The National Teachers College
629 Nepomuceno St., Quiapo, Manila, 1001 Metro Manila

“Student Grade Management System”

A Documentation Paper
Presented to **Ms. Paula M. De Guzman**
National Teachers College
Quiapo, Manila

In Partial Fulfillment
of the Requirements for the Subject
Introduction to Computer Programming 2

by:

Esteban, Maria Victoria N.

June 26, 2025

TABLE OF CONTENTS

Introduction

1.1 Overview

The Student Grade Management System (SGMS) is a software solution designed to manage the grades of their students, created using Turbo C++ software. The system is developed using the C++ programming language and leverages advanced C++ concepts like data structures and functions to manage student records, student registration, and grade entry. The system avoids using file handling to keep it simple and focused on core functionalities.

1.2 Project description

Student Grade Management System (SGMS) key features include student registration, grade entry, grade calculation, student search, grade reports, sorting functions, and data validation .The system uses structured data models to represent students, and grades. These data models are defined using struct in C++. Student Grade Management System (SGMS) core functionalities can register students, input and calculate grades, can search students, and display grade reports.

1.3 Purpose and scope of the system

Student Grade Management Systems (SGMS) are powerful, flexible, and easy to use and are designed and developed to deliver real conceivable benefits to schools. The main purpose of developing SGSM is to help Schools to be more efficient in registration of their students and manage students' grade records. The system will be used as the application that serves schools or other educational institutions. The intention of the system is to help school administration to manage the grade file records of their students properly. If the School management system is file based, management of the school has to put much effort on securing the files. They can be easily damaged by fire, insects and natural disasters. Also could be misplaced by losing data and information.

System Objectives

The main objectives of a Student Grade Management Systems (SGMS) is to efficiently manage students records and ensure accurate and updated records. Other objectives of SGMS is to design a system for better file handling, and provide MIS (Management information system) report on demand to management for better decision making, and better coordination among the different departments.

System Features

The Student Grade Management System features and description are shown below:

- Student Registration
  - Add new students to the system.
  - Each student has a unique ID and name.
  - Stored in an array of structures.
- Grade Entry
  - Input grades for **5 subjects** per student.
  - Accepts only valid grades (0–100) using input validation.
  - Supports multiple students.
- Grade Calculation
  - Calculates **average marks** for each student.
  - Assigns **letter grades** based on the average:
    - A: 90–100
    - B: 80–89
    - C: 70–79
    - D: 60–69
    - F: < 60
- Student Search
  - **Linear search** by name.

    Binary search by student ID (after sorting).
  - Displays student details if found.
- Grade Reports
  - View **individual student reports** (name, ID, average, grade).

- ➢ View **all students** in a summary table format.
- ➢ Shows complete performance overview.

❖ Sorting Function

- ➢ **Bubble Sort** by name (alphabetical).
- ➢ **Selection Sort** by ID (ascending).
- ➢ **Insertion Sort** by average grade (ascending).
- ➢ Improves report readability and search performance.

❖ Data Validation

- ➢ Ensures:
  - Valid grades (0–100).
  - Menu options within valid range.
  - Correct data formats.
  - Prevents program crashes due to invalid input.

❖ Exit

- ➢ Its function is to leave the system.

❖ Menu-Driven Interface

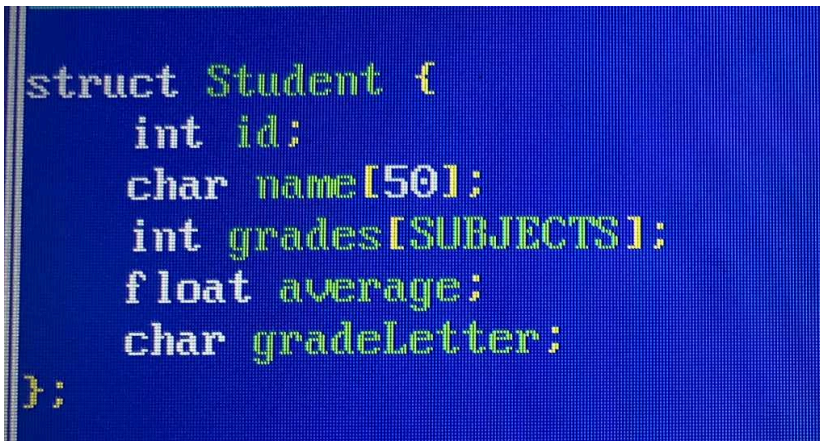
- ➢ A user-friendly menu to navigate through different options and functionalities.

System Design

The **Student Grade Management System** in Turbo C++ is designed as a modular, menu-driven console application that allows users to manage student academic records efficiently. The system is built using structured programming principles, focusing on core C++ concepts such as data types, control statements, arrays, structures, and sorting and searching algorithms. Below are the descriptions of each data structure:

1. Student Structure

The Student structure is used to store information about each student. This includes their unique ID, name, subject grades, average score, and final letter grade.

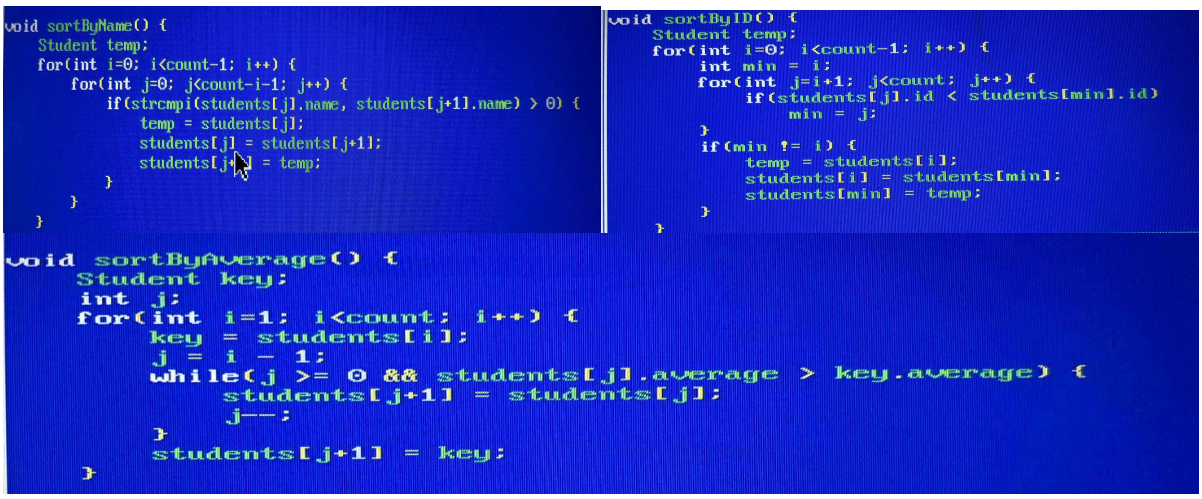


Picture 2. Struct student

- **id**: An integer that uniquely identifies each student.
- **name**: A character array (string) to store the student's name.
- **grades**: An integer array to store marks for 5 subjects.
- **average**: A float to store the calculated average of the student's grades.
- **letterGrade**: A character to store the assigned letter grade (A, B, C, D, F).

2. Sorting Module

This module provides sorting functionalities for organizing student data based on different fields.



```
void sortByName() {
    Student temp;
    for(int i=0; i<count-1; i++) {
        for(int j=0; j<count-i-1; j++) {
            if(strcmpi(students[j].name, students[j+1].name) > 0) {
                temp = students[j];
                students[j] = students[j+1];
                students[j+1] = temp;
            }
        }
    }
```

```
void sortByID() {
    Student temp;
    for(int i=0; i<count-1; i++) {
        int min = i;
        for(int j=i+1; j<count; j++) {
            if(students[j].id < students[min].id)
                min = j;
        }
        if(min != i) {
            temp = students[i];
            students[i] = students[min];
            students[min] = temp;
        }
    }
```

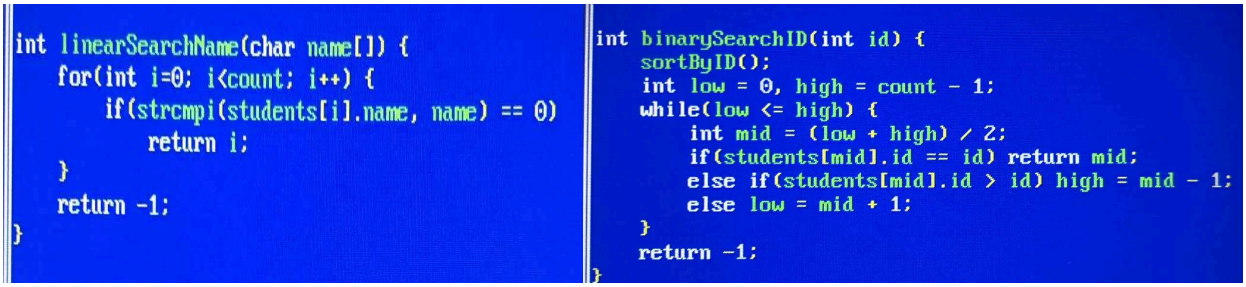
```
void sortByAverage() {
    Student key;
    int j;
    for(int i=1; i<count; i++) {
        key = students[i];
        j = i - 1;
        while(j >= 0 && students[j].average > key.average) {
            students[j+1] = students[j];
            j--;
        }
        students[j+1] = key;
    }
```

Picture 3. Sort by Name, ID, and Average

- **Bubble Sort**: Used to sort students alphabetically by their names.
- **Selection Sort**: Used to sort students by their unique IDs.
- **Insertion Sort**: Used to sort students by their average grades.

3. Searching Module

This module handles the logic for finding student records.



```
int linearSearchName(char name[]) {
    for(int i=0; i<count; i++) {
        if(strcmpi(students[i].name, name) == 0)
            return i;
    }
    return -1;
}
```

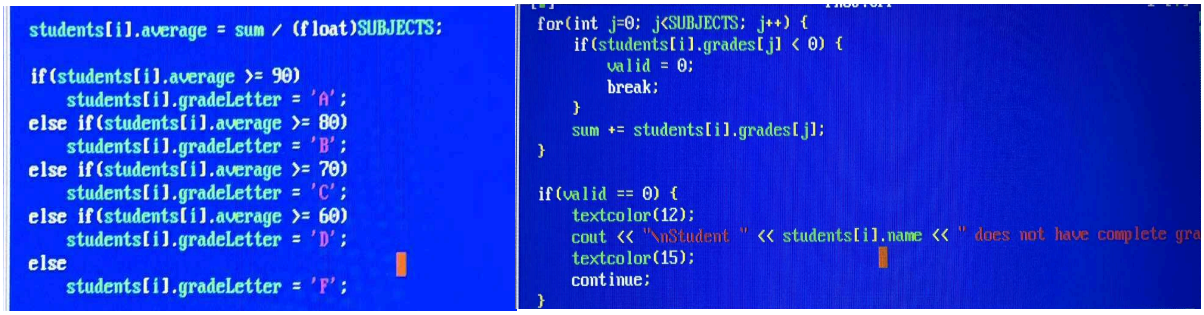
```
int binarySearchID(int id) {
    sortByID();
    int low = 0, high = count - 1;
    while(low <= high) {
        int mid = (low + high) / 2;
        if(students[mid].id == id) return mid;
        else if(students[mid].id > id) high = mid - 1;
        else low = mid + 1;
    }
    return -1;
}
```

Picture 4. Linear and Binary Searching Module

- **Linear Search**: Used to search for a student by name.
- **Binary Search**: Used to search for a student by ID (requires sorted data by ID beforehand).

4. Grade Calculation Module

This module calculates the average score for each student and assigns a letter grade based on the following scale:



```
students[i].average = sum / (float)SUBJECTS;

if(students[i].average >= 90)
    students[i].gradeLetter = 'A';
else if(students[i].average >= 80)
    students[i].gradeLetter = 'B';
else if(students[i].average >= 70)
    students[i].gradeLetter = 'C';
else if(students[i].average >= 60)
    students[i].gradeLetter = 'D';
else
    students[i].gradeLetter = 'F';
```

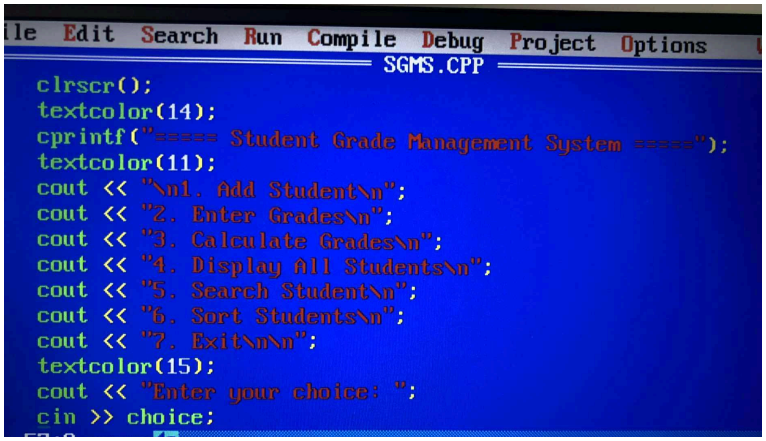
```
for(int j=0; j<SUBJECTS; j++) {
    if(students[i].grades[j] < 0) {
        valid = 0;
        break;
    }
    sum += students[i].grades[j];
}

if(valid == 0) {
    textcolor(12);
    cout << "\nStudent " << students[i].name << " does not have complete gra
    textcolor(15);
    continue;
}
```

Picture 5. Grade calculation

- **A**: 90–100
- **B**: 80–89
- **C**: 70–79
- **D**: 60–69
- **F**: Below 60

5. Menu and User Interface

A text-based menu is used to interact with the user. It allows access to the following functions:



```
ile  Edit  Search  Run  Compile  Debug  Project  Options
                SGMS.CPP
  clrscr();
  textcolor(14);
  cprintf("===== Student Grade Management System =====");
  textcolor(11);
  cout << "\n1. Add Student\n";
  cout << "2. Enter Grades\n";
  cout << "3. Calculate Grades\n";
  cout << "4. Display All Students\n";
  cout << "5. Search Student\n";
  cout << "6. Sort Students\n";
  cout << "7. Exit\n\n";
  textcolor(15);
  cout << "Enter your choice: ";
  cin >> choice;
```

