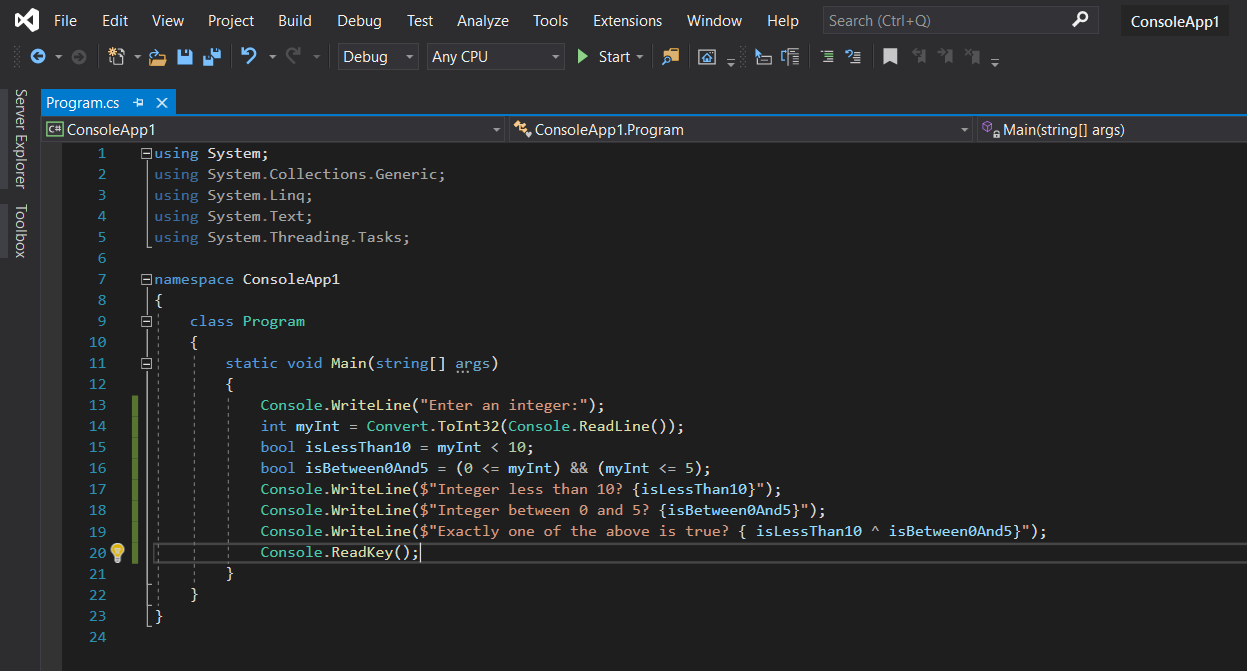
### **Important Note**

Note that the &= and |= assignment operators do not make use of the && and || conditional Boolean operators that is, all operands are processed regardless of the value to the left of the assignment operator.

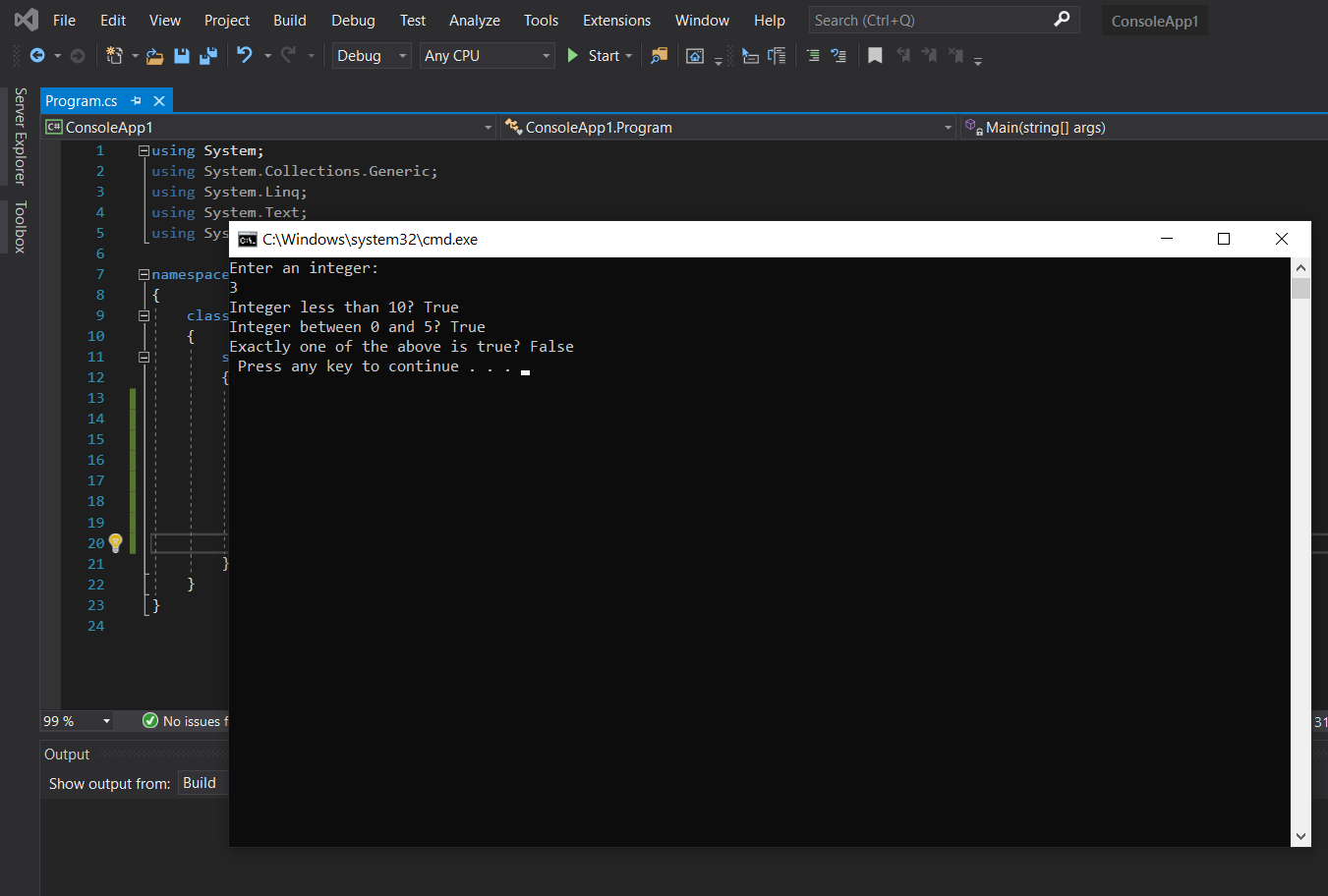
We have gone too much far with these Boolean operators and I guess that almost all of you guys have understood it very well by now. So now it's time to implement these Boolean operators into our very small Console program. Let's see that what we have learned by now, will benefit us in our coding.

