**Class Project Part 3**

Michael Pultorak

05/08/2022

**Problem Description:**

This program implements a user menu that allows the user to read, write, add, display, and search for grade items in a .csv format. The user will be able to 1: open a .csv file. This file will be read and placed into a Grade object. Each grade object has a max grade, grade, description, type of grade, and date. Each grade in the .csv file will be placed into a vector. The user can also 2: write to a file. This will write everything in the grade vector into a new file based on user input. The user can also 3: add a grade item to the vector. The user can 4: print each grade item into the terminal in a table format. This will also print the total score and the current grade. The user can also 5: search for grade items by date or description Finally, option 6: closes the program and deletes the grade vector.

**Overall Software Architecture:**

The menu() function within database is the core function of this program. By calling this function, it will create the grades vector to store data. This function also implements the menu loop. At the beginning of this function, it will call display\_menu() which prints out the UI. After this, get\_input() will be called, which prompts for an integer input and ensures input given is an integer. With the user input now taken, we now scan for the option the user selected. There are 6 menu options, one to exit program, one to read a .csv file, one to write a .csv file, one to display data read in the terminal, one to add grade items, one to search items, and one to exit program. All of these, other than exiting the program, require the grade class. This class is composed of a max grade, grade, description, type of grade, and date. When reading a file with read\_file(), in R1.cpp, the first line of the file, or heading, is ignored. The rest of the lines are split by comma and put into a grade object and then added to a grade vector. This is repeated until an empty line is read, then the grade vector is returned. When writing a file in write\_file(), in R6.cpp, the heading is first inserted then grade vector is looped though and grade items are added to the file separated by a comma. Each grade object is a new line. In display\_list(), in R7.cpp, the heading is first printed into the terminal and the grade vector is looped though and printed. Each grade object is a new line. In add\_item(), in R2.cpp, the user will be prompted to input values that will be placed into a grade item then put into the grade vector. In search\_grades(), in R3.cpp, the user will be asked which value to search by then for what value. The Grade vector will then be searched though for the item. Finally, in R8.cpp, format\_summary() will find the total and recived score for all grade items of specified type. This data will then be fed into print\_summary(), which will print these result. Print\_summary() will also call find\_letter\_grade(), which will return the letter grade for the percentage score. Menu() calles this function when prompted to print grade items. Thus, R8.cpp, R7.cpp. R1.cpp ,R6.cpp, R2.cpp, and R3.cpp are dependent on the grade class and database.cpp is dependent on R7.cpp, R1.cpp, R6.cpp, R2.cpp, and R3.cpp. Finally main.cpp only calls menu(), which implements all of this.

**Input Requirements:**

* Menu inputs:
  + To select a menu item: int 1-6
    - All incorrect int inputs or non int inputs are caught
    - Input value is not limited
  + To open and write files: string of any size
    - All incorrect string inputs for file opening is caught
    - Expected to be in .csv format
  + To add grade items: int and string of any size
    - For Grade and Max grade int expected (non int caught)
    - For type, description, and date string input expected
  + To search for grade items: int between 1-2 and string of any size
    - To select type of search an int of 1 or 2 must be entered
      * 1 – date
      * 2 – description
      * Other inputs caught
    - To input value to search: string of any size
* File inputs:
  + File values returned as strings of any length
    - Type, date, description, grade, and maxgrade will be received in as string values.
    - Grade and maxgrade are converted into int after file input

**Output Requirements:**

* File output
  + Output to files is done though grade items
  + Each grade item has a type, date, description, grade, and maxgrade.
  + Type, date, and description are strings
    - Date max length: 10 chars
    - Description max length: 20 chars
    - Type max length: 12 chars
  + Grade and maxgrade are ints
    - Both are limited to 3 chars

**Problem Solution Discussion:**

Algorithm use within this program is mostly relegated to loops based upon the size of the grade vector or file size. Within R1.cpp, the read\_file() function loops though a .csv file until a blank line is read. Within the process, it adds each individual item, separated by a comma, to a grade vector. Thus the algorithm depends on the number of n items in the file, so T(n) = O(n). For R6.cpp, the function write\_file() is a loop that depends on the number of items in the grade vector. For each item it writes the values in the grade vector to a file. Thus, the algorithm depends only on the number of items n in the grade vector, so T(n) = O(n). In R7.cpp, the function print\_list() is a loop that prints each grade item within the grade vector. Thus, the algorithm depends only on the number of items n in the grade vector, so T(n) = O(n).

In R3.cpp, in search\_grades() the whole vector must be searched as all items that match search input must be printed to console. Thus, the algorithm is T(n) = O(n). In, R8.cpp, print\_summary() loops through the whole vector of grades and places scores into a int array. Thus, T(n) = O(n).

**Data Structures:**

When choosing the data structure for the storage of the grade objects, I considered both the linked list and vector data structures. The pros and cons of the link list was that adding new items would be in constant time, but it would be less convenient to implement and require more space. The pros and cons of the vector was that it would be easy to work with and finding items would be in constant time but adding items would be costly. Ultimately, I chose the vector as the data set is relatively small so searching and convenience was an adequate tradeoff for O(n) time when adding items.

**User Interface Scheme:**

* Main menu
  + Option 1: Read a file
    - Enter input file
  + Option 2: Write a file
    - Enter output file
  + Option 3: Add grade item
    - Enter grade value
    - Enter maxgrade value
    - Enter grade type
    - Enter grade description
    - Enter grade date
  + Option 4: Display grade item
    - Prints all grade items
    - Finds score for all grade items of specified type.
  + Option 5: Search grade item
    - Enter grade value to search by
      * 1. Date
      * 2. Description
    - Enter item to search
  + Option 6: Exit program

**Status of Application:**

The program complies and runs. All requirements have been implemented. No known bugs or glitches. IDE used: CLION.