Testing

# Poor Testing Practices (Unfortunately, these are the most common practices used by students)

* Running lots of **random testing**. Using **serendipity** isn't an engineering practice and will fail you as the problems become larger and more complex. You must engineer how you will design, implement and test your programs to have long-term success.
* Testing without knowing what the result of the test should be before you execute the test (this directly relates to why many students want to **use a debugger to track down errors** in their code; **they don't actually know what should happen at each step** of the program so they use the debugger to see what is happening).
* Believing it will work because you wrote it and it compiled.
* Writing large sections of code then trying to test it as one large section.

# Better Testing Practices

* Tests that focus on small specific areas where errors are most likely to occur such as **boundary conditions**. Think about 0 occurrences of something, 1 occurrence of something, 2 occurrences of something, the maximum elements of something occurring, and +1 the maximum number of elements occurring.
* **Develop a model** of the system you need to test so accurate you can use it to predict what will happen under every circumstance. Next use the model to develop tests where each output of the system compared to the predicated output of your model. **Verify** that the model and the system produce the same results. When they don't figure out which is wrong the actual system or your model of the system. This is the ***Scientific Method*** and how to use it in testing. Without this you are just running random test and that is "crossing your fingers and hoping for the best", which is neither science nor engineering**. If you don't have a model of how our assembler, virtual machine and compiler will work how can you ever expect to develop them?**
* Use **incremental, design, development and testing**. Create small sections of code either top-down or bottom-up and test them before moving on. Just because you've been **successful developing toys systems with poor development/debugging/testing practices doesn't mean you can continue to do so**.
* Develop **regression tests** in your code that can be run and re-used at any point in the development process. Strongly consider automating your regression tests. Piping output from standard out into a file then comparing the output to a regression test file (e.g., use a diff command) can tell you if you are getting the same or different results for previous tests.
* Need more guidance, look into **Test Driven Development**. The basic idea is develop a test before you finish your code. Ensure the test fails when your code is not complete. Keep coding until the test completes successfully. Make simple unit tests and more complex integration tests. Keep these tests around to perform **regression** testing.
* While sharing code is **cheating**! Share the results of a test run is a wonderful use of your time. Sit down with **two other members** of the class you don't normally talk with and compare the output of your programs. **The probability of three people having the exact same mistake unless they are working together is very small**. If your outputs differ figure out who is correct. If there is any doubt, **ask me** don't just take peoples advise because they think they're correct. I've never meet a student who has done this and gotten a problem wrong.

Debugging

# Poor Debugging Practices

* **Single stepping** through your code in the debugger.
* Setting **break-points** then single stepping through your code in the debugger.
* Virtually anything you do that begins or ends with you using your **debugger** to find the error. **Not developing your own tracing, logging and testing code**.
* Why is using a debugger such a poor practice? Very simple. People start **relying on the debugger to find the error rather than using their skill** to find the bugs. Debuggers are useful tools but they don't replace skill and knowledge. If you are using a debugger more than 10% of the time you're wasting time and most likely don't really understand how the program should work.
* Thinking that **taking shortcuts** will get you done faster.

# Better Debugging Practices

* **Isolation** is the key to debugging.
* Use **simple print statements** to isolate which part of you program is having the problem
* **Systematically comment out code** until you find the smallest collection code that can replicate the problem.
* **Systematically delete data** until you find the smallest collection of data that can replicate the problem.
* The asm/vm project to some degree and the compiler are such that a developer can spend **100's of hours looking for a bug** and never find it unless they use isolation first. Many of the harder bugs to find in your compiler will take you **3-10 hours to find even using good isolation** **practices**.
* On multiple occupations I've had students (and even experienced developers) come to my office after spending 20, 40 even 70+ hours looking for a bug only to have me help them find the bug in under 15 minutes. **In every case I've done this using a combination of print statements, commenting out sections of code and reducing the data set size**. Some problems are actually really hard to find, most are not!