EaVal
(Ease of Evaluation):
Simplifying the Grading Process

Mendoza, Roy Gabriel I.
Pangilinan, Bienvenido Jr. B.
Pasion, Al Twain D.
Pellazar, Stephen Kurt R.

Technological Institute of the Philippines
Quezon City

November 2025

Table of Contents

Introduction

In today's ever changing world, and in the academic setting, teachers or professors often rely on digital tools such as Excel or online grading systems to compute and record the students' grades. However, these tools may not be accessible at all times or to every situation that exists, such as lack, slow or limited access to the internet, lack of access to licensed software because of paywalls, or because of the data privacy or security of these softwares or sites.

In addition to all these factors, these softwares are just mostly Input, Process, and Output without considering or identifying the data and how it is arranged. Additionally, such software allows everyone to view everyone' information without regard to the owners of such information.

As such to these scenarios or factors, this project will create a coded software where a teacher or professor will be able to manage, edit, create, or remove data from his inputted data, with a lock that will enable only the professor or teacher to do such actions. In this code, the students will be able to view only his grades without being able to abuse or edit the results. Using C++ language, this code will be able to virtually run anywhere without the need of the internet, thus securing the privacy of the Professors and Students.

The Project

This project is designed to create a program using C++ that calculates, determines the ranking, and displays the results of the students' grades in a specific subject and determines if that student either passed or failed on that subject. This program begins by asking the user to input the name of the subject and asking the user for the amount of students enrolled in that subject. After providing the information of the students, such as the name, the gender and the grades of each component, the program will now process these inputs and calculate the final grades while applying the percentage of each component to the final one. Once the final grade is computed for each student, the program now will determine if the student passed or not via asking the user on what is the passing mark and the failing mark for that subject.

Beyond simple grade computation, the program will also rank the students from either highest to lowest, then sort it by gender, and finally to sort it by alphabetically. To ensure the data security of the grades, the proponents of the program will put a lock where only the professor or the teacher will know in which he or she will be the only one who will be able to manage, edit, or change the data of the students. The program will first ask who is using it, if they type the professor, the program will ask for the passcode of it to enter the editing panel, if they type students, the program will ask for the student ID, after checking the database or the list, the program will only show the data for that specific student's information.

Overall, this project provides a practical, secure and user-friendly solution for teachers to calculate and manage the grades efficiently without depending on paid or free softwares and websites. By automating the computations and the display of the output, the user of this program will be able to easily show the results of the students for that subject, without manually putting the name on the intended list.

Objectives

The objective of this project is to create a code where users such as professors and teachers will be able to input and compute the grades of their students and manage the data that they gain or have. This project also ensures that the students' data will not leak anywhere by making the project not required to be connected to the internet or be in a paywall before being able to be accessed.

This project also contains Two specific objectives, and they are:

1. This project's goal is to create a code where everyone will be able to run it freely locally on their own without the reliance on the internet or to a paid software.
2. This project's goal is to create a program that allows professors and teachers to manage, edit and calculate their students' grades and have a secured program where only they would be able to edit the data, and only allows the students actions to be just to view their individual grades and not change any of it.

Flowchart of the System

The flowchart shows the logical process of a grading system program in c++ that allows both professors and students to interact with the program. The program begins with the user selecting their role whether they are the **Professor**, **Student**, or **Quit/close the program.**.

If the user selects **Student (S)**, they are prompted to enter their Student ID. If the ID is not found, the system displays an error message and returns to the main menu. If the ID is found, the student is directed to a menu with options to either view their grades or log out. Selecting "View Grades" displays their grades in a table format before returning to the main menu, while "Log Out" simply ends the student session and returns to the main program.

If the user selects **Professor (P)**, they are taken to the Professor Menu, however they must first enter the passcode in which they will be directed to the five main options unique only to them, and these options are:

1. **Add Student** – The professor inputs student information such as name, gender, quiz, recitation, and exam scores. The system then calculates and stores the grades using the student's ID before returning to the main menu of the program.

2. **Edit Student** – The professor inputs a Student ID. If found, they can update quiz, recitation, or exam scores, after which the system recalculates the grades and returns to the main program.

3. **Delete Student** – The professor enters the Student ID. If found, the system asks for confirmation. If the professor confirms, the student's data will be cleared, and the program returns to the main menu.

4. **View Student** – The professor can choose to sort the list of students by ascending grades, descending grades, passing status, or alphabetically. The program then displays the sorted list in a table format to the output terminal before returning to the main program.

5. **Log Out** – Lets the professor exit back to the professor's menu or to the main program.

If the user selects **Quit (Q)** from the start, the program terminates.

Overall, the flowchart shows a structured and user-friendly process that manages student records efficiently while providing both access control and data management for professors and students.