Picture 6. Menu and User Interface

- Add Student
- Enter Grades

- Calculate Grades
- Display Individual Student
- Display All Students
- Search Student
- Sort Students
- Exit

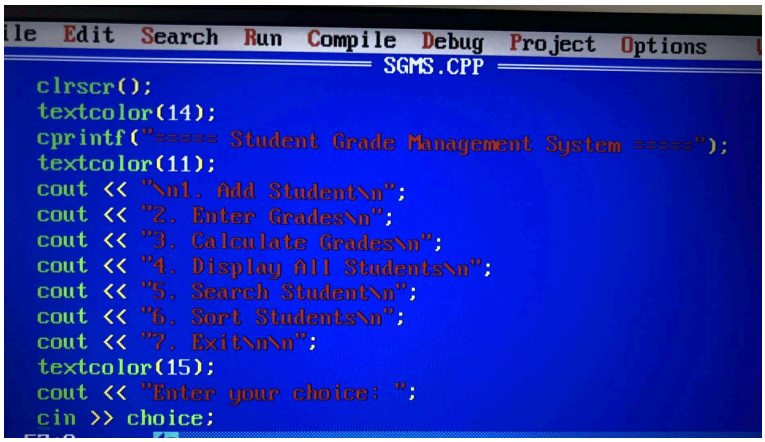
The menu is implemented using a switch-case structure inside a do-while loop, ensuring the user remains in control until they choose to exit.

System Implementation

The compiler tool used in creating theStudent Grade Management System (SGMS) in C++ program is an old version of TURBO C++ compiler that can be accessed by downloading the software. This system consists of three core modules: Student structure,Sorting and Searching Module, and Grade Calculation Module. Below are the key components of each module along with code snippets and explanations.

1. Main Menu Module

The main menu is the control center of the program where the user chooses actions like adding students, entering grades, or displaying data. It uses switch-case inside a do-while loop to keep displaying the menu until the user chooses to exit. Input is validated to prevent crashes when invalid entries are made.



Picture 7. Screenshot of the Main Menu Module

2. Student Registration Module

This module allows the user to add a new student by entering a unique ID and full name. It also initializes each subject grade with a placeholder value like -1 to indicate that no grade has been entered yet. The use of cin.getline() enables multi-word names.

```
cout << "Enter Student ID: ";
cin >> students[count].id;
cout << "Enter Student Name: ";
cin.ignore();
cin.getline(students[count].name, 50);

for(int i=0; i<SUBJECTS; i++)
    students[count].grades[i] = -1;

students[count].average = 0.0;
students[count].gradeLetter = 'F';
count++;
```

Picture 8. Screenshot of the Student Registration

3. Grade Entry Module

The user is prompted to enter grades for five subjects. A do-while loop ensures that each grade is within the valid range (0–100) before it's accepted. This prevents incorrect or out-of-range data from being stored.

```
for(int i=0; i<SUBJECTS; i++) {
    do {
        cout << "Enter grade for Subject " << (i+1) << " (0-100): ";
        cin >> students[index].grades[i];
    } while(students[index].grades[i] < 0 || students[index].grades[i] > 100);
}
```

Picture 9. Screenshot of Grade Entry

4. Grade Calculation Module

This part of the system calculates the average grade of each student by summing all subject scores and dividing by the total number of subjects. It also assigns a letter grade (A–F) based on the average using if-else conditions. The result is stored in the student's record.

```
calculateGrades() {
for(int i=0; i<count; i++) {
    int sum = 0;
    int valid = 1;

    for(int j=0; j<SUBJECTS; j++) {
        if(students[i].grades[j] < 0) {
            valid = 0;
            break;
        }
        sum += students[i].grades[j];
    }
```

```
students[i].average = sum / (float)SUBJECTS;

if(students[i].average >= 90)
    students[i].gradeLetter = 'A';
else if(students[i].average >= 80)
    students[i].gradeLetter = 'B';
else if(students[i].average >= 70)
    students[i].gradeLetter = 'C';
else if(students[i].average >= 60)
    students[i].gradeLetter = 'D';
else
    students[i].gradeLetter = 'F';
```

Picture 10. Screenshot of Grade Calculation

5. Search Module

This module allows the user to search for a student by name using linear search, or by ID using binary search. Binary search is faster but requires the list to be sorted by ID beforehand. Linear search works for names and is case-insensitive using strcmpi().

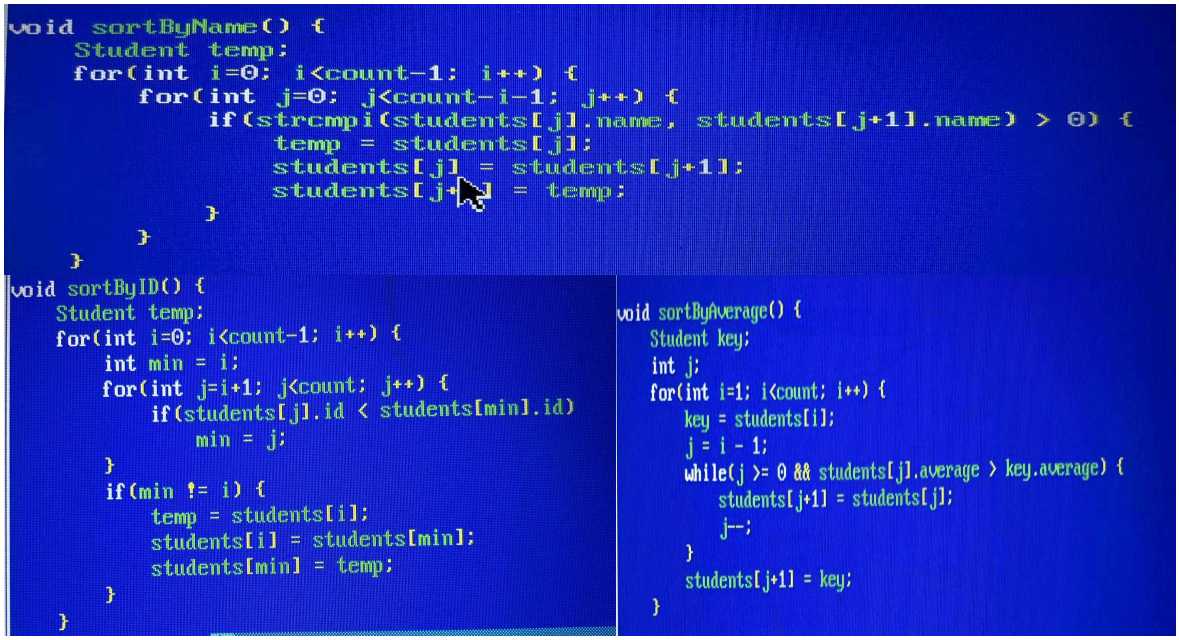
```
int linearSearchName(char name[]) {
    for(int i=0; i<count; i++) {
        if(strcmpi(students[i].name, name) == 0)
            return i;
    }
    return -1;
}
```

```
int binarySearchID(int id) {
    sortByID();
    int low = 0, high = count - 1;
    while(low <= high) {
        int mid = (low + high) / 2;
        if(students[mid].id == id) return mid;
        else if(students[mid].id > id) high = mid - 1;
        else low = mid + 1;
    }
    return -1;
}
```

