CSCI Phase 3 Testing Document

CSCI 3060U

Dr. Michael Miljanovic

Levi Willms, Daniel Hinbest, Syed Rizvi, Raje Singh

Run the program

The environment is compiled with makefile. If not using a system that supports makefile please switch to a system that does support makefile, such as a Unix OS or WSL.

To compile code run `make build`. Then to run the code use `./distribution-system.exe <current_user_accounts> <available_games> <games_collection> (optional)<daily_transaction>` to run the executable. Where the first parameter is always the current user accounts file, the second is the available games, the third is the games collection and the final parameter is an optional daily transaction file. Note: The daily transaction file passed to the system on startup **must** be empty! `make clean` will clear out all output files from the system.

e.g. `./distribution-system.exe current_user_accounts.txt available_games.txt games_collection.txt`

To ensure code in the `main` branch of the repository always runs, run `make clean` and `make build` PRIOR to every commit. You can also do `make rebuild` to do both of these actions in one command.

Note: g++ must be installed on the host machine for compilation and running the system.

Run testing suite

Testing is done using the shell scripts run_tests.sh and compare_results.sh. The `run_tests` script generates the folders and files in `testing/tests` and the `compare_results` script actually compares the expected results to the actual ones.

Though the script files can be used directly, it is recommended to use the rules that are implemented in the makefile as follows:

`make test <transaction>`: Tests all cases for a single transaction. `all` can be used to do all test cases at once. `make compare <transaction>`: After running `make test`, use this with the same argument to check the results of the test.

RECOMMENDED: You can do both of these at once by doing `make run_tests <transaction>` to generate the test files and do the comparison.

Use `make clean_tests` to remove all test-generated files, including daily transaction files, if the directory becomes bigger than you want it to.

run_tests.sh

The run_tests.sh script first finds all the subdirectories under the `testing/cases/<category_name>` directory. If multiple category names are provided then it will search for each of those testing categories. After finding each category it will run through all the test cases it found, performing the input from `input.txt`, saving the commands run to `test.txt`, saving the terminal output to `results.txt` and copying over the `output.txt`, `daily_transaction.txt` and newly created daily transaction file in the format `dtf_test_<test_name>_<date/time>` to the `testing/tests/<category>/<case_name>/ directory`.

compare_results.sh

The compare_results.sh script begins by locating all subdirectories under the `testing/tests/<category_name>` directory. If several category names are inputted, it searches through each specified testing category. Once each category is identified, the script iterates over the discovered test cases. For each case, it validates the content of `output.txt` against `results.txt` within the test case directory. It also checks for the presence and accuracy of a daily transaction file by comparing a generated `dtf_test_<test_name>` file against an expected `daily_transaction.txt` file. The script concludes by summarizing the results, detailing the pass or fail status for each test case and providing a cumulative report of all tests, distinguishing between passed and failed ones.

Assumptions

Below are a list of assumptions made by the system that will clarify any architectural questions that may arise in the system design.

Daily transaction file

A few assumptions have been made for writing to the daily transaction file. First, the system does **not** write transactions to the daily transaction file on a failed transaction. Also, a login transaction has been added with the transaction code 99 to signify the beginning of a session.

Passing files on startup

On starting the executable there are three mandatory files that the system needs to startup and one optional file. The three mandatory files are the current user accounts, available games, and games collection. A preformatted daily transaction file is an optional parameter to be passed, however, it **must** be an empty file.

Fixes from phase one

Specific attention was paid to fixing the missed requirements in phase 1. These requirements are:

- Exact credit limit of $999,999.00 for a users credit balance.
- Files referenced as inputs/outputs.
- Actual test files for daily transactions, available games, current user accounts, and games collection. Can be found under csci3060/storage/ directory.
- **ALL** tests were updated to include an expected daily transaction file where necessary.

Testing table

Below is the testing table for the frontend tests. Each test will have a name, purpose for the test, problem with it's output, the cause for the problem in code or test case, and what was done to fix the problem and allow the test case to pass.

It should be noted that the testing table is outlined in no particular order to when the test was conducted. Some tests that were completed earlier might appear after a test that was completed later. Also, all tests that appear in the table end on a COMPLETED note. That is, the table outlines what was done for each test to reach a passing state. If a test case does not exist in the table that means that the test case need no action to reach a passing state.