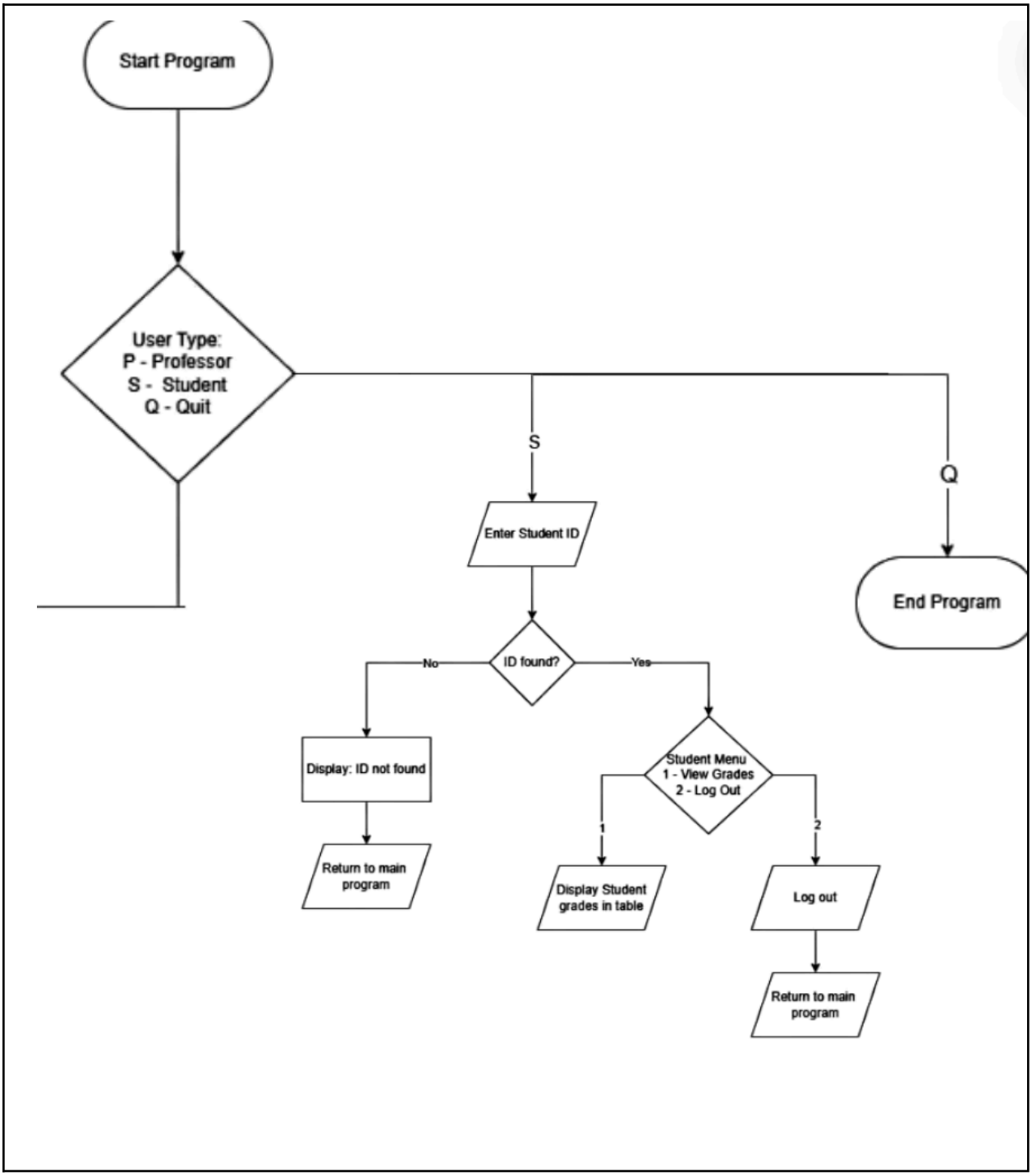

Figure 1: Program Start and Student Menu and Closing of Program

Figure 1 illustrates the flow of the program from the start, focusing on the **Student Menu** and the process for closing the program. The program begins with the **Start Program** step, where the user is asked to identify their type: **Professor (P)**, **Student (S)**, or **Quit (Q)**.

If the user selects **Q**, the program immediately ends. If the user selects **S**, the system prompts for the **Student ID**. The program then checks whether the entered ID exists in the database.

- If the **ID is not found**, the system displays the message "ID not found" and returns to the main program.
- If the **ID is found**, the student gains access to the **Student Menu**, which contains two options:

    1. **View Grades** – Displays the student's grades in a table format, then returns to the main program.
    2. **Log Out** – Logs the student out and returns to the main program.

This flowchart shows a simple and logical process that ensures proper user identification and smooth navigation for students while maintaining an option to exit the program anytime.

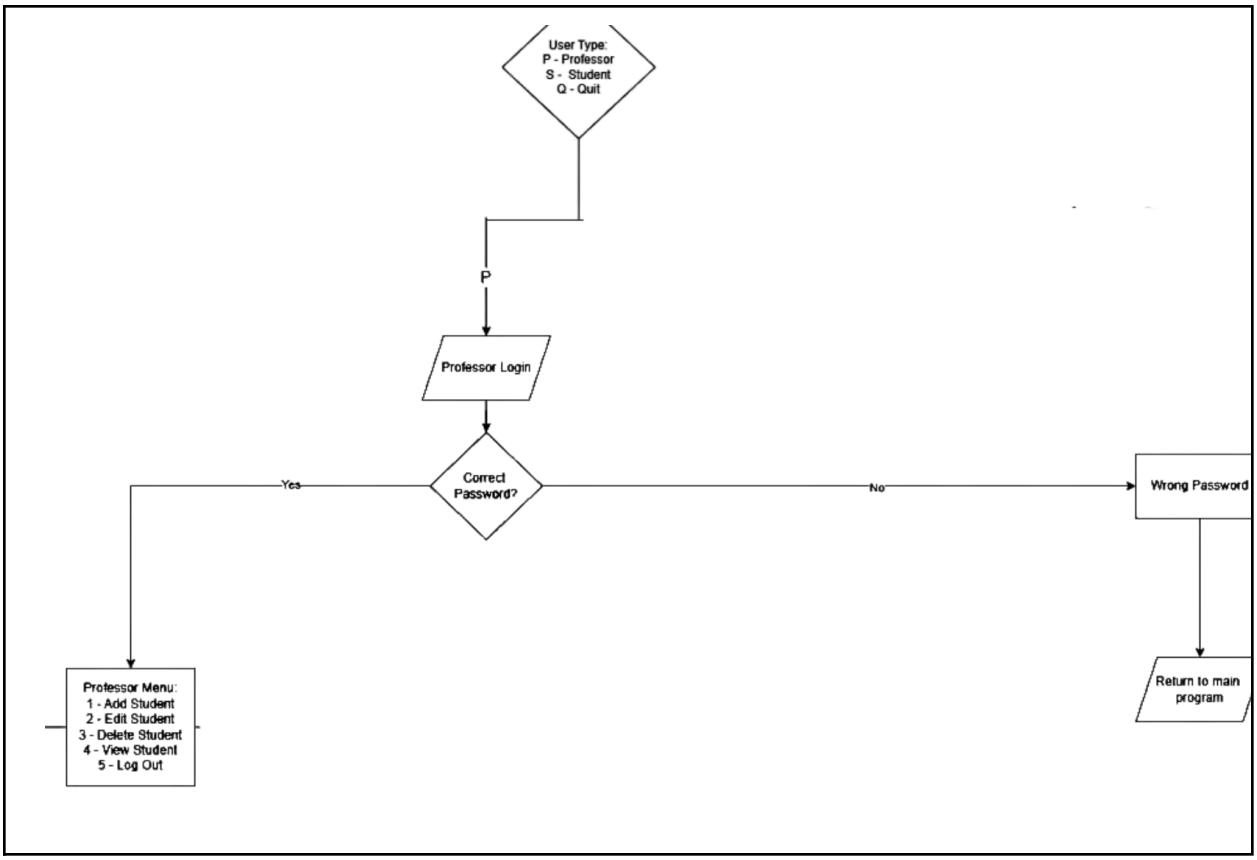

Figure 2: Professor Login and Menu

Figure 2 illustrates the overall flow of the grading system program from the start until the Professor Menu stage. The process for this figure begins with the user selection process, where the user is prompted to choose a user type: Professor (P), Student (S), or Quit (Q).