# **Coding a program on Boolean Operators**

Let's code our very first program using in which we will use Boolean operators.



Let's run this program as usual by pressing Ctrl+F5 and see what happens



As you can see that by entering the integer ‘3’ we are getting the correct answer if you preview the above-written code. With this let's move to our next topic which is using the branching method for Flow Control.

# **Branching**

Branching is the act of controlling which line of code should be executed next. The line to jump to is controlled by some kind of conditional statement. This conditional statement is based on a comparison between a test value and one or more possible values using Boolean logic.

In C# we use three branching techniques as

* The Ternary Operator
* The If Statement
* The Switch Statement

Let's discuss them one by one.

## **Ternary Operator**

The simplest way to perform a comparison is to use the ternary (or conditional) operator mentioned in the last tutorial. You've already seen unary operators that work on one operand, and binary operators that work on two operands, so it won't come as a surprise that this operator works on three operands.

You might use this as follows to test the value of an int variable called var1:

string char1 = (var1 < 10) ? "Less than 10": "Greater than or equal to 10";

The result of the ternary operator is one of two strings, both of which may be assigned to char1. The choice of which string to assign is made by comparing the value of var1 to 10. In this case, a value of fewer than 10 results in the first string being assigned, and a value of greater than or equal to 10 results in the second string being assigned. For example, if var1 is 4, then char1 will be assigned the string Less than 10.

## **If Statement**

The if statement is a far more versatile and useful way to make decisions. Unlike ternary statements, if statements don't have a result so you can't use them in assignments, instead you use the statement to conditionally execute other statements.

The result of the if statement cannot be assigned to a variable, you have to assign a value to the variable in a separate step:

string char1;

if (var1 < 10)

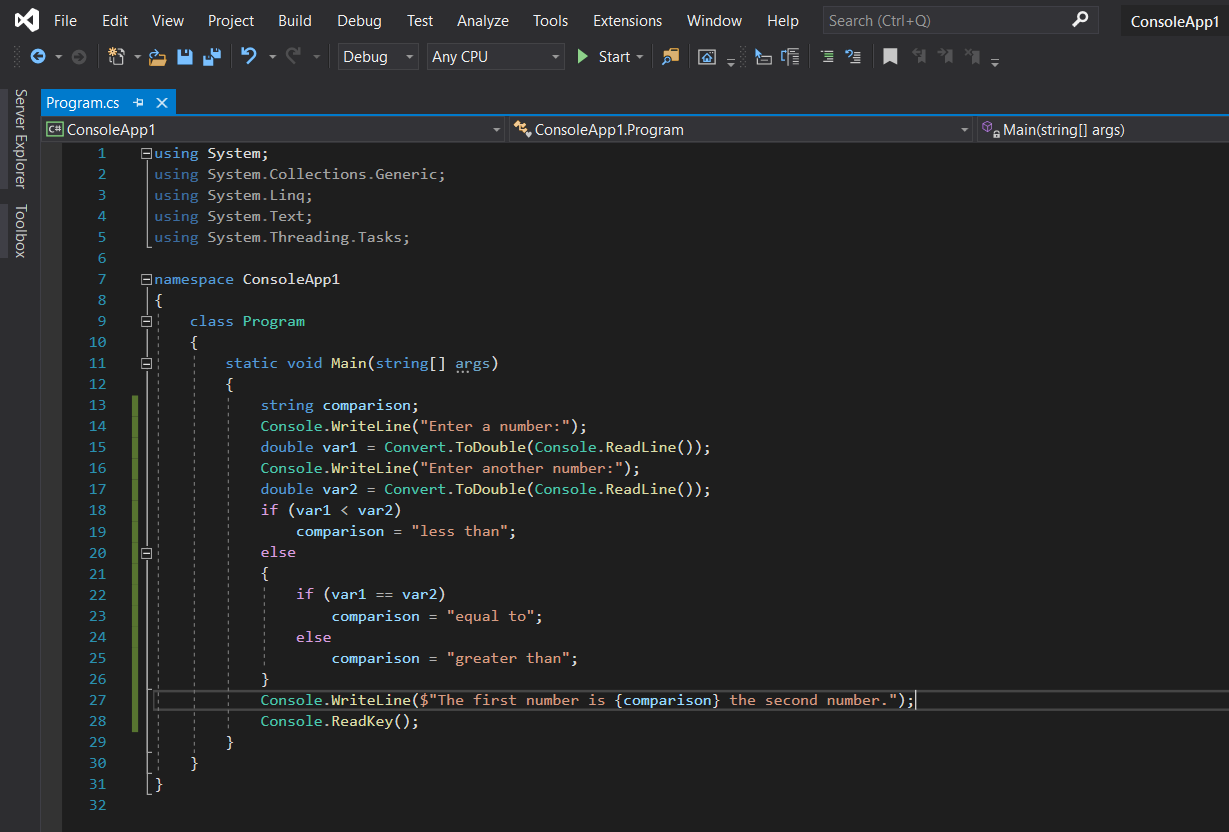
char1 = "Less than 10";

