Alexandru Majeru

CSC 330

Project #2

Test Plan

As with any program, the Flin Flon Airlines Booking Management System built for this project requires testing to ensure its proper operations according to specifications listed in “Requirements Specifications” of the project.

In this case, testing of the system consisted mostly of ensuring the GUI of the system is properly set up and that the data flow between the SQL database and the GUI does not create any conflicts. Moreover, certain information that can be input by the employee – for example, dates of birth or flight for passengers – need certain verification so that the data makes sense when it is read at some point in the future.

The first step of any run of the system is determining if the user utilizing the system is in the database. The user is asked to input his or her Employee ID followed by a personal password. After this the system either finds the user in the database or asks to contact an administrator – who can add the user into the database and essentially “register” an employee. It is assumed that the user has an assigned ID, so the system does not assign a new value to the employee - the administrator must use it when creating a new user.

Furthermore, the system must be able to recognize the status of the user – either they are a regular booking agent or part of the administration. Depending on this, a different menu will be presented – and administrator has all the menu choices a booking agent has with addition of options of managing client, flights, and staff.

The testing was conducted assuming the following possibilities:

1. User ID has special symbols in it
2. User inputs a long ID (over 10 characters)
3. User password is incorrect
4. User password is two short
5. User is a booking agent
6. User is an administrator

Next, we presented the menu to the user. The menu was built using multiple panels on a single form – with panels becoming visible or not depending on what choices user makes. The testing of this part primarily focuses on the functionality – it is important to prove that all the buttons and overlays are functioning properly.

The menu for a booking agent presents four options, for administrator – eight. Depending on the option clicked, a different menu setup will be displayed. It is important to test each button individually.

Following the addition of the user to the system (or confirmation of employee record), the user is presented with a menu of choices, which include:

1. Add New Item to History
2. See List of Employees
3. Display History
4. Exit

The above choices are numbered, with user selecting one of the options. To ensure the proper run of the program, we need to test the user input in the menu:

1. User inputs a value outside of the menu range (number above 4)
2. User inputs a symbol that is not a number
3. User inputs multiple symbols

Once the user selects an option, the program either displays a table of contents (Employees list) or provide the user with a second menu – either for adding a vehicle case or select the type of history the user wants to display. In both menus, the system presents the user with a number of choices. The testing in this part of the program is as above – it is important to make sure we do not get a failure if the user inputs a value outside of the expected ones.

Aside from user interactions with the program, the testing process also covers the general run of the program. The system operates with data from two separate text files – “databases” of Employees and previous Orders (History of the system). Testing of externals consists of catching the following cases:

1. “History.txt” file does not exist
2. “Employess.txt” file does not exist
3. Files are corrupted/unable to open
4. Employee record already exists

Since the program operates mostly with strings and characters, testing also involves ensuring proper analysis of data input by the user. This involves:

1. Correct input types from user when collecting Employee/Vehicle data
2. Proper parsing of strings collected from “History” and “Employees” text files
3. Proper formatting of strings when writing (adding) data to text files

Moreover, since the system highly depends on advanced class functionality, testing is necessary to ensure its proper implementation:

1. Proper inheritance between purely abstract base class Vehicle and derived classes Cargo, Business, and Loan
2. Correct implementation of polymorphism between class Order, which is composed of abstract class Vehicle, and the derived classes of base class Vehicle
3. Usage of STL (Standard Template Library) containers, iterators, and algorithms when recording data from external file for parsing/analysis/output.

Finally, testing also covers proper display of data collected from the users, i.e. readability of tables displayed as the result of user interactions with the system. Main cases are:

1. Tables are easy to read and have proper labeling
2. No data overlapping, resulting on two separate data inputs sharing the same field on the display
3. Correct display of time and date of the Orders placed in the system (as a local time and date and not an encoded integer)
4. Tables containing the History data in proper order (newest entries on top, oldest at the bottom)