Alexandru Majeru

CSC 330

Project #1

Test Cases, Results, and Outputs

Previously, it was stated that every functioning system must be thoroughly tested before it can be presented to the user. In the previous submission, a test plan was stated. The primary focus of the tests was on the user input at different stages of the system run – however, additional requirements were placed, which described the proper location of the system files and how to operate them.

The first stage of the program run required the user to input his or her Employee ID. If the ID was found in the system, it would be displayed on the console and the program will move to the second step – the menu. However, if the ID was not found, the user is requested to input his or her first name and last name. The testing of the program was required to set up this step of the program run to operate properly:

|  |  |
| --- | --- |
| **Test Case** | **Results** |
| User inputs a number as part of their name | System informs the user that the numbers are not allowed as part of the name; asks to input the name again |
| User inputs the name consisting of multiple parts/words | System reads the name successfully as a string |
| User inputs a long name | System reads the name successfully (limit of characters – 18) |

After acquiring the users ID and name (the program assumes that the user IDs are assigned outside of the system, with each user possessing a unique combination of letters and numbers), the main menu is displayed. The user is presented with four options, two of which lead to secondary menus, one displays the list of employees, and one exits the program. This step was tested as follows:

|  |  |
| --- | --- |
| **Test Case** | **Results** |
| User inputs a number outside of menu range | System informs the user that the value is not valid, resets menu |
| User input is not a number | System informs the user that the value is not valid, resets menu |
| User inputs multiple numbers/symbols | System informs the user that the value is not valid, resets menu |

When the user input the proper value (1 through 4), the system continues its run further. If the user decides to add an order to the history, the program will present three possible choices, representing the type of the vehicle (Cargo, Business, or Loan), as well as the “Back” option to return to the previous menu. The testing for this step was the same as in the previous one – it was important to make sure the user provides proper input to the program, otherwise the menu resets.

Once the user chooses the type of vehicle to add to the system, he or she is required to enter the data about the order – vehicle model, license plate, driver, and mileage, as well as some type-specific information. User inputs are mostly unlimited – the size of most fields is enough to accommodate most vehicle types. However, there was still necessity in the following cases:

|  |  |
| --- | --- |
| **Test Case** | **Results** |
| User inputs a non-numeric value for mileage field | System informs the user that the value is not valid, asks for a different input |
| User inputs a non-numeric value for passenger number for Business Trip type | System informs the user that the value is not valid, asks for a different input |
| User inputs a non-numeric value for cargo weight for Cargo Transportation type | System informs the user that the value is not valid, asks for a different input |
| User inputs Driver ID that is in the system | System uses the Driver data saved in the system to fill in the necessary fields |
| User inputs Driver ID that is not in the system | System asks the user to input data about the Driver |

Once the user inputs all the necessary data, he or she is returned to the main menu. Next, if the user chooses the second option – displaying the employees – the console displays the list in form of a table. Since the program requires the user to register in the system, the list of employees is never empty and contains at least one record of the current user. Testing of the availability of external files will be described later.

If the user decides to view the history of company vehicle use by following the third option, he or she will be shown another menu containing five options – three for different vehicle types, one for the entire history of company vehicles, and one to return to the main menu. The testing of this step is similar to testing of menus described previously – the program is making sure that the input by the user is numeric and contains values that describe menu items. The program then displays the table of results, color-coded by type (Green for Time, Cyan for Category, and White for History Data).

Due to the constraints of the console size (width of 800 pixels), the data for each case is displayed in three rows – first row contains the type of vehicle and driver information, second row describes the vehicle and mileage of the trip, and the third row shows additional information about the trip, depending on the type.

The final option in the main menu is the exit option, which terminates the program.

Aside from testing the user interface for irregular input, the system required a proper way to operate with two external files. The way the system was setup is that all the data from files was read at the system start and stored in two list containers. When the user introduced a new vehicle order, it was not only added to the list, but also written into the corresponding file at the runtime. This avoided the necessity to create copies of text files and delete the existing ones to avoid any confusion when the program was commanded to exit.

The main problems regarding the file reading were based on file presence in the system and proper parsing of data to separate different information about vehicles:

|  |  |
| --- | --- |
| **Test Case** | **Results** |
| “History.txt” is not found in working directory | System creates a new “History.txt” file |
| “Employees.txt” is not found in working directory | System creates a new “Employees.txt” file |
| User inputs new data into files | Each data member (for either Vehicle or User objects) is written into its respective file (“History” or “Employees”) separated by a “|” symbol |
| User requests data from files | Each individual field is read using “getline” command until it reaches the “|” symbol. The data read is saved in a variable. If the data is saved in form of an integer/double, the “ifstream” data type is used. |
| “.exe” file created to run the system is launched in a separate folder, with none of the source code present | The folder is designated the working directory, in which two “.txt” files necessary for operation are created and the system runs as intended |

**Outputs:**

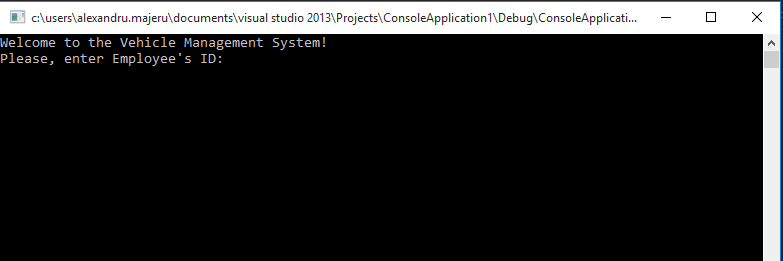


Figure Welcome Screen

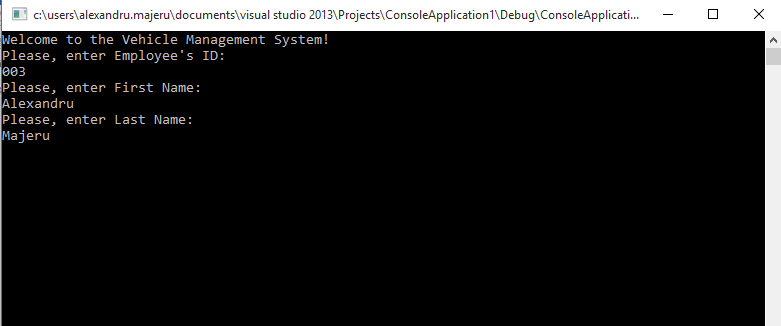


Figure Console in case of a new employee using the system

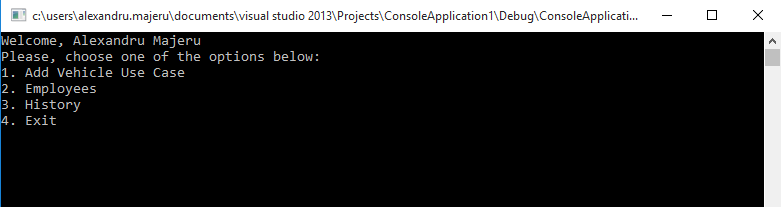


Figure Main Menu of the system

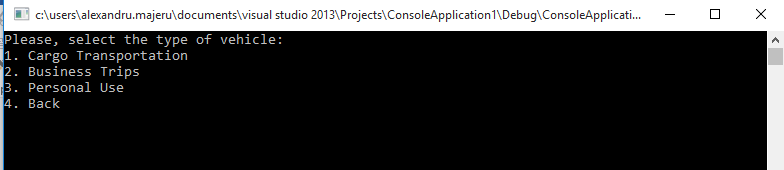


Figure Adding a New Case menu

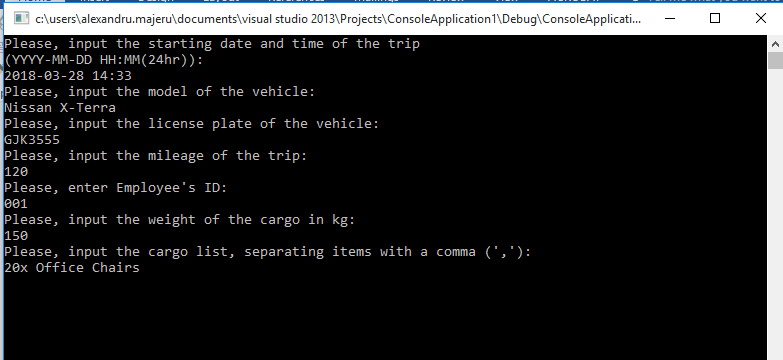


Figure User input to create a new case

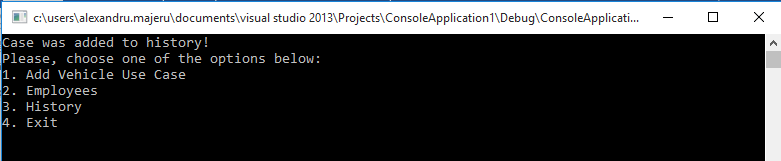


Figure After inputting a new case

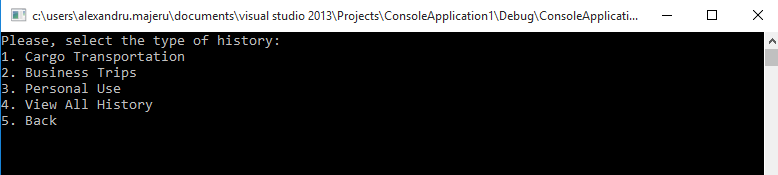


Figure Seeing the history of vehicle use

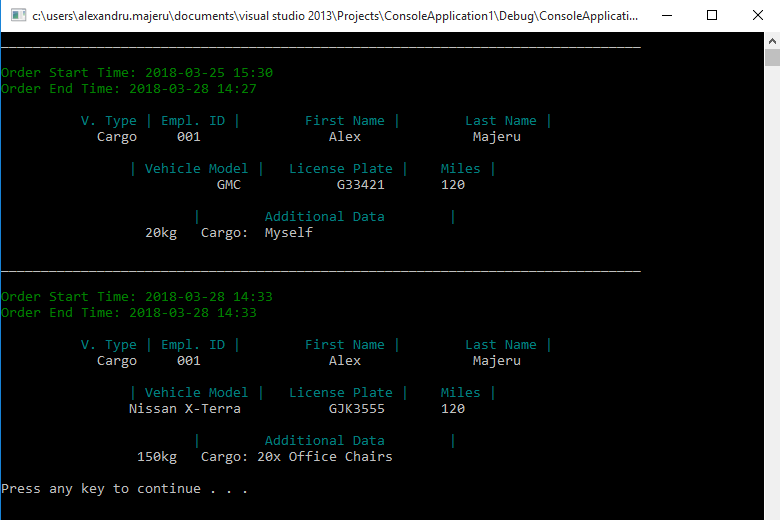


Figure After inputting a new case

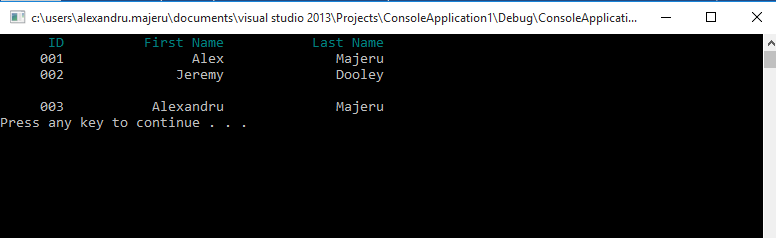


Figure List of Employees after adding a new ID

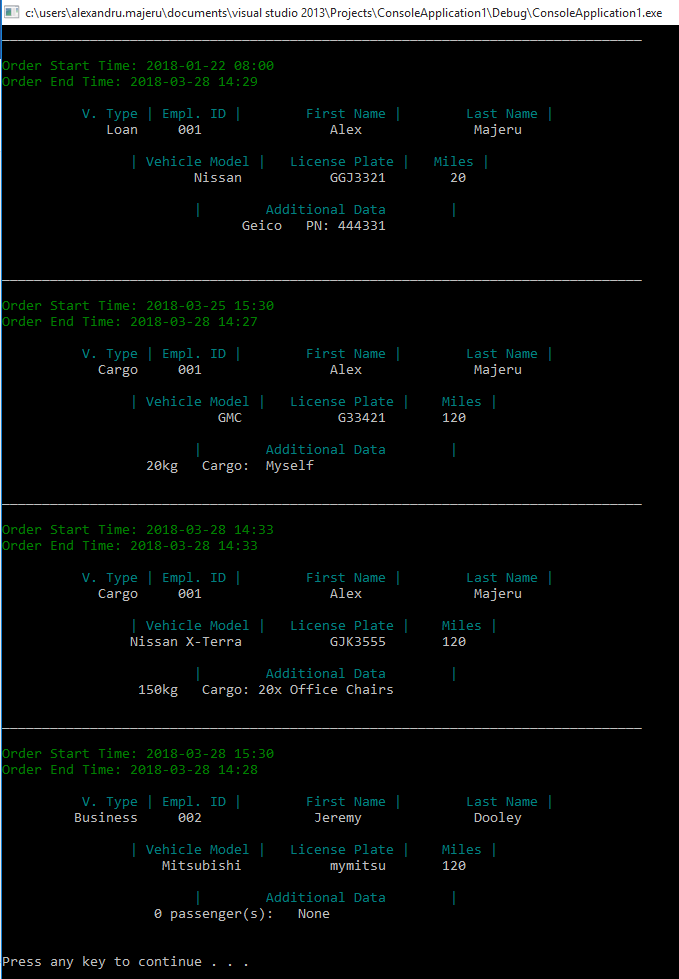


Figure Entire history, including new input, in chronological order (based on start time)