else

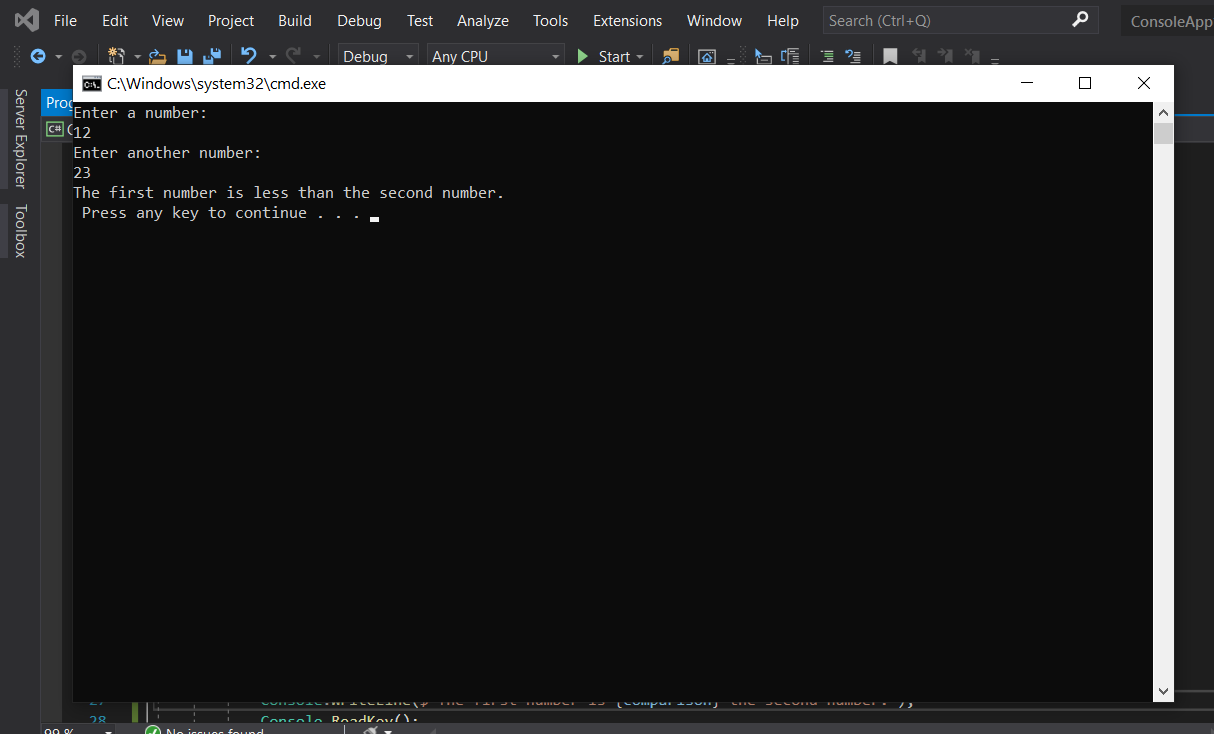
char1 = "Greater than or equal to 10";

Code such as this, although more verbose, is far easier to read and understand than the equivalent ternary form, and enables far more flexibility.

## **Implementing If Statement in Code**



Let's run and see what result we get from this code



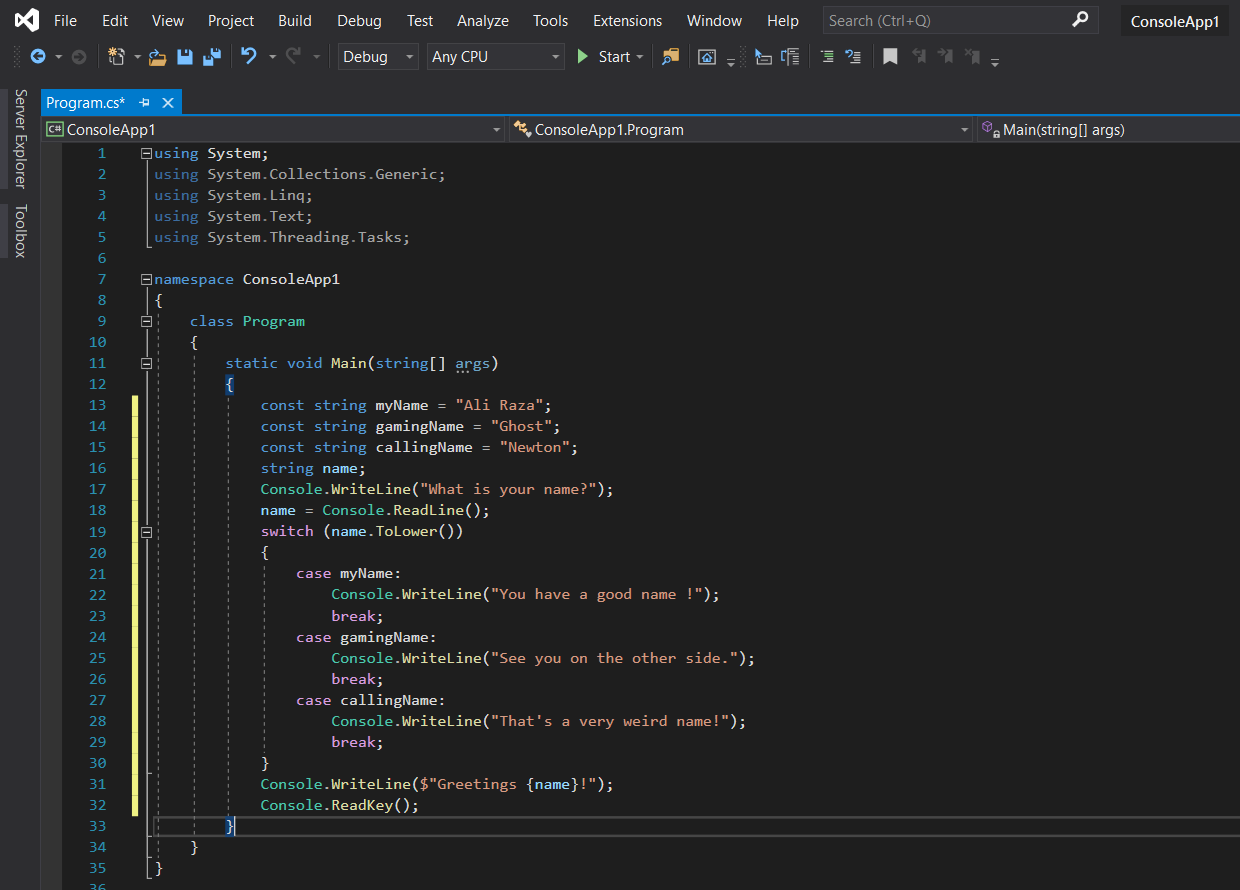
In these situations, consider using a slightly different indentation scheme and contracting the section of code for the else blocks (that is, using a single line of code after the else blocks, rather than a block of code). That way, you end up with structured and nested else-if and if-else statements.

