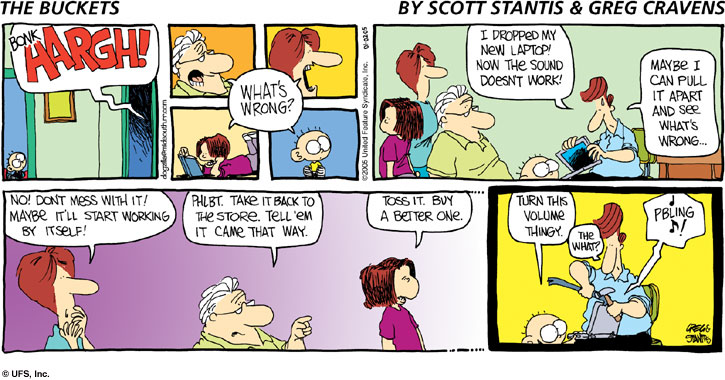
# CMS 3145 Week Three Class Notes: Chapter 6 **Test** and **Debug**



**OBJECTIVES:**

1. Understand ***Testing***
2. **Practice**, **Practice**, **Practice** *Debugging* *Skills*

**Review**

There are 3 classifications of errors

1. **Syntax**: the language of the code is wrong. Typically caused by a typing error.
2. **Run Time** error / exception. Typically caused by bad input.
3. **Logic**. The program does not do what it is supposed to do. Caused by the development team.

**Testing a program**

Finding **syntax** errors is easy because the complier will find them for us. A good IDE will also help by showing the errors in the code. Sometimes the error message will not be helpful. An error in one line of code might be caused by a syntax error in a different line of code. Keeping a log of the errors you make can help you learn what different error messages mean. For example, “**symbol not found**” often means that a variable is not spelled correctly, including being in the wrong case. The program will NOT compile when there are syntax errors. When asking about a syntax error include the EXACT error message that is created by the error.

Another common error for novice programmers is not having **balanced** parentheses or brackets. Remember that these symbols follow the buddy system: every **opening** bracket must have a corresponding **ending** bracket. As soon as you type the **opening** bracket immediately make sure there is an **ending** bracket if the IDE does not automatically type it for you. Then type the new code inside of the block of code defined by the brackets.

Be careful when code is added or deleted. Changing the balance of brackets can result in a difficult to find error. Remember that you can use **Control-Z** to undo typing and bring the code back to a state where there are no syntax errors. It is helpful to fix syntax errors as soon as they occur. Don’t let them accumulate.

Finding **runtime** errors is a little harder because they might not occur every time the program is run. Often bad input is the cause of a runtime error and only sometimes is the input ‘bad’. Whenever there **might** be a runtime error, a Try…Catch block of code should be used to prevent the program from crashing! Runtime errors often occur with problems with **input** into the program which is why the I/O operations we cover later in the semester require Try…Catch code.

Finding a **logic** error can ONLY be done with testing. We can define what the program must do before the code is written. Given a set of **valid inputs** we should get a **valid output**. This is called a **test** **script**.

A program must check for valid input, which is called “**Data Validation**”. Thus we should also test the program with **invalid** inputs to make sure that the program gracefully handles bad input. It is important to give useful and specific **feedback** to the user about why the inputs are invalid.

**Debugging a program**

If we test a program and there are no **logic** **errors** we will release the program for distribution. A program always needs **debugging** when there are logic errors that are hiding in the program. Debugging is also useful for understanding **runtime errors** because we can look at the exact line of code that causes the error as the program is running

Debugging is a useful way to find logic errors. By putting breakpoints in the code and using the “**Debug Project**” run option (Ctrl-F5) we can watch the code run and examine the values of variables as the program is running. Using the **Step Into** button (F8) will run one line of code at a time starting from where the breakpoint is set. The **Variable** Window will show the variable and there values.

When a line of code is highlighted, in yellow, it means that the complier is about to run that line of code. Read the line of code carefully. Check the values in the Variable window to make sure they are the values you expect. Predict what will happen when the line of code is executed. Then run that one line of code. Check that the changes match your expectation. Do this for each line of code until you find the one line of code that is causing the logic error. Does the program get the input the way you expect it to? Does the program calculate values the way you expect? Does it generate output they way you expect?

This can be a tedious process. When you ask for help report what you have seen in the **debugging** process. Report the values of variables when at break points with the program running.