Debugging Your Code

Errors

There are basically two types of errors in code; *compile errors* and *runtime errors*.

A *compile error* (or *build error*) prevents the C# compiler from being able to process the code and an appropriate message explaining the problem is generated for the programmer, for example, using an incorrect data type, leaving out a semi-colon at the end of a line, missing a closing brace, etc.

A *runtime error* is encountered when the project is being run. Runtime errors are usually the result of trying to perform an invalid operation on a variable. For example, for two integer variables, if:

```
variable1 = 10 / variable2
```

will not generate a compile error, and under most circumstances, will not generate a runtime error. However, if `variable2` has the value 0, the result is undefined and therefore cannot be assigned to the integer `variable1`. A runtime error will occur; the error is referred to as an *exception*, and it is said to be *thrown*.

Debugging Tools

The debugger helps you find and fix runtime errors and logic errors in your application. Using the debugger, you can step through your code, set breakpoints and watches, and inspect and modify program values. As you debug your application, the debug windows provide information about the state of your application.

Task 1: Using the Debugger

1. Open the 12-debugging folder, click on the .sln file.

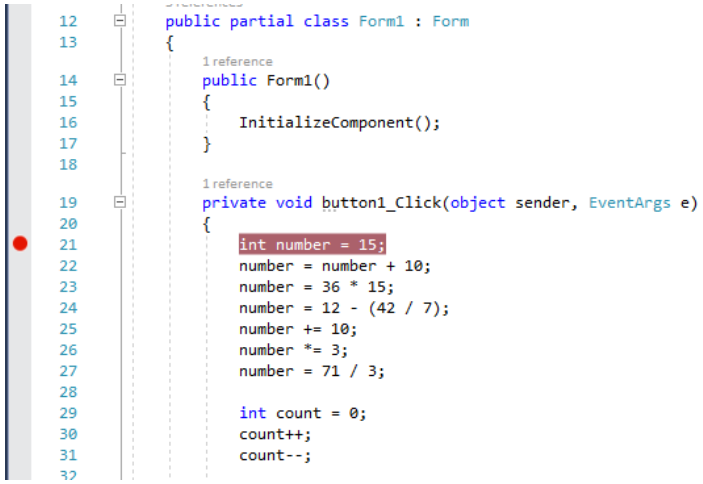
2. In the `button1_Click` event handler, there is the following code:

```
int number = 15;
number = number + 10;
number = 36 * 15;
number = 12 - (42 / 7);
number += 10;
number *= 3;
number = 71 / 3;

int count = 0;
count++;
count--;

string result = "hello ";
result += "again " + result;
MessageBox.Show(result);
result = "the value is: " + count;
result = "";

bool yesNo = false;
```

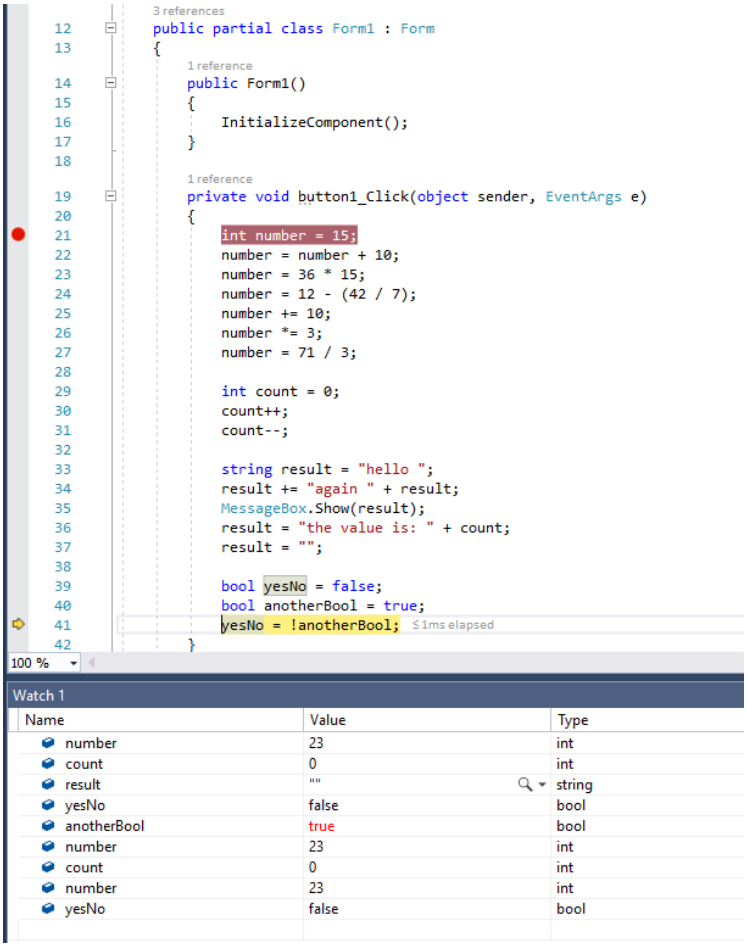
```
bool anotherBool = true;
yesNo = !anotherBool;
```

3. Set a breakpoint by right clicking on the first line of code, and choosing *Breakpoint/Insert Breakpoint.* You could also use the *Debugging* menu, F9 (the breakpoint toggle), or just click in the grey area (*gutter*) to the left of the statement. A red dot will appear in the gutter.

   A *breakpoint* stops the execution of a program at the breakpoint.



4. Run the program. Your program will start as usual, with the form. Click on the button. The IDE will bring up the code editor will stop at the first line. Note how it is highlighted in yellow.



The *watch* window appears at the bottom of the IDE. It has 3 tabs: Autos, Locals and Watch1. Open Watch1, it keeps track of the value of each of the variables as you work through your code.

5. *Step through the code*, using F11, the *StepInto* button on the *Debug* toolbar, or from the *Debug* menu. Stepping through code lets you run your program one line of code at a time. The next line of code does not execute until you tell the debugger to continue. Notice the values of the variables as you step through your code.
   What is the difference between *StepInto* (F11) and *StepOver* (F10)?
   StepOver allows you to progress through the program one line at a time – this does not take you into a method if one is called – but steps over it and stops at the first line after the function.

   StepInto is very similar, except that you are now taken into any method. *StepInto* stops at the first line of code inside the method.

   If you want to stop stepping through a program at any stage you can choose *Debug/Stop Debugging* or *Shift+F5,* or the program reset button on the Debug toolbar.

6. When you want to resume, press *F5* or *Debug/Continue.*

Catching Exceptions

While your program may compile it may still crash when it runs. When an error occurs predictably and under known conditions (eg dividing a number by zero or a user entering a string when an integer value is expected) these errors can be caught as exceptions with appropriate feedback and your program can continue running.

The mantra is “it is easier to ask for forgiveness than for permission”.

The *try* block contains the code that might throw the exception.
The *catch* block contains the code that is executed when exceptions are caught.
The *finally* block contains code that is always executed; after a *try* block if no exception occurs or after a *catch* block if an exception occurs.

For example:

```
int number1 = 10;
int number2 = 0;

try
{
     int number3 = number1 / number2;
}
catch (DivideByZeroException d)
{
     MessageBox.Show("An error has occurred: " + d.Message);
}
finally
{
     MessageBox.Show("Put clean up code here");
}

MessageBox.Show("And now continuing with the program...");
```
