Announcements

- Program #3
  - Available on Web Page
- Exam #1
  - Thursday, March 8, 6:00 – 7:30 in Armory 0126
  - Makeup Exam Friday March 9, 2:00 PM – room TBA
- Reading
  - Notes (Today)

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?
    - Does it 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 allocated memory freed?
    - Is all memory used initialized?

Notes on Testing

- Often Tests may be hybrid of white/black box
  - Example: Test cases for project #3
    - 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 classes 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: hardware simulator for testing cell phone software
- Language Features
  - assert(express);
- Code Coverage Tools
  - Instrument program to measure what runs
  - Run program on a set of tests
  - Report on what was run or not:
    - Can be at the statement, line, branch, or path level

Using lcov

- 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 #2
- Drilling down from summary to individual files
- Looking at a given statement
  - Notice coverage counts for individual statements
- Show before/after for adding test case

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 P1

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>

#include "machine.h"
#include "assembler.h"
#include "disassemble.h"

memoryLocation badMemory[MEM_SIZE];

memoryLocation goodMemory[] = {
    { .insn.opCode = 0, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 1, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 2, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 3, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 4, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 5, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 6, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 7, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 8, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 9, 0x0, 0x0, 0x0, 0x0 },
    { .insn.opCode = 10, 0x0, 0x0, 0x0, 0x0 },
};

int main(int argc, char *argv[])
{
    int i;
    int ret;
    int exitCode = 0;

    ret = 0;

    /* invalid op codes */
    badMemory[ret++].insn.opCode = 11;
    badMemory[ret++].number = 0xdddddddd;
    badMemory[ret++].number = 0xeeeeeeee;
    badMemory[ret++].number = 0xffffffff;

/* load with invalid register #2 */
    badMemory[ret].insn.opCode = 0x1;
    badMemory[ret].insn.r1 = 0x1;
    badMemory[ret].insn.r2 = 0x2;
    badMemory[ret].insn.r3 = 0x2;
    badMemory[ret++].insn.address = 0x2;

    /* iterate through program testing instructions */
    for (i=0; i < ret; i++) {
        if (validInstruction(badMemory[i].insn)) {
            printf("Error: considered memory word of %x as a valid
           instruction\n", badMemory[i].number);
            exitCode=1;
        }
    }

    ret = sizeof(goodMemory)/sizeof(instruction);

    /* now test the valid instructions */

    for (i=0; i < ret; i++) {
        if (!validInstruction(goodMemory[i].insn)) {
            printf("Error: considered memory word of %x as an invalid
           instruction\n", goodMemory[i].number);
            exitCode=1;
        }
    }

    ret = 0;

    /* now test the valid instructions */

    if (exitCode) {
        exit(-1);
    } else {
        printf("Passed all tests\n");
        exit(0);
    }
}
```