If the user selects P, the program proceeds to the Professor Menu, which provides several key functions for managing student records. However the professor must first enter the passcode first before they begin accessing the

professor menu. Under the professor menu, there are several functions, and these include adding new students, editing existing student data, deleting students, viewing all student records, and logging out. Each option leads to a specific set of actions such as inputting student information, recalculating grades, confirming deletions, or sorting displayed student data.

This section of the flowchart emphasizes the professor's control over the system, allowing for efficient data management, grade calculation, and record updates while maintaining clear navigation and logical program flow.

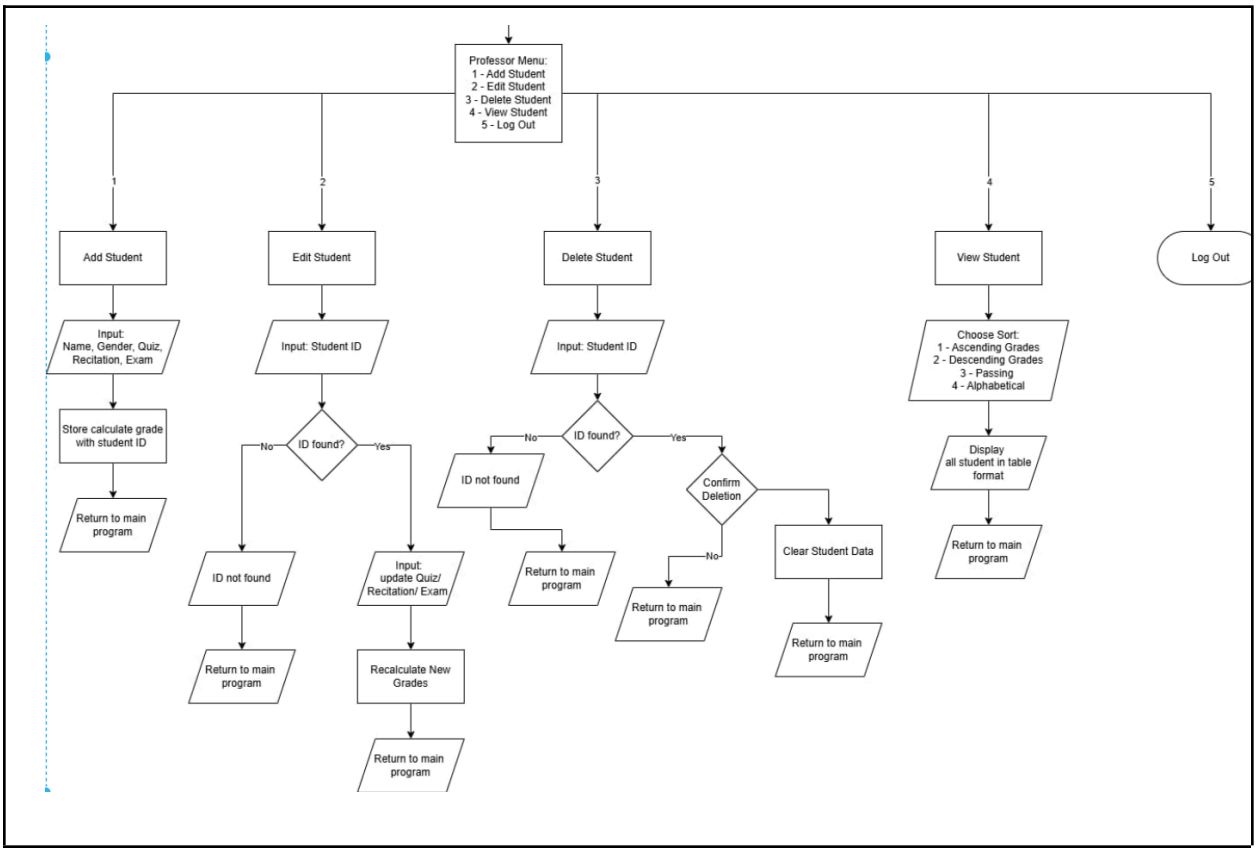

Figure 3: Professor Menu and Functions

Figure 3 illustrates the Professor Menu and functions section of the flowchart which shows how professors manage student records within the program. Once the professor selects their role, the system displays five main options: Add Student, Edit Student, Delete Student, View Student, and Log Out.

Add Student – The professor inputs the student's name, gender, and scores for quizzes, recitations, and exams. The system then calculates the final grade, stores the information with the assigned student ID, and returns to the main program.

Edit Student – The professor enters a Student ID. If the ID is found, the professor can update quiz, recitation, or exam scores. The system recalculates the grades and saves the new data before returning to the main program. If the ID is not found, a message is displayed, and the program returns to the main menu.

Delete Student – The professor inputs the Student ID. If found, the system requests confirmation. Once confirmed, the student's data is deleted, and the system returns to the main program.

View Student – The professor chooses how to sort the student records: by ascending grades, descending grades, passing status, or alphabetical order. The system displays the results in a table format, then returns to the main program.

Log Out – Ends the professor's session and returns to the main program.

This portion of the flowchart highlights how the system enables professors to efficiently add, modify, organize, and remove student data, maintaining a clear and structured process until the program concludes.

