Abraham Meza

Christos Papachristos

CS 202 – 1001

February 27, 2019

Project 4 Documentation

**Description:**

This project will significantly expand upon Project 3 by adding additional functionality, and implementing more abstract data types (ADTs) and their operations through classes. **Pointers must be used for all array manipulation**, including arrays with ADTs (structs, classes) e.g, rental cars, rental agencies. **Pointers must be used in function prototypes and function parameter lists** - not square brackets. Make sure all your C-string functions (e.g. string copy, string compare, etc.) work with pointers (parameters list and function implementation). **Const** should be used in **parameter lists**, **functions**, and **function signatures** as appropriate. Square brackets should be used only when declaring an array, or if otherwise you specify your own overloaded operator[] . **Pointer arithmetic** (e.g., ++ , +=, - -, -=) and **setting the pointer back to the base address** using the array name **can be used to move through arrays**.

The additional functionality is as follows: You are given an updated data file where there is 1 Agency location (**Agency**) which has **5** cars (**Car**) which can potentially be of a high-tech type. Each car can incorporate **up to 3** (0-3) special driving sensors (**Sensor**). You will have **similar menu options**, but the **functionality has been updated** below. Note: using multiple helper functions to do smaller tasks will make this project significantly easier.

Continuing through Computer Science II (202), our instructor assigned us a project that demonstrates our abilities to utilize multiple class skeletons that incorporate abstract data types and their operations, in order to access, read, and write into a file. As an extension from project 3, this project utilizes the same methods to transfer data to/from the file to the terminal; however, it takes the assignment a step further by making us edit the file in order to extract the information. Students had to utilize multiple programs to create a Car Rental menu that allows the user to see the car data of the corresponding agency. This includes: agency name, zip code, model, make, year, high tech sensors, renter, etc. The menu options would include sensor-type display commands, to rental prompts that are activated when looking for the most expensive car. As the project continues to be an add-on to our previous projects, it still challenges students to learn how to utilize pointer arithmetic within abstract data types as it covers operations, new key words, and structures. In addition, the project continues to polish the students’ use of switch statements when creating a user menu. As students tackle on data structures and multiple-file linking, they are instructed to utilize separate programs for each kind of information (agency, car, sensors, etc.) for smaller tracing and debugging.   
 For my design, I began with creating separate header files for each category of information. Within each header file, I created a class skeleton that illustrated all of the required information for the project, along with the functions that will toggle certain menu options. Once I declared all of my members and variables, I constructed program files that manipulated the header files to initialize all of the categories. Based on the instructions within the project parameters, I created my constructors and member getters/setters to reflect the requirements (specific parameters, re-setters, etc.)). Overall, the class skeletons were very intuitive as they reflected the same structure from the previous project. Even so, I had to dictate how to connect data structures to my menu functions. Firstly, I created my file read algorithm that would scan through the document for the required information. However, I had a difficult time scanning through the unnecessary symbols that had to be omitted from the given text file. After relooking at some of the slides, I realized that I could utilize keywords such as “friend” operators and “this” referencing. Once I was able to create a proper algorithm with the friend operators ([], +, <<), I was able to conduct the algorithms within the operators without having to do them within the menu function. After creating and testing the function of an operator, I could just reference it through keyword “this” and it would have the proper functionality. As a result of being able to read the file information, I was able to create relational functions that would utilize specific parts of the information to execute a certain task. For menu options such as printing the corresponding car data, the structure would be closely similar to the read-in operator, thus utilized the same concepts. However, I had to created my own algorithms for the other menu functions since they required specific operations and loop implementations. Even so, the functions were very similar to the ones in previous projects; I only had to modify the algorithms to implement data type structures accordingly (connections through the dot operator, arrow operator, class array manipulation, and pointer arithmetic). To do so, I took each function and spit it into multiple functions, if needed. Through this, I was able to create it piece-by-piece instead of creating a big function, thus making it easier to construct and debug.

After completing all of the program files and linking them appropriately, I created a main function in a separate program file that connected all of the files together. Within the main function, the switch statement structure was identical to the previous projects as it expressed the different options that the user may select. In addition to that, I declared and defined all of the appropriate string functions that mimicked the string library in order to make all of the files function globally. From there, I simply called the class functions I created to manage each certain task in their corresponding cases. Afterwards, I tested and managed my program to meet the requirements within the instructions. Even though I was able to properly establish a flowing relationship between all of the programming files, I ran into some simple debugging due to pointer arithmetic errors. As a result of being a lengthy project, I realized why we were instructed to split the assignment into separate parts. When debugging the code, it was easier to pinpoint the faults as it addressed it specifically within the terminal. At times, fixing errors would cause more errors in different files; even so, I knew what needed to be fixed since the error came from the same file.

Though my design is identical to the previous projects we’ve had this semester, the implementation of data structures and pointer arithmetic made me think harder about the algorithms needed for proper functionality. Unfortunately, I could not express the specific sensors for each car within my printing function. Every time I would attempt to access the inventory from a differentiating file, it would not allow me to access its contents. In addition, when I attempted to access the getSensors() function, and the private sensor member, I would continuously get an address instead of characters that were read from the file. I played around with the pointer arithmetic and operators but could not find a way around it. Even though this project revolved around the concept of implementing data structures within class skeletons, I still have a long way to go in understanding its full functionality.

If I were to have more time with the project, I would improve its functionality by adding more elements of convenience and efficiency by manipulating the car sensor menu options (2, 3) by addressing which car had which sensors.

All in all, the project was very efficient in teaching students about data structures and pointer arithmetic as they tackle on new algorithmic implementation through the utilization of keywords and operators. Personally, I enjoyed making operator functions as it allowed them to be used anywhere; even so, I feel more comfortable with the traditional function prototype and algorithmic definitions.