

CHAPTER 3

Dealing with Output

You already know all the main steps that you should take when developing a program in the C# language. In addition, you have already seen the important statement `Console.WriteLine`, which displays data on your user's screen. In this chapter, you will extend your knowledge of this statement. I will also show you other possibilities for the output.

Producing Numeric Output

You already know how to display some text. In this section, you will learn how to display a number.

Task

You will write a program that displays the number 37 (see Figure 3-1).

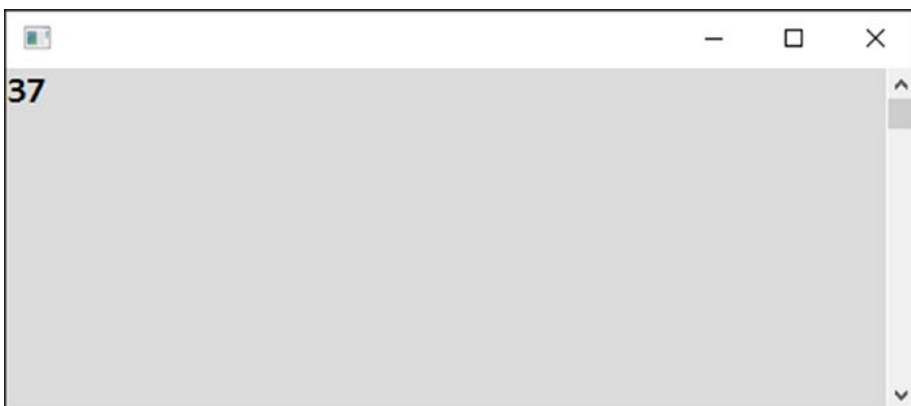


Figure 3-1. *The program in action*

Solution

In Visual Studio, create new project called Numeric Output. The code is similar to the previous program you wrote in Chapter 2, as shown here:

```
static void Main(string[] args)
{
    // Output of a number to the user
    Console.WriteLine(37);

    // Waiting for Enter
    Console.ReadLine();
}
```

Note In this example, and all the following examples in the book, I show you just the block of code after the line with the `Main` word. This is the block of code you are in control of; in other words, it's the block of code you change. The rest of the `Program.cs` source code should remain intact the same way you left it in your first program from the previous chapter.

To be sure you understand me, the whole source code looks like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Numeric_output
{
    class Program
    {
        static void Main(string[] args)
        {
            // Output of a number to the user
            Console.WriteLine(37);
        }
    }
}
```

```
        // Waiting for Enter
        Console.ReadLine();
    }
}
```

But, again, this is the last time you will see the whole source code. There is no need to repeat the Visual Studio–generated code each time I show an example because you will never change it. If you are ever in doubt, you can consult the complete sample projects accompanying the book.

After typing in the code, launch the program using the F5 key. To terminate the program, press Enter.

Discussion

Unlike with text, you do not surround numbers with quotes.

Of course, you could surround “37” in quotes, but there is a profound difference between the number 37 and the text “37”—you can calculate with numbers. That is why you are learning now how to work with numbers correctly.

Making Calculations

The next task is to make a simple calculation.

Task

You are going to display to the user what 1 plus 1 is (see Figure 3-2).



Figure 3-2. 1 plus 1

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Output of a calculation
    Console.WriteLine(1 + 1);

    // Waiting for Enter
    Console.ReadLine();
}
```

Type it in and launch the program!

Note

In programming, this kind of calculation (generally, a combination of values) is called an *expression*.

Making More Complex Calculations

Of course, you do not need a computer to add 1 to 1. But what about 1 plus 2 times 3? Do you think this is ridiculously trivial again? Wait just a minute because even in this simple case mistakes are easy to make!

Task

You'll create a program to add 1 plus 2 times 3.

