**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.

- The script will pipe the console outputs into a temporary file for comparison after completion.

- The script will save the transactions file.

- After completion, the console outputs file and the transaction file will be “diff”'d to the .out and .trans file that correspond with the .in file.

- If it returns a difference then the test case fails, otherwise it passes.

- If there's an error then the test case fails immediately.

- The output of the test case after completion will be sent to the console as well as appended to a file.

- The file will store all the pass/fails for each test case for that build and will be created at the start of a build.

- The temporary files will then be deleted so that they do not affect any future test cases.

This will be constructed as the script outputs to the terminal the pass/fails.

- e.g. 10Feb0237.pass representing the 10th of February at 2:37am.

- All of the testing files will be in a directory for testing (which will be on the same level as the project directory).

- The testing directory will contain; the script, the temporary files that are created for each test case (that will be deleted after each test case), the input directory and the output directory.

- The pass/fail files will be stored in a directory so that they can be referenced if needed in the future.

**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 be named with 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**

When the script starts it will read a file, ‘testcases’, which will contain a descriptive name and the previously mentioned identifier for each case. This will allow the script to find the disparate files of the test case while pairing it with a meaningful description.

1. **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.

1. **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.

Two 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 transaction output can simply be copied from the transaction file already utilized by the front end. 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.

The directory structure will be as follows...

Directory structure:

├── project/

│ └── src/

│ └── resources/

└── tests/

├── inputs/

│ ├── chng/

│ ├── crte/

│ ├── delt/

│ ├── depo/

│ ├── dsbl/

│ ├── enab/

│ ├── logn/

│ ├── logt/

│ ├── payb/

│ ├── tran/

│ └── with/

├── outputs/

│ ├── chng/

│ ├── crte/

│ ├── delt/

│ ├── depo/

│ ├── dsbl/

│ ├── enab/

│ ├── logn/

│ ├── logt/

│ ├── payb/

│ ├── tran/

│ └── with/

└── passes\_fails/

With the contents of each directory being...

Contents of each directory:

- wrapper/

- project/

- .exe file

- src/

- source files

- resources/

- transactions.txt

- accounts.txt

- tests/

- script file

- two temporary file during a test case that captures outputs

- inputs/

- subdirectories for functionality points/

- input files .in

- outputs/

- subdirectories for functionality points/

- expected output files .out

- expected transaction files .trans

- passes\_fails/

- pass/fail files .pass/.fail