Test Plan

Akira Aida – 100526064
Kathryn McKay – 100524201
Alexander Wheadon – 100514985

Test Input and Output Organization
Each of our test cases has a unique identifier with the format CCCC### where CCCC contains the function name expressed in 4 characters and #s are digits uniquely identifying each test case for that function. For each test case, there are input files containing the driver commands with each input on a newline that represents the command being entered. These input files correspond with two output files, containing the expected data: One file will store appropriate console prompts and responses to the test input, while the other will contain transactions detailing how those responses should be recorded for the back end. Altogether, each test has three data files: the input, the console output, and a transaction file. These are respectively labelled with file extensions .in, .out, and .trans.

Test Run Plan
The goal of the test run will be to determine which cases are and are not handled properly by the program. This will be represented by an output pass/fail file, showing which tests proceeded as expected and which ones did not. Each test result file will contain the timestamp of its completion, thus allowing the pass/fails to be easily assessed for comparisons to previous and future builds. These tests will be performed and recorded by a shell script, acting as a test harness. In order for our tests to run successfully it needs to be able to (1) identify each test case, (2) load each test case's corresponding input/output files, (3) control the test environment, (4) start up the program and execute the inputs, (5) capture the outputs, (6) compare the yielded against the expected data, and last but certainly not least (7) record the results.

(1) Identify Each Test Case
There are two possibilities for loading test cases. The first option would be to traverse each directory and file of the inputs folder. When the names of those files are known, it is then possible to find the corresponding output files. The second option would be to have the script load up a file for this purpose. When the script starts it could read a file, 'testcases', containing a descriptive name (likely the scenario outlined in our test cases file) and the previously mentioned identifier for each case. This would allow the script to find the disparate files of the test case while pairing it with a meaningful description. Since these details are not seen by the customer nor the back-end client, they do not need resolution at this time and may be deferred to phase #3.

(2) Load Each Test Case's Corresponding Input/Output Files
Each test cases' function name and case number will already be known, as described above. This will allow the script to locate the test's input and output files in the directory structure, which we have already defined. Each test suite is nested inside of a folder labeled with the four-character function name, so the script will need to pull the name from the case number at this time. Once done, the script will be able to load the test's .in, .out, and .trans files as parameters for its various actions.

(3) Control the Test Environment
The front end program stores it's data externally, including the accounts file, the transaction file, and any other files that may become necessary to allow sessions to persist. Before the execution of each test it will be necessary to reset these resources to a controlled state; it may also be desirable to keep a copy of the old files to be restored at the end of the test run. In tandem with the tests we have compiled a sample accounts.txt file that is to be placed where the front end can reach it. In addition, any prior transactions should be cleared. This default state should be reset at the beginning of each test case execution.

(4) Start Up the Program and Execute the Inputs
The front end program will be run once for each test case by the shell script, with the corresponding input file commands fed to the shell script. The mechanism for doing this will depend on the implementation of the program.

(5) Capture the Outputs
Two temporary files will be created for each test case: One to capture the console outputs and one to capture the transactions. To glean the console outputs, the stdout stream for the program may be redirected: creating the first file. The second file, the transaction output, can simply be copied from the existing transaction file before it is reset. After the test run is over, these files should be removed.

(6) Compare the Yielded against the Expected Data
Once the data is captured, it may be "diff"'d to the corresponding output files. If the output of the "diff" for both is nothing then the test case passes, otherwise it fails. With this, it is possible to make assertions for each test case, resulting in a Pass/Fail record. There are no degrees of passing, if even one assertion is off then the whole test is failed.

(7) Record the Results
Using the diff as an assertion, the results of each test case may be paired with the descriptive name loaded earlier. The results of each test case may be output to console, to a file, or both. If recording the results to the file is automated, we will label it with the timestamp stored in a directory so that they can be referenced if needed in the future.

Directory structure: