

BA 272 Testing, Documentation, and Debugging

If you want to provide business value when developing software, you need to understand testing, documentation, and debugging.

Testing: If you don't know how to test your software, you don't really know how to build it right. Current software development methodologies and tools support a "test as you go" paradigm. For example, Microsoft Team Foundation tools allow designers to build automatic testing into large development processes so that a programmer cannot "check in" code unless it passes a series of tests. Team Foundation is not the only such tool, but it is the one used in the College's Business Solutions Group (BSG) processes. We are not going to get into automating the test procedure in this class, but we do want to give you practice in quality assurance through testing. So, all programming assignments include creation of a rudimentary test plan. You are asked to come up with exemplary test data before you write your programs. Specify these things before you start coding. You may need to adjust as you go along, that's ok. So for example.

Based on the simple programming model we have used so far in BA272, what if you wanted to make a program to calculate the area of a rectangle? List a couple of examples of inputs and outputs. A test case might look like this:

```
Prompt:      Enter width:      User types: 2
Prompt:      Enter length:     User types: 2
Output: The area of a 2 X 2 rectangle is 4.
```

Explain why that is not a very good test. If you use such weak test data for your examples, expect to get marked down.

Of course rectangle calculation is VERY simple. What if you were to compute the area and circumference of a circle given its diameter? Different things come into play such as how many places of PI and how accurate an answer is required? By thoughtfully defining your test cases in advance and from a user perspective, you avoid many traps programmers commonly fall into.

The testing we require in BA272 is simplistic. There is much more to consider. Load testing, integration testing, and regression testing are just a few key ideas. But, at least we want you to begin to form good habits early in your programming career.

Documentation: Page 45 in the text describes the basics of adding comments to code and gives some related, good advice. ACTG 378 and the rest of the Business Information Systems classes will deal extensively with documentation and how it contributes to the success of developed systems. Successful BA272 students will practice commenting code and receive feedback. Assignments call for explanatory comments describing input, processing, and output. We will also expect to see comments integrated into your programs. Test plans are also a kind of documentation. Understand the mechanics of adding comments to your programs and think about helpful vs. problematic commenting practices.

Debugging: It is not a bad idea to practice finding helpful programming tips on the internet. Please look at the [MSDN Debugging Basics](http://msdn.microsoft.com/en-us/library/aa290881(VS.71).aspx) article [http://msdn.microsoft.com/en-us/library/aa290881\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/aa290881(VS.71).aspx)). Please note the difference between syntax and logic errors. We will talk about a third type of error (runtime errors) when we look at the `try/catch` and `tryparse` commands. Syntax errors are invalid instructions: they violate the rules of the programming language. Logic errors are acceptable from a language rules standpoint but make the application do the wrong thing. At the end of this document you will find some code you can paste into a test project. Lest walkthrough a few helpful debugging skills. As Mr. Monk would say: - you'll thank me later.

First consider syntax and logic errors Are there any syntax errors in the code? How can you tell? Why is it an error to leave off a semicolon? We may not get that last question answered satisfactorily! It just is.

Comment out the errant `Console.WriteLine(...` line and make a new one. Isn't it cool how typing `Console.` brings up a list of methods and properties for the `Console` object? (You could just add the semicolon at the end...) Like other Integrated Development Environments, the Visual Studio IDE helps out a lot with syntax. This can substantially improve programmer productivity.

Do you remember the implicit narrowing conversion error we talked about previously? Note how Visual Studio highlights the error and blocks compilation of the program. Comment that line out too.

How about the line: `double dDiameter = int.Parse(Console.ReadLine());`

Be sure you can discuss the answers to these questions: Does it generate a syntax error? Does it do the right thing? What kind of error would we get if we put 45.763 in as the diameter? Should the test data catch this? What different business rules that might determine if the code is right or wrong?

Debugging tools in Visual Studio – Here are a few things to try, then explore debugging some on your own.

Add a break by clicking in the gutter next to the line: `int iFeetPerMile = 5280; // Everyone knows that!`

Run the program in debug mode - clicking the green arrow is one way. Note how you can hover over a variable name and see its value. You can also add variables to the watch window - `Debug/Windows/Watch/Watch1`. Add `dAvgSpeedFeetPerHour` and `dDiameter` to the watch list. Step through several lines of code by clicking the Step Into button several times.



As you step through the code you can see the console change and the variable values change. Try putting another break point a few lines of code ahead and pressing the green arrow again. This make the program execute until it reaches the break point.

Note how control passes from the debugger to the console when you get to `Console.ReadLine();`. Press enter to move on to the next line of code. Try entering a value with a decimal in for the diameter of the circle. An exception occurs because changing a decimal value to an integer is a narrowing conversion – the program isn't sure what is right. You can try it again by clicking the Step Into button.