Picture 11. Linear and Binary Search

6. Sorting Module

The sorting module organizes student records by name (using Bubble Sort), ID (using Selection Sort), or average grade (using Insertion Sort). This improves data readability and allows efficient searching. Each algorithm was selected for simplicity and teaching clarity.



```
void sortByName() {
    Student temp;
    for(int i=0; i<count-1; i++) {
        for(int j=0; j<count-i-1; j++) {
            if(strcmpi(students[j].name, students[j+1].name) > 0) {
                temp = students[j];
                students[j] = students[j+1];
                students[j+1] = temp;
            }
        }
    }
}
```

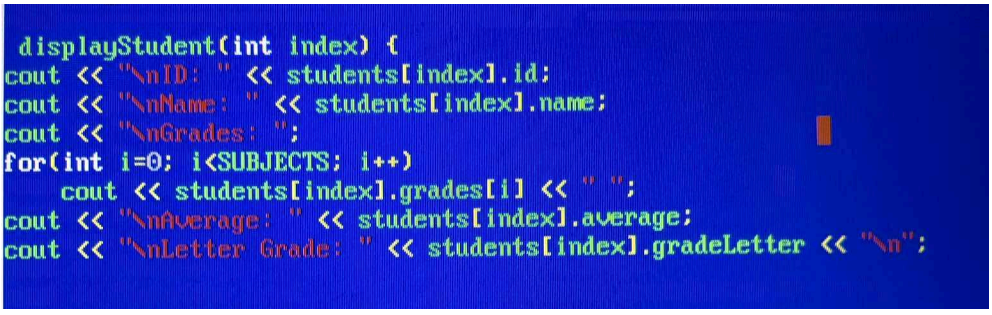
```
void sortByID() {
    Student temp;
    for(int i=0; i<count-1; i++) {
        int min = i;
        for(int j=i+1; j<count; j++) {
            if(students[j].id < students[min].id)
                min = j;
        }
        if(min != i) {
            temp = students[i];
            students[i] = students[min];
            students[min] = temp;
        }
    }
```

```
void sortByAverage() {
    Student key;
    int j;
    for(int i=1; i<count; i++) {
        key = students[i];
        j = i - 1;
        while(j >= 0 && students[j].average > key.average) {
            students[j+1] = students[j];
            j--;
        }
        students[j+1] = key;
    }
```

Picture 12. Screenshot of Sort by Name, ID, and Average

7. Display Module

This module prints the full details of each student, including ID, name, subject grades, average, and letter grade. It is used for both individual and full class displays. You can also enhance the output with textcolor() for visual clarity.

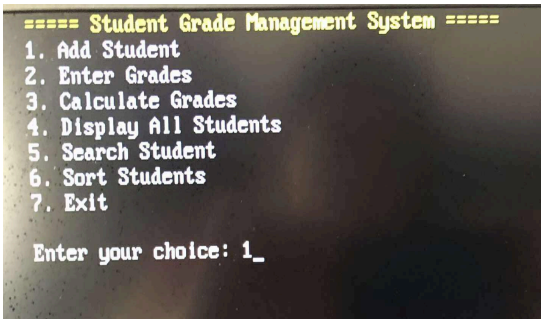
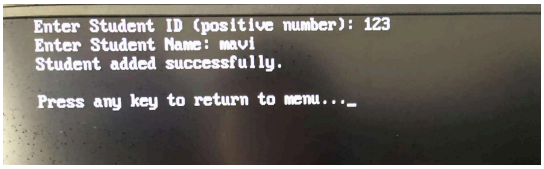
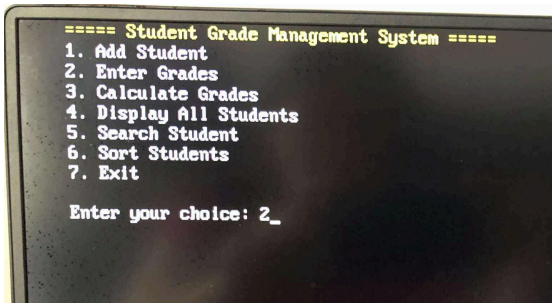
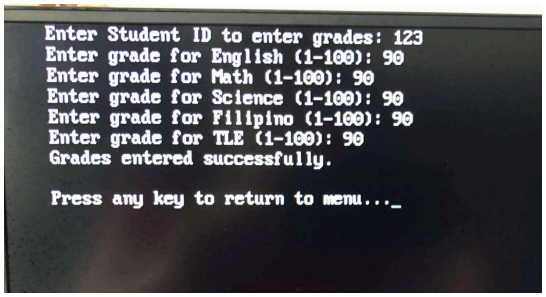


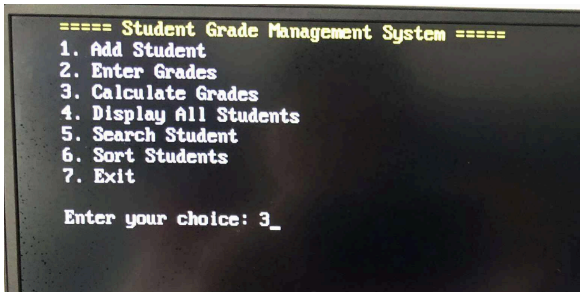
```
displayStudent(int index) {
cout << "\nID: " << students[index].id;
cout << "\nName: " << students[index].name;
cout << "\nGrades: ";
for(int i=0; i<SUBJECTS; i++)
    cout << students[index].grades[i] << " ";
cout << "\nAverage: " << students[index].average;
cout << "\nLetter Grade: " << students[index].gradeLetter << "\n";
```

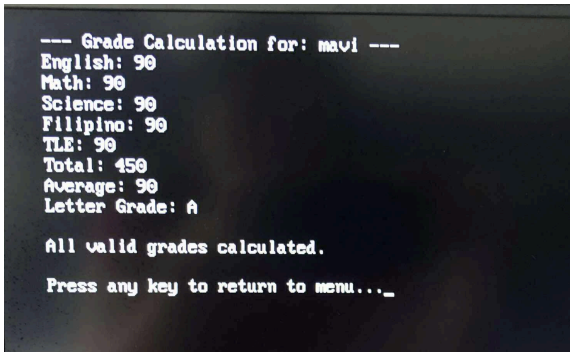
Picture 13. Screenshot of Display Information

Testing

This section will display a series of screenshots demonstrating the testing and functionality of the Student Grade Management System (SGMS) software.

Screenshot	Steps
 	**Steps:** 1. Run the program 2. Select option 1 from the menu to add a new students 3. Enter the following details when prompted: ● Student ID #: 123 ● Student Name: Mavi 4. Verify the output to ensure the student is added successfully with a unique ID. **Expected Result:** ● The student should be added to the system with a unique ID (e.g., 123). ● The system should display a confirmation message indicating the successful addition of the student.
 	**Steps:** 1. Press any key to go back to the main menu. 2. Select option 2 from the menu to add an Enter grade. 3. Enter the following details when prompted: ● ID #: 123 ● Enter grade for Eng: 90 ● Enter grade for Math: 90 ● Enter grade for Sci: 90 ● Enter grade for Fil: 90 ● Enter grade for TLE: 90 4. Verify the output to ensure the grade entered is added successfully. **Expected Result:** ● The student grade should be added to the system with a unique ID (e.g., 123). ● The system should display a confirmation message indicating the successful addition of the grades.