Pseudocode

The pseudocode provides a structured outline of the Student Grade Management System's logical flow and main operations. It describes how the program responds to user input, processes data, and performs actions without using specific programming syntax. This section helps the readers in understanding the step-by-step logic of the system before implementing it in C++. It begins with the user selecting a role, it could be a professor, a student, or quit the program which will terminate all process, and proceeds with the corresponding operations such as adding, editing, deleting, and viewing student records for professors, and viewing grades for students. The pseudocode ensures clarity in the program's design by illustrating the logical decisions, loops, and data manipulations that occur during execution.

```
Start

DISPLAY "Enter Type: S - Student | P - Professor | Q - Quit"
  INPUT userType

  IF userType = 'P'
     THEN
     DISPLAY "Professor Login"
     INPUT password

     IF password = correctPassword
        THEN
        DISPLAY "Professor Menu"
        DISPLAY "1 - Add Student"
        DISPLAY "2 - Edit Student"
        DISPLAY "3 - Delete Student"
        DISPLAY "4 - View Student"
        DISPLAY "5 - Log Out"
        INPUT choice

     SWITCH choice
```

```
CASE 1:
  DISPLAY "Add Student"
   INPUT name, quiz, recitation, exam
   CALCULATE grade = (quiz + recitation + exam) / 3
   STORE student record with grade
   DISPLAY "Student Added Successfully"
   RETURN to main program

 CASE 2:
   DISPLAY "Edit Student"
    INPUT studentID
      IF studentID found
      THEN
         INPUT new quiz, recitation, exam
         RECALCULATE grade
         UPDATE record
         DISPLAY "Record Updated"
      ELSE
         DISPLAY "ID not found"
      ENDIF
      RETURN to main program

CASE 3:
    DISPLAY "Delete Student"
    INPUT studentID
    IF studentID found
      THEN
      CONFIRM deletion
      IF confirmed
        THEN
        DELETE student record
        DISPLAY "Record Deleted"
          ENDIF
        ELSE
          DISPLAY "ID not found"
        ENDIF
       RETURN to main program

   CASE 4:
      DISPLAY "View Students"
      DISPLAY "Choose Sort:"
      DISPLAY "1 - Descending Grades"
      DISPLAY "2 - Ascending Grades"
      DISPLAY "3 - Passing"
      DISPLAY "4 - Alphabetical"
      INPUT sortChoice
      DISPLAY student list in table based on sortChoice
      RETURN to main program

   CASE 5:
```

```
            DISPLAY "Logging Out..."
            RETURN to main program
        ENDSWITCH
```

```
        ELSE
            DISPLAY "Wrong Password"
            RETURN to main program
        ENDIF
```

```
    ELSE IF userType = 'S'
        THEN
        DISPLAY "Enter Student ID"
        INPUT studentID

        IF studentID found
            THEN
            DISPLAY "Student Menu"
            DISPLAY "1 - View Grades"
            DISPLAY "2 - Log Out"
            INPUT studentChoice

        IF studentChoice = 1
            THEN
            DISPLAY student grades in table
            RETURN to main program
                ELSE
                DISPLAY "Logging Out..."
                RETURN to main program
            ENDIF

    ELSE
        DISPLAY "ID not found"
        RETURN to main program
    ENDIF
```

```
    ELSE IF userType = 'Q'
      THEN
      DISPLAY "End Program"
      TERMINATE PROGRAM

    ELSE
      DISPLAY "Invalid Option"
      RETURN to main program
    ENDIF

END
```

Figure 4: Pseudo code of the System

Figure 4 represents the pseudocode of the entire code for the project of EaVal (Ease of Evaluation): Simplifying the Grading Process. The pseudocode describes a simple grading system or program that is designed for professors and students. When the program starts, it prompts the user to enter their type: S for student, P for professor, or Q to quit the program. If the user selects the professor option by typing P, the program requests a password for verification. Upon entering the correct password, the professor gains access to a menu with several options such as: adding students, editing grades, deleting the student info, or viewing student records, as well as logging out. When adding a student, the professor inputs the student's name and scores for quiz, recitation, and exam, after which the program calculates the average grade and stores the record. In the edit option, the professor searches for a student using their ID, updates the grades, and recalculates the average. The delete option allows the professor to remove a student record after confirmation, while the view option displays all student records based on a chosen sorting method—such as by grade order, passing status, or alphabetically.

If the user chooses the student option, the program asks for a student ID. If the ID exists, the student can view their grades or log out. If the ID is not found, the program notifies the user and returns to the main menu. Finally, if the user enters Q, the program ends with a message indicating that it has been terminated. Any invalid input results in an error message and returns the user to the main menu.

Overall, the pseudocode outlines an interactive system that allows professors to manage student data and students to check their grades, ensuring basic access control and functionality.

Data Dictionary

This data dictionary provides a detailed description and explanation of all data elements within the program. It serves as a reference for understanding the structure of the program, and the purpose of each variables to the program. By organizing the information clearly, the data dictionary would be ensure that anyone viewing the code can easily understand and interpret the data clearly, and the process that it is used for.

The table presents the data elements used in the program, including their names, data types, sizes in bytes, and descriptions of their specific roles. It identifies key components such as the studentInfo structure, which stores student-related information, and global variables like quiz, recitation, and exam, which hold the grade weight percentages. The table also includes supporting elements like counter, course, and password, which contribute to the program's functionality. Overall, the data dictionary table summarizes how each variable contributes to managing, calculating, and displaying student records effectively.

Table 1: Data Dictionary

Data Name	Size	Data Type	Description
1. Name	32 bytes	string	Stores the student's name.
2. Gender	1	char	Stores the student's gender, can only be M or F.

3. ID	4	int	Stores the unique ID automatically assigned to each students.
4. Quiz	8	Double	Stores the student's quiz grade.
5. Recitation	8	Double	Stores the student's recitation grade.
6. Exam	8	Double	Stores the student's exam grade.
7. Final Grade	8	Double	Computed final weighted grade average grade of the student.
8. Occupancy	1	bool	Indicates whether a student slot is used.
9. Password	32	Const string	Password for the professor menu.
10. studentSize	4	Const int	Maximum number of students in the class.
11. students	-	string	Array to store up to 50 students
12. counter	4	int	Counts how many students are entered.
13. course	32 bytes	String	Name of the course that the program is being used on.
14. quiz	1	float	Stores the percentage of the quiz for the final computation.
15. recitation	1	float	Stores the percentage of the recitation for the final computation.
16. Exam	1	float	Stores the percentage of the exam for the final computation.
17. temp	-	string	Temporarily holds a student info during the sortation/edit operations.

