# DETAILED FAILURE REPORT

Team: Zero Defects

| Name | Student Number |
|---|---|
| Muhammad Usman Majeed | 10086980 |
| Jessica Nahulan | 10029341 |
| Johan Cornelissen | 10098176 |

## Contents

## QBasic Application Usage:

The following provides a brief overview of how to use the QBasic python front end application.
The Front End takes two arguments: the name of a Valid Accounts List file, and the name of a Transaction Summary file. Additionally, the Front-End reads input from standard input, which can either be typed in by the user or redirected from a file (when testing).

**For the first case (production mode):**

```
./QBasic.py validaccounts.txt transactionsummary.txt
```

**When testing, use redirection to feed in the "user" input, say testinput.txt:**

```
./QBasic.py validaccounts.txt transactionsummary.txt < testinput.txt
```

**You can also save the terminal output created by the test:**

```
./QBasic.py validaccounts.txt transactionsummary.txt < testinput.txt > testoutput.txt
```

## Testing Scripts:

To automate the requirements testing performed on the QBasic application two shell scripts were developed. These scripts together allow for test **execution and verification** to be performed using automation rather than the tedious task of performing these operations manually.

- **testExecutionScript.sh**
    - Dependent on testsToRun.txt file. This file is used as a comma delimited list of the tests to be run/verified, including the test area (LOGIN, LOGOUT etc.), test name, and transaction file name.
    - This simple shell script iterates through the testsToRun file and executes the QBasic.py application as appropriate for each test.
        - The valid accounts file path used is constructed using the following: "Testing/${testArea}/Input Files/valid_accounts_file_${testName}.txt"
        - The transaction file path used is constructed using the following: "Testing/${testArea}/Output Files/${transactionFileName}.txt"
        - The standard input file path used is constructed using the following: "Testing/${testArea}/Input Files/input_${testName}.txt"
        - The standard output file path used is constructed using the following: "Testing/${testArea}/Output Files/output_${testName}.txt"
    - Sample output from running the script:

```
$ ./testExecutionScript.sh
Executing LOGIN/test_single_login test:
./QBasic.py "Testing/LOGIN/Input Files/valid_accounts_file_test_single_login.txt"
"Testing/LOGIN/Output Files/transaction_file_empty.txt" < "Testing/LOGIN/Input
Files/input_test_single_login.txt" > "Testing/LOGIN/Output Files/output_test_single_login.txt"
LOGIN/test_single_login test output saved to Testing/LOGIN/Output Files/output_test_single_login.txt

Executing LOGIN/test_txn_login test:
./QBasic.py "Testing/LOGIN/Input Files/valid_accounts_file_test_txn_login.txt" "Testing/LOGIN/Output
Files/transaction_file_empty.txt" < "Testing/LOGIN/Input Files/input_test_txn_login.txt" >
"Testing/LOGIN/Output Files/output_test_txn_login.txt"
LOGIN/test_txn_login test output saved to Testing/LOGIN/Output Files/output_test_txn_login.txt
```

- **testOutputVerificationScript.sh**
    - Dependent on testsToRun.txt file. This file is used as a comma delimited list of the tests to be run/verified, including the test area (LOGIN, LOGOUT etc.), test name, and transaction file name.
    - This simple shell script iterates through the testsToRun file and executes the "diff" command to compare the standard output files (log file) and the transaction file created. Each file is compared to the respective file found in the "Expected Output Files" directory to verify the expected results were created during the testing phase.
        - The **test** transaction file path used is constructed using the following: "Testing/${testArea}/Output Files/${transactionFileName}.txt "
        - The **expected** transaction file path used is constructed using the following: "Testing/${testArea}/Expected Output Files/${transactionFileName}.txt"
        - The **test** standard output file path used is constructed using the following: "Testing/${testArea}/Output Files/output_${testName}.txt"
        - The **expected** standard output file path used is constructed using the following: "Testing/${testArea}/Expected Output Files/output_${testName}.txt"
    - Sample output (showing both passing and failing test cases) from running the script:

```
$ ./testOutputVerificationScript.sh
Verifying LOGIN/test_single_login test:
diff "Testing/LOGIN/Output Files/transaction_file_empty.txt" "Testing/LOGIN/Expected Output
Files/transaction_file_empty.txt"

diff "Testing/LOGIN/Output Files/output_test_single_login.txt" "Testing/LOGIN/Expected Output
Files/output_test_single_login.txt"

LOGIN/test_single_login TEST PASSED!

…….

Verifying CREATEACCT/test_acc_seven_digits test:
diff "Testing/CREATEACCT/Output Files/transaction_file_empty.txt" "Testing/CREATEACCT/Expected
Output Files/transaction_file_empty.txt"

diff "Testing/CREATEACCT/Output Files/output_test_acc_seven_digits.txt"
"Testing/CREATEACCT/Expected Output Files/output_test_acc_seven_digits.txt"
11c11,12
< Please enter the name of account owner:
---
> Error Invalid account number
> Sorry an error occurred, please try again:
CREATEACCT/test_acc_seven_digits TEST FAILED!

…….

####################TESTING STATISTICS####################
Tests Passed: 72
Tests Failed: 13
Tests Run: 85
##########################################################
```

