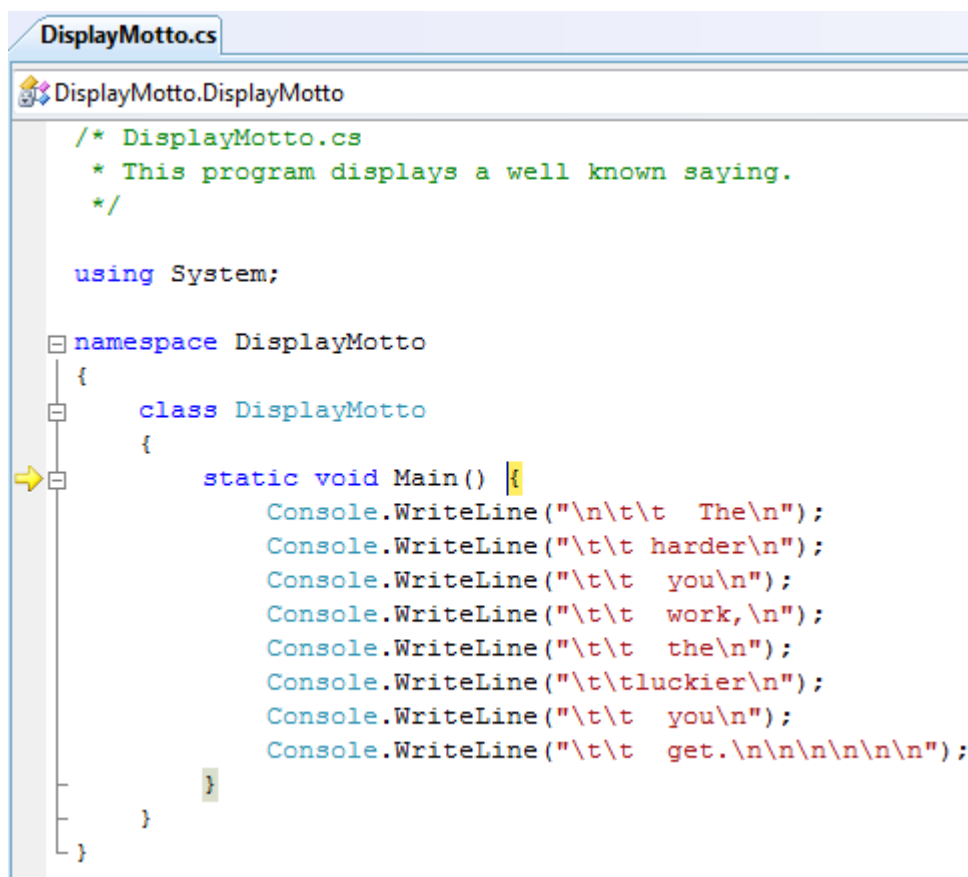


First Steps in the Visual Studio Debugger

NB: in the following material, symbols such as **F5** and **F10** refer to **functions keys**, which are located in the top row of keys, on most desktop PCs. However, notebook computers often differ.

1. If you have not already done so, download the file **DebuggerSection.zip** and unzip it into any convenient location, e.g. **My Documents**.
2. From the **DisplayMotto** subfolder, open the **DisplayMotto** solution in Visual Studio.
3. To see what the program does, run it by using the menu command **Debug → Start Without Debugging** (or use the shortcut, **Ctrl+F5**). Once you've seen what the program displays, close its console window (but not Visual Studio).
4. Now select the command **Debug → Step Over** (or use the shortcut, **F10**). The **DisplayMotto.cs** file should automatically open (unless you've opened it already), and a yellow arrow will appear in the left-hand margin, as show below. This shows that the program has started running, but the Debugger has **paused** its execution right at the start, i.e. *before* the **Main()** method is executed.

