CSCI 3060U - Winter 2025

Phase #3 - Front End Requirements Testing

Due Monday, March 7, 2025 (11:59pm)

In this assignment, you will test the prototype Front End you built in Phase #2 using the acceptance tests you created in Phase #1. Remember to update your tests based on the marking feedback.

You should proceed as follows:

(1) If you have not already done so, organize your test inputs into an input directory (or set of input directories) containing the individual input transaction files for each test run.

(2) If you have not already done so, organize the actual expected output files corresponding to your input files into an expected directory (possibly the same directory as the inputs, with a different naming convention) where the names of the expected output files are the same as the names of the corresponding input files in the input directory (or directories).

(3) Create an (empty) output directory (or use the same directory, with yet another naming convention) to store the outputs (both transaction files and terminal outputs) from your test runs.

(4) If you have not already done so, change your program to be a command line program - one that can be run from the Unix shell prompt. Make sure that the input to your program is the standard (terminal) input, that the log output is the standard (terminal) output, and that the name of the input file (current bank accounts) and output file (bank account transaction file) are accepted as command line arguments, so that your program can be run from the command line something like this:

```
bank-atm currentaccounts.txt transout.atf
```

(5) Make an automated script (Unix shell script) to run your program on each of the input files in your input directory, creating corresponding output files in your output directory. Remember to save as output both the terminal log and the daily transaction file from each test. For example, your script file may look something like this:

```
chdir inputs
foreach i (*) echo "running test $i"
    bank-atm currentaccounts.txt ../outputs/$i.atf < $i > ../outputs/$i.out
end
```

(6) Validate the results of your test runs by comparing your actual output daily account transaction files to the ones in your expected output directory. If you made actual expected output files, then make a script to automate checking that the actual outputs are the same as the expected outputs. For example, your script might look like this:

```
chdir inputs
foreach i (*) echo "checking outputs of test $i"
   diff ../outputs/$i.atf ../expected/$i.etf
end
```

Write a script to do the same kind of comparison for your expected terminal output files as well.

(7) You should make a table of each failure observed, noting which test failed, what it was testing and what was wrong about its output.

(8) Fix each of the observed failures, noting in the failure table what actions or changes were made to address it. For example, if the test was wrong, you would write in the table that you removed or fixed the test inputs. If your program was wrong, you would write in the table what the error in your code was and how you changed the program to fix it.

(9) After you've fixed all the observed failures, commit your source code to GitHub, go back to step 5 and run **ALL** your tests over again to see if the problems are fixed (or if you've created any new ones). Keep repeating until all tests work properly.

What to Hand In

In this assignment, you will hand in through Canvas:

(1) The Unix script files you used to automatically run your tests and validate their output.

(2) The detailed failure table for your test runs, with a row for each observed test failure and columns showing the test name, what it was testing, how its output was wrong, what the error in you code was and how you changed the program (or test) to fix it.

(3) The source code of your final Front End program that passes all your tests.

Marking Scheme

Marking is out of ten, according to the following criteria. In each category, marks are assigned between zero and the number of marks shown, to a resolution of 1/2 mark.

Script Files, Macros or Programs used to run your tests

2 marks

Quality
- clear, readable scripts
- understandable variable names
- internal comments or documentation explaining script or test process

1 mark

Automation
- scripting does majority of test work, little or no hand work to be done

Failure Log

5 marks

Organization and Completeness
- failure log presented in clear table form, listing:
- test name or number,
- what was being tested,
- the nature of the failure,
- what the error in the code was, and
- what actions were taken to fix it

New Source Code

New Code Passes All Tests

2 marks

=======

TOTAL 10 marks