

Introduction to the Debugger

Debugging, or the process of finding logical errors in your code, can be a very tedious process. Luckily, the MS Visual Studio.NET debugger can allow you to accurately pinpoint your logical errors so that you can fix them and continue. This exercise will briefly introduce you to the debugger by teaching you the simple features of the debugger along with some examples.

Introduction

The Debug Menu (see Figure 1) is where all the functionality for the debugger is located. The major functions are *Start (F5)*, *Step Into (F11)*, *Step Over (F10)*, and *New Breakpoint (Ctrl +B)*. This is the menu that you see while you are coding. If you are in debug mode, you will see a totally different menu (which will be covered later).

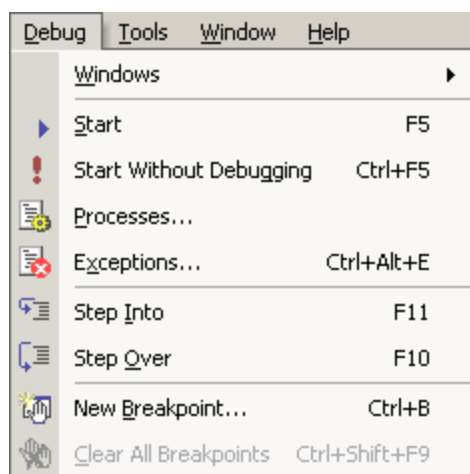


Figure 1 Debug Menu in Visual Studio.NET

Terms

- 1) Breakpoint- Represents a breaking point in your program where execution stops. If a breakpoint is placed on a line, all code up to (but not including that line) has been executed.
 - a. Conditional Breakpoint- Break in execution *after* some condition has been met. Useful in looping conditions.
- 2) Step Over- Steps over the current line. If the current line is a function call, the code for the function will not be debugged but will be executed.
- 3) Step Into- Steps into the current line if the current line is a function call. This is useful when debugging errors in functions.

Debug Process

- 1) You have some code (Figure 2) that you wish to debug. There are two things should happen:
 - a. You read two input values from a file called "Input.txt"
 - b. You are calculating a return on investment by calculating the gain and dividing by initial balance.

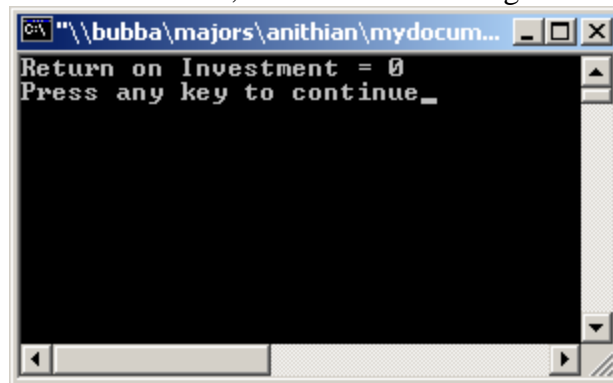
```

#include <string>
#include <fstream>
using namespace std;
int main()
{
    ifstream fin("Input.txt");
    int iInitialBalance=0;
    int iFinalBalance=0;
    fin >> iInitialBalance>>iFinalBalance;
    double dReturnOnInvestment=(iFinalBalance-iInitialBalance)/iInitialBalance;
    return 0;
}

```

Figure 2 C++ Code that you wish to debug

You run the code, and see the following:



The return on investment should be 2.38% instead of 0.

2) In order to determine where the error lies, you can either step through one line at a time from line 1, or make a guess as to where the error is. To step one line at a time, press the **F10** key; and after repeated presses of F10, you will see something like Figure 3. The other way to debug is to set breakpoints at certain lines of code

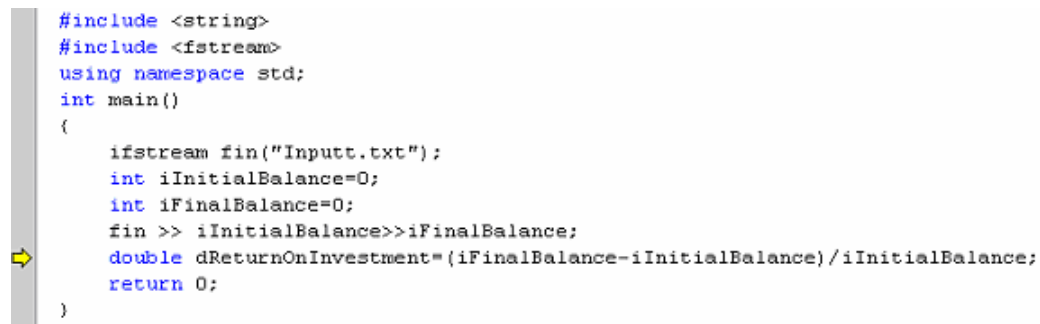


Figure 3 Step Over (F10)

3) There are two ways to determine the value of a variable when debugging. One is by simply placing the mouse cursor over the variable to see its value (Figure 4) or by looking at the Autos window in the lower left corner of the IDE (Figure 5)

```

.= (iFinalBalance-iInitialB
vent = " << iFinalBalance = 10075

```

Figure 4

Name	Value	Type
dReturnOnInvestment	0.0000000	double
iFinalBalance	10075	int
iInitialBalance	10050	int

Figure 5: Autos Window

Looking at the Autos window, one may wonder why return on investment is 0 even though the variable values used in the calculation contains the right values. The debug process has helped to determine that the input values in the formula is correct, but knowledge of C++ is required because at this stage, one should note that the iFinalBalance, and iInitialBalance are integers and integer division will not result in a decimal answer. Changing the variable types to doubles will result in correct values.

- 4) As mentioned above, the second method of debugging is to guess where the error lies and to set breakpoints where you think the error may have occurred. In this case, we know that the logical error is on the line that calculates the return on investment. There are a few ways to set a breakpoint; however, the easiest way is to click on the line in question and press **F9** (see Figure 6)

```

#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream fin("Input.txt");
    int iInitialBalance=0;
    int iFinalBalance=0;
    fin >> iInitialBalance>>iFinalBalance;
    double dReturnOnInvestment=(iFinalBalance-iInitialBalance)/iInitialBalance;
    cout << "Return on Investment = " << dReturnOnInvestment << endl;
    return 0;
}

```

Figure 6: Breakpoint set after pressing F9 on the line in question

- 5) The next step is to run the code by pressing F5 and you will notice that the execution stops on the line that you added the breakpoint. The line containing the breakpoint has not been executed; but using breakpoints allow you to skip over code that you know works and start debugging at the point where you think the error occurs.

```
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream fin("Input.txt");
    int iInitialBalance=0;
    int iFinalBalance=0;
    fin >> iInitialBalance>>iFinalBalance;
    double dReturnOnInvestment=(iFinalBalance-iInitialBalance)/iInitialBalance;
    cout << "Return on Investment = " << dReturnOnInvestment << endl;
    return 0;
}
```

Figure 7 Execution Stopping on a breakpoint.

Using the techniques outlined above, one can easily reach the same conclusion about the integer division problem and having to change the data types of the initial and final balance to doubles.