This **Step Over** command allows you to run your program *one step at a time*. (That explains the **Step** part of the name. We'll explain the **Over** part later.)




```
DisplayMotto.cs
DisplayMotto.DisplayMotto

/* DisplayMotto.cs
 * This program displays a well known saying.
 */

using System;

namespace DisplayMotto
{
    class DisplayMotto
    {
        static void Main() {
            Console.WriteLine("\n\t\t The\n");
            Console.WriteLine("\t\t harder\n");
            Console.WriteLine("\t\t you\n");
            Console.WriteLine("\t\t work,\n");
            Console.WriteLine("\t\t the\n");
            Console.WriteLine("\t\t luckier\n");
            Console.WriteLine("\t\t you\n");
            Console.WriteLine("\t\t get.\n\n\n\n\n\n");
        }
    }
}
```

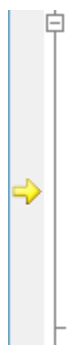
5. Use the same command, **Debug → Step Over** (or **F10**), so that the program now enters the **Main()** method and is *ready* to execute the first **WriteLine** statement, as shown below.



```
static void Main() {
    Console.WriteLine("\n\t\t The\n");
    Console.WriteLine("\t\t harder\n");
    Console.WriteLine("\t\t you\n");
    Console.WriteLine("\t\t work,\n");
    Console.WriteLine("\t\t the\n");
    Console.WriteLine("\t\tluckier\n");
    Console.WriteLine("\t\t you\n");
    Console.WriteLine("\t\t get.\n\n\n\n\n\n");
}
```

It's important to realise that the yellow colour means that the line is *about to be executed*, i.e. it hasn't happened yet. Use **Alt+Tab** (or similar) so that you can see the program's console window. You will see that no program output has occurred so far. Then use **Alt+Tab** (or similar) to return to Visual Studio. (Don't close the program's console window.)

6. Use the same command (repeatedly) to make the program do one step at a time, until the yellow line reaches the position shown below.



```
static void Main() {
    Console.WriteLine("\n\t\t The\n");
    Console.WriteLine("\t\t harder\n");
    Console.WriteLine("\t\t you\n");
    Console.WriteLine("\t\t work,\n");
    Console.WriteLine("\t\t the\n");
    Console.WriteLine("\t\tluckier\n");
    Console.WriteLine("\t\t you\n");
    Console.WriteLine("\t\t get.\n\n\n\n\n\n");
}
```

As before, look at the program's console window to confirm that the program has executed the first four **WriteLine** statements, and then paused. Then return to Visual Studio (without closing the console window).

7. Use the same command to keep stepping through the program, until the yellow line passes the closing brace (}) and execution is complete. (The console window will automatically close.)

Warning: if you accidentally use the same command too many times, your program will start executing from the top again. That's ok if you do want to run it again, but can be confusing if you don't realise that you've made it start again.

Using the Shortcut Keys

As the previous section shows, you often need to use the same command *over and over again*, when using the Debugger. When you are doing it for the first few times, the selection of menu commands *by name* – such as **Debug → Step Over** – is often easier to do. But with a bit of practice, using the equivalent function key – such as **F10** – *is less effort* than using the mouse to select a menu item, especially when you're using the same command repeatedly. (Some notebook computers make it hard to use the function keys, so your mileage may vary.)

As a central reference, the following table summarises the function key equivalents of the main debugging commands. Some of these have not been covered above, and will be discussed later in this workshop, or in future workshops.

Function Key	Debug Command
F5	Start debugging or continue

Shift+F5	Stop debugging
Ctrl+Shift+F5	Restart debugging
F9	Insert or remove a breakpoint
F10	Step over
F11	Step into
Shift+F11	Step out

Some of these commands also appear as buttons in the Debugger's toolbar shown below. Note that this only appears when the Debugger is running. If you move your mouse over any of buttons in Visual Studio, a flyout will appear describing the button's purpose.