## Failure Report:

| Failure # | Test Area | Test Name | Test Intention | Error in output | Error in code | Fix applied | Re-Test details |
|---|---|---|---|---|---|---|---|
| **1** | N/A | N/A | N/A | In the testing list created in Assignment 1, it was assumed that an empty (only EOS line) transaction file would be created when the QBasic application is started. However, in the application design for Assignment 2 it was determined that a transaction file should only be created during a "LOGOUT" transaction. Error seen in verification script output: diff: Testing/DELETEACCT/Output Files/transaction_file_empty.txt: No such file or directory | Test verification script was looking for non-existent transaction file. | testOutputVerificationScript.sh script adjusted to use a new keyword in testsToRun.txt file to indicate when no transaction file should be created. This should also detect the erroneous case where a transaction file is created when none is expected. | Ran testExecutionScript.sh and testOutputVerificationScript.sh to ensure previously failing test cases are now passing. Also, manually created erroneous case of where a transaction file is created when there shouldn't be one to ensure the verification script successfully found the error. |
| **2** | N/A | N/A | N/A | All tests involving testing before a "LOGIN" command is issued experienced erroneous results when test automation was executed. The standard output created by these tests ended in a python trace back as the program was expecting a new transaction code past the end of the test. However, the test input files contained an end of file | QBasic application did not have a method to properly quit the application without performing "LOGOUT" command. | The QBasic application was modified to always accept "q" or "quit" as a valid transaction code. This allows the user to quit the application at any time. Note however a "LOGOUT" command is still required to end the current session and to produce a transaction summary file. | Unit test was performed manually after implementing fix to ensure "q" or "quit" inputs safely quit the application during multiple points in the application sequence. Tests that originally failed were re-run to ensure they were now producing the expected output. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | character which caused the QBasic application to crash. | | | |
| **3** | LOGIN | test_teller_all_txn_login_agent | If in agent mode all transactions are accepted. | Transaction file and standard output files were missing indications that CREATEACCT command were successful. Specifically, the standard output file indicated that an invalid account name was used. After investigation, it was determined that the test input file was missing the account name when calling CREATEACCT transaction. Without the account name, the QBasic application was interpreting the end of file character as the account name, resulting in an error. | No error in code. | Input_test_teller_all_txn_login_agent.txt file was modified to include the account name after the CREATEACCT transaction code. Updated format:<br>CREATEACCT<br>1000328<br>accountOne | Tests were re-run to ensure that the CREATEACCT command produced no errors in the standard output file, and that the appropriate "NEW" line was included in the transaction summary file. |
| **4** | LOGIN, LOGOUT, DELETEACCT, CREATEACCT | test_mode_agent, test_mode_machine, test_after_logout, test_txn_logout | *Multiple | When test cases were created in Assignment 1, it was assumed that multiple LOGIN and LOGOUT commands would be accepted. However, during the creation of the QBasic application in Assignment 2 it was identified that logically it makes the most sense to have only one LOGIN/LOGOUT combination throughout the application flow. This is due to the LOGOUT | No error in code. | Test intentions were changed to reflect the new application workflow (indicating only one LOGIN/LOGOUT combination at a time). The tests are now intended to ensure that extra input commands are not accepted after the application completes (after the first LOGOUT command). | Re-ran test cases to ensure that standard output matched expected output. Ensures that extra input commands are not accepted after the application completes (at the first LOGOUT command). |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | command being used to create the transaction summary file at the end of the session. | | | |
| **5** | DELETEACCT | test_accept_agent, test_txn_on_delacc | *Multiple | Output for both tests included unexpected "Invalid account number" message when performing DELETEACCT transaction. After some investigation, it was identified that the proper account numbers were never included in the valid_accounts_file for these two tests. | No error in code. | Valid accounts file's for both tests were modified to include the appropriate account numbers. | Both tests were re-run using testExecutionScript.sh with the help of testOutputVerificationScript.sh to verify that the expected output files and expected transaction summary files now matched the test output files. |
| **6** | CREATEACCT | test_acc_first_digit_non_zero | Test if the account number starts with non-zero decimal digit | Standard output file created showed that account number starting with a zero was accepted by the CREATEACCT transaction. The expected output was that the application would reject an account number with a leading zero. | QBasic application code does not block CREATEACCT transactions on account numbers starting with a zero. | A check was added to each transaction processing function to ensure that the integer account number is exactly 7 digits long. This ensures that there are no leading zeroes and also ensures the account number is exactly 7 digits long (two separate requirements). The reason this check works is because the cast from string to integer will automatically remove any leading zeroes (this was originally overlooked and therefore never caught). | Performed manual unit test on several transaction types to ensure an account number with a leading zero is denied. Re-ran tests to ensure test failure was no longer observed. To avoid these errors in other areas test_acc_first_digit_non_zero tests were added to deposit, withdraw, and transfer commands as well. These tests were originally overlooked in assignment 1. |
| **7** | CREATEACCT | test_acc_seven_digits | Test if the account is exactly 7 decimal digits | Standard output file created by test_acc_seven_digits test case showed that that CREATEACCT command still accepts account | QBasic application code does not block CREATEACCT transactions on | A check was added to each transaction processing function to ensure that the integer account number is exactly 7 digits long. This ensures that the account number is exactly | Performed manual unit test on several transaction types to ensure an account number with less than or greater than 7 digits |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | numbers even if they are over 7 digits long. | account numbers that are longer than 7 digits long. | 7 digits long and that the account number has no leading zeroes (two separate requirements). The reason this check works is because the cast from string to integer will automatically remove any leading zeroes (this was originally overlooked and therefore never caught). | is denied. Re-ran tests to ensure test failure was no longer observed. To avoid these errors in other areas test_acc_seven_digits tests were added to deposit, withdraw, and transfer commands as well. These tests were originally overlooked in assignment 1. |
| **8** | DEPOSIT | test_valid_amount_leading_zero | Test amount doesn't accept leading zeros | Standard output file created showed that an amount starting with zero was accepted by DEPOSIT transaction. The expected output was that the application would reject an amount starting with zero. | The amount entered was immediately converted from a string to an int before the input leading zero test was being done. The string to int conversion was automatically removing the leading zero. | Before converting the string, amount entered to an int. A check was added to validate the input and deny with error if it starts with a zero. | Performed manual unit test on several DEPOSIT transactions to ensure an amount starting with zero is denied. Re-ran tests to ensure test failure was no longer observed. |
| **9** | WITHDRAW | test_valid_amount_leading_zero | Test amount doesn't accept leading zeros | Standard output file created showed that an amount starting with zero was accepted by WITHDRAW transaction. The expected output was that the application would reject an amount starting with zero. | Same issue stated in **(8)**. The amount entered was immediately converted from a string to an int before the input leading zero test was being done. The string to int conversion was | Before converting the string, amount entered to an int. A check was added to validate the input and deny with error if it starts with a zero. | Performed manual unit test on several WITHDRAW transactions to ensure an amount starting with zero is denied. Re-ran tests to ensure test failure was no longer observed. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | automatically removing the leading zero. | | |
| **10** | TRANSFER | test_valid_amount_leading_zero | Test amount doesn't accept leading zeros | Standard output file created showed that an amount starting with zero was accepted by TRANSFER transaction. The expected output was that the application would reject an amount starting with zero. | Same issue stated in **(8)**. The amount entered was immediately converted from a string to an int before the input leading zero test was being done. The string to int conversion was automatically removing the leading zero. | Before converting the string, amount entered to an int. A check was added to validate the input and deny with error if it starts with a zero. | Performed manual unit test on several TRANSFER transactions to ensure an amount starting with zero is denied. Re-ran tests to ensure test failure was no longer observed. |
| **11** | VALID ACCOUNT LIST FILE | test_acc_first_digit_non_Zero | Test that the account number doesn't start with zero | Standard output file created showed no error and accepted the valid accounts list file with a zero-starting entry. Expected output was an error after the user logged in and entered Machine/Agent. | When reading the valid accounts list file, each line was being converted to an int while handling non-digits errors but there was no check for if the lines started with zero. | When processing each line from valid accounts list file, before converting the string, amount entered to an int. A check was added to validate the input and deny with error if it starts with a zero. | Performed manual unit test on several transactions including various valid account list files with account numbers starting with zero, to ensure an account number starting with zero is denied. Re-ran tests to ensure test failure was no longer observed. |
| **12** | VALID ACCOUNT LIST FILE | test_acc_seven_digits | Test exactly 7 digits + new line for each account | Standard output file created showed no error and accepted the valid accounts list file with entries of length exactly 7. Expected output was an error after the user logged | When reading the valid accounts list file, the length of each line wasn't being checked. | When processing each line from valid accounts list file, a check was added to ensure the line was of length exactly 7. If not print an error. | Performed manual unit test on several transactions including various valid account list files with account numbers were not exactly of length 7, to ensure an account |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | in and entered Machine/Agent. | | | number not of length 7 are denied. Re-ran tests to ensure test failure was no longer observed. |
| **13** | VALID ACCOUNT LIST FILE | Test_acc_non_digits | Test that the account number doesn't have non-digit characters | Standard output file created showed no error and accepted the valid accounts list file with entries of non-digits. Expected output was an error after the user logged in and entered Machine/Agent. | When reading the valid accounts list file, the characters of each line weren't being checked. | When processing each line from valid accounts list file, a check was added to ensure the line was of only digits. If not print an error. | Performed manual unit test on several transactions including various valid account list files with account numbers were not digits, to ensure non-digit account numbers are denied. Re-ran tests to ensure test failure was no longer observed. |