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Multiplication has greater priority
    Console.WriteLine(1 + 2*3);

    // Forcing priority using parentheses
    Console.WriteLine((1 + 2)*3);

    // Waiting for Enter
    Console.ReadLine();
}
```

The launched program looks like Figure 3-3.

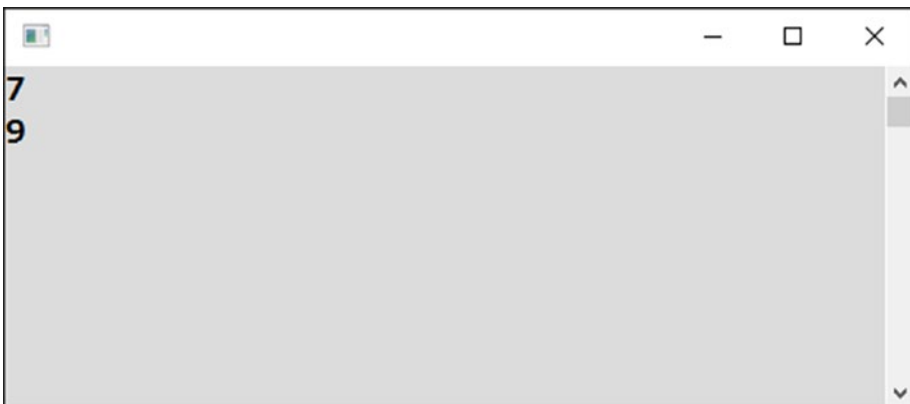


Figure 3-3. *Doing more complex calculations*

Discussion

Note the following about this program:

- The purpose of this task was to show you that you always have to know what exactly needs to be calculated. In this example, you have to make up your mind about whether you want to do addition first or multiplication first.
- In basic math rules, multiplication and division have higher priority than addition or subtraction. It is the same in programming as in mathematics. If you first want to add 1 to 2 and then multiply by 3, you need to use parentheses around the 1 and 2.
- I have not used spaces around the multiplication symbol (asterisk), but this has nothing to do with the calculation order. It just looks better to me. In C#, spaces and line breaks do not matter. (Of course, you should not break a word in the middle.)
- Finally, the example shows that the computer executes program statements in the order they are written. This means from the top down.

Joining Text

You will now explore that the plus operator (+) can be used also with text, not just with numbers. In other words, it adds text together.

Task

The task is to explore how to add text together (see Figure 3-4).

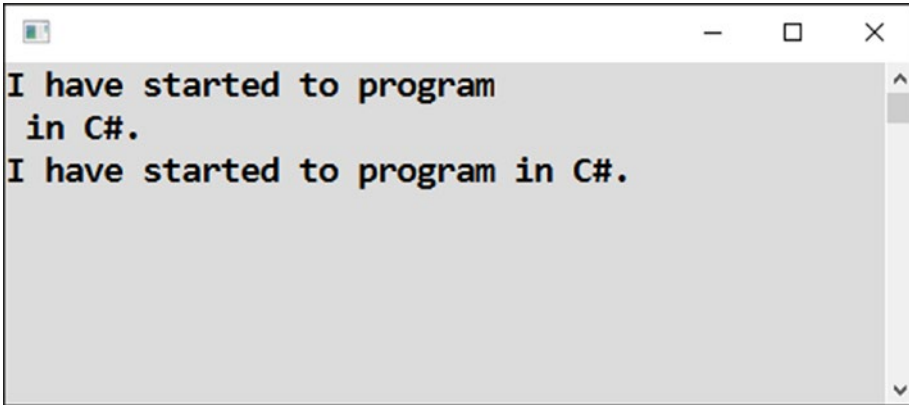


Figure 3-4. Joining text

Solution

Here is the code:

```
static void Main(string[] args)
{
    // Normal text
    Console.WriteLine("I have started to program");

    // Also normal text
    Console.WriteLine(" in C#.");

    // Joining two texts using plus sign
    Console.WriteLine("I have started to program" + " in C#.");

    // Waiting for Enter
    Console.ReadLine();
}
```

Note the space before the *in* preposition!

Outputting Special Characters

Sometimes you need to output a special character to the screen. Here are some examples:

- Output Enter to terminate a line.
- Output a quote mark. (Quotes in C# serve as text delimiters, so they must be treated specially.)
- Output a Unicode character (of course, if your font knows how to draw it).

Task

Now you will write a program that shows how to work with special characters.

Solution

To work with special characters, you use *escape sequences*. In C#, an escape sequence starts with a backslash.