## **Switch Statement**

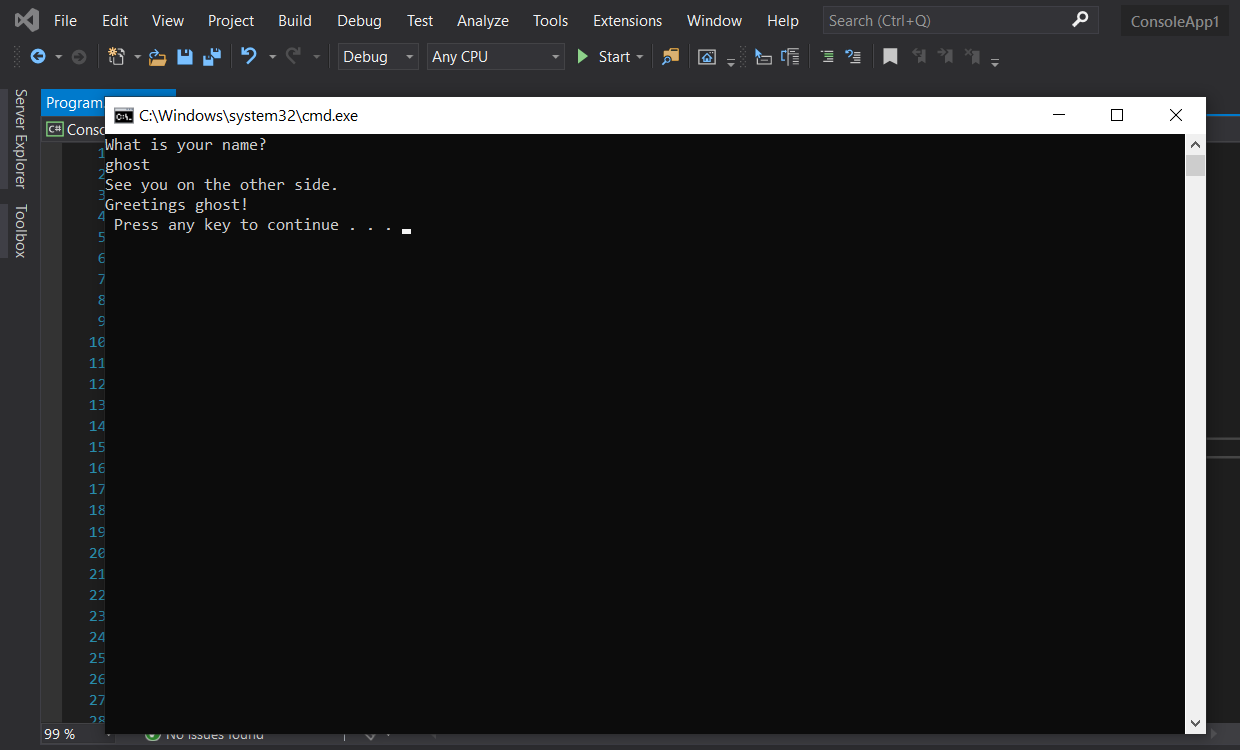
The switch statement is similar to the if statement in that it executes code conditionally based on the value of a test. However, the switch enables you to test for multiple values of a test variable in one go, rather than just a single condition. This test is limited to discrete values, rather than clauses such as “greater than X,” so its use is slightly different. However, it can be a powerful technique?

On completion of the code in each section, you have an additional command, break. It is illegal for the flow of execution to reach a second case statement after processing one case block, but though the break command it becomes easier for the compiler where one statement ends or breaks and where the other one starts. Let's code an example for you guys to understand what is happening and what I am talking about here.

## **Implementing Switch Statement in Code**



Let's run this code and see what happens



# **Looping**

Looping refers to the repeated execution of statements. This technique comes in very handy because it means that you can repeat operations as many times as you want (thousands, even millions, of times) without having to write the same code each time.

Writing the same code 10 times seems a bit wasteful, and what if you wanted to change the duration from 10 years to some other value? You'd have to manually copy the line of code the required amount of times, which would be a bit of a pain! Luckily, you don't have to do this. Instead, you can have a loop that executes the instruction you want the required number of times. Another important type of loop is one in which you loop until a certain condition is fulfilled.

## **do Loops**

do loops operates in a very different way than other loops. The code you have marked out for looping is executed, a Boolean test is performed, and the code executes again if this test evaluates to true, and so on. When the test evaluates to false, the loop exits.

For example, you could use the following to write the numbers from 1 to 10 in

a column:

int i = 1;

