Project Report

Student Performance Analyzer using C

1. Title Page Project Title:

Student Performance Analyzer

Project Language: C Programming Language

Submitted by: Shardul Aggarwal

SAP:590022064

Date: November 28, 2025

Organization: University of Petroleum and Energy Studies

2. Abstract

The Student Performance Analyzer is a console-based application developed in the C programming language designed to efficiently manage, process, and analyse academic data for a small to medium sized class or group of students. The system takes student details and scores across multiple subjects/assessments as input. It then applies robust algorithms to calculate performance metrics such as overall average, assign letter grades (based on predefined criteria), categorize performance levels (e.g., Excellent, Good), and even implement a simple predictive model to estimate potential final scores. The primary goal is to provide immediate, actionable insights into student academic standing, replacing manual calculation processes with a fast, reliable, and user-friendly digital solution. The project emphasizes data structures, file I/O for persistence, and modular programming principles.

3. Introduction

Academic performance analysis is critical for educators to identify students requiring intervention, evaluate teaching methodologies, and report progress. Traditional methods involving manual calculations or rudimentary spreadsheets

are often time-consuming and prone to human error. This project addresses the need for a dedicated, efficient, and easy-to-deploy tool. Utilizing the C language allows for a high degree of control over memory management and computational efficiency, making the system lightweight and fast, suitable for environment where specialized database or GUI tools are unavailable. The application acts as a foundation for more sophisticated educational data mining tool

4. **Problem Definition**

The core problem addressed by this project is the lack of an integrated, automated system for:

1. Efficient Data Management: Storing and retrieving student performance data (name, scores) persistently.

2. Accurate Metric Calculation: Automating the calculation of key academic metrics (average, grade, performance category).

3. Performance Forecasting: Providing an objective, data-driven estimate of a student's final score potential to facilitate early intervention strategies. 4. Simplified Reporting: Generating a clear, summarized report for each student instantly

4. Simplified Reporting: Generating a clear, summarized report for each student instantly.

5. Objectives of the Project

The primary objectives of the Student Performance Analyzer project are:

- To design and implement a structured data model (using C struct) to hold student records.

- To develop functions for the basic CRUD operations: Create (add new student), Read (view records), Update (modify scores), and Delete (remove student).
- To implement algorithms for standard grade and average calculation.
- To incorporate a simple predictive model to project scores based on current data.
- To ensure data persistence using file handling (e.g., binary or text files) so that data is retained between program executions.
- To provide a simple, intuitive, menu-driven command-line interface (CLI) for user interaction.

6.System Requirements

Hardware Requirements

Component	Minimum Specification
Processor	Intel Pentium / AMD equivalent or higher
RAM	256 MB
Hard Disk	10 MB free space
Input Device	Standard Keyboard
Display	Console / Terminal (80×25 characters)

Software Requirements

Component	Required Version
Operating System	Windows 7 or above, macOS, or any Linux distribution
Programming Language	C
Compiler	GCC (GNU Compiler Collection) or any ANSI C–compatible compiler
Development Environment	Text Editor (e.g., VS Code, Vim) or IDE (e.g., Code::Blocks, Dev-C++, CodeLite)

7. Methodology

7.1 Input Design

The system uses a Command-Line Interface (CLI) for user interaction. All data is received through standard input functions such as scanf and fgets.

Types of Inputs Collected

1. **Student Identifier**
    - Student ID: A unique integer value.
    - Student Name: A string value.
2. **Assessment Scores**
    - A fixed number of subject or test scores are entered.
    - Example: 3 internal tests and 1 final exam, or 5 subject marks.
    - Scores are floating-point values in the range 0.0 to 100.0.
    - Input validation ensures:
        - Marks are within the valid range.
        - Only numerical values are accepted

Output Design

The output is displayed on the console in a clear, formatted, tabular manner, showing**:**

1. Student ID and Name.

2. Raw scores for each assessment.

3. Calculated Metrics: Overall Average, Letter Grade, and Performance Category.

4. Predicted Final Score (based on the model).

Algorithms

a. Average Calculation

The average is calculated as the arithmetic mean of all recorded assessment scores, with specific weights applied to internal tests and the final exam.

$$\text{Average Score} = \frac{W1 + W2 + \cdots + Wn}{(W1 \cdot S1) + (W2 \cdot S2) + \cdots + (Wn \cdot Sn)}$$

Where:

- S_i= score of the i-th assessment
- W_i= weight assigned to that assessment

Implementation Note:
In the sample program, