Dennis Krupitsky, Matthew Connelly, Andrew Teterycz
CSC 330 – Object Oriented Programming
Project #01 – Library Systems Information - Phase 5 (Test Cases and Results)
Due Date: 04/17/2019

Test Cases and Results

Testing occurred during and after feature development. Informal tests were conducted during development to check for correct behavior of methods and correct instantiation of class objects. Formal tests were then written to be reused in the future and to establish a pattern for testing as well as for intended use of classes.

The majority of test cases were for each class's functions, to ensure that each object would behave predictably and as intended on its own. In addition, we also tested the end product's behavior and capability as an interactive program.

There were test drivers written for classes BookInformation, BorrowerInformation, LibraryInventory, and Date. Outside of these class-oriented test drivers, there was a test driver written to check LibraryInventory's data importing functions and the program's main driver, which served to also test the Menu class's functionality.

Class-Oriented Test Drivers

The class-oriented test drivers involved calling all methods and comparing the results with expected or desired results. Both sets of results are written to standard out for the developer to observe.

A set of data is defined at the start of program; this data is to serve as the inputs for the object's methods. For example, LibraryInventory's inventory-setting function requires a string argument. This string will contain the name of the data file to import. This string is defined before any methods are called to help establish test controls.

A large portion of these tests was setter and getter testing, which ensures that a given object will contain the correct data after construction and mutation. This facet of these tests sets them apart from functionality testing drivers.

Functionality Test Drivers

Extensive testing of functionality was employed in the functionality test drivers. The program's main driver served to test the menu class's methods, which involved user interaction. The user would test a feature first with a good input and then a bad one.

For example, for withdrawing books from the book database through the menu, a user must input a valid book name. Testing this feature would involve first using the name of a book known to exist in the database, followed by using a book or any other string that does not correspond with the known data. A developer would run the program several times, entering data.

For testing data import, a driver calling the import functions and then calling the print functions was written. This simple driver was then run at different points of development to test the accuracy of the data being imported; making sure the data was not corrupted during the import process.

Once the main driver utilizing the menu was written, however, this small test driver was not used as much as when data was initially being created.

The Effect of Data on Testing

The data used in this program was stored in CSV (comma-separated values) format. This format is simple and almost universally easy for programs to read. A developer can easily understand at a glance what the dataset entails by viewing the CSV file in a spreadsheet or even as plain text.

Specifying commas and newlines as delimiters, the CSV file can be read by a C++ program and stored in lists.

The handling of data was tested the most in the main program when interfacing with the menu object. Some issues encountered were with handling user input. While whitespace is ignored in the data import function via a getline call, user-entered whitespace is recorded by a C++ program when an input stream operator is used to store a value. The issue here was that more than a few books had titles with whitespace in their name. The solution was to get user input with the STL cin getline method.

Previous user input had to be ignored as well, as old buffered newlines would be counted in user input when the user was prompted for a book title, which would make for a bad book title despite the user entering a good one. This was countered with the cin ignore method.

Test Results

The results of testing made for a clearer guide to API implementation and a program with few visible bugs.

A majority of bugs were removed during feature development, which was confirmed during the writing of formal tests. Additional bugs were found and removed during the main program's testing; these were bugs dependent on runtime variables like user input.