## Appendix

### Test Execution Script (testExecutionScript.sh)

```
#!/bin/bash

# STDOUT Colour Definitions (NC stands for No Colour)
RED='\033[0;31m';
NC='\033[0m';

# Iterate over all the lines in the testsToRun.txt file.
while read p; do

      # Skip blank lines (used purely for readability of testsToRun.txt file)
      if [ -z "$p" ]; then
            continue
      fi

      # Create an array of the comma seperated values found for each test case.
      # Where
      # pathArray[0] = test area name, ex: LOGIN, LOGOUT, etc.
      # pathArray[1] = test case name
      # pathArray[2] = transaction summary file name
      IFS=',' ;
      index=0;
      for path in $p; do
            pathArray[index]=$path;
            #echo $path;
            index=$index+1;
      done

      # Create the path to valid accounts file, transaction summary file, standard
      input file, and the standard output file path
      ValidAccountsFile="Testing/${pathArray[0]}/Input
      Files/valid_accounts_file_${pathArray[1]}.txt"
      TransactionFile="Testing/${pathArray[0]}/Output Files/${pathArray[2]}.txt"
      InputFile="Testing/${pathArray[0]}/Input Files/input_${pathArray[1]}.txt"
      OutputFile="Testing/${pathArray[0]}/Output Files/output_${pathArray[1]}.txt"

      # Display in red text what test is currently being verified.
      echo -e "${RED}Executing ${pathArray[0]}/${pathArray[1]} test:${NC}";
      # Display what execution line is being executed in the rare case where
      manual
      intervention is necessary.
      echo "./QBasic.py \"${ValidAccountsFile}\" \"${TransactionFile}\" <
      \"${InputFile}\" > \"${OutputFile}\"";
      ./QBasic.py ${ValidAccountsFile} ${TransactionFile} < ${InputFile} >
      ${OutputFile};

      # Display where test output can be found for ease of debugging test
      failures.
      echo "${pathArray[0]}/${pathArray[1]} test output saved to ${OutputFile}";
      # Print new line to seperate the test case output and results
      echo "";
done < testsToRun.txt
```