Code

This program or code of a program is the back bone of the proccess for the student grade management system that is designed to help professors and students to calculate and compute the grades of the students, and view the results in an organized manner. This program allows two types of users: the Professor, and the Students. Professors can securely log in using the password for their menu ensuring that students would not be able to access or change grades of their own grades or others.The professor would be able to add, edit and delete existing data, basically having access to change the data inside of the program. While if the user is logged in as a student, they could only find their student ID, and see only their own grades. The program uses structured data storage through arrays adn the structure feature of the C++ to handle multiple students entries, and allows the to be stored correctly in a single session.

```
#include <iostream>
#include <windows.h>
#include <iomanip>
using namespace std;

//structure for student info
struct studentInfo{
    string name;
    char gender;
    int id;
    double quiz;
    double recitation;
    double exam;
    double finalGrade;
    bool occupancy;
};

//password for professor
const string password = "professor";

//size of students
const int studentSize = 50;

//array for student info
studentInfo students[studentSize];

//counter for inputted students
int counter = 0;

//weight grades
float quiz = 0;
float recitation = 0;
float exam;

//function for getting weighted grades
```

```
void grades();


//main menu
void userType();

//two types of users: "professor" and "student"

//picks professor ---> professorUser --> professorMenu --> add/edit/delete/view(ascending, descending,
passing||failing, alphabetical)/logout
bool professorUser();
   void professorMenu();
      void add();
      void edit();
      void del();
      void view();
         void display();
         void asc();
         void dec();
         void passfail();
         void alpha();
      void quit();



//picks student --> studentUser --> studentMenu --> view/logout
int studentUser();
   void studentMenu(int id);
      void studentView(int id);

//for clearning console screen
void clear();

//course name
string course;

int main(){

   cout << "********************" << endl;
   cout << "*                  *" << endl;
   cout << "*   EAVAL INC      *" << endl;
   cout << "*                  *" << endl;
   cout << "********************" << endl;


   //coursename
   cout << "Please enter course code/name. (This is permanent. Restart program to change.): ";
   getline(cin, course);

   //confirmation
```

```
   while (true){

            cout << endl << "Is this final? (Y/N): ";
            char choice;
            cin >> choice;
            cin.ignore();
            choice = toupper(choice);

            if (choice == 'Y'){
                        break;
                }
                else if (choice == 'N'){
                        cout << "Please re-enter course name: ";
                        getline(cin, course);
                }
                else {
                        cout << "Invalid Input." << endl;
                }
            }

   grades();

   //student size warning
   cout << endl << "The amount of students to be entered is limited to " << studentSize << endl << endl;

   //clears garbage characters
   for (int i = 0; i < studentSize; i++){
      students[i].name = " ";
      students[i].gender = ' ';
      students[i].id = 0;
      students[i].quiz = 0;
      students[i].recitation = 0;
      students[i].exam = 0;
      students[i].finalGrade = 0;
      students[i].occupancy = false;
   }

   userType();

}


//gets weighted grade
void grades(){

            cout << endl << "Now please enter the weighted grades for Quizzes, Recitation, and Exam. (This is
permanent. Restart program to change.)" << endl << endl;

            cout << "Quiz Weight in percentage (ex. 50): ";
            cin >> quiz;
```

```
            cout << "Recitation Weight in percentage (ex. 50): ";
            cin >> recitation;
            cout << "Exam Weight in percentage (ex. 50): ";
            cin >> exam;

            while (quiz + recitation + exam != 100){

                        cout << endl << "The given weighted grade exceeded or did not reach 100%. Please re-enter."
<< endl;

                        cout << "Quiz Weight in percentage (ex. 50): ";
                        cin >> quiz;
                        cout << "Recitation Weight in percentage (ex. 50): ";
                        cin >> recitation;
                        cout << "Exam Weight in percentage (ex. 50): ";
                        cin >> exam;

            }

            cout << endl;
            cout << left << setw(30) << "Quiz Percentage: " << quiz << "%" << endl;
            cout << left << setw(30) << "Recitation Percentage: " << recitation << "%" << endl;
            cout << left << setw(30) << "Exam Percentage: " << exam << "%" << endl;
            cout << left << setw(30) << "Final Grade: " << 100 << endl;

            quiz = quiz / 100;
            recitation = recitation / 100;
            exam = exam / 100;


}

//pick your user or quit
void userType(){
   //controlled loop sentinel
   while(true){
   //asks for userType
      cout << "Log in as Professor or Student or Quit? (P/S/Q): ";
      char userType;
      cin >> userType;
      userType = toupper(userType);

      if (userType == 'P'){
         if (professorUser()){
            clear();
            professorMenu(); break;
         }
         else {
            cout << "Wrong Password." << endl;
         }
```

```
        }

        else if (userType == 'S'){
            int entry = studentUser();
            bool verify = false;
            for (int i = 0; i < studentSize; i++){
                if (students[i].id == entry){
                    verify = true;
                    break;
                }
            }
            if (verify){
                clear();
                studentMenu(entry);
            }
            else {
                cout << "Student ID not found." << endl;
            }
        }

        else if (userType == 'Q'){
            cout << "Thank you for using the program." << endl;
            quit(); abort(); break;
        }

        else {
            cout << "Invalid Input." << endl;
        }
    }
}

//professor route part 1
bool professorUser(){
    cout << "Enter Password: ";
    string entry;
    cin.ignore();
    getline(cin, entry);
    return entry == password;
}


//professor route part 2
void professorMenu(){


    while(true){

        cout << endl << "Professor Menu" << endl;
        cout << "(1) - Add student" << endl;
        cout << "(2) - Edit student's Grade" << endl;
```

```
        cout << "(3) - Delete student Info" << endl;
        cout << "(4) - View student" << endl;
        cout << "(5) - Log-out" << endl;
        cout << "Choose action: ";
        int choice;
        cin >> choice;

        switch(choice){
            case 1: add(); break;
            case 2: edit(); break;
            case 3: del(); break;
            case 4: view(); break;
            case 5: clear(); userType(); break;
            default: cout << "Invalid Action."; break;
        }
    }
}

//professor route 3.1
void add(){

    clear();

    if (counter == studentSize){
        cout << "You have entered the maximum amount of students.";
        return;
    }

    //declare a temporary variable
    studentInfo temp;

    //asks for student name
    cout << "Enter Student Name (Surname, First Name M.I): ";
    cin.ignore();
    getline(cin, temp.name);

    //asks for gender