Going Faster, Then Pausing — Breakpoints

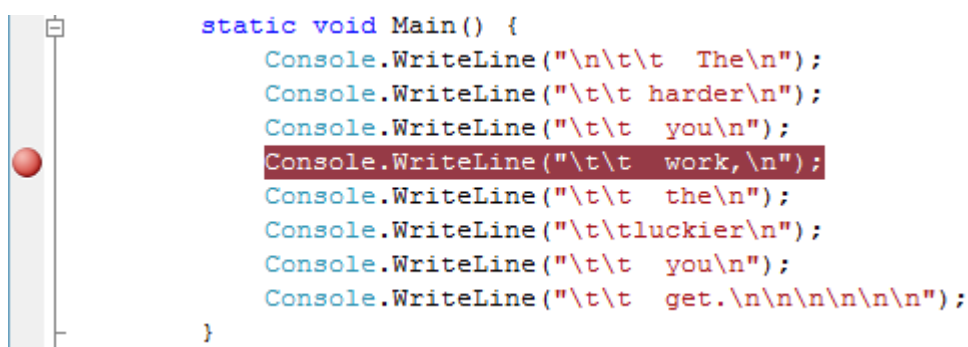
Using the **Step Over** command is a good way to see exactly what a program is doing, but in general, it would become painful if we had to keep telling the computer to do each step. This is particularly true for programs that contain hundreds of lines of code, and for programs that contain loops.

Often what we want is a way to tell the computer to run our program from the start, but to pause the program when it reaches a particular place in the program. The most common way of doing this is to put one or more breakpoints into the program.

A **breakpoint** is a line in your program that you select and when it is reached, the program is paused, and placed into what is called **break mode**.

There are several ways that you can add or set a breakpoint. We start with the simplest.

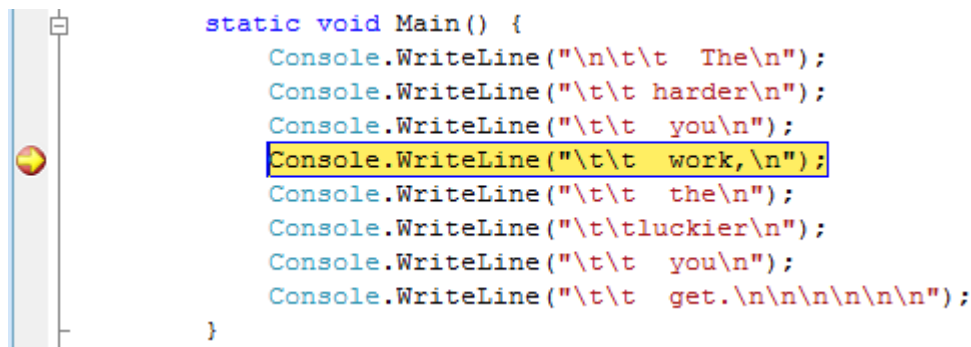
1. If you do not have it open from before, open the **DisplayMotto** project in Visual Studio 2012 and then open **DisplayMotto.cs**.
2. Click *anywhere* on the line that contains the word **work**. Select **Debug → Toggle Breakpoint** from the menu, or use the **F9** keyboard shortcut. The line should appear in red, as shown below.



As the menu option name implies, pressing **F9**, or selecting **Toggle Breakpoint** a second time turns off the breakpoint. Try it a few times if you like, but leave the breakpoint set when you're done.

3. Run the program by selecting the **Debug → Start Debugging** (or **F5**) command. The program runs until it reaches the breakpoint. At that line, the program pauses. As before,

the yellow colour shows the line that is *the next one* to be executed by the program. (As earlier, you can confirm this by looking at the program's console window, if you like.)



4. At this point, the program is paused just as it was earlier. So use **Step Over (F10)** to step to the next line. What happens to the yellow arrow on the red background, shown in the left-hand margin above, and why?
5. Using the same approach as in step 2 just above, set a second breakpoint on the last **WriteLine** line (containing the word **get**). Select the **Debug → Continue (or F5)** command. Again, the program runs until it reaches this second breakpoint.

As well as showing that you can have multiple breakpoints, this shows that you can set breakpoints after your program has started running. (You can even set a breakpoint on a line that your program has already passed, but it will have no effect until the next time that the program reaches that line, if ever.)