```
static void Main(string[] args)
{
    // Multiline output
    Console.WriteLine("First line\r\nSecond line");

    // I prefer specifying "Enter" in more human form
    Console.WriteLine("First line" + Environment.NewLine + "Second line");

    // Text containing a quote
    Console.WriteLine("The letter started so sweet: \"My Darling\"");

    // Unicode characters, in this case Greek beta
    Console.WriteLine("If the font knows, here is Greek beta: \u03B2");
}
```



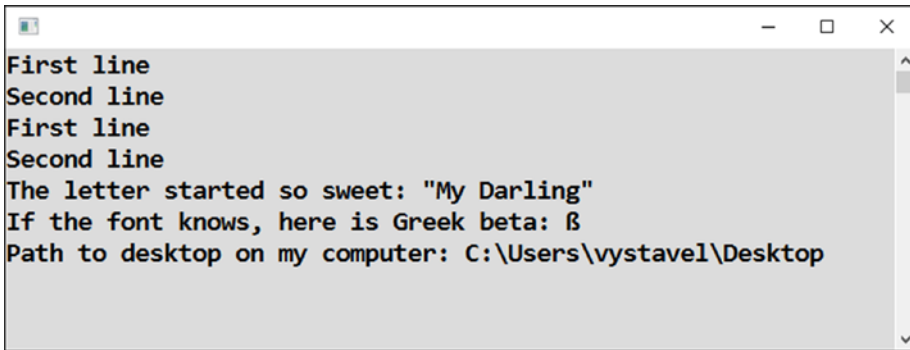
```

// Backslashes themselves need to be doubled
Console.WriteLine("Path to desktop on my computer: " + "C:\\Users\\
vystavel\\Desktop");

// Waiting for Enter
Console.ReadLine();
}

```

The result should look like Figure 3-5.



```

First line
Second line
First line
Second line
The letter started so sweet: "My Darling"
If the font knows, here is Greek beta: β
Path to desktop on my computer: C:\Users\vystavel\Desktop

```

Figure 3-5. Working with special characters

Discussion

Note the following about this program:

- In C#, a backslash in text introduces a so-called escape sequence. But what if you want to output a backslash? Then you need to double it. This is often the case when dealing with file paths in the Windows operating system.
- Console applications will recognize even the simple `\n` as a line terminator (meaning Enter). However, in many other C# programs, you need “the whole Enter,” which is signified with `\r\n`. That is why you used it in this program. You also used `Environment.NewLine`, which is definitely the best alternative since it is nicely human readable.

Using Preformatted Text

Sometimes you might want to display multiline text in one go (see Figure 3-6).

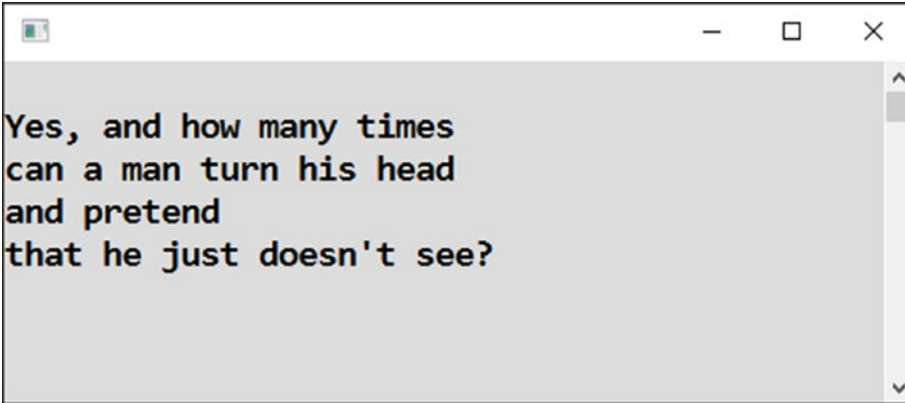


Figure 3-6. Multiline text

Task

You will create a program to display multiline text.

Solution

You prepend the opening quote mark of the text with the at (@) sign, as shown here:

```
static void Main(string[] args)
{
    // Bob Dylan...
    Console.WriteLine(@"
Yes, and how many times
can a man turn his head
and pretend
that he just doesn't see?
");

    // Waiting for Enter
    Console.ReadLine();
}
```

Note

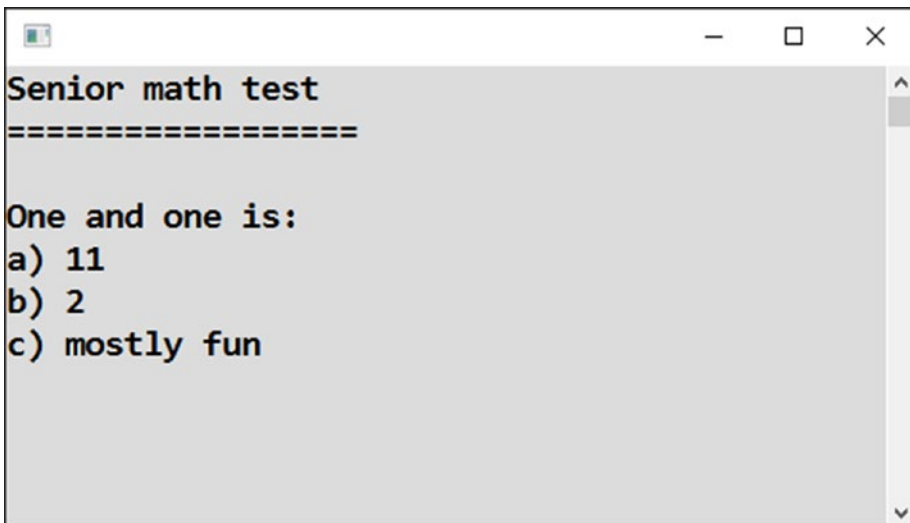
The at (@) sign also switches off escape sequences. That is why you might find it useful when dealing with file paths in Windows (mentioned earlier); in that case, you do not have to double each backslash.

Adding 1 and 1

In the next task, you will return to the problem of adding 1 to 1. Are you wondering why I am returning to such a trivial task? Well, even doing something as simple as adding 1 to 1 can go wrong. Let's see.

Task

The task is to explore different ways of putting two numbers together (see Figure 3-7).



```
Senior math test
=====

One and one is:
a) 11
b) 2
c) mostly fun
```

Figure 3-7. Putting numbers together

Solution

Here is the code:

```

static void Main(string[] args)
{
    // Pay special attention when mixing texts with numbers!
    Console.WriteLine(
@"Senior math test
=====
One and one is:");
    Console.WriteLine("a) " + 1 + 1);
    Console.WriteLine("b) " + (1 + 1));
    Console.WriteLine("c) " + "mostly fun");

    // Waiting for Enter
    Console.ReadLine();
}

```

Discussion

When you mix numbers with text, the result might appear different from what you expect!

Let's consider the first answer (a). The computer calculates the whole expression from left to right. First, it takes the text a) and a number (the first 1). It joins them together to be a) 1. Then, it takes this new text and the final number (the second 1) and again joins them together to obtain the text a) 11.

The second answer (b) is different. The parentheses make the computer perform the addition of the numbers first, joining the text on the left only afterward.

Sometimes it may be more transparent to precalculate the intermediate results and store them in variables. This is what you are going to study in the next chapter. Of course, variables have many more uses than this, as you are going to see.

Summary

In this chapter, you explored several possibilities that the `Console.WriteLine` statement gives you for different kinds of output. Specifically, you have learned the following:

- In addition to text, you can work with numbers in your programs. Unlike with text, you do not surround numbers with quotes.
- You can combine several values into expressions. For this purpose, you use operators such as `+`, `-`, and `*`. With numbers, they do ordinary arithmetic. The plus operator works also with text, in which case it joins two pieces of text into a single one.
- In calculations, you always have to be careful about the order in which the result is evaluated. Multiplication and division have precedence over addition and subtraction. To force a different evaluation order, use parentheses.
- Special characters such as quotes or newlines are output using escape sequences starting with backslash.
- You can conveniently output preformatted multiline text by prepending it with an at (`@`) sign.