do

{

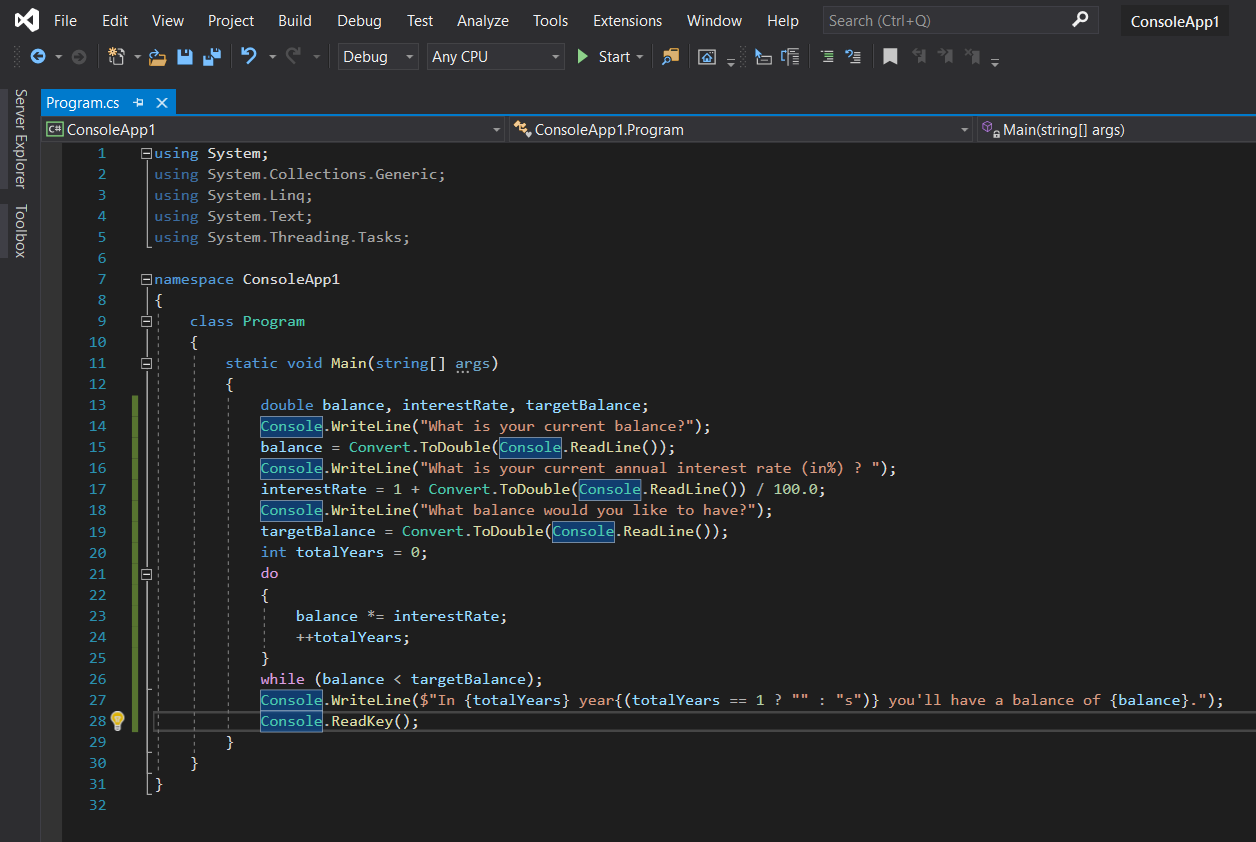
Console.WriteLine("{0}", i++);

}

while (i <= 10);

Here, you use the suffix version of the ++ operator to increment the value of i after it is written to the screen, so you need to check for i <= 10 to include 10 in the numbers written to the console.

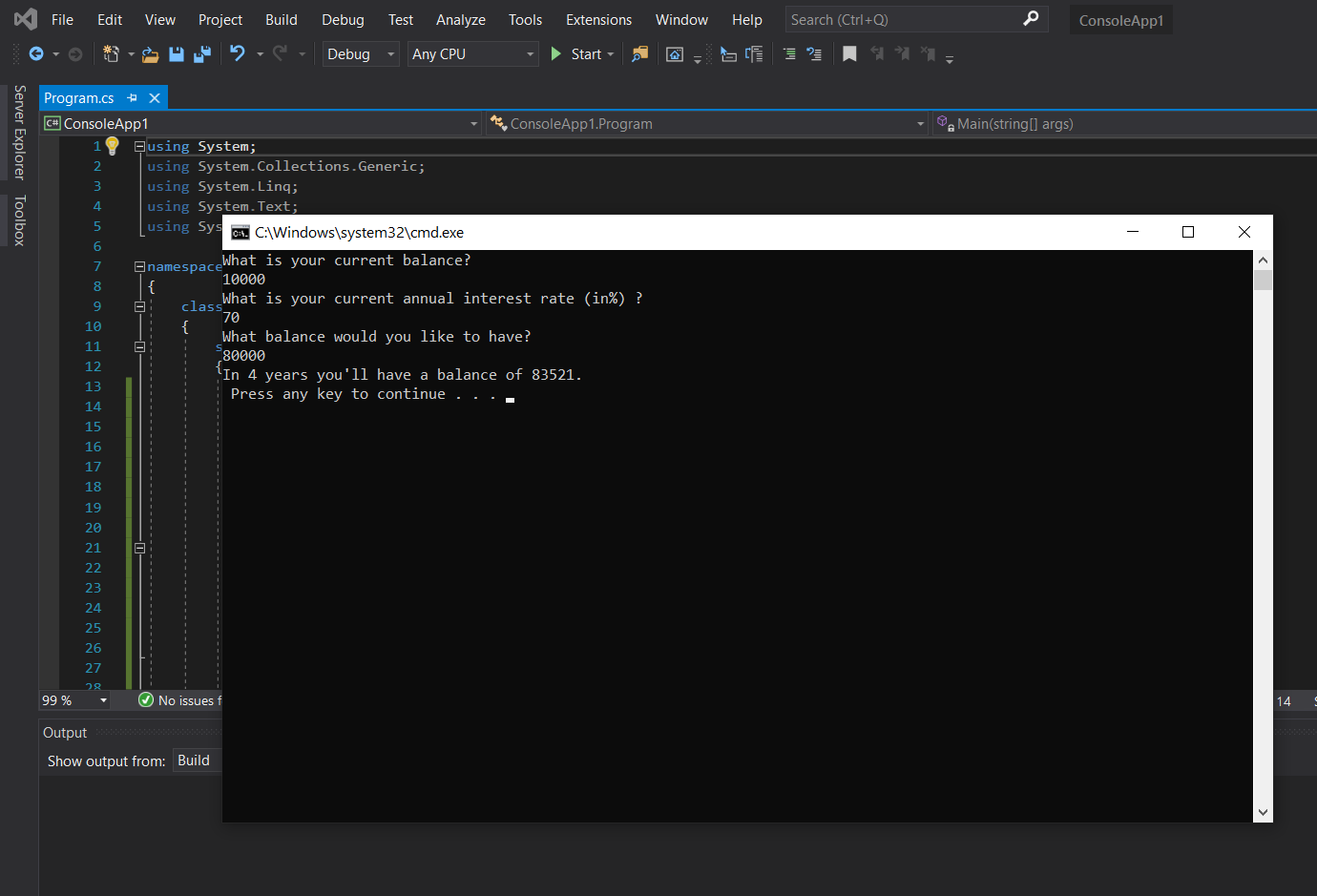
Let's code an example for you guys.



Let's run the above code and see what happens.

