Software Testing Notebook Worksheet #3
Automating Testing & Fixing Bugs
Due: November 12, 2004

Name: ______________________________

Lab Time: ______________________________

Grade: __________/75

On my honor, as a University of Colorado at Boulder student, I

have neither given nor received unauthorized assistance on this work.

Signature: ______________________________

Automating Testing

This week you will be writing three `tcsh-based` shell scripts to automate testing. You will then use these scripts to help debug the ezpay program until you get it to pass all of its test cases. Place your shell scripts in your architecture-independent `bin` directory, and make sure that this directory is in your path.

Setting Up

This week's focus is not on structural testing, so we will not use a jcoverage-instrumented version of ezpay. To switch back to a version of ezpay that has not been modified by jcoverage, perform the following steps.

1. In your ezpay directory, type `ant clean-all`. This will delete `jcoverage.ser` and the `build`, `dist`, and `report` directories.
2. Rename your `build.xml` file to `build-old.xml`.
3. Copy `ezpay3.tar` from `~csci3308/src` directory into your `src` directory and unpack it. This tar file is designed to update your existing ezpay installation. All it contains is a new `build.xml` file.
4. Type `ant` in your `ezpay` directory to build a non-instrumented version of `ezpay`.

Now we need to modify the `ezpay` shell script that currently sits in your `src/ezpay` directory so that it can be moved to your architecture-independent `bin` directory.

1. Create a copy of your `ezpay` script and name it `ezpay.old`. Use this file as a way to revert your changes just in case you run into problems with the instructions below.

2. In the `ezpay` script, change the following line:

   ```
   set ezpayJars = (ezpay-jcov.jar ezpay.jar)
   ```

   to look like this:

   ```
   set ezpayJars = (ezpay.jar)
   ```

3. Change the following line:

   ```
   set ezJars = ($ezJars dist/lib/$jar)
   ```

   to look like this:

   ```
   set ezJars = ($ezJars $HOME/csci3308/src/ezpay/dist/lib/$jar)
   ```

4. Move this modified script to `$HOME/csci3308/bin`

5. Type rehash and verify that this new ezpay script is now first in your command path. That is, the output of "`which ezpay`" should be:

   ```
   $HOME/csci3308/bin/ezpay
   ```

   .

6. You are now ready to write your new shell scripts.

A Script to Run One Test Case

Create a shell script in `$HOME/csci3308/bin` called `run-test` that takes as arguments a test set and a test case, and an optional verbose flag, `-v`. This script might be invoked like this:

```
run-test ts1 tc05
run-test -v ts2 tc13
```

If a test case passes, the run-test script should exit with a status code of 0, otherwise it should exit with a status code of 1. In verbose mode, the script should print the test set and test case number, and either `test passed`, or `test failed`, like this:

```
ts1 tc05:  test failed
ts2 tc13:  test passed
```

Without the verbose flag, the script should only produce output when a test case fails. The run-test script should produce no other output; in particular, any output created by an individual test case should not be allowed to clutter the display. Be sure to comment your script.

Note: your script should be able to handle different types of test cases including those that involve invoking ezpay with no command line arguments, invoking ezpay with command line arguments, as well as the ability to capture and test output that ezpay sends to standard out or its output file. For instance, if a test case is testing ezpay's verbose option, it must redirect ezpay's standard output to a file and then compare that file with output.expected, rather than comparing output.expected with ezpay's output file.

Handling the `-v` flag

When handling the `-v` flag, do NOT capture the existence of this flag in a shell variable called `verbose`. The reason is that you might write code that looks something like:

```
set verbose = 1
```

The problem with this is that the `verbose` shell variable has a special meaning in `tcsh` and the above line will be interpreted as:

```
set verbose
```

(ignoring the "= 1" part) and the script will start displaying its commands to standard out, i.e., tcsh will turn on its **own** verbose mode.

To avoid this problem, simply use a different name for your `-v` flag. For instance, `vflag`.

A Script to Run An Entire Test Set

Create a second shell script in `$HOME/csci3308/bin` called `run-set` that takes as arguments the test set and an optional verbose flag, `-v`. It may be invoked like this:

```
run-set ts2
run-set -v ts1
```

This script should call `run-test` on every single test case in the specified test set. The verbose flag should be passed to the `run-test` script, if present. At the end of a run, the script should print the total number of test cases executed, the number of test cases that passed (along with a percentage value) and the number of test cases that failed (along with its percentage value). **It should print this line regardless of the presence or absence of the verbose flag.** An example summary line is:

```
Test cases:  20; Passed:  10 (50.00%) Failed:  10 (50.00%)
```

Be sure to comment your script.

Note: Since `tcsh` does not support floating point division, we have supplied a script in <`~csci3308/bin/`> called `percentage`. `percentage` expects two integer arguments. It will take those arguments and divide the first by the second, multiply the result by 100, and produce a formatted percentage string. Thus, the invocation

```
percentage 4 23
```

produces the string "17.39%". The behavior of `percentage` is undefined if anything but integers are passed as its arguments. This script should allow you to produce the summary line for the `run-set` script with ease.

A Script to Run All Test Sets

Create a shell script called `run-all` in `$HOME/csci3308/bin` that calls `run-set` on all of the test sets for the program (just ts01 and ts02 in our case.) This script can take an optional verbose flag, `-v`, which it passes to `run-set`, if present. Be sure to comment your script.

Script Design

You will have to make some choices about how each script works. If you need to change directories, should you do it in `run-test`, or in `run-set` before calling `run-test`? What error checking should you do? Document your design decisions in a README file. **Additionally, be sure to document how to use each script.**

IMPORTANT: These scripts should assume that they are being invoked in the architecture-independent `build` directory that we used in worksheet 1. You should create a `ts02` directory in `$HOME/csci3308/build/ezpay/test` so that you can run test cases from both test sets in the `$HOME/csci3308/build/ezpay` directory. Your test sets still live in your `ezpay src` directory, but they are "executed" in the `build`directory (again, just like we did in worksheet 1).

FURTHERMORE: The `-v` flag for the `run-test`, `run-set`, and `run-all` scripts **has nothing to do** with ezpay's `-v` flag. Your functional and structural test plans determine when ezpay is invoked with its `-v` flag.

Debugging

Now, you need to debug the ezpay program.

Execute `run-all -v` on the ezpay program; save the results of this test run to a file called `pre-debug.txt`. Now, modify the ezpay source code to fix bugs. Note: the source code is contained in `$HOME/csci3308/src/ezpay/src/ezpay`. After each bug fix, recompile the ezpay program (by invoking "ant" in the ezpay `src` directory) and then run the `run-all -v` script in the ezpay `build` directory to see if more test cases pass. You do not need to save the output of these incremental test runs. Keep debugging the program until all of your test cases pass for both test sets. If you can't get 100% of your test cases from ts01 and ts02 to pass, then get as close as you can, and explain why it is impossible to achieve 100%. When you have finished debugging, execute `run-all -v` one last time and save the results to a file called `post-debug.txt`.

What to Hand In

This worksheet is worth 75 points. You should hand in the following items:

- a printout of the README file that describes the scripts and your design decisions. (15 points)
- printouts of the three scripts themselves. (30 points)
- printouts of pre-debug.txt and post-debug.txt. Be sure to label them! (5 points)
- a description of the debugging process, including a list of bugs fixed (e.g. describe the bug, its location in the source code, and how you fixed it). If you did not succeed in getting 100% of your test cases to pass, include an explanation as to why it was impossible to achieve that goal. (25 points)