Name	Purpose	Error in Output	Cause in code	Fix
sell_bs_fail	Ensure that buy standard accounts cannot put games up for sale.	Buy standard accounts were able to put games up for sale. Leading to an infinite loop in the testing script.	Lack of validation for user account types.	Added validation that checks the current logged in user to see if they are a buy standard user, if so it gives an unauthorized error.
sell_duplicate_name	Ensure that a game is not being added to the sell list that already exists.	Infinite loop in testing script.	Lack of validation for game name in code leads to an infinite loop in the testing script.	Added validation that checks the available games file to see if a game name already exists. Also had to update the read() function for the available games file manager to ignore trailing underscores in the file processing.
sell_file_write_success	Ensure that the sell transaction is properly being written to the daily transaction file.	Infinite loop while running testing script.	Problem with test case: Test case did not include numerical value for price.	Updated test case to include a price.
sell_invalid_name	Ensure that the name length does not exceed maximum.	Infinite loop while running test script.	No validation was implemented for game name length. Causing the test to continue after it	Updated sell transaction function to include validation for game name length.

Name	Purpose	Error in Output	Cause in code	Fix
			should have failed.	
sell_without_login	Ensure that a user cannot enter the sell interface without logging in.	Infinite loop while running test script.	Session did not end after test was ran, causing the system to loop over the same inputs.	Solution: extended the logout command to act as an exit command if the user is not logged in.
Sell_aa_success, sell_fs_success	Ensure that users are properly able to list games for sale on valid input.	Terminal outputs were not being written to the results.txt file correctly. Sale price: was missing.	Std::cin stream was not flushed after collecting input, resulting in the same command to be repeated in the results.txt file.	Solution: Flushing the cin stream each time the user is prompted for input.
Sell_ss_success, sell_invalid_price, sell_fs_success, sell_duplicate_name, sell_invalid_name,	Various tests to ensure the validity or invalidity of sell parameters.	Actual output does not contain string “Entering listing interface…”	Code does not exist that prints that line.	Removed string from expected output files. It was deemed unnecessary UI that just clogs up the terminal.
sell_bs_fail	This test ensures that buy standard users are not able to put a game up for sale.	Error message not showing in results.txt file.	Script was not optimized to read std:cerr output.	A flag was added to the script to allow for cerr streams to be written to the test output file.
sell_bs_fail	This test ensures that buy standard users are not able to put a game up for sale.	Output was testing for admin username and was missing a logout message.	Expected output was using ‘admin1’ as username and did not include ‘Goodbye’ message.	Username was changed to “buystandard” and “Goodbye!” message was added.
buy_without_login	This test ensures that the buy transaction cannot be	Testing script gets stuck in an infinite loop on this case.	No validation was implemented for the buy	Added login validation to buy transaction.

Name	Purpose	Error in Output	Cause in code	Fix
	accepted unless a user is logged in.		transaction. Thus, the buy interface would begin which did not allow the system to properly exit.	
Buy_aa_success, buy_already_owned, buy_bs_success, buy_fs_success, buy_game_not_found, buy_insufficient_funds, buy_ss_fail,	These are a mixture of tests to ensure the restrictions on the buy transaction are properly working.	Infinite loop when running testing script.	Input.txt file did not have logout transaction as the final input.	Added logout at the end of each input.txt.
user_list_not_signed_in	Ensures the user list cannot be displayed with the user is not signed in	Infinite loop when running testing script	Input.txt file did not have exit as final input	Added exit at the end of the input
user_list_fs_fail user_list_bs_fail user_list_ss_fail	Ensures a full/buy/sell standard user cannot access the user list	Full/Buy/Sell standard user was able to access the users list	Transaction::list( ) did not have validation to ensure only admins could access the user list Test case input and output did not match the program input and output Missing daily transaction file in test cases	Added validation to ensure only admins can list users Updated test cases to match the program input and output Added daily transaction in test cases
user_list_aa_success	Ensures an admin user can access the user	“Users” was not accepted as input, returned	transaction.cpp did not accept “Users” as input	Updated transaction.cpp to accept

Name	Purpose	Error in Output	Cause in code	Fix
	list	“Invalid user type” Admin user was able to access the users list	Test case output did not match the program output Missing daily transaction file in test case	“Users” Updated test case to match program input and output Added daily transaction in test case
user_list_aa_not_found	Ensures the admin user cannot access the user list when it cannot be found	The user list exists so it was finding the user list	No situation where the users list could not be found	Changed the input file to ensure the program was looking for the wrong file
game_list_not_signed_in	Ensures a user cannot access a game list when they are not signed in	Creates an infinite loop when running test script	Input.txt did not have Exit as final command	Added Exit as the final input command
game_list_aa_success game_list_fs_success game_list_bs_success game_list_ss_success	Ensures each user type can see a list of available games	Invalid list type error was generating	“Games” was not accepted input, required “games”. Test case did not match program input	Updated transaction.cpp to accept “Games” as user input Test case was updated to match program input
game_list_aa_not_found game_list_fs_not_found game_list_bs_not_found game_list_ss_not_found	Ensures that a list of games cannot be returned if the list is not found	The program is set up by default to point to the file that has the correct list and cannot cause this issue	Returns the list of available games	Updated input.txt to accept the incorrect game list.
*_after_logout	Ensures transactions cannot be submitted after	No issue. Client requirement change.	N/A	Removed ALL after logout tests. System will exit on

Name	Purpose	Error in Output	Cause in code	Fix
	logout			logout so no transaction can be submitted. Logout tests will ensure that the system exits successfully.
sell_file_write_fail	This test ensures that the daily transaction file is not being written to on a failed attempt to sell a game.	Actual output does not match expected output	Issue with test case. Expected output was using different strings for confirmation of commands. There was also a punctuation error.	Changed expected output to match the rest of the sell test cases and code output. Case now passes.
buy_fs_success, buy_insufficient_funds, buy_bs_success, buy_game_not_found, buy_already_owned	These are a collection of tests to ensure the system is working correctly with the buy transaction.	Expected output did not match actual output.	Prompt for game name in buy interface was “Enter game name:” where the tests were expecting an output of “Name of game to buy:”	Changed prompt in buy code to “Name of game to buy” to match test cases. Only test not passing after this was buy_game_not_found.
buy_game_not_found	This test ensures that a game must exist in the available_games.txt file to be purchased.	No error message was found in actual output.	Code lacked validation message when a game name was not found.	Validation message was added as well as found flag to ensure that the proper transaction string is written if found or nothing written with a console error otherwise. Case now passes.
buy_ss_fail	This tests ensures that sell standard accounts cannot purchase a game in the	Actual output did not match expected output.	Issue with test case: Expected output had both the error message and the prompt for	The prompt was removed and the error message changed to match the code. Test case now

Name	Purpose	Error in Output	Cause in code	Fix
	system.		the buy interface.	passes.
*_file_write_fail	These tests were to ensure that the daily transaction file was not being updated on failed transactions.	N/A	N/A	These tests were removed because each test case individually includes a daily transaction file that will be compared with the actual daily transaction file generated by that case. Therefore it was a redundant test.
*_file_write_success	These tests were to ensure that the daily transaction file was being written correctly on successful transactions.			These tests were removed because each test case individually includes a daily transaction file that will be compared with the actual daily transaction file generated by that case. Therefore it was a redundant test.
delete_current_user	This test is created to attempt to delete the current user unsuccessfully	Attempted test case run created an infinite loop	Input file did not match test case Test case document had incorrect delete user command	Test case was corrected to run the correct input Test case document updated to have the correct delete command
delete_non_exis	This test is	Attempted test	Input file did not	Test case was

Name	Purpose	Error in Output	Cause in code	Fix
ting_user	created to attempt to delete a non-existing user unsuccessfully	case run created an infinite loop	match test case Test case document had incorrect test case name Test case and document did not have exit command	corrected have an exit command and have the correct input Test case document updated to have the correct exit command and correct test case name
delete_user_aa_success	Attempts to delete a user while logged in as admin	Caused an infinite loop No output confirming successful deletion	Test case input was incorrect Updated transaction.cpp to add a confirmation output	Updated test case input file to have the correct script Updated test case name to accurately represent the admin user can delete users
delete_user_fs_fail delete_user_bs_fail delete_user_ss_fail	Attempts to delete a user logged in as admin	Non-existent cases, added to ensure only admin user could delete users All users could delete users	Test cases missing from test case document Verification missing to make sure only admins can delete users Program output and test case output did not match	Test cases were added to the test case document Verification was added to ensure admins are the only users who can delete other users Updated test cases to match program output
delete_without_login	Attempts to delete a user without login	Caused an infinite loop	Test case input did not include an exit command	Updated test case input and test case document to

Name	Purpose	Error in Output	Cause in code	Fix
			transaction.cpp did not verify that a user is logged in to run the delete command	include exit command Updated transaction.cpp to verify a user is logged in
create_duplicate_user	Ensure that duplicate users cannot be created.	Problem with input: not all commands were included in the input.	Test case input was incorrect	Updated test case input to have the correct input procedure.
invalid_user_type	Ensure that only AA, BS, FS, or SS users are being created	Problem with input: not all commands were included in the input.	Test case input was incorrect	Updated test case input to have the correct input procedure.
create_invalid_username	Ensure that the username is capped at 15 characters.	Problem with input: not all commands were included in the input.	Test case input was incorrect	Updated test case input to have the correct input procedure.
Create_new_fs_user, create_new_bs_user, create_new_ss_user, create_new_aa_user	Ensure that a FS, BS, and SS users can be created	N/A	Problem with cases. Only one case handled creating new users that only checked for FS users.	Added cases to test creation of all user types.
Create_ss_fail, create_bs_fail, create_fs_fail	Cases that test the validation of creating user accounts.	Non-admin users were able to create user accounts	No validation was implemented in create_user transaction code.	Validation was added to ensure only an admin can create a user.
create_max_credit_exceeded	Ensures that a user cannot be created with more than the max credit amount.	N/A	Case was not implemented.	Case was added as well as validation in code for credit amount.

