Software Testing Notebook Worksheet #1
Functional Testing
Due: November 1, 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: ______________________________

The Program Under Test

You will start the testing notebook by getting familiar with the program you will be testing, the Easy Pay System, or ezpay. The tar file `~csci3308/src/ezpay.tar` contains the files for worksheet 1. Unpack this tar file in your `src` directory. In the `ezpay` directory, you should find the following:

spec.txt	ezpay's requirements specification
tests/	A directory containing files that represent two test cases for the ezpay program

The program itself is located at `~csci3308/bin/ezpay`. This shell script is a wrapper for a java program contained in the file: <`~csci3308/lib/ezpay-bugs.jar`>. You do not need to copy these files to your account. If you do, be sure to update the path in the shell script to point to your version of the jar file.

To "sanity test" the program, try typing:

```
~csci3308/bin/ezpay --help
```

The program should print usage information. To make it easier to work with this program, you should add `~csci3308/bin` to your path. Once this is done, type `rehash` and make sure that the program is in your path.

The second thing you should do is move the test case files into a test case structure simliar to the one we used in lab 7. That is, create a directory called `test` within ezpay's `src` directory and within that directory create a directory called `ts01`. Then create directories for the two test cases `tc01` and `tc02` within the `ts01` directory and move the test case files to the appropriate directories. You should rename the files such that the test case directories contain three files, one called `ezpay.in`, one called `output.expected`, and one called `documentation`. You can delete the `tests` directory once you have finished creating the `ts01`, `tc01`, and `tc02` directories.

The third thing you should do is create an architecture-independent `build` directory for ezpay (even though we don't need to actually build the program this week) and create a testing structure within it similar to the one created in step 2 above. (Note: you may need to create a `build` directory within your `~/csci3308` directory. Then create an `ezpay` directory within this newly created `build` directory.) Thus, your architecture-independent `build` directory should contain a `test` directory which contains a `ts01` directory which contains the `tc01` and `tc02` directories.

Now, similar to what we did in lab 7, run the following code within the architecture-independent `build` directory (indeed you should probably put this code within a script, so you can run it multiple times):

```
set srcdir = $HOME/csci3308/src/ezpay
foreach testdir (test/ts01/*)
     cp $srcdir/$testdir/ezpay.in $testdir
     cd $testdir
     ezpay
     diff $srcdir/$testdir/output.expected ezpay.out > /dev/null
     echo $status
     cd ../../..
end
```

Recall that a "0" indicates success while a "1" indicates a failure. If all went well, then one of the sample test cases passed and one failed.

IMPORTANT: This code is only intended to get you started in automating test cases...you will need to modify it, if you want to completely automate every test case that you write for this week of the testing notebook. For instance, this script only invokes ezpay with no command line arguments; thus, it cannot be used to test ezpay's -v and –help command flags.

ALSO IMPORTANT: The focus of your work for this assignment is the test plan and making your test cases match your test plan. The focus is NOT on automating your test cases. As such, it is perfectly acceptable to run your test cases "by hand" this week. In particular, DO NOT WASTE TIME trying to perfect a script to automate your test cases this week.

You should now develop a funcational test plan (described below) that will help you create additional test cases that you can use to test the ezpay program. You are welcome to use the two test cases that we have provided but be careful, you may have to rename these test cases to match your test plan. That is, your test plan is going to develop a set of test cases and it will specify the contents for `tc01`, `tc02`, `tc03`, etc. So if, for whatever reason, your test plan has indicated that our `tc01` needs to be called `tc20`, then be sure to rename it accordingly. Since this is so important, it bears repeating:

Make sure that your test cases match your test plan.

Functional Testing

Functional testing provides a criteria to determine how many test cases to create for a test suite based on the functional specification of a program. A functional test plan is created to help identify test cases. In order to write your test plan, perform the following steps:

1. The first step in functional testing is to analyze the program's specification and identify its functional categories. These categories are typically broad (such as handling a program's command line options, sorting, detecting errors, etc.) but can sometimes be quite specific (such as testing a program's ability to calculate a particular type of value).
2. Once the functional categories have been identified, analyze the specification to determine the program's specification items. A specification item is a specific function that the program must perform. Each specification item can typically be assigned to one of the functional categories.
3. Then, for each specification item, determine its functional equivalence classes. That is, the specification item may need to behave differently given different types of input. In lecture 17, an example of selecting functional equivalence classes for the GreatestCommonDivisor function is provided on page 16. (Note: most items will have only one or two equivalence classes, although more is possible.)
4. Once you have determined the functional equivalence classes for each specification item, determine test inputs for each functional equivalence class. Many equivalence classes will only require one test input (just like the GreatestCommonDivisor example), but some may require additional inputs to test boundary conditions (similar to the car database example in lecture 19).

You are now ready to write your test plan.

The Test Plan

1. The first section of your test plan should contain a brief description of the ezpay program and list the functional categories that you identified.
2. The second section of your test plan should be subdivided by functional categories, where each subsection lists the specification items for a particular functional category. Each specification item should be assigned a number (e.g. S01, S02, S03, ...) and should be described as precisely as possible. Below the description, you should list the item's functional equivalence classes along with the test inputs that you have selected for each class. Since each test input corresponds to a test case be sure to assign each one a testcase number (e.g. tc01, tc02, tc03, ...). These numbers need to increase sequentially across all specification items in this section of your test plan. Note that these numbers are not yet final (see below).

3. In the third section of your test plan, identify redundant test cases. For instance, if tc01 and tc23 use the exact same input to test two different specification items, then you can delete tc23 and use tc01 to test both items. As a result of this analysis, your test case numbers may change (e.g. if you delete tc23, then tc24 becomes tc23, tc25 becomes tc24, etc.). As such, once you have finished identifying redundant test cases, create a final list of test cases that identifies which key specification items (which have also been assigned a number) each test case covers. Your final list should use a format similar to this:

   tc01: S01, S02, S23
   tc02: S04
   tc03: S05
   tc04: S06
   ...

4. In the fourth section of your test plan, show the results of running your test suite on the buggy ezpay program. The results should show which test cases passed, which test cases failed, and end with a summary that lists how many cases passed and failed. Your listing should look like:

   tc01: passed
   tc02: failed
   ...
   tc30: failed
   14 tests passed, 16 tests failed

   Remember, you do not need to produce the above listing via a script. You can run each test case by hand, keep track of the results and then generate the above listing manually.

5. In the fifth section of your test plan, list functions of EZPay that cannot be tested. Untestable items are those in which ezpay's behavior (i.e. expected output) cannot be predicted from the specification.

The Test Cases

For each test case in your functional test plan, create a test case directory in the `$HOME/csci3308/src/ezpay/test/ts01` directory. Each test case should contain an input file (if needed), an output.expected file, and a documentation file. You will need to create a corresponding test case directory in the `ts01` directory of your architecture-independent build directory. When you have finished implementing your test plan, run all of your test cases and use the output to produce the report described in section 4 of the test plan. Then, create a tar file of the `$HOME/csci3308/src/ezpay/test/ts01` directory and upload the tar file via Moodle. Your tar file should be named `lastname-ts01.tar`. In other words, John Smith would name his tar file, `smith-ts01.tar`.

What to Hand In

You should hand in your completed test plan at lecture on November 1st and have uploaded your tar file by that time as well.

Evaluation

Your functional test plan will be graded on your ability to identify the specification items within the ezpay requirements specification and on completing each of the requested sections. In addition, we will check your tar file to make sure that your "implemented" test cases match your test plan. The point break down is as follows:

- Functional test plan complete. (50 pts.)
- Functional test cases written correctly and match functional test plan. (25 pts.)