    while (true){
        cout << "Enter Student's Gender (M/F): ";
        cin >> temp.gender;
        temp.gender = toupper(temp.gender);

        if (temp.gender == 'M' || temp.gender == 'F'){
            break;
        }
        else {
            cout << "Invalid Input." << endl;
        }
    }
```

```
//asks for overall quiz grade
while(true) {
    cout << "Enter Student's Quiz Grade (0 - 100): ";
    cin >> temp.quiz;

    if (temp.quiz > -1 && temp.quiz < 101){
        break;
    }
    else {
        cout << "Invalid Input." << endl;
    }
}

//asks for overall recitation grade
while(true) {
    cout << "Enter Student's Recitation Grade (0 - 100): ";
    cin >> temp.recitation;

    if (temp.recitation > -1 && temp.recitation < 101){
        break;
    }
    else {
        cout << "Invalid Input." << endl;
    }
}

//asks for overall exam grade
while(true) {
    cout << "Enter Student's Exam Grade (0 - 100): ";
    cin >> temp.exam;

    if (temp.exam > -1 && temp.exam < 101){
        break;
    }
    else {
        cout << "Invalid Input." << endl;
    }
}

//calculates the resulting grade
temp.finalGrade = ((temp.quiz * quiz) + (temp.recitation * recitation) + (temp.exam * exam));


//finds a vacant spot in the array
int index = 0;
for (int i = 0; i < studentSize; i++){
    if (students[i].occupancy == false){
        students[i] = temp;
        students[i].id = i + 1;
        students[i].occupancy = true;
```

```
            counter = counter + 1;
            index = i;
            break;
        }
    }

    clear();

    cout << endl;
    cout << "Student's Name: " << students[index].name << endl;
    cout << "Student's Gender: " << students[index].gender << endl;
    cout << "Student's ID: " << students[index].id << endl;
    cout << "Student's Quiz Grade: " << students[index].quiz << endl;
    cout << "Student's Recitation Grade: " << students[index].recitation << endl;
    cout << "Student's Exam Grade: " << students[index].exam << endl;
    cout << "Student's Final grade: " << students[index].finalGrade << endl;
    cout << "Array Occupancy: " << students[index].occupancy << endl;
    cout << "Number of Students Entered: " << counter << endl;
}

//professor route 3.2
void edit(){

    clear();

    //asks for student id
    cout << "To edit, enter Student's ID: ";
    int id = 0;
    cin >> id;

    //finds the student id
    int index = -1;
    for (int i = 0; i < studentSize; i++){
        if (students[i].id == id){
            index = i;
            break;
        }
    }
    if (index == -1) {
        cout << "Student ID not found." << endl;
        professorMenu();
    }

    //declare temp variable
    studentInfo temp;

    cout << "Currently Editing " << students[index].name << "." << endl;

    //quiz grade edit
    while (true){
```

```
        cout << "Current Quiz Grade: "  << students[index].quiz << "." << endl << " Enter new Grade (0-100) or Enter
-1 to keep: ";
        cin >> temp.quiz;
        if (temp.quiz > -1){
            students[index].quiz = temp.quiz;
            cout << "Quiz Grade Successfully Changed." << endl;
            break;
        }
        else if (temp.quiz == -1){
            cout << "Quiz Grade Kept." << endl;
            break;
        }

    }

    //recitation grade edit
    while (true){
        cout << "Current Recitation Grade: "  << students[index].recitation << "." << endl << " Enter new Grade
(0-100) or Enter -1 to keep: ";
        cin >> temp.recitation;
        if (temp.recitation > -1){
            students[index].recitation = temp.recitation;
            cout << "Recitation Grade Successfully Changed." << endl;
            break;
        }
        else if (temp.quiz == -1){
            cout << "Recitation Grade Kept." << endl;
            break;
        }
        else {
            cout << "Invalid Input." << endl;
        }

    }

    //exam grade edit
    while (true){
        cout << "Current Exam Grade: "  << students[index].exam << "." << endl << "Enter new Grade (0-100) or
Enter -1 to keep: ";
        cin >> temp.exam;
        if (temp.exam > -1){
            students[index].exam = temp.exam;
            cout << "Exam Grade Successfully Changed." << endl;
            break;
        }
        else if (temp.exam == -1){
            cout << "Exam Grade Kept." << endl;
            break;
        }
        else {
```

```
            cout << "Invalid Input." << endl;
        }
    }

    //computes final grade
    students[index].finalGrade = ((students[index].quiz * quiz) + (students[index].recitation * recitation) +
(students[index].exam * exam));

    clear();

    cout << students[index].name << "'s current grade." << endl;
    cout << endl;
    cout << "Student's Quiz Grade: " << students[index].quiz << endl;
    cout << "Student's Recitation Grade: " << students[index].recitation << endl;
    cout << "Student's Exam Grade: " << students[index].exam << endl;
    cout << "Student's Final grade: " << students[index].finalGrade << endl;

    professorMenu();
}

//professor route 3.3
void del(){
    //enter student id to delete
    cout << "To delete student's info, enter ID: ";
    int id = 0;
    cin >> id;

    //find student id
    int index = -1;
    for (int i = 0; i < studentSize; i++){
        if (students[i].id == id){
            index = i;
            break;
        }
    }
    if (index == -1) {
        cout << "Student ID not found." << endl;
        professorMenu();
    }


    while (true){
        cout << "Are you sure you want do delete " << students[index].name << "'s info? (Y/N): ";
        char choice;
        cin >> choice;
        choice = toupper(choice);

        if (choice == 'Y'){
                                    //deletion of info
            students[index].name = " ";
```

```
            students[index].gender = ' ';
            students[index].id = 0;
            students[index].quiz = 0;
            students[index].recitation = 0;
            students[index].exam = 0;
            students[index].finalGrade = 0;
            students[index].occupancy = false;

            counter = counter - 1;
            cout << "Student Info succesfully deleted." << endl;
            clear();
            professorMenu(); break;
        }

        else if (choice == 'N') {
            cout << "Did not delete student info." << endl;
            professorMenu(); break;
        }

        else {
            cout << "Invalid Input." << endl;
        }
    }
}

//professor route 3.4
void view(){

    clear();

    while(true){
        cout << endl;
        cout << "View sorted by.." << endl;
        cout << "(1) - Ascending Grades" << endl;
        cout << "(2) - Descending Grades" << endl;
        cout << "(3) - Passing -> Failing" << endl;
        cout << "(4) - Alphabetical" << endl;
        cout << "(5) - Return to Professor Menu" << endl;
        cout << "Choose Aciton: ";
        int entry = 0;
        cin >> entry;

        switch(entry){
            case 1: asc(); break;
            case 2: dec(); break;
            case 3: passfail(); break;
            case 4: alpha(); break;
            case 5: clear(); professorMenu(); break;
            default: cout << "Invalid Input."; break;
        }
```

```
        break;
    }

    display();

    while (true){
        cout << "Enter -1 to go back: ";
        int choice = 0;
        cin >> choice;

        if (choice == -1){
            clear();
            professorMenu(); break;
        }
        else {
            cout << "Invalid Input." << endl;
        }
    }
}

//professor route 3.4.1
//bubble sort but ascending
void asc(){
    for (int i = 0; i < studentSize - 1; i++) {
        for (int j = 0; j < studentSize - i - 1; j++) {