Rob Miles has some good thoughts on Debugging on pages 167 through 170. Still, the simple things we just tried: setting breaks, hovering over variables, and setting watches can help you immensely as you try to get you program working. Of course the debugger can do much much more.

```

using System;
namespace Data_Manipulation_Exercise
{
    class ExploreDataManipulation
    {
        static void Main(string[] args)
        {
            /* This program explores some basic programming issues
            * Inputs vary and are mostly hard coded, Processing is simple but varied,
            * Outputs are intended to demonstrate how techniques work
            * THIS IS A COMMENT BLOCK
            */

            //Console.WriteLine("This code explores some class exercises")

            int iFeetPerMile = 5280;    // Everyone knows that!
            double dAvgSpeedFeetPerHour = 2427;    // Computed avg for the job
            double dHoursSpent = 22.6;    // Billable time
            int iRatePerMileInCents = 5435;    // Negotiated rate or $54.35 / mile

            double dFeetDone = dAvgSpeedFeetPerHour * dHoursSpent;
            int iMilesDone = (int)(dFeetDone / iFeetPerMile); // charge only for complete miles
            int iAmountPaidInCents = iMilesDone * iRatePerMileInCents;
            double dAmountPaidInDollars = (double)iAmountPaidInCents / 100;
            Console.WriteLine("Charge: " + dAmountPaidInDollars);

            double avgSpeedFeetPerHour = 2427;    // Computed avg for the job
            double hoursSpent = 22.6;    // Billable time
            int ratePerMileInCents = 5435;    // Negotiated rate or $54.35 / mile
            Console.WriteLine("Charge: " + (double)((double)(
                (int)((avgSpeedFeetPerHour * hoursSpent) / 5280)
            ) * ratePerMileInCents) / 100));

            int i = (int)((25f / 2f) + (25 / 2));
            Console.WriteLine("Line A: int i = (int)((25f / 2f) + (25 / 2)); i = " + i + " Rounding Error");
            i = (25 + 25) / 2;
            Console.WriteLine("Line B: int i = (25 + 25) / 2; i = " + i + " Right!");
            i = (25f / 2f) + (25f / 2f);
            Console.WriteLine("Line C: int i = (25f / 2f) + (25f / 2f); i = Floats don't fit in ints ");
            i = (int)((25f / 2f) + (25f / 2f));
            Console.WriteLine("Line D: int i = (int)((25f / 2f) + (25f / 2f)); i = " + i + " Right!");
            i = (25 / 2) + (25 / 2);
            Console.WriteLine("Line E: int i = (25 / 2) + (25 / 2); i = " + i + " Rounding Error");
            i = (int)(25f / 2f + 25f / 2f);
            Console.WriteLine("Line F: int i = (int)(25f / 2f + 25f / 2f); i = " + i + " Right!");
            double d = (25 / 2) + (25 / 2);
            Console.WriteLine("Line G: double d = (25 / 2) + (25 / 2); d = " + d + " Rounding Error");
            Console.WriteLine();

            double dbl = 25f;
            Console.WriteLine("Explore float values: dbl = 25f;  dbl = " + dbl);
            dbl = 86.7;
            Console.WriteLine("Explore float values: dbl = 86.7;  dbl = " + dbl);
            dbl = 86.7f;
            Console.WriteLine("Explore float values: dbl = 86.7f;  dbl = " + dbl);

            Console.WriteLine();
            dbl = 867 / 8670;
            Console.WriteLine("Explore float values: dbl = 867 / 8670;  dbl = " + dbl);
            Console.WriteLine("Note that it treated everything as an intger so it truncated ");
            Console.WriteLine();

            Console.WriteLine("So you fix it by adding f to the values in the computation");
            dbl = 867f / 8670f;
            Console.WriteLine("Explore float values: dbl = 867f / 8670f;  dbl = " + dbl);
            Console.WriteLine("Fine! so now you decide to put fs after all values ");

            Console.WriteLine();
            dbl = 86.7f / 8670f;
            Console.WriteLine("Explore float values: dbl = 86.7f / 8670f;  dbl = " + dbl);
            Console.WriteLine("Not waht you expected.... ");

            Console.WriteLine();
            dbl = 86.7 / 8670;
            Console.WriteLine("You were thinking: dbl = 86.7 / 8670;  dbl = " + dbl);
        }
    }
}

```

```

Console.WriteLine("Hmmm... so Careful!... ");

Console.WriteLine("");
Console.WriteLine("Press enter to clear the screen an continue");
Console.ReadLine();
Console.Clear();

Console.WriteLine("Enter diameter of a circle");
double dDiameter = int.Parse(Console.ReadLine());

/*
 * Diameter =      5
 *           Pi to 15 Digits      Pi to 2 Digits
 *           3.1415926535897900    3.14
 * Circumference 15.70796327      15.7
 * Area          19.63495408      19.625
 */

double PI2 = Math.Round(Math.PI,2);
double PI15 = Math.Round(Math.PI, 15);

double dCirc2 = 2 * PI2 * (dDiameter / 2);
double dArea2 = PI2 * Math.Pow(dDiameter / 2, 2);
Console.WriteLine("Using PI2, Circumference=" + dCirc2);
Console.WriteLine("Using PI2, Area=" + dArea2);

double dCirc15 = 2 * PI15 * (dDiameter / 2);
double dArea15 = PI15 * Math.Pow(dDiameter / 2, 2);
Console.WriteLine("Using PI15, Circumference=" + dCirc15);
Console.WriteLine("Using PI15, Area=" + dArea15);

Console.WriteLine("Notice that the 15 digit results don't match up with the test data");
Console.WriteLine("Which answers were right? Wrong? How do you decide in a business app?");

Console.ReadLine();
}
}
}

```