Announcements

- Program #2
  - Due TODAY at 8pm
- Reading
  - Notes (Today)
  - Notes (Thursday)
- Dr. Hollingsworth's Thursday Office hours
  - will not be held this week

Testing

- Why Test Software
  - Find bugs in the implementation
  - Find bugs in the specification
- Testing is a major part of Software Development
  - Around 50% of time in many systems are spent on testing
  - For life critical software, can be 3-5 times development
- Testing Process
  - Execute a program with the intent of finding an *error*.
  - Run special code (or inputs) called a test case.
  - Goal is that each new test should increase likelihood of finding a yet undiscovered bug.

What do we try to test?

- Normal Cases
  - Does the program work on good inputs/parameters
    - Does it not crash?
    - Does it perform as expected?
  - Generally the easiest type of test to write
- Error Cases
  - Does the program work on bad inputs/parameters?
    - Debugged software should never crash.
    - What happens **after** an error (can we continue)?
  - Often most bugs lie in this part of the code
    - Many infrequently executed cases
- Environmental Errors
  - Often very hard to test
  - Examples: Computer out of memory, disk drive removed

Tests are an integral part of the software

- Write Tests as you write software
  - Don't wait until code is done to write tests
  - Sometimes write test code **before** software
  - Test each function/method as you write it
- Tests code can be big
  - Sample Project:
    - Dyninst 133,000 lines, 16,000 lines of testing code
- When the software is "done", tests continue as part of it
  - Need to be able to re-test when code changes
  - Need to validate code on new platforms
    - Sometimes even individual systems

White Box Testing

- Examine Statements of a Program
  - Look for places errors can occur
  - Writes tests cases designed to have program run all possible combinations
- Try to get maximum **coverage** of a test and test suite
  - Coverage can be expressed in:
    - Lines of a program (statement coverage)
    - Control points in a program (branch coverage)
    - Sequences of statements (path coverage)
  - Try to write minimum set of tests to achieve
    - Acceptable coverage level
    - 100% coverage rarely possible

Examples of Statements and How to Test

- If then else
  - Are all cases run?
- Loops
  - Inputs that run the loop:
    - Not at all
    - Only one time
    - Two times
    - A large number of times
    - Test all possible termination conditions
- Assignment statements
  - With range of possible values
    - For ints: positive, negative and 0

Black box testing

- Test Program based on specifications
  - Does it perform as expected
  - Don't examine how it does the job, just does it do it?
- Try to look for:
  - Incorrect or missing functions
  - Interface errors
  - Errors in data structures or external database access
    - After the calls is the file/db correct?
  - Performance errors
    - Timing specs can be a critical as correctness specs
  - Initialization and termination errors
    - Is all memory initialized before its value is tested?
    - Is all allocated memory freed?

Notes on Testing

- Often Tests may be hybrid of white/black box
  - Example: Test cases for project #1B
    - Really designed as black box tests for students
    - After sample solution, initial test cases
      - Look at sample solution for un-tested items
      - Added additional test cases to increase coverage
- Can spend near infinite amounts of effort on testing
  - Concentrate on tests that demonstrate a wide variety in categories of problems
- Approach with the mindset:
  - What ever can go wrong will go wrong

Tools To Aid in Testing

- Test Harness Tools
  - Environment for running functions or methods
  - Allows small bits of code to be run without entire system
  - Example: Junit
  - Can sometimes even be hardware built for this
    - Example: simulator for testing cell phone hardware
- Code Coverage Tools
  - Instrument program to measure what runs
  - Run program on a set of tests
  - Report on what was run and what was not run:
    - Can be at the statement, line, branch, or path level
- C-language code feature
  - assert(ptr != NULL)
  - if the asserted condition is false, then the program will give an error message and the program will terminate

Using lcov

- One example of a Statement Coverage tool
- Steps:
  - Compile program with flags -fprofile-arcs -ftest-coverage
  - Run programs on test cases
  - lcov -c –d . -o coverage.out
  - genhtml coverage.out
  - Creates index.html file in current directory

Demo of lcov viewer

- Coverage of Tests on project #1B
- Drilling down from summary to individual files
- Looking at a given statement
  - Notice coverage counts for individual statements

Process of Writing Test Cases

- Figure out what to test
- Write code to setup conditions for test
  - For example, call init API functions as needed
- Invoke routines to be tested
- Write Code to verify desired property
  - Example: Does the table have the correct items?

- Tricky Issues:
  - How to keep the tests small
  - How to test what you want to and not other things
    - Does an insert test depend on lookup working?

Example: Public Test #0 from P2

```
int main()
{
    int ret;
    char *ptr;
    int errorCount=0;
    assocTable *table;

    table = createTable(4);
    if (!table) {
       printf("ERROR: unable to create a table\n");
       exit(-1);
    }

    ptr = lookupTable(table, "foo") ;
    if (ptr) {
        printf("ERROR: found non-existent element foo in table\n");
       errorCount++;
    }

    ret = addTableItem(table, "foo", "bar");
    if (ret != 0) {
       printf("ERROR: AddTableItem returned an error\n");
       errorCount++;
    }

    ptr = lookupTable(table, "foo") ;
    if (!ptr || strcmp(ptr, "bar")) {
        printf("ERROR: unable to find element bar in table\n");
       errorCount++;
    }

    if (errorCount) {
        printf("Test simpleOps failed: %d sub-tests failed\n", errorCount);
        exit(-1);
    } else {
        printf("Test simpleOps passed\n");
        exit(0);
    }
}
```

Building a Test Suite

- API:
  - Int createEmployee(char *lastName, char *firstName);
    - Return unique id for that employee
  - Int lookupEmloyee(char *lastName, char *firstName, int *id);
    - Return 0 on success, -1 on failure (not found or > 1 match)
    - Fills out id on success
    - fristName can be null
  - Int deleteEmployee(int id);
  - Int setSalary(int id, float salary);

Possible Test Cases

- Int createEmployee(char *lastName, char *firstName);
  - Pass valid data
  - Pass same last name, different first name
  - Pass same last name, first name same as previous
  - Pass null for lastName and/or for firstName
  - Pass an lastName/firstName that has 10MB before null termination.
  - Create 10,000 employees

Test Cases Continued

- Int lookupEmloyee(char *lastName, char *firstName, in *id);
  - Call before inserting anyone
  - Lookup someone that was created
  - Lookup someone with the same lastname, but different firstName
  - Lookup null last/first names
  - Pass id as null
- Int deleteEmployee(int id);
  - Pass valid id
  - Pass invalid id
  - Pass id of someone who has just been deleted
- Int setSalary(int id, float salary);
  - Invalid id
  - Negative salary