Don’t get confused about the interest rate or the balance calculation program here it is very simple just go and watch the video where I’ll explain each and everything here.

Let's move towards the execution of the program.



I hope that you have understood the concept of using the do loop here so let's move to our next loop which is the while loop.

## **while Loop**

while loops are very similar to do loops, but they have one important difference. The Boolean test in a while loop takes place at the start of the loop cycle, not at the end. If the test evaluates to false, then the loop cycle is never executed. Instead, program execution jumps straight to the code following the loop.

Here's how while loops are specified:

They can be used in almost the same way as do loops:

int i = 1;

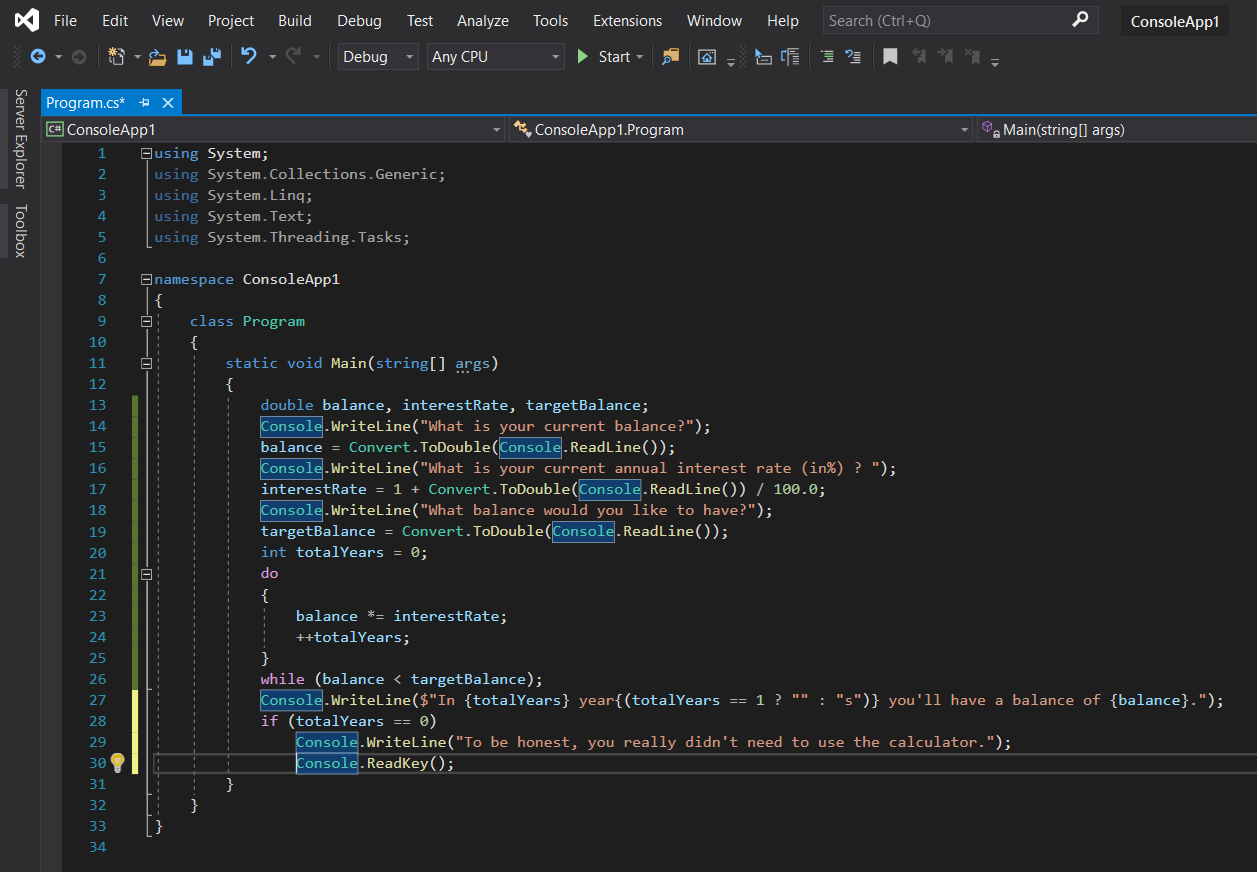
while (i <= 10)

{

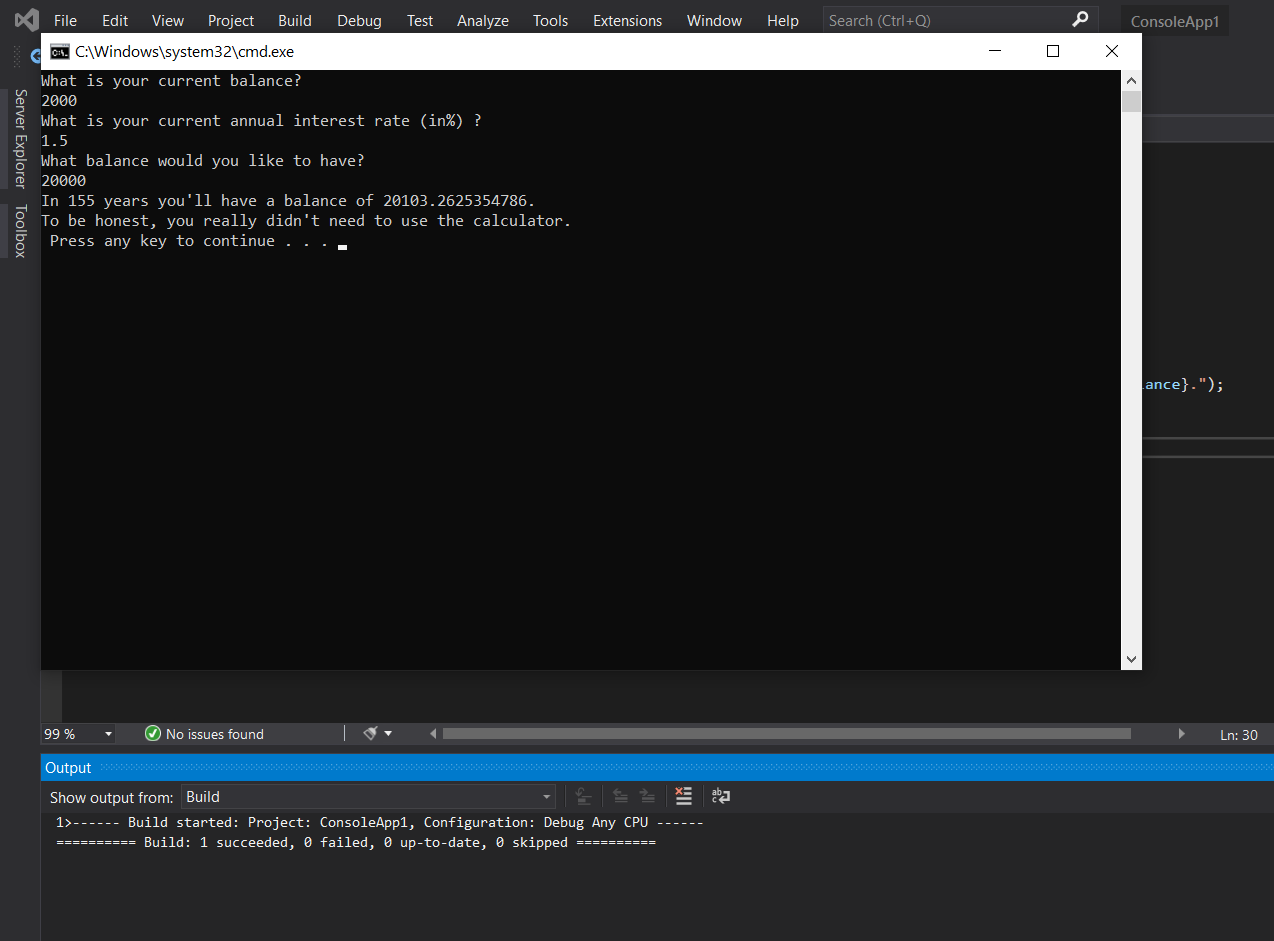
Console.WriteLine($"{i++}");

