CISC / CMPE 327 - Fall 2018

Assignment #3 - Front End Requirements Testing

Due Wednesday, Oct. 31, 10pm

In this assignment, you will test the prototype Front End you built in Assignment 2 using the acceptance tests you created in Assignment 1. 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 transactions for each test run.

2. If you haven't already created actual expected output files corresponding to your input files, do that now. Organize them into an expected directory (or set of expected directories) with the same structure as the input directory (or directories), and 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 set of output directories) with the same structure as the input directory (or directories).

4. If you have not already done so, change your program to be a command line program—one that can be run from the Windows command prompt or the Linux/macOS 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 names of the input current events file and output daily transaction file are given as command line file name arguments, so that your program can be interactively run from the command line, something like this:

```
./frontend activeaccts.txt transsummary.txt
```

5. Make an automated script (Windows .bat file or Linux/macOS 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 event transactions file from each test. For example, on Linux/macOS, your script file might look something like this:

```
#!/bin/bash
cd inputs
for i in *.txt
do
   echo "running test $i"
   ../frontend activeaccts.txt ../outputs/$i.txt  < $i.txt  \
       > ../outputs/$i.log
done
```

6. Validate the results of your test runs by comparing your actual output event transaction files and logs to the ones in your expected output directory. Make a script to automate checking that the actual outputs are the same as the expected outputs. For example, on Linux, your script might look like this:

```
#!/bin/bash
cd inputs
for i in *.txt
do
   echo "checking outputs of test $i"
   diff ../outputs/$i.txt ../expected/$i.txt
   diff ../outputs/$i.log ../expected/$i.log
done
```

7. Compile a report 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, note in the failure report what actions or changes were made to address it. For example, if the test input itself was wrong, you would write in the report that you removed or fixed the test input. If your program was wrong, you would write in the report 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, go back to step 6 and run ALL your tests over again to see if the problems are fixed (or if you've created any new ones). Keep repeating this loop until all your tests work properly.

What to Hand In

In this assignment, you will hand in, as a PDF in onQ:

1. The Windows, Linux, or other kind of script files or macros you used to automatically run your tests and validate their output.

2. The detailed failure report 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 your code was, and how you changed the program (or test input) to fix it. You may wish to make this a spreadsheet.

Also hand in:

3. A **.zip file** containing:

    a. your final Front End source code that passes all of your tests

    b. your test scripts

    c. your test directories

    The easiest way to do this is to make a .zip of your entire project directory.

Note: The redundancy between items 1 and 3 (and possibly 2 and 3) is intentional. The TAs will focus their attention on items 1 and 2, and they shouldn't need to dig through your .zip file to find that information. The .zip file allows us to run your code, and check that items 1 and 2 are consistent with what you actually did.

Marking Scheme (SUBJECT TO CHANGE)

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.

Scripts or Programs to Run Your Tests — 4 marks

Organization
- clear, readable scripts
- understandable variable names
- clear internal comments or documentation explaining the script or testing process

Automation
- script or program does vast majority of test work, minimal hand work to be done

Failure Report — 4 marks

Organization
- failure report 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

Completeness
- all tests have been run, and all failures addressed

New Source Code
- listing of final source code that passes all tests

Overall Presentation & Quality — 2 marks

=======

10 marks