CS 162
Intro to Programming II
Testing and Debugging
Definition

From the dictionary-
the means by which the presence, quality, or genuineness of anything is determined; a means of trial.

For software (from Wikipedia)-

Software testing can be stated as the process of validating and verifying that a computer program/application/product:

meets the requirements that guided its design and development,
works as expected *,
can be implemented with the same characteristics,
and satisfies the needs of stakeholders.

* This is the only item we are concerned with. The others are important as software engineering requirements.

OSU Oregon State University
Testing Basics

• The key is to localize the problem
• How to do this?
  – Debug compilation units as they’re written
  – Start testing with small units of code (unit tests)
  – Integrate these pieces and test
  – Build to the full system and continue to test
• A test plan formalizes this process
Works as Expected

• How do we establish this?
  – Unit Testing *
  – Integration Testing
  – Regression Testing

* We are mainly interested in this testing for this course.
Integration Testing

- Individual software modules are combined and tested as a group
- Keep integrating the smaller pieces into larger pieces and test them
- This process may continue though many layers as software systems may include large modules with communications links or even large distributed systems
Regression Testing

• You fix a bug
• Make a new test to verify that you fixed that bug
• Do you throw your new test away?
  • No! Keep it and all subsequent versions
• This collection of test cases (including data) is called a test suite
• Very often, a bug that you fixed will appear in a future version (known as cycling)
• Use the saved test suites to ensure that known failures in prior versions do not reappear
Unit Testing

• Unit testing validates that a module or class is working correctly

• You write test cases for all functions (or methods)

• Done by the software developers- the author and possibly others.
Unit Testing

• Why don’t we wait and test the complete program?
  – If there is an error, how do we know where it occurred?
  – Many of the exceptions in the code can’t be intentionally triggered from the user interface

• Ensure each class or module (unit) works correctly first

• Then we test how they work together (integration testing)
Test Cases

• The set of data and procedures used to test a class or module
  – Procedures can be:
    • Instructions for manual entry of data or selection of options in the code
    • Driver functions, test harnesses (many names) that call the functions in the class to run the code
    • Expected outcomes for each (manual or automated) test
Test Cases

• Developing test cases (or at least thinking about them) can (should?) be part of the software design and implementation process
  – You may realize that you need more exceptions
  – You may realize that you have too many exceptions
  – It will help you focus on the logic and flow of control in the source code
Test Cases

• Developing test cases can even help in the design phase, before you code!

• Consider an algorithm to convert a raw score to a letter grade (using no plus or minus grades)
  
  If score > 90 then grade = ‘A’
  Else if score > 80 then grade = ‘B’
  Else if score > 70 then grade = ‘C’
  Else if score > 60 then grade = ‘D’
  Else grade = ‘F’

  So 90 is a B, 80 is a C, and so on?
  Are scores limited to the range 0 to 100?
Choosing Good Test Cases

• Positive test cases
  – Legitimate values you expect your code to use, such as 79 or 55 in the grade algorithm

• Boundary test cases
  – Legitimate values at the edge of ranges in your code, such as a score of 90 in the grade algorithm

• Negative test cases
  – Values you expect your code to reject, such as -13 or 234 in the grade algorithm
Test Coverage

• All logic paths and test cases should be included in the test

• A logic path is a flow of control through the program, i.e. each sequence of decisions (if, switch/case, or loop control statements) in the program

• For the grade algorithm you would want to test (at a minimum) scores for too high, too low, A, B, C, D, and F
Test Plan

• Documents how the program will be tested
• It should specify the test of test case, the data values or files required, the expected output, and any driver functions (test harness) to use
• Often is tabular:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Input Values</th>
<th>Driver Functions</th>
<th>Expected Outcomes</th>
<th>Observed Outcomes</th>
</tr>
</thead>
</table>

• Each entry is as detailed as necessary- for complex systems one case could take an entire page
• Test plans vary greatly and are unique to each program