Unit Testing

Managing implementation complexity

- How many parts does the Space Shuttle have?
  - Estimates range from 2.5 – 6 million
- What would happen if NASA assembled a space shuttle with "untested" parts (i.e., parts that were built but never verified)?
- It wouldn't work, and in all likelihood you would never be able to make it work
  - Cheaper and easier to just start over

Managing implementation complexity

- Individual parts should be verified before being integrated with other parts
- Subassemblies should also be verified before being combined with others
- If adding a new part breaks the system, the problem must be related to the recently added part
- Track down the problem and fix it
- This ultimately leads to a complete system that works

2 approaches to programming

- Approach #1
  - "I wrote ALL of the code, but when I tried to compile, nothing seemed to work!"
- Approach #2
  - Write a little code (e.g., a method or small class)
  - Test it
  - Write a little more code
  - Test it
  - Integrate the two verified pieces of code
  - Test it
  - ...

Unit testing

- Large programs consist of many smaller pieces
  - Classes, functions, packages, modules, etc.
- "Unit" is a generic term for these smaller pieces
- Two important types of software testing are:
  - Unit Testing
  - System Testing
- Unit Testing is done to test the smaller pieces in isolation before they are combined with other pieces
  - Usually done by the developers who write the code
- System Testing is done to test the entire system after all of the pieces have been integrated together
  - Usually done by a separate software testing team

What unit tests do

- Unit tests create objects, call methods, and verify that the returned results are correct
- Actual results vs. Expected results
- Unit tests should be automated so that they can be run frequently (many times a day) to ensure that changes, additions, bug fixes, etc. have not broken the code
- Notifies you when changes have introduced bugs, and helps to avoid destabilizing the system

Test driver program

- The tests are run by a "test driver", which is a program that just runs all of the unit test cases
- It must be really easy to add new tests to the test driver
- At the end the program, the test driver either tells you that everything worked, or gives you a list of tests that failed
- Make files should be used to automate the building and running of the test driver (e.g., `$ make test`)
- Little or no manual labor required to run tests and check the results

Unit test demo

- Classes have Test methods
- Unit testing framework (UnitTest.h, TEST macro)
- Test driver calls test methods
- Run the tests
- Modify some tests to fail
- Rerun the tests to show the failure messages

Selecting test cases

- How do you design test cases for a method?
- Ad Hoc testing – try a variety of valid and invalid inputs, whatever comes to mind
- Exhaustive testing – create a test for each distinct possible path through the method
  - Trace the line numbers that are executed when the method executes (1, 2, 10, 11, 12, 10, 11, 12, 13, 17)
  - Each possible trace is a different "path"
  - Often the number of possible paths is so large that exhaustive testing is infeasible

Selecting test cases

- We can't be exhaustive, so we instead use code coverage criteria to guide our selection of test cases
  - Line coverage – test each line at least once
    - This is the minimal level of testing, and is usually not sufficient
  - Branch coverage – test the TRUE and FALSE cases of each branch at least once
  - Condition coverage – test ALL possible ways that each condition can be TRUE or FALSE
    - Find all primitive boolean subexpressions, create a truth table, design a test case to cover each row in the truth table

```
if ( a > b && c == 5 ) {
    …
}
```

a > b	c == 5
F	F
T	F
F	T
T	T

Code coverage example

```
if ( a > b && c == 5 ) {
   …
}
d = 5;
```

Possible test cases

(1) a > b and c == 5

(2) a > b and c != 5

(3) a <= b and c == 5

(4) a <= b and c != 5

- Which test cases are required for:
  - Line coverage?
    - (1) only
  - Branch coverage?
    - (1) and any of the other three
  - Condition coverage?
    - (1), (2), (3), (4)

What about missing code?

- Code coverage criteria helps to ensure that existing code is tested thoroughly
- What if there is no code at all to handle certain kinds of input?
  - Programmer forgot to handle some cases
- Code coverage doesn't find these errors
- Ad Hoc testing can be used to detect unhandled cases