Lab #7
Testing
Due In Lab, October 22, 2003

Name: _____

Lab Time: _____

Grade: _____/10

Testing

Today you will be testing a program to expose failures. In Homework 7, you familiarized yourself with the operation of a quick sort program, `quick-okay`. Now that you know the type of input the program takes, and the output it produces, you will be writing test cases for it.

Two executables can be found in `~csci3308/arch/$ARCH/bin`. Their names are `quick-okay` for the program without bugs, and `quick-bugs` for the program with bugs. We are providing the bug-free program as a reference so you can generate the expected output for your test cases. The buggy program is the one you will be testing.

The Testing Directory Structure

Get the file `~csci3308/src/quick-bugs.tar` and unpack it in your source directory. It does not contain the source code for the quick-bugs program, but it does contain a README file and a test case.

In the `quick-bugs` directory you will find a directory named `test`. All of the test cases will be placed in here. Inside of `test` there is a directory named `ts01`. This stands for test set 1. A test set is a group of test cases. There may be several test sets for a particular program. Inside `ts01` there is a directory named `tc01`. This stands for test case 1. Each test case has the same files and so has to be placed in its own directory.

Go into `tc01` and look at the files. There is a `README` file explaining the purpose of the test case. There is an `input` file, and there is an `output.expected` file. Run the `quick-okay` program on the `input` file. Did the test pass or fail? How do you know? (The answer to the second question is not because we told you the program is correct.)

Create four more test cases. Try to write test cases that reveal bugs in the quick-bugs program (i.e., test cases that cause it to produce incorrect output). Create the directories `tc02`, `tc03`, `tc04`, and `tc05` for the new test cases, and create the files `README`, `input`, and `output.expected` for each one. Write your test cases below (all three parts for each test case). You can use the back of this page if you need extra space.

When turning in this lab, be sure to submit an archive that includes your answers to the questions in this lab along with your new test cases.

Running Test Cases from the Build Directory

Now I'm going to discuss some software engineering philosophy behind testing. You should think of test cases as a type of source code. As such, test cases should be placed within a subdirectory of a source directory, as we have done here for this lab. When running a test case, the output and comparisions should be done in a build directory, since the output files are automatically generated.

A test case is like source code because it is generated by a human. A person has to decide what input should be tested for each case. Normally, the expected output is also generated by a person who knows what the program is supposed to do. We are cheating by using a correct version to tell us the expected output, but normally you will not have access to such a program.

The output of a test case is automatically generated from the input and the program. It can be recreated at any time and thus can be safely deleted. You should be very careful when deleting anything in a source directory. Things that can be deleted and easily re-created go in a build directory. Another difference is that a test output can change, but the test case does not. If you fix a bug, a test case may produce different output, but the expected output, the output the program is *supposed* to have, stays the same.

There is one final reason to run tests in the build directory. The build directory is architecture specific. It is not inconceivable that a program would pass a test on one architecture, but fail on another. Sometimes the program requires different code for different architectures. In the C programming language, such code can contain `#ifdef` statements to use different code for different architectures. Since different architectures often require different assumptions about a program's operating environment, there may be a bug in the code that only fails on one architecture but not others.

Hopefully, I have convinced you to leave your test cases in the source directory and run (build) them in the build directory. Think about a test run as a compiler whose output is a single value, pass or fail for that test case.

Go to your architecture-specific build directory and create a subdirectory for quick-bugs. In this directory create the same `test/ts01/tc01` directory structure, and create directories for test cases 2 through 5.

Return to the `build/quick-bugs` directory and type the following commands into a file called run-tests:

```
#!/bin/tcsh -f
set srcdir = $HOME/csci3308/src/quick-bugs
foreach testdir (test/ts01/*)
      quick-bugs < $srcdir/$testdir/input > $testdir/output
      diff $srcdir/$testdir/output.expected $testdir/output
      echo $status
end
```

Make run-tests executable and run it. Record the output that was produced by this script below:

Which test cases passed or failed? How do you know?

Explain below what this script does and how it works

So, now you have some experience writing and running test cases, and determining if they pass or fail. Hopefully, you also see how shell scripts can be used to automate the testing process. Indeed, this lab should have you adequately prepared for the testing notebook!