<table>
<tr>
<td>




</td>
<td>
Steps:

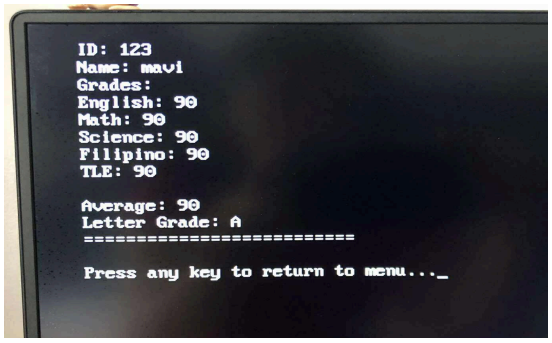
Press any key to go back to the main menu.
Select option 3 from the menu to calculate grade details.
Enter the following when prompted and details will be displayed.

Student ID#: 123

Verify the output to ensure the details of grades entered.

Expected Result:

The details of the student's grades will be displayed.

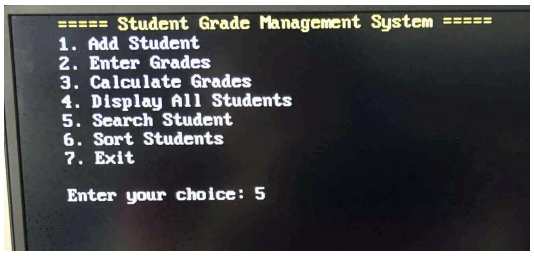
</td>
</tr>
<tr>
<td>


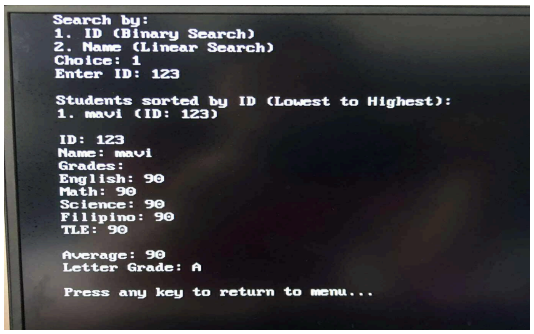
</td>
<td>
Steps:

Press any key to go back to the main menu.
Select option 4 from the menu to Display Students.
Verify the output to ensure the details of the students

Expected Result:

The information of the students will be displayed.

</td>
</tr>
<tr>
<td>




</td>
<td>
Steps:

Press any key to go back to the main menu.
Select option 5 from the menu to search Students.
Enter the following details when prompted:

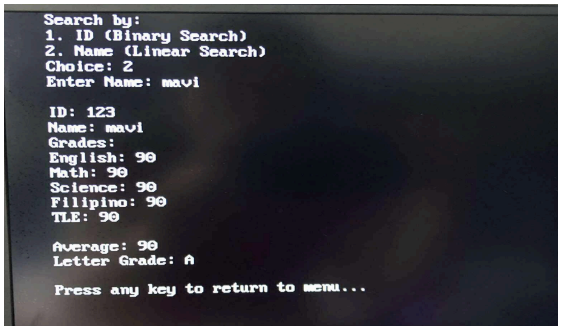
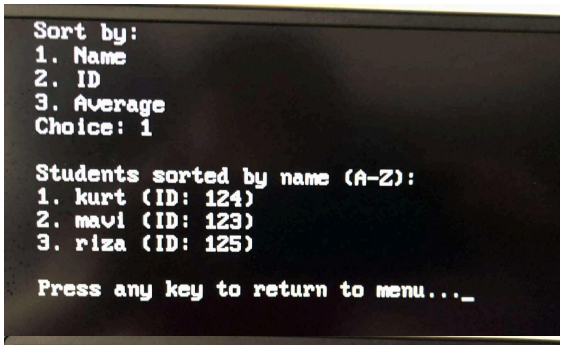
ID
Name1
Choice:

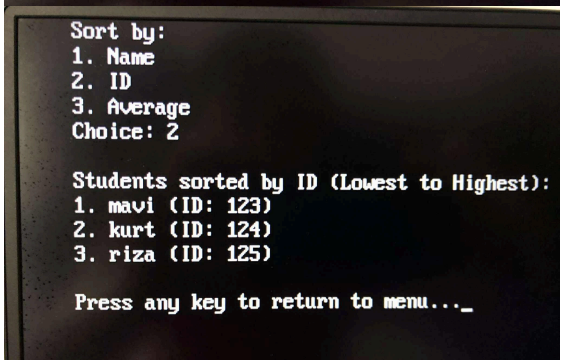
Verify the output to ensure the details that we search are correct with a unique ID and Name.

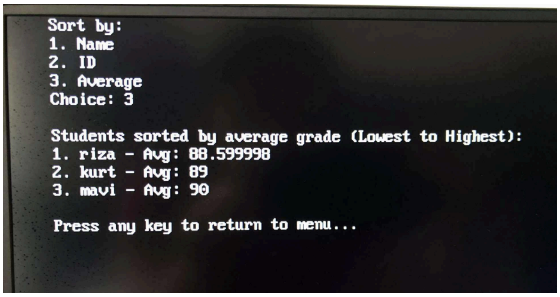
Expected Result:.

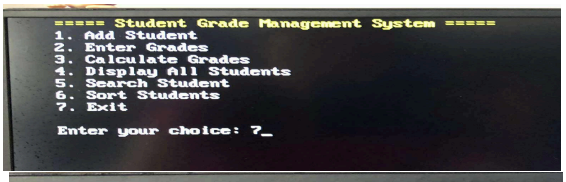
The system should display a confirmation message indicating the successful student search by ID or Name.

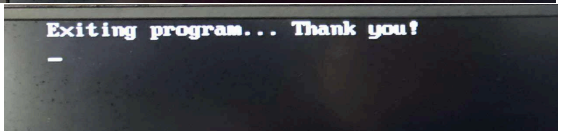
</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td></td>
</tr>
<tr>
<td> 




</td>
<td>Steps:
1. Press any key to go back to the main menu.
2. Select option 6 from the menu to Sort students.
3. Enter the following details when prompted:
● Sort by Name:
● Sort by ID:
● Sort by Average:
● Choice:
4. Verify the output to ensure that the sorting function is correctly from A-Z and Lowest to Highest.