Name	Purpose	Error in Output	Cause in code	Fix
create_duplicate_user	Ensures a user can't be created with an existing name.	Infinite loop was triggered by the testing suite.	No validation was implemented for if a username already exists in the system.	Added validation to ensure the username does not already exist.
create_invalid_user_type	Ensures that AA, FS, SS, or BS is entered as the user type when creating a user.	Infinite loop when running test case.	No validation for user type in code.	Added validation for user type in code.
create_invalid_username	Ensures that the username cannot exceed 15 characters	Infinite loop when running test scripts.	No validation for username in code.	Added validation to ensure username cannot exceed 15 characters.
ALL TESTS	Every test that involves writing to daily transaction file	An extra _ in the daily transaction file	Username fields were being written with incorrect field length due to a typo in format transaction string functions	Change the field width from the incorrect numbers to 15.
login_already_loged_in	Ensures that a logged in user cannot attempt to login as another user	Infinite loop in test (asks user for username repeatedly)	Validating if the user was logged in was done after prompting for the username	Move the check to the top of the login procedure
logout_daily_transaction_update	Was to ensure that the daily transaction file was written upon logout	N/A	N/A	Test was removed since all test cases check for the daily transaction file write
logout_failure	Redundant test	N/A	N/A	Test was removed since it was redundant (already have logout_without_login)

Name	Purpose	Error in Output	Cause in code	Fix
refund_game_not_found refund_game_not_sold refund_game_not_owned	Test case verifies the game in the system exists	N/A	N/A	Removed cases because there is no requirement to have game input, making these cases unnecessary
refund_aa_success	Tests a successful admin refund process	Caused an infinite loop	No Logout command in user input	Added a Logout command in user input
refund_bs_fail refund_ss_fail refund_fs_fail	Tests a unsuccessful non-admin refund process	Caused infinite loops	No logout command in user input	Added logout command in user inputTh
refund_without_login	Test an attempt to refund without being logged in	Caused an infinite loop	No exit command in user input	Added an exit command in user input
addcredit_xx_max_session_amount_exceeded	Prevent a user from adding more than 1000 credits in their current session	Would always pass	No current session variable that keeps track of credit addition	Added current session variable that keeps track of credit addition. (Currently starts at 10.00 for testing purposes)
buy_already_owned	Prevent a user from buying a game that already exists in their collection	No output for user failing to purchase game.	No validation existed that prevented a user from buying a game they owned.	Added validation to ensure a user cannot purchase a game they already own.
buy_insufficient_funds	Ensure that a user cannot buy a game they cannot afford	No output for a user having insufficient funds to purchase a game.	No validation existed in code that checked for a user having the correct amount of funds to purchase a game.	Added validation to check that a user has enough available_credit to purchase a game.
addcredit_xx_m	Prevent a user	Would always	No validation for	Added validation

Name	Purpose	Error in Output	Cause in code	Fix
ax_credit_fail	from simply entering a credit amount greater than 1000 (would automatically exceed session limit)	pass	this case existed	for this case
addcredit_xx_negative_credit_fail	Prevent a user from entering a negative float as the credit amount	Would always pass	No validation for checking for negative input	Added validation for this case
addcredit_xx_*	Make sure all tests that require a daily_transaction file were not missing one	Tests would always show a warning indicating that the daily_transaction.txt file was missing	No daily_transaction.txt were present in any of the cases	Added daily_transaction.txt file to every test case to prevent warning from popping up
refund_buyer_not_found	Ensure that the input for the buyer is a valid user	Infinie loop reached when test was run	No validation in code as to whether the buyer was a real user or not.	Added validation to check the current users file for the buyers username.
refund_fs_fail, refund_ss_fail, refund_bs_fail	Ensure that non-admin users cannot issue refunds.	Infinite loop reached when test was run.	Code assumed that fs, ss, and bs users could refund their own transaction but not that of others users.	Code was changed to check for admin user type and return an unauthorized error otherwise.
refund_seller_not_found	Ensure that the seller is an active user in the system.	Infinite loop reach when test was run.	No validation for whether the seller was a real user or not.	Added validation to check the current users file for the sellers username.
refund_seller_insufficient_funds	Ensure that the seller has enough credits to refund the game.	Refund was processed when seller has insufficient funds.	No validation for whether the seller had enough credits for the refund or	Added validation to check the sellers available credits to see if they had enough

Name	Purpose	Error in Output	Cause in code	Fix
			not.	to process the refund.