Alexandru Majeru

CSC 330

Project #1

README

The purpose of this file is to describe the functionality of the Vehicle Management System as well as elaborate on the process of creating it. The system is relatively simple – user is expected to launch the .exe file provided, which will start the program. It will also create two .txt files in the folder at which .exe is located – for this reason it is recommended to start the system in a dedicated folder to avoid any confusion and file error/mismatch. If the system has been utilized before and the above files are already present within the directory, the system will open the existing .txt files instead of creating new ones.

The user is asked to input his or her ID – it can be any combination of symbols and the system assumes that the employee already has an assigned ID. If the ID is not present in the “Employees.txt” file, the user is asked to input his or her name. If the ID is present, the system greets the user and displays the menu.

Once the menu is displayed, the user has four options – adding new vehicle use case, reviewing the list of employees, and reviewing previous vehicle use cases. The final option is terminating the program, which does not affect the external files.

In case the user decides to add a new vehicle case, he or she will be asked to choose the type/purpose of the vehicle and asked to input the information, describing the case. If the user decides to display the list of employees, it will be output on the console in a form of a table, containing user ID, First Name, and Last Name. If the user decides to display the history, he will be provided with a choice of history type – either a specific vehicle type, or the entirety of history. This will display the vehicle type of choice in form of a table, displaying vehicle data.

The main issue with the system built is its simplicity. The program acts as a database manager with very limited capabilities – the user can either add vehicle cases to history or display data from the database (employees or vehicle cases). It does not follow the vehicles that are currently being used by the employees or the vehicles available to the employees for various uses. The user is not capable of redacting any data within the system. In addition, the user can introduce any new employees and vehicles on demand, without verifying that they actually exist.

Despite its simplicity, the system utilizes the notions of polymorphism, composition, inheritance, as well as STL. The implementation of the above did not prove problematic – the relationships between classes were understandable and easy to create, helped in simplifying the system implementation, and decreased the amount of code and memory required for execution.

The main issue encountered in the process of building the system was formatting of the data from the user and external files. Proper analysis of data required very accurate parsing of data and, in some cases, relies on the user inputting proper data. However, in every case evaluated in “Test Plan” and “Test Cases and Results” documentation, the program was successful in catching inconsistencies or undesired data in user input and preventing the errors.

Aside from data formatting issues, another prevalent problem encountered was the proper comparison of time. The idea was for the system to sort the history by the starting date of the order using the list container algorithm. To do so, the date could not be saved in format provided by the “ctime.h” header.

Overall the project, while having relatively simple results, was successful in testing and implementing the materials covered in class and operated as intended. However, its limitations are problematic and cannot be overlooked if the project is to be used in a professional environment. To solve this limitations, first of all, the system functionality would have to be significantly expanded from simply taking in the user input and saving it as history, to actually working with two databases. To change the program, it would be required to add an additional external file – a list of vehicles that are available to users. This list will be redacted depending on whether the user is taking a vehicle from the system or returning one.

If the above changes were to be implemented, the system would move from history addition/display functionality to actually following and managing current orders made by users at different times. However, these changes would require addition of significant complexity to the system – the User class would need to store the data regarding the vehicle they are currently using (if any). This would bring significant changes to the process of reading the files, string parsing, and saving of data. At the same time, it will limit user input into the system, which would significantly decrease the possibility of error in the system – however, it will also make addition of new vehicles problematic and require it to be done externally, not through the system (or expand the capabilities of the system even further and make it depend on the user input again).