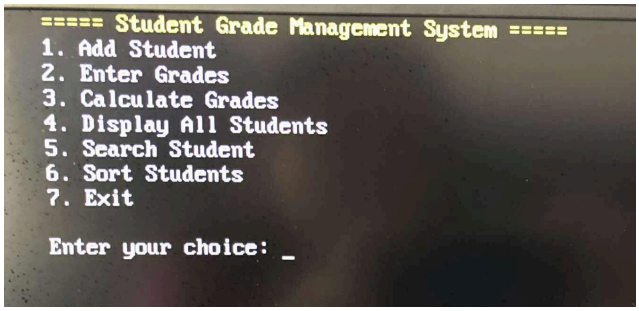
Expected Result:
● The Sorting functions should display a confirmation message indicating the successful sort.</td>
</tr>
<tr>
<td> 

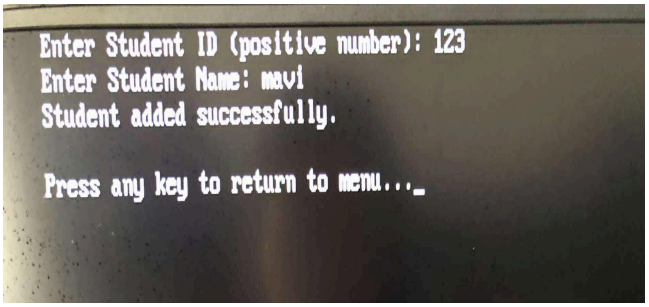

</td>
<td>Steps:
1. Press any key to go back to the main menu.
2. Select option 7 from the menu to exit the program.
3. The program will automatically exit.

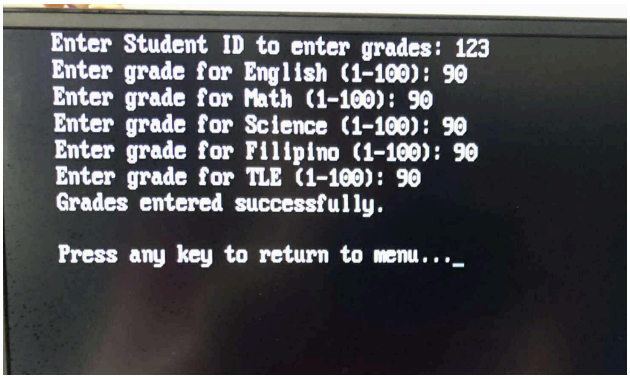
Expected Result:
● The system should display a message “leaving the program.. Thank you”.</td>
</tr>
</table>

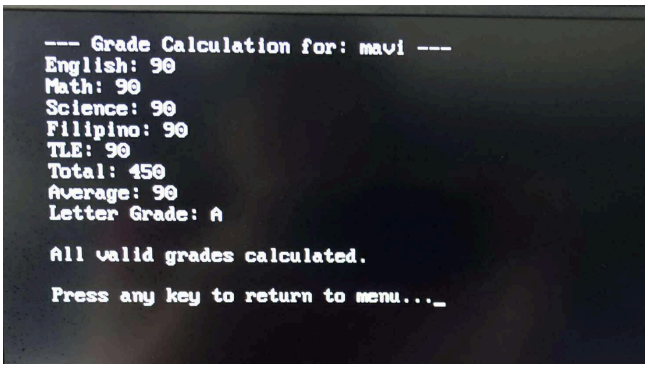
Results

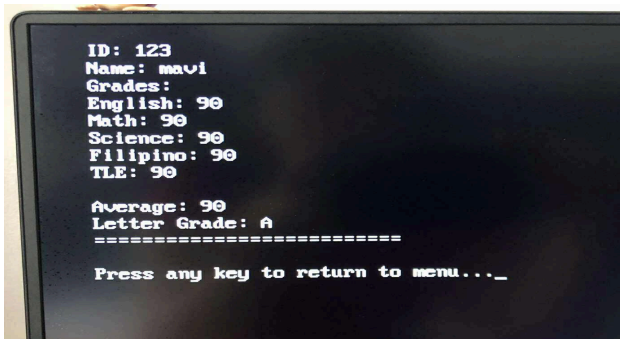
After testing the functionality of the software, the results demonstrate its capability to perform successfully. The picture below illustrates the complete session, showcasing each step:

