100 XP

Exercise - Perform addition with implicit data conversion

10 minutes

Often, you'll want to perform mathematical operations on numeric data. You'll start with addition in this unit, and expand to other operations in the next unit because there's an important lesson to learn about how the C# compiler parses and interprets your code.

Add two numeric values

To add two numbers together, you'll use the *addition operator*, which is the plus symbol `+`. Yes, the same plus symbol `+` that you use for string concatenation is also used for addition. The reuse of one symbol for multiple purposes is sometimes called "overloading the operator" and happens frequently in C#.

In this instance, the C# compiler understands what you're attempting to do. The compiler parses your code and sees that the `+` (the operator) is surrounded by two numeric values (the operands). Given the data types of the variables (both are `int`s), it figures out that you intended to add those two values.

1. Enter the following code into the .NET Editor:

```
C#

int firstNumber = 12;
int secondNumber = 7;
Console.WriteLine(firstNumber + secondNumber);
```

2. Run the code and you'll see the following result in the output console:

```
Output

19
```

Mix data types to force implicit type conversions

What happens if you try to use the `+` symbol with both `string` and `int` values?

1. Modify the code you wrote to match the following code:

```
C#

string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " + widgetsSold + " widgets.");
```

2. Run the code and you'll see the following result in the output console:

```
Output

Bob sold 7 widgets.
```

In this case, the C# compiler understands that you want to use the `+` symbol to concatenate the two operands. It deduces this because the `+` symbol is surrounded by operands of `string` and `int` data types. So, it attempts to implicitly convert the `int` variable `widgetsSold` into a `string` temporarily so it can concatenate it to the rest of the string. The C# compiler tries to help you when it can, but ideally, you would be explicit about your intentions.

Attempt a more advanced case of adding numbers and concatenating strings

1. Modify the code you wrote to match the following code:

```
C#

string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " + widgetsSold + 7 + " widgets.");
```

2. Run the code and you'll see the following result in the output console:

```
Output

Bob sold 77 widgets.
```

Instead of adding the `int` variable `widgetsSold` to the literal `int` `7`, the compiler treats everything as a string and concatenates it all together.

Add parentheses to clarify your intention to the compiler

1. Modify the code you wrote to match the following code:

```
C#

string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " + (widgetsSold + 7) + " widgets.");
```

2. Run the code and you'll see the following result in the output console:

```
Output

Bob sold 14 widgets.
```

The parentheses symbol `()` becomes another overloaded operator. In this case, the opening and closing parentheses form the *order of operations* operator, just like you might use in a mathematical formula. You indicate that you want the inner-most parentheses resolved first resulting in the addition of `int` values `widgetsSold` and the value `7`. Once that is resolved, then it will implicitly convert the result to a string so that it can be concatenated with the rest of the message.

> **Note**
>
> You should probably avoid performing both a calculation and concatenation in a single line of code. The point here is to help you understand how to view operators and operands the way the compiler does.

Recap

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

- You can perform mathematical-like addition operations on numbers.

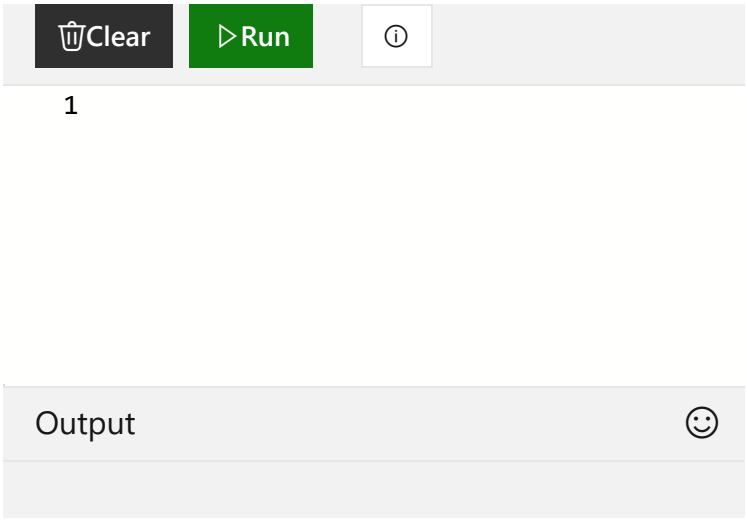
- Both string concatenation and addition use the plus `+` symbol. This is called *overloading an operator*, and the compiler infers the proper use based on the data types it's operating on.
- When it can, the C# compiler will implicitly convert an `int` into a `string` if it's obvious that the developer is trying to concatenate the string representation of a number for presentation purposes.
- Use parentheses to define an order of operations to explicitly tell the compiler that you want to perform certain operations before other operations.

Module complete:

Unlock achievement

.NET Editor

Press CTRL+M, TAB to exit the editor


Clear
Run
1
Output