Data Structures Project Part 3

May 5th, 2022

By: Lavan Vivekanandasarma

Problem Description

The core problem/challenge that this program is aiming to solve/address, is designing a C++ program that can receive data via a file of comma-separated values (.csv) that includes a student's assignment names, descriptions, categories, and point related information concerning those assignments and then be able to copy that information into another .csv file or display it to the console in a tabular format. Furthermore, you should be able to add grade items to the data structure, in addition to searching for them by description and date.

Overall Software Architecture

The function that runs everything, known as the main() is located within the database.cpp file. Following standard C++ protocol, it returns a 0 if everything runs properly. It contains the code for the main menu that uses a do-while loop to allow the user to choose one of the four primary functions by asking the user to enter a "1", "2", "3", "4", "5" or "6" depending on what the user would like to do. The proper choice is selected via a switch statement.

If the user enters a "1" into the console while on the main menu, the functionality for reading .csv files will be enabled. It will prompt the user for a .csv file name and will confirm that it is a .csv file by checking the length of the input string given (since the smallest possible length for a .csv file name is five) and by checking the last four characters to make sure that the inputted name is actually a .csv file. Then afterward, it will call the .read() function on an R1.cpp object, derived from the R1 class which has the implementation of the read() method. The read() method in R1.cpp uses ifStream's infile(), getline(), and a while loop for each line in the .csv file to store each value into a variable. Then those variables are used as arguments for the GradeItem object constructor, which is then created and put into the gradeInfo vector which is a private

instance variable of the R1.cpp, along with the header variable. Then after .read() finishes running in the main() it will check the header private instance variable of the R1.cpp to see if it was filled which is a symptom of either empty .csv files or nonexistent .csv files, and as such the user will be notified. If there is data within the file, then the .csv file exists and is filled with data, and a local vector and string variable within main() will be filled with the gradeInfo and header objects respectively from the R1 object, and in doing so completing the read function.

If the user enters a “2” the functionality to copy the data will be engaged. First, it checks if the local variables within main have been filled with the data from the read method, if not then it will instruct the user as such and will give the user the option to either exit or return to the main menu. If the data is filled, then it will ask the user for a file to export to, and store the name to a local variable. Then it will create an instance of the R6.cpp class where the functionality for copying is implemented and call the .write() method where the header and gradeInfo variables are parameters. The .write() method within R6.cpp uses ofStream, .open(), and a for loop for each element within the gradeInfo variable to output information to a new file using getter methods within the GradeItem class to access items. After the .write() method is called in the main() it will prompt the user asking to either continue or terminate the program just as before.

If the user enters a “3” the functionality to display the data will be engaged. First, just as it does for the copy method, it checks if there is any data to display. If there isn’t, it will inform the user, otherwise, it will create an instance of the R7.cpp class and use the .displayGradeInfo() method, where a vector of GradeItem objects is the parameter. Then using printf, .c_str(), and getter methods the .displayGradeInfo() method prints out the information within the vector of GradeItem objects in a tabular manner. Furthermore using similar functionality it will call R8.cpp’s method .displayGradeSummary() method to display a summary in a tabular format by

calculating the four major categories (Quiz, Homework aka HW, Class work, Exam), their associated point totals, and the overall final grade after passing the vector of grade objects as an input.

If the user enters a “4” the functionality to add a grade item will be engaged. First, an instance of the R2.cpp class is created. Then it calls the addGrade function with the gradeItem vector as a parameter, which will prompt the user for a date, description, type, Maximum grade, and grade information, and use it to create an instance of the GradeItem class, and append it to the vector that was fed into the method parameter.

If the user enters a “5” the functionality to search for a grade item will be engaged. First, an instance of the R3.cpp class is created. Then it prompts the user to enter a “1” to search by date or anything else to search by description. Then it calls one of the two member functions searchGradeByDesc or searchGradeByDate accordingly, asking the user for either a date or description to feed as a parameter to the function.

Finally, if the user enters a “6” then the program will terminate and return a 0. One thing to note is that there is a function within the main called continueFunctions() which will be called upon any error or successful termination of a task, where it will prompt the user for a “0” or a “1” to either continue doing tasks by going back to the main menu or to terminate the program respectively. Some uses of this function were mentioned prior to this. While not necessary, this function makes the user experience easier to use, less confusing by making it easier to follow.

Input Requirements

As of now the main external input requirements are for the read() and write() functionalities. Currently, the input requirement necessitates that inputs for file read() functionality are .csv files and that they are filled such that there is at least one line of data and a name of at least five characters. For the write() functionality, there are no real stipulations on what the file has to be named as long as it takes lines of values. For searching, and adding, all inputs should also be strings.

Output Requirements

All outputs to files are currently done with string values. The output file's data that is written by the program are a series of strings since those are what is taken in from the input.

Problem Discussion

The read() function runs in O(n) time, where n is the number of lines in the input file. The write() function also runs in O(n) time, where n is the number of elements in the gradeInfo vector. The displayGradeInfo() and displayGradeSummary functions run in O(n) time as well, where n is the number of items in the gradeInfo vector and uses c_str() so string inputs can be used with printf. The search function runs in O(n) time as it needs to scan the entire vector once, and is O(n) in space as it returns all matching items. The add function runs in constant time since all it is doing is a few elementary operations.

Data Structures

The notable deliberate decisions of data structure use include the choice of vector over an array or linked list and the String element for the grade values. A vector was chosen over a standard array due to the nature of flexibility regarding size, and was chosen over a linked list due to convenience and ease of directly accessing elements. As for the grade values, choosing string values may seem counterintuitive at first, but as of now there are no mathematical operations that need to be performed with the grade info, and all of the current operations that require the grade information necessitate that they are in string format. Furthermore, a toInt() method using C++'s stoi() function is used in R8.cpp to calculate the summary of the grades.

User Interface Scheme

The current UI Scheme includes a screen that asks the user to input specific numbers (ex. "Enter 1 to read a .csv file) with very clear instructions to complete such a task. Likewise, after any tasks are completed or if any issues arise there is a menu that asks the user if they would like to continue or exit the program.

Status of Application

The application fully works as of now, and was created using Microsoft's Visual Studio Code. As per the author's understanding, it follows all of the requirements illustrated by the rubric.