            if (students[j].finalGrade > students[j + 1].finalGrade) {
                studentInfo temp = students[j];
                students[j] = students[j + 1];
                students[j + 1]= temp;
            }
        }
    }
}

//professor route 3.4.2
//bubble sort the descending way
void dec(){
    for (int i = 0; i < studentSize - 1; i++) {
        for (int j = 0; j < studentSize - i - 1; j++) {
            if (students[j].finalGrade < students[j + 1].finalGrade) {
                studentInfo temp = students[j];
                students[j] = students[j + 1];
                students[j + 1]= temp;
            }
        }
    }
}

//professor route 3.4.3
//sorts by passing or failing
```

```
void passfail(){

   dec();

   cout << endl;
   cout << "                                    " << course << "                                    " << endl << endl;
   cout << left << setw(45) << "Name" << setw(15) << "Final Grade" << setw(12) << "Status"  << endl;
   cout << "-----------------------------------------------------------------------------------------------------" << endl;
   for (int i = 0; i < studentSize; i++){
      if (students[i].finalGrade >=75 && students[i].occupancy == true){
         cout << left << setw(45) << students[i].name << setw(15) << students[i].finalGrade << setw(12) <<
"Passed" << endl;
      }
   }

   for (int i = 0; i < studentSize; i++){
      if (students[i].finalGrade < 75 && students[i].occupancy == true){
         cout << left << setw(45) << students[i].name << setw(15) << students[i].finalGrade << setw(12) << "Failed"
<< endl;
      }
   }

   cout << "\n\t\t\tEnd of List...\t\t\t\n";
   cout << "-----------------------------------------------------------------------------------------------------" << endl;

   cout << "Weighted Grades";

   cout << endl;
            cout << left << setw(30) << "Quiz Percentage: " << quiz * 100 << "%" << endl;
            cout << left << setw(30) << "Recitation Percentage: " << recitation * 100 << "%" << endl;
            cout << left << setw(30) << "Exam Percentage: " << exam * 100 << "%" << endl;
            cout << left << setw(30) << "Final Grade: " << 100 << endl;
            cout << endl;

   while (true){
      cout << "Enter -1 to go back: ";
      int choice = 0;
      cin >> choice;

      if (choice == -1){
         clear(); professorMenu(); break;
      }
      else {
         cout << "Invalid Input." << endl;
      }
   }


}
```

```
//sorts by alphabetical
//bubblesorting but for string
void alpha(){
    for (int i = 0; i < studentSize - 1; i++){
        for (int j = 0; j < studentSize - i - 1; j++){
            if (students[j].occupancy && students[j+1].occupancy) {
                if (students[j].name > students[j+1].name) {
                    studentInfo temp = students[j];
                    students[j] = students[j+1];
                    students[j+1] = temp;
                }
            }
        }
    }
}

//professor route 4
//to display the array
void display(){
    //counter to confirm occupied values in the array
    int amount = 0;

    cout << endl << endl;
    cout << "                                    " << course << "                                    " << endl << endl;
    cout << left << setw(45) << "Name" << setw(20) << "Student ID" << setw(10) << "Quiz" << setw(15) <<
"Recitation" << setw(10) << "Exam" << setw(25) << "Final Grade" << endl;
    cout << "-----------------------------------------------------------------------------------------------------" << endl;
    for (int i = 0; i < studentSize && amount < counter; i++){
        if (students[i].occupancy){
            cout << left << setw(45) << students[i].name << setw(20) << students[i].id << setw(10) << students[i].quiz
<< setw(15) << students[i].recitation << setw(10) << students[i].exam << setw(25) << students[i].finalGrade <<
endl;
            amount = amount + 1;
        }
    }
    cout << "\n\t\t\tEnd of List...\t\t\t\n";
    cout << "-----------------------------------------------------------------------------------------------------" << endl;

    cout << "Weighted Grades";

    cout << endl;
            cout << left << setw(30) << "Quiz Percentage: " << quiz * 100<< "%" << endl;
            cout << left << setw(30) << "Recitation Percentage: " << recitation * 100 << "%" << endl;
            cout << left << setw(30) << "Exam Percentage: " << exam * 100 << "%" << endl;
            cout << left << setw(30) << "Final Grade: " << 100 << endl;
            cout << endl;

}

//student route 1
```

```
int studentUser(){
    cout << "Enter Student ID: ";
    int id;
    cin >> id;
    return id;
}

//student route 2
void studentMenu(int id){

    //find student id
    int index = -1;
    for (int i = 0; i < studentSize; i++){
        if (students[i].id == id){
            index = i;
            break;
        }
    }
    if (index == -1) {
        cout << "Student ID not found." << endl;
        return;
    }

    while(true){
        cout << endl << "Hello, " << students[index].name << endl;
        cout << endl << "Student Menu" << endl;
        cout << "(1) - View Grades" << endl;
        cout << "(2) - Log-Out" << endl;
        cout << "Choose action: ";
        int choice;
        cin >> choice;
        switch(choice){
        case 1: studentView(index); break;
        case 2: clear(); userType(); break;
        default: cout << "Invalid Action."; break;
        }
    }
}

//student route 3
void studentView(int id){
    cout << endl << endl;
    cout << "                                " << course << "                                " << endl << endl;
    cout << left << setw(45) << "Name" << setw(20) << "Student ID" << setw(10) << "Quiz" << setw(15) <<
"Recitation" << setw(10) << "Exam" << setw(25) << "Final Grade" << endl;
    cout << "-----------------------------------------------------------------------------------------------------" << endl;
    cout << left << setw(45) << students[id].name << setw(20) << students[id].id << setw(10) << students[id].quiz <<
setw(15) << students[id].recitation << setw(10) << students[id].exam << setw(25) << students[id].finalGrade <<
endl;
    cout << "\n\t\t\t\tEnd of List...\t\t\t\n";
```

```
    cout << "-----------------------------------------------------------------------------------------------------" << endl;


    cout << "Weighted Grades";

    cout << endl;
            cout << left << setw(30) << "Quiz Percentage: " << quiz * 100<< "%" << endl;
            cout << left << setw(30) << "Recitation Percentage: " << recitation * 100 << "%" << endl;
            cout << left << setw(30) << "Exam Percentage: " << exam * 100 << "%" << endl;
            cout << left << setw(30) << "Final Grade: " << 100 << endl;
            cout << endl;

    while (true){
        cout << "Enter -1 to go back: ";
        int choice = 0;
        cin >> choice;

        if (choice == -1){
            clear(); return; break;
        }
        else {
            cout << "Invalid Input." << endl;
        }
    }
}


//to quit the program
void quit(){

    cout << "Made by EaVal." << endl;

}
//clears the screen for cleaner output
void clear() {
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    DWORD count;
    DWORD cellCount;
    COORD homeCoords = {0, 0};

    if (hStdOut == INVALID_HANDLE_VALUE) return;

    if (!GetConsoleScreenBufferInfo(hStdOut, &csbi)) return;
    cellCount = csbi.dwSize.X * csbi.dwSize.Y;

    FillConsoleOutputCharacter(hStdOut, (TCHAR) ' ', cellCount, homeCoords, &count);
    FillConsoleOutputAttribute(hStdOut, csbi.wAttributes, cellCount, homeCoords, &count);
    SetConsoleCursorPosition(hStdOut, homeCoords);
}q
```