}

Let's also code an example for while loop.



Let's run it and see what happens here.



## **for Loop**

Another type of loop is the for loop. This type of loop executes a set number of times and maintains its counter. To define a for loop you need the following information:

* A starting value to initialize the counter variable
* A condition for continuing the loop, involving the counter variable
* An operation to perform on the counter variable at the end of each loop cycle

For example, if you want a loop with a counter that increments from 1 to 10 in steps of one, then the starting value is 1, the condition is that the counter is less than or equal to 10, and the operation to perform at the end of each cycle is to add 1 to the counter.

Earlier, you used to do and while loops to write out the numbers from 1 to 10. The code that follows shows what is required to do this using a for loop:

int i;

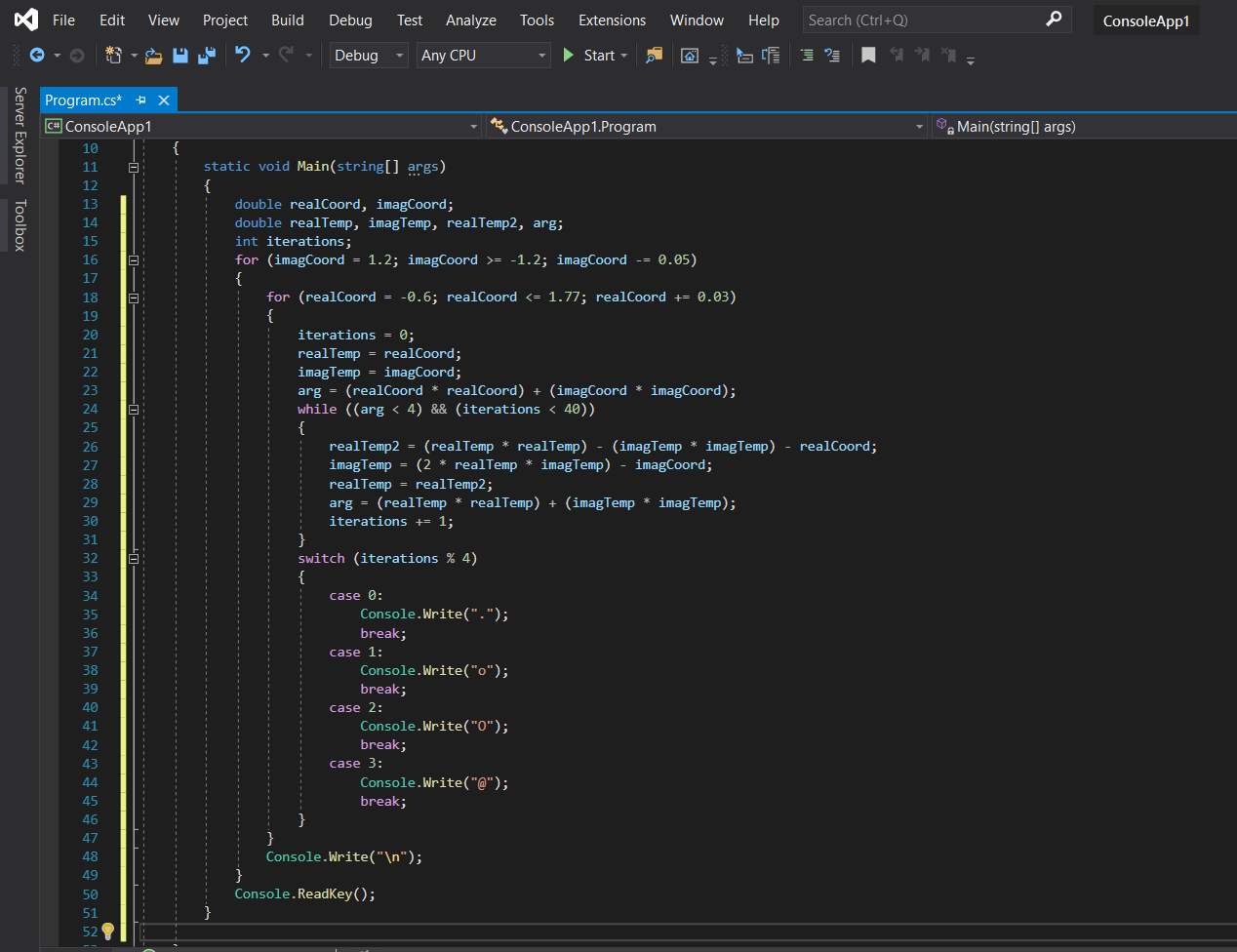
for (i = 1; i <= 10; ++i)