## Test Output Verification Script (testOutputVerificationScript.sh)

```
#!/bin/bash

# STDOUT Colour Definitions (NC stands for No Colour)
RED='\033[0;31m';
BLUE='\033[0;34m';
GREEN='\033[0;32m';
NC='\033[0m';

# Statistic counters for number of passed/failed test cases
passed=0;
failed=0;

# Iterate over all the lines in the testsToRun.txt file.
while read p; do

      # Skip blank lines (used purely for readability of testsToRun.txt file)
      if [ -z "$p" ]; then
      continue
      fi

      # Create an array of the comma seperated values found for each test case.
      # Where
      # pathArray[0] = test area name, ex: LOGIN, LOGOUT, etc.
      # pathArray[1] = test case name
      # pathArray[2] = transaction summary file name
      IFS=',' ;
      index=0;
      for path in $p; do
            pathArray[index]=$path;
            #echo $path;
            index=$index+1;
      done

      # Create the path to actual/expected versions of the transaction summary
      file and
      standard output files.
      ActualTransactionFile="Testing/${pathArray[0]}/Output
      Files/${pathArray[2]}.txt"
      ExpectedTransactionFile="Testing/${pathArray[0]}/Expected Output
      Files/${pathArray[2]}.txt"
      ActualOutputFile="Testing/${pathArray[0]}/Output
      Files/output_${pathArray[1]}.txt"
      ExpectedOutputFile="Testing/${pathArray[0]}/Expected Output
      Files/output_${pathArray[1]}.txt"

      # Display in blue text what test is currently being verified.
      echo -e "${BLUE}Verifying ${pathArray[0]}/${pathArray[1]} test:${NC}";
      # If no transaction file should be created, verify no transaction got
      created.
      # These situations use special keyword NO_TRANSACTION_FILE.
      if [ ${pathArray[2]} = "NO_TRANSACTION_FILE" ]; then
            echo "Verifying no transaction file was created by this test."
            if [ ! -f ${ActualTransactionFile} ]; then
```

```
                echo -e "Verified no transaction file was created.\n";
            else
                OUTPUT1="Error, a transaction file WAS created by the test. See
            file,
                ${ActualTransactionFile}";
                echo -e "${OUTPUT1}\n";
            fi
        else
            # Otherwise if a transaction file IS expected, test if transaction
            file matches
            the expected one.
            if [ ! -f ${ActualTransactionFile} ]; then
                OUTPUT1="${ActualTransactionFile} file not found!"
                echo "${OUTPUT1}";
            else
                echo "diff \"${ActualTransactionFile}\"
            \"${ExpectedTransactionFile}\"";
                OUTPUT1="$(diff ${ActualTransactionFile}
            ${ExpectedTransactionFile})"''
                echo "${OUTPUT1}";
            fi
        fi

        # Test if the stdout file matches the expected one.
        if [ ! -f ${ActualOutputFile} ]; then
            OUTPUT1="\"${ActualOutputFile}\" file not found!"
            echo "${OUTPUT1}";
        else
            echo "diff \"${ActualOutputFile}\" \"${ExpectedOutputFile}\"";
            OUTPUT2="$(diff ${ActualOutputFile} ${ExpectedOutputFile})"''
            echo "${OUTPUT2}";
        fi

        # PRINT PASS OR FAIL for the test cases (using green or red text
        appropriately)
        if [ -z "${OUTPUT1}" ] && [ -z "${OUTPUT2}" ]; then
            echo -e "${GREEN}${pathArray[0]}/${pathArray[1]} TEST PASSED!${NC}";
            passed=$((passed+1));
        else
            echo -e "${RED}${pathArray[0]}/${pathArray[1]} TEST FAILED!${NC}";
            failed=$((failed+1));
        fi

        # Print new line to seperate the test case output and results
        echo "";

done < testsToRun.txt

# Print to the user at the end of the verification phase the statistics of tests
passed versus failed.
echo "#####################TESTING STATISTICS####################";
echo -e "${GREEN}Tests Passed: $passed ${NC}";
echo -e "${RED}Tests Failed: $failed ${NC}";
testsRun=$((passed+failed));
echo -e "${BLUE}Tests Run : $testsRun ${NC}";
echo "############################################################";
```