Figure 5. Entirety of the code for the Final Project: EaVal (Ease of Evaluation):
Simplifying the Grading Process

Code discussion

The program begins by displaying a title and asking the user to input the course name or code. After confirming the course, it proceeds to the grades function, where the user enters the percent weights for the quizzes, recitations, and exams. These values must total 100, ensuring the grading system is balanced. Once validated, the program converts these percentages into decimal form to prepare for grade computations. It then initializes all student records in the [students] array by setting default values and marking their [occupancy] as false, which indicates that no records are currently stored.

Next, the program enters the [userType()] function, which serves as the main control point for user access. The user must choose between three options: the Professor, Student, or Quit. If the user selects 'P' for professor, the program requests a password through the [professorUser()] function. A correct password grants access to the [professorMenu()], where several options are presented: add, edit, delete, view, or log out. The [add()] function allows professors to input student details, including name, gender, and grades. The program computes the final grade using the previously set weight percentages and stores the record in the [students] array. Each added student automatically receives an ID number based on their position in the array.

The [edit()] function lets professors modify an existing record by searching for the student's ID. If found, it allows the professor to update quiz, recitation, or exam grades. The program then recalculates the final grade and updates the student's record. The [del()] function enables deletion by confirming the student ID and resetting the record's data to default values, marking the slot as available again. The [view()] function allows professors to display all stored student records. It provides sorting options such as viewing in ascending or descending order, showing only passing students, or listing alphabetically by name. Sorting is performed using a simple bubble sort algorithm, which repeatedly swaps records based on the selected criteria.

If the user selects 'S' for student, the [studentUser()] function is called. The program asks for the student's ID and, if valid, opens a student menu that allows viewing grades or logging out. When a student chooses to view grades, the program displays a table showing their quiz, recitation, and exam scores, along with the computed final grade. Finally, if the user selects 'Q' from the main menu, the [quit()] function executes and the program terminates.

Overall, the code uses loops, conditionals, and user-defined functions to manage data effectively. It relies on a structured approach using the [studentInfo] struct to group related information together. The flow of the program is modular, meaning each function handles a specific tasks, like adding, editing, or viewing students information, making the logic clear and easy to follow.

Results and Discussion

Subsection Introduction:

This section presents the results obtained from running the Student Grade Management System program. It includes the outputs generated during execution and a discussion of how the program performs its intended tasks, including data input, processing, and display. The results demonstrate how effectively the program manages student records and computes final grades based on the set criteria.

Results Discussion:

Upon execution, the program first displays a title banner, or the name of the program, and it then prompts the user to enter the course name and confirm it. It then asks for the weight percentages of quizzes, recitations, and exams, verifying that their total equals 100%. Once confirmed, the system initializes all student data and presents the main menu, where users can choose their role: Professor, Student, or Quit.

When the user selects the Professor option and enters the correct password, the program displays the Professor Menu. Selecting “Add Student” prompts the user to input the student’s name, gender, and scores for quizzes, recitations, and exams. The program calculates the final grade automatically based on the given weights and stores the student’s information in memory. A confirmation message such as “Student Added Successfully” is displayed, followed by a formatted table showing the student’s details and computed grade.

Choosing “Edit Student” allows modification of existing student grades. If the entered ID exists, the program prompts for new grade values, recalculates the final grade, and displays “Record Updated.” The “Delete Student” option confirms deletion before removing the record, ensuring accuraScy and avoiding accidental deletions. The “View Students” feature presents all student data in a table format, organized based on sorting preferences—ascending, descending, alphabetical, or passing grades. Each view presents neatly aligned columns for ID, name, gender, and grades, making it easy to review performance.

If the user selects the Student option, the program asks for a student ID. Upon finding a match, it displays the Student Menu with options to “View Grades” or “Log Out.” Choosing “View Grades” shows the individual student’s quiz, recitation, exam, and final grade in a clear table layout. The program returns to the main menu after each operation, maintaining smooth and organized navigation. When “Quit” is selected, the system displays “End Program” and terminates properly.

The results show that the program executes successfully, and performs accurate grade computation. Its logical structure allows users to manage data easily, while its use of conditionals and loops ensures that invalid inputs or missing data are properly handled. Sorting and display functions work as intended, giving clear and organized results. Overall, the program functions as an effective tool for managing and viewing student grades.

Conclusion

The Student Grade Management System successfully performs all its intended functions, including adding, editing, deleting, and viewing student records. The program effectively computes final grades based on user-defined weight percentages, ensuring flexibility across different grading systems. Its structured approach, using functions and conditional statements, allows for organized and user-friendly operation. The results confirm that the program is reliable for managing small-scale student data within a class or department setting.

Recommendations:

This project is completed and shows the application of learning that the students has managed to learn throughout the first semester of the course: Programming Logic and Design. However, due to limitations or the lack of information regarding to some part of the code to make it more applicable or more distinguished, the group that developed this code or program recommends that others who will use this as a reference will learn or add a function that will let the data be stored offline or when the program is terminated, the inputted values or data from the previous session will be stored somewhere.

Another is that, when the data for the quizzes, recitations and exams, the user will be asked first on how many of these components are there per component, like ask the user on how many quizzes exist, how many recitations are there, or how many exams are there, before calculating the whole final grade. The group also ask or try for the future developer for this code to try to make a Graphics User Interface for this project to make it more pleasing to the eyes of future user. Also, due to the lack of sortation with the gender of the student, we recommend to the future programmer/s to use the gender variable to sort the student by alphabetically and gender based.

References

Cain, J., Medina, M., Romanelli, F., & Persky, A. (2022). Deficiencies of Traditional Grading Systems and Recommendations for the Future. *American Journal of Pharmaceutical Education*, *86*(7), 8850. https://doi.org/10.5688/ajpe8850

Decker, A., Edwards, S. H., McSkimming, B. M., Edmison, B., Rorrer, A., & Pérez, M. A. (2024). *Transforming Grading Practices in the Computing Education Community*. https://doi.org/10.1145/3626252.3630953

Kreinovich, Kosheleva, O., & Servin, C. (2021). *How To Guarantee Fairness Of Grading Without Sacrificing Privacy?* ScholarWorks@UTEP. https://scholarworks.utep.edu/cs_techrep/1541/

Kumar, R. (2023). Faculty members' use of artificial intelligence to grade student papers: a case of implications. *International Journal for Educational Integrity*, *19*(1). https://doi.org/10.1007/s40979-023-00130-7

Morris, S. R., & Townsley, M. (2025). Traditional vs. Modern-Grading Practices: A Comparative Case Study. *Journal of Educational Leadership in Action*, *9*(3). https://doi.org/10.62608/2164-1102.1169

Sinan II, Tong VVT, Nwoacha V, et al. (2024). Enhancing security and privacy in educational environments: A secure grade distribution scheme with Moodle integration. Journal of Infrastructure, Policy and Development. 8(7): 3737. https://doi.org/10.24294/jipd.v8i7.3737

Tisha, S. M., Oregon, R. A., Baumgartner, G., Alegre, F., & Moreno, J. (2022, May 1). *An Automatic Grading System for a High School-level Computational Thinking Course*. IEEE Xplore. https://doi.org/10.1145/3528231.3528357