{

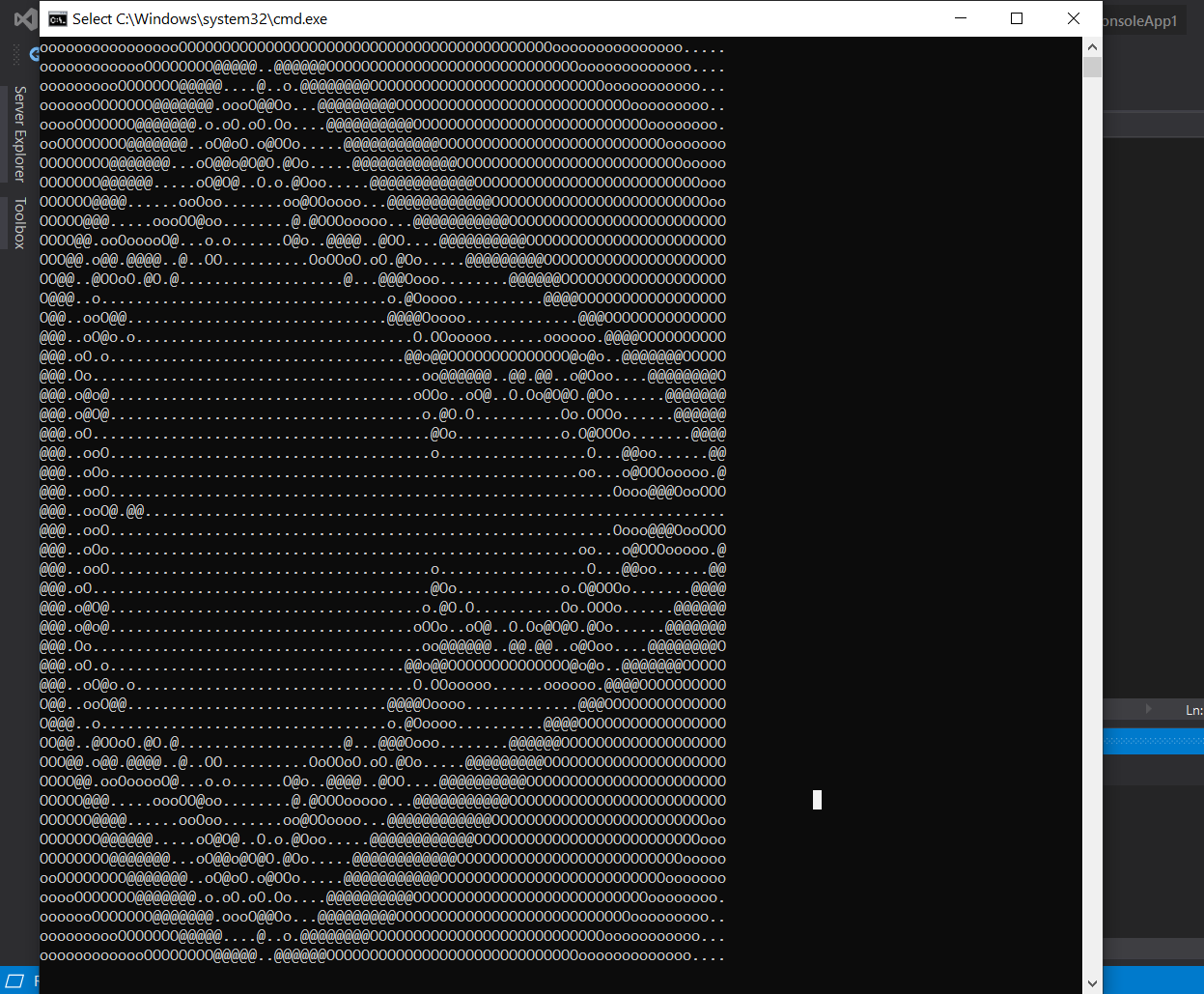
Console.WriteLine($"{i}");

}

Let's code an example for a better understanding of you guys.



Let's run this code and let me show you magic!



Amazing right? It took like 2.5 hours to build such animation on the screen but worth it. You can call it the dark water ship from Star Wars or an eagle ray whatever you would like to call it. Let's pack this up and move to the next loop.

## **Interrupting Loops**

Sometimes you want finer-grained control over the processing of looping code. C# provides commands to help you here:

break — Causes the loop to end immediately

The break command simply exits the loop, and execution continues at the first line of code after the loop, as shown in the following example:

int i = 1;

while (i <= 10)

{

if (i == 6)

break;

Console.WriteLine($"{i++}");

}

This code writes out the numbers from 1 to 5 because the break command causes

the loop to exit when i reaches 6.

continue — Causes the current loop cycle to end immediately (execution continues with the next loop cycle)

continue only stops the current cycle, not the whole loop, as shown here:

int i;

for (i = 1; i <= 10; i++)

{

if ((i % 2) == 0)

continue;

Console.WriteLine(i);

}

In the preceding example, whenever the remainder of i divided by 2 is zero, the continue statement stops the execution of the current cycle, so only the numbers 1, 3, 5, 7, and 9 are displayed.

return — Jumps out of the loop and its containing function

## **Infinite Loops**

It is possible, through both coding errors and design, to define loops that never end, so-called infinite loops. This can be useful, and you can always exit such loops using code such as break statements or manually by using the Windows Task Manager. However, when this occurs by accident, it can be annoying. Consider the following loop, which is similar to the for loop in the previous section:

int i = 1;

while (i = 10)

{

if ((i % 2) == 0)

continue;

Console.WriteLine($"{i++}");

}

Here, i isn't incremented until the last line of code in the loop, which occurs after the continue statement. If this continue statement is reached (which it will be when i is 2), the next loop cycle will be using the same value of i, continuing the loop, testing the same value of i, continuing the loop, and so on.

That’s it from my side I hope you have enjoyed this session. Stay tuned for future tutorials. 😊