100 XP

Exercise - Increment and decrement values

10 minutes

The final basic operations you'll learn about in this module is how to increment and decrement values using special operators that are combinations of symbols.

Increment and decrement

Frequently, you'll need to increment and/or decrement values, especially when you're writing looping logic or code that interacts with a data structure.

The `+=` operator adds and assigns the value on the right of the operator to the value on the left of the operator. So, lines two and three in the following code snippet are the same:

C#

```
int value = 0;     // value is now 0.
value = value + 5; // value is now 5.
value += 5;        // value is now 10.
```

The `++` operator increments the value of the variable by 1. So, lines two and three in the following code snippet are the same:

C#

```
int value = 0;     // value is now 0.
value = value + 1; // value is now 1.
value++;           // value is now 2.
```

These same techniques can be used for subtraction, multiplication, and more. The following exercise steps will highlight a few.

> **Note**
>
> Operators like `+=`, `-=`, `*=`, `++`, and `--` are known as *compound assignment* operators because they compound some operation in addition to assigning the result to the variable.

The `+=` operator is specifically termed the *addition assignment* operator.

Write code to increment and decrement a value

1. Select all of the code in the .NET Editor, and press `Delete` or `Backspace` to delete it.
2. Enter the following code in the .NET Editor:

C#

```
int value = 1;

value = value + 1;
Console.WriteLine("First increment: " + value);

value += 1;
Console.WriteLine("Second increment: " + value);

value++;
Console.WriteLine("Third increment: " + value);

value = value - 1;
Console.WriteLine("First decrement: " + value);

value -= 1;
Console.WriteLine("Second decrement: " + value);

value--;
Console.WriteLine("Third decrement: " + value);
```

3. Run the code. You should see the following output:

Output

```
First increment: 2
Second increment: 3
Third increment: 4
First decrement: 3
Second decrement: 2
Third decrement: 1
```

Note

In the "second increment", you used `value += 1;`. However you could have used any literal `int` value (or a variable) to increment that amount. The same holds true for the "second decrement": `value -= 1;`.

Position the increment and decrement operators

Both the increment and decrement operators have an interesting quality — depending on their position, they perform their operation before or after they retrieve their value. In other words, if you use the operator before the value as in `++value`, then the increment will happen *before* the value is retrieved. Likewise, `value++` will increment the value after the value has been retrieved.

Use the increment operator before and after the value

1. Delete the code from the previous steps and enter the following code into the .NET Editor:

```
C#

int value = 1;
value++;
Console.WriteLine("First: " + value);
Console.WriteLine($"Second: {value++}");
Console.WriteLine("Third: " + value);
Console.WriteLine("Fourth: " + (++value));
```

2. Run the code. You should see the following output:

```
Output

First: 2
Second: 2
Third: 3
Fourth: 4
```

Notice this line of code:

```
C#

Console.WriteLine($"Second: {value++}");
```

There's two steps to this line:

1. Retrieve the current value of the variable `value` and use that in the string interpolation operation.
2. Increment the value.

The next line of code confirms that the value was, in fact, incremented.

```
C#

Console.WriteLine("Third: " + value);
```

In contrast, consider the last line of code:

```
C#

Console.WriteLine("Fourth: " + (++value));
```

Here, the order of operations is switched because the `++` operator is placed before the operand `value`.

1. Increment the value.
2. Retrieve the new incremented value of the variable `value` and use that in the string operation.

While not strictly necessary, you added parenthesis around the expression `(++value)` to improve readability. Seeing so many `+` operators next to each other seems like it could be misunderstood by other developers. Stylistic decisions like this are subjective. However, since you'll write the code once but read it many times, you should prioritize readability.

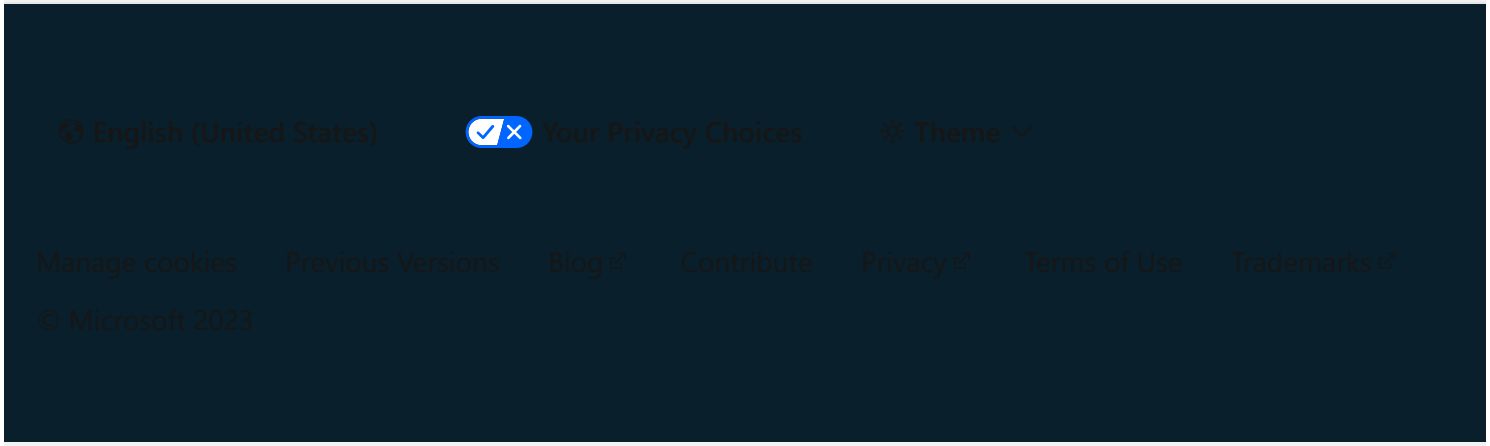
Recap

Here's what you've learned so far about mathematical operations in C#:

- Use compound assignment operators like `+=`, `-=`, `*=`, `++`, and `--` to perform a mathematical operation like increment or decrement, then assign the result into the original variable.
- Increment and decrement operators perform differently depending on whether the operator is before or after the operand.

Module complete:

Unlock achievement



.NET Editor

Press CTRL + M, TAB to exit the editor

Clear Run

```
1
```

Output