6. In general, the program pauses at each breakpoint it encounters, until the program stops running. Select the **Debug → Continue (or F5)** command again. Because there are no more breakpoints, the program executes its remaining statements and then terminates.
7. *Breakpoints remain even when the program stops.* Even if you were to close Visual studio entirely, the breakpoints will still be present when you open this project again (unless you start deleting its files). I.e. breakpoints only go away when you explicitly remove them.

Use **Debug → Start Debugging (or F5)** to run the program again, so that it again stops at the first breakpoint. When it does, use **F9 (or Toggle Breakpoint)** to remove that breakpoint. You don't need click on the line first.

8. At this point, we *could* use **F5** to continue to the next breakpoint. However, there will be times when you're in break mode and you just want to stop the program – perhaps to change it in some way – rather than letting it run. Stop the program by selecting **Debug → Stop Debugging (or Shift+F5)**. It will terminate immediately, i.e. without executing the remaining parts of the program.

For what it's worth, another way to set a breakpoint involves positioning the mouse in the gray margin, to the left of the line of code where you want to set a breakpoint, and then clicking the left mouse button. Some people find this easier than using the menu or **F9**, while others find it harder to position the mouse in the right place. It is also a source of confusion for raw beginners who accidentally click in that margin: "Why has my code turned red? Is there something wrong with it?"

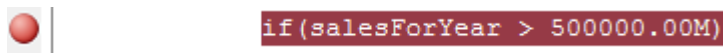
You can now close this solution in Visual Studio.

Examining an if Statement

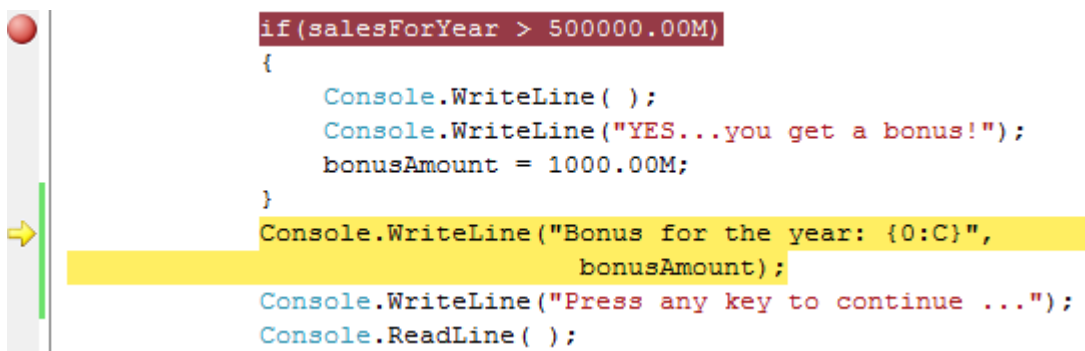
1. From the **Ch05_bonusApp** subfolder, open the **bonusApp** solution in Visual Studio 2012.

This is essentially the same program as Example 5-9 in the Doyle textbook. You can refer to the textbook for further explanation about the parts of the program if you like, but you don't need to have the textbook with you for this part of the workshop.

2. In the Solution Explorer window, double-click on **BonusCalculator.cs** to open it.
3. Set a breakpoint on the line containing the `if`.



4. Run the program by pressing **F5** (or selecting **Debug → Start Debugging**).
5. When the console window appears, type in a number *larger than* 500000, e.g. the value 654321, and then press the **Enter** key.
6. Once you do that, the program will pause because the breakpoint is reached. Though paused, the console window still has the keyboard focus, so you must click back in Visual Studio so that the following keystrokes go to it. Click anywhere within the **BonusCalculator.cs** window.
7. Use **F10** (or **Debug → Step Over**) to see whether the code inside the `if` statement is executed. Keep doing this, until you reach the **WriteLine** line shown in yellow below.