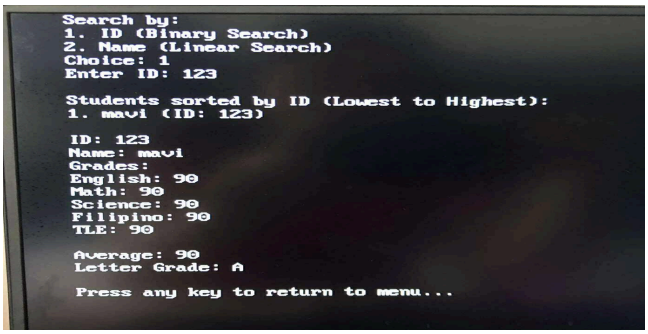


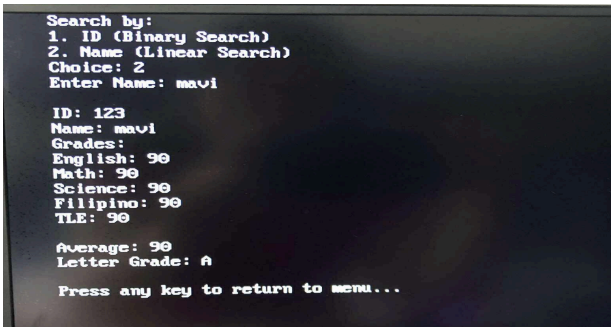


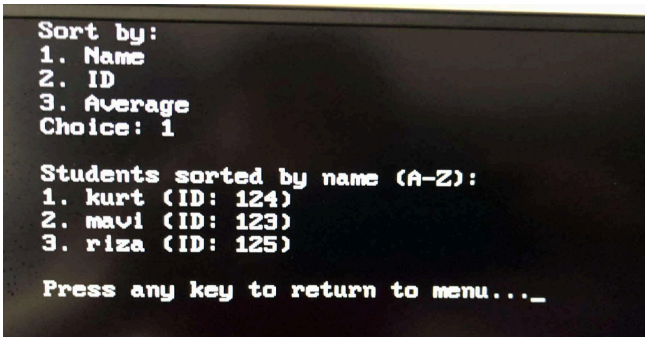


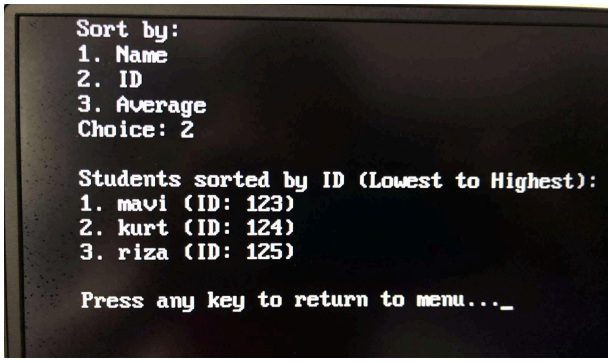


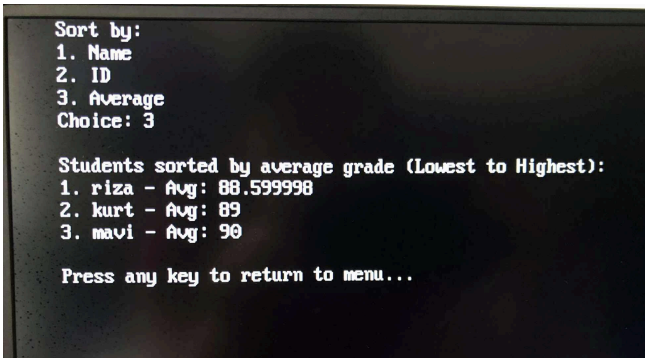


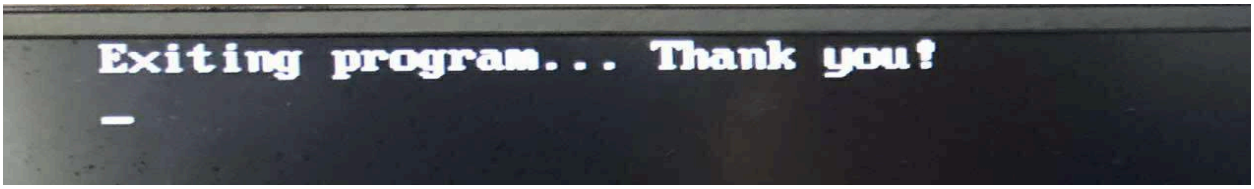












Error Handling

Error handling in the Student Grade Management System developed using Turbo C++ is crucial for ensuring that the program runs smoothly and does not crash when users provide incorrect or unexpected input. One of the most common errors encountered is when the user inputs a letter or symbol instead of a number, especially when selecting from the menu. To address this, the system checks for input failure using cin.fail(). If an error is detected, the input stream is cleared using cin.clear() and the rest of the invalid input is ignored using cin.ignore(). A colored message is then displayed using textcolor() to inform the user that their input was invalid, and the program safely returns to the main menu without breaking.

In addition to input type validation, the system also ensures that all grades entered fall within a valid range of 0 to 100. This is implemented using a do-while loop that repeatedly asks the user to input a valid grade until it meets the specified criteria. This prevents invalid data from being processed during calculations. Furthermore, the program prevents adding more students than the allowed maximum limit by checking the count against a predefined constant (MAX). If the limit is reached, an error message is shown and no new student is added. Before calculating average grades, the system checks that all subject grades have been entered. It does this by verifying that none of the grades are left at the initial placeholder value (such as -1). If any grades are missing, the program skips that student's calculation and informs the user with a

warning. Similarly, when a user attempts to search for a student by ID or name, the program handles the possibility of not finding a match. It displays a clear message instead of failing silently or producing incorrect output.

Overall, error handling in this system involves validating inputs, using logical checks for conditions like grade range and data completeness, and giving user feedback through color-coded messages. These practices not only improve program reliability but also enhance user experience by guiding them toward correct usage and preventing system crashes.

Conclusion

The **Student Grade Management System** developed in Turbo C++ is a practical and educational console-based application that allows users to efficiently manage student academic records. It features secure input validation to ensure that student IDs are positive, names contain no numeric characters, and grades fall strictly within the valid range of 1 to 100. With clearly defined operations such as adding students, entering grades, calculating averages, and assigning letter grades, the system provides a structured and user-friendly experience. It also includes searching and sorting functionalities, helping users organize and retrieve student information effectively. By utilizing key programming concepts such as structures, arrays, conditional logic, and basic search and sort algorithms, this system serves as a strong learning tool for beginners aiming to build their skills in C++. Overall, it demonstrates how structured programming and proper input handling can be combined to create a reliable and interactive management system.

Source code:

```
#include <iostream.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

#include <dos.h>

#include <ctype.h>


#define MAX 100

#define SUBJECTS 5


char subjectNames[SUBJECTS][15] = {"English", "Math", "Science", "Filipino", "TLE"};


struct Student {

   int id;

   char name[50];

   int grades[SUBJECTS];

   float average;

   char gradeLetter;

};
```

```
Student students[MAX];
int count = 0;

// Function declarations
void addStudent();
void enterGrades();
void calculateGrades();
void displayStudent(int index);
void displayAll();
void searchStudent();
void sortStudents();
int binarySearchID(int id);
int linearSearchName(char name[]);
void sortByName();
void sortByID();
void sortByAverage();
int isValidName(char str[]);

void main() {
    clrscr();
    int choice;
```

```
do {
    clrscr();
    textcolor(14);
    cprintf("===== Student Grade Management System =====");
    textcolor(11);
    cout << "\n1. Add Student\n";
    cout << "2. Enter Grades\n";
    cout << "3. Calculate Grades\n";
    cout << "4. Display All Students\n";
    cout << "5. Search Student\n";
    cout << "6. Sort Students\n";
    cout << "7. Exit\n\n";
    textcolor(15);
    cout << "Enter your choice: ";
    cin >> choice;

    if (cin.fail()) {
        cin.clear();
        cin.ignore(100, '\n');
        textcolor(12);
```

```
    cout << "Invalid input! Please enter a number.\n";
    textcolor(15);
    delay(1000);
    continue;
}

clrscr();
switch(choice) {
    case 1: addStudent(); break;
    case 2: enterGrades(); break;
    case 3: calculateGrades(); break;
    case 4: displayAll(); break;
    case 5: searchStudent(); break;
    case 6: sortStudents(); break;
    case 7:
        textcolor(10);
        cout << "Exiting program... Thank you!\n";
        textcolor(15);
        delay(1000);
        exit(0);
    default:
```

```
            textcolor(12);
            cout << "Invalid choice!\n";
            textcolor(15);
        }
        cout << "\nPress any key to return to menu...";
        getch();
    } while(1);
}

int isValidName(char str[]) {
    for (int i = 0; i < strlen(str); i++) {
        if (isdigit(str[i]))
            return 0;
    }
    return 1;
}

void addStudent() {
    if(count >= MAX) {
        textcolor(12);
        cout << "Max student limit reached!\n";
```

```
    textcolor(15);
    return;
}

int tempID;
do {
    cout << "Enter Student ID (positive number): ";
    cin >> tempID;
    if (cin.fail() || tempID <= 0) {
        cin.clear();
        cin.ignore(100, '\n');
        textcolor(12);
        cout << "Invalid ID! Please enter a positive number.\n";
        textcolor(15);
    }
} while (tempID <= 0);
students[count].id = tempID;

char tempName[50];
do {
    cout << "Enter Student Name: ";
```

```
        cin.ignore();
        cin.getline(tempName, 50);
        if (!isValidName(tempName)) {
            textcolor(12);
            cout << "Invalid name! Numbers are not allowed.\n";
            textcolor(15);
        }
    } while (!isValidName(tempName));
    strcpy(students[count].name, tempName);

    for(int i = 0; i < SUBJECTS; i++)
        students[count].grades[i] = -1;

    students[count].average = 0.0;
    students[count].gradeLetter = 'F';
    count++;

    textcolor(10);
    cout << "Student added successfully.\n";
    textcolor(15);
}
```

```
void enterGrades() {
    int id;
    cout << "Enter Student ID to enter grades: ";
    cin >> id;
    if (cin.fail()) {
        cin.clear();
        cin.ignore(100, '\n');
        textcolor(12);
        cout << "Invalid input!\n";
        textcolor(15);
        return;
    }

    // 🔁 Use unsorted linear search (no sorting)
    int index = -1;
    for (int i = 0; i < count; i++) {
        if (students[i].id == id) {
            index = i;
            break;
        }
```

```
}

if(index == -1) {
    textcolor(12);
    cout << "Student not found.\n";
    textcolor(15);
    return;
}

for(int i = 0; i < SUBJECTS; i++) {
    int valid = 0;
    do {
        cout << "Enter grade for " << subjectNames[i] << " (1-100): ";
        cin >> students[index].grades[i];
        if (cin.fail()) {
            cin.clear();
            cin.ignore(100, '\n');
            cout << "Invalid input! Please enter a number.\n";
            continue;
        }
```

```
            if (students[index].grades[i] <= 0 || students[index].grades[i] > 100) {
                cout << "Invalid grade! Must be between 1 and 100.\n";
            } else {
                valid = 1;
            }
        } while (!valid);
    }

    textcolor(10);
    cout << "Grades entered successfully.\n";
    textcolor(15);
}

void calculateGrades() {
    for(int i = 0; i < count; i++) {
        int sum = 0;
        int valid = 1;

        for(int j = 0; j < SUBJECTS; j++) {
            if(students[i].grades[j] < 0) {
                valid = 0;
```

```
        break;
    }
    sum += students[i].grades[j];
}

if(!valid) {
    textcolor(12);
    cout << "\nStudent " << students[i].name << " has incomplete grades.\n";
    textcolor(15);
    continue;
}

students[i].average = sum / (float)SUBJECTS;

if(students[i].average >= 90) students[i].gradeLetter = 'A';
else if(students[i].average >= 80) students[i].gradeLetter = 'B';
else if(students[i].average >= 70) students[i].gradeLetter = 'C';
else if(students[i].average >= 60) students[i].gradeLetter = 'D';
else students[i].gradeLetter = 'F';

textcolor(14);
```

```
        cout << "\n--- Grade Calculation for: " << students[i].name << " ---\n";
        textcolor(15);
        for(int j = 0; j < SUBJECTS; j++) {
            cout << subjectNames[j] << ": " << students[i].grades[j] << "\n";
        }
        cout << "Total: " << sum << "\n";
        cout << "Average: " << students[i].average << "\n";
        cout << "Letter Grade: " << students[i].gradeLetter << "\n";
    }

    textcolor(10);
    cout << "\nAll valid grades calculated.\n";
    textcolor(15);
}

void displayStudent(int index) {
    cout << "\nID: " << students[index].id;
    cout << "\nName: " << students[index].name;
    cout << "\nGrades:\n";
    for(int i = 0; i < SUBJECTS; i++) {
        cout << subjectNames[i] << ": " << students[index].grades[i] << "\n";
```

```
    }
    cout << "\nAverage: " << students[index].average << "\n";
    cout << "Letter Grade: " << students[index].gradeLetter << "\n";
}

void displayAll() {
    if(count == 0) {
        textcolor(12);
        cout << "No student records to display.\n";
        textcolor(15);
        return;
    }
    for(int i = 0; i < count; i++) {
        displayStudent(i);
        cout << "============================\n";
    }
}

int binarySearchID(int id) {
    sortByID();
    int low = 0, high = count - 1;
```

```
    while(low <= high) {
        int mid = (low + high) / 2;
        if(students[mid].id == id) return mid;
        else if(students[mid].id > id) high = mid - 1;
        else low = mid + 1;
    }
    return -1;
}

int linearSearchName(char name[]) {
    for(int i = 0; i < count; i++) {
        if(strcmpi(students[i].name, name) == 0)
            return i;
    }
    return -1;
}

void searchStudent() {
    int ch;
    cout << "Search by:\n1. ID (Binary Search)\n2. Name (Linear Search)\nChoice: ";
    cin >> ch;
```

```
if (cin.fail()) {
    cin.clear();
    cin.ignore(100, '\n');
    textcolor(12);
    cout << "Invalid input!\n";
    textcolor(15);
    return;
}
if(ch == 1) {
    int id;
    cout << "Enter ID: ";
    cin >> id;
    if (cin.fail()) {
        cin.clear();
        cin.ignore(100, '\n');
        cout << "Invalid ID input.\n";
        return;
    }
    int index = binarySearchID(id);
    if(index != -1) displayStudent(index);
    else {
```

```
            textcolor(12);
            cout << "Student not found.\n";
            textcolor(15);
        }
    } else if(ch == 2) {
        char name[50];
        cout << "Enter Name: ";
        cin.ignore();
        cin.getline(name, 50);
        int index = linearSearchName(name);
        if(index != -1) displayStudent(index);
        else {
            textcolor(12);
            cout << "Student not found.\n";
            textcolor(15);
        }
    } else {
        textcolor(12);
        cout << "Invalid search option.\n";
        textcolor(15);
    }
```

```
}

void sortStudents() {
    int ch;
    cout << "Sort by:\n1. Name\n2. ID\n3. Average\nChoice: ";
    cin >> ch;
    if (cin.fail()) {
        cin.clear();
        cin.ignore(100, '\n');
        textcolor(12);
        cout << "Invalid sort option.\n";
        textcolor(15);
        return;
    }
    if(ch == 1) sortByName();
    else if(ch == 2) sortByID();
    else if(ch == 3) sortByAverage();
    else {
        textcolor(12);
        cout << "Invalid sort option.\n";
        textcolor(15);
```

```
    }
}

void sortByName() {
    Student temp;
    for(int i = 0; i < count-1; i++) {
        for(int j = 0; j < count-i-1; j++) {
            if(strcmpi(students[j].name, students[j+1].name) > 0) {
                temp = students[j];
                students[j] = students[j+1];
                students[j+1] = temp;
            }
        }
    }
    textcolor(10);
    cout << "\nStudents sorted by name (A-Z):\n";
    textcolor(15);
    for(int i = 0; i < count; i++)
        cout << i+1 << ". " << students[i].name << " (ID: " << students[i].id << ")\n";
}
```

```
void sortByID() {
    Student temp;
    for(int i = 0; i < count-1; i++) {
        int min = i;
        for(int j = i+1; j < count; j++) {
            if(students[j].id < students[min].id)
                min = j;
        }
        if(min != i) {
            temp = students[i];
            students[i] = students[min];
            students[min] = temp;
        }
    }
    textcolor(10);
    cout << "\nStudents sorted by ID (Lowest to Highest):\n";
    textcolor(15);
    for(int i = 0; i < count; i++)
        cout << i+1 << ". " << students[i].name << " (ID: " << students[i].id << ")\n";
}
```

```
void sortByAverage() {
    Student key;
    int j;
    for(int i = 1; i < count; i++) {
        key = students[i];
        j = i - 1;
        while(j >= 0 && students[j].average > key.average) {
            students[j+1] = students[j];
            j--;
        }
        students[j+1] = key;
    }

    textcolor(10);
    cout << "\nStudents sorted by average grade (Lowest to Highest):\n";
    textcolor(15);
    for(int i = 0; i < count; i++)
        cout << i+1 << ". " << students[i].name << " - Avg: " << students[i].average << "\n";
}
```

Reference

Sourcecodester. (2020, November 17). *Student grading system using PHP/MySQL with source code*. SourceCodester. https://www.sourcecodester.com/php/14522/student-grading-system-using-phpmysql-source-code.html