8. Use **F5** (or **Debug → Continue**) to continue running the program.
9. When the “Press any key to continue ...” prompt appears, do so. The program then terminates.
10. Restart the program by pressing **F5** (or **Debug → Start Debugging**).
11. When the console window appears, type in a number *smaller than* 500000, e.g. the value 4321, and then press the **Enter** key.
12. As before, the program will pause because the breakpoint is reached. As before, click anywhere within the **BonusCalculator.cs** window so that Visual Studio gets the following keystrokes.
13. Use **F10** (or **Debug → Step Over**) to see whether the code inside the `if` statement is executed.
14. Once you reach the **WriteLine** line shown in yellow above, use **F5** (or **Debug → Continue**) to continue running the program.
15. When the “Press any key to continue ...” prompt appears, do so. The program then terminates.

The above steps show the basics of how an `if` statement works. If you want to try it again – perhaps with different input values – just go back to step 10 above.

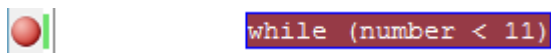
When you’ve seen enough, close this solution in Visual Studio.

Examining a while Statement

1. From the **Ch06_SummedValues** subfolder, open the **SummedValues** solution in Visual Studio 2012.

This is essentially the same program as Example 6-1 (p. 296) in the Doyle textbook. As above, you can refer to the textbook for further explanation about the parts of the program, if you like, but you don't need to have the textbook with you, for this part of the workshop.

2. In the Solution Explorer window, double-click on **SummedValues.cs** to open it.
3. To see the program's output, run it by pressing **F5** (or selecting **Debug** → **Start Debugging**). When the "Press any key to continue ..." prompt appears, do so.
4. Set a breakpoint on the line containing the while.



5. Run the program by pressing **F5** (or selecting **Debug** → **Start Debugging**).

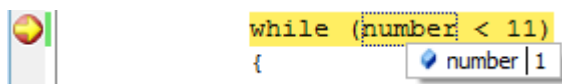
The program will pause at the breakpoint.

6. For the following steps, we want to make sure that numbers are displayed in decimal. Check that the Hex button in the Debugger's toolbar is **not** down. I.e. check that it appears as show in the left-hand image below. If it is down – as shown in the right-hand image below – click it once (and then move your mouse away) to reset it.



7. Hover your mouse above the word **number** in the **while** statement, so a flyout appears as shown below.

"Hover" means to move your mouse so that it's on top of an item, and then leaving the mouse motionless for a brief period. You **don't click** on the item. (If you forget and *do click*, then just move the mouse and try again.)



Here, the flyout tells us that the **number** variable currently has the value 1 (as we would expect, from the assignment statement on the preceding line in the program, i.e.

```
int number = 1;
```

8. Use **F10** (or **Debug** → **Step Over**) to step through the statements that are inside the **while** loop. What happens when you reach the loop's closing brace (}) for the first time?
9. Keep using **F10** (or **Step Over**) to go around the loop a few times. As you do so, occasionally hover the mouse over each of the **number** and **sum** variables, so you can check that you understand what is going on.

10. Keep using **F10** (or **Step Over**) until the program leaves the **while** loop and reaches the **WriteLine** statement.

Hover the mouse over the **number** variable. What is its value? Why has the program left the **while** loop?

11. Use **F5** (or **Debug → Continue**) to continue running the program. When the “Press any key to continue ...” prompt appears, do so.
12. Restart the program by pressing **F5** (or **Debug → Start Debugging**).
13. When it hits the breakpoint this time, you should use **F5** (or **Debug → Continue**) to continue running the program, rather than **F10**.

When you do this, it may seem that nothing has happened. However, when you hover the mouse over the **number** variable, you will see that it now has the value 2.

Use **F5** (or **Debug → Continue**) again, and you will see that it now has the value 3. *Make sure that you understand why this is happening.*

It is a highly useful way of being able to pause each time the program goes around a loop, without having to step through every program line (using **F10**).

14. When you are sure that you understand what the program is doing, use **Shift+F5** (or **Debug → Stop Debugging**) to stop.
15. When you've seen enough, close this solution, and Visual Studio.

We have focused on using the Debugger to show how basic programming constructs, such as statement-sequences, **if** statements, and **while** statements, are executed by your computer. There's a lot more that you can do with the Debugger and future workshops will show you more about how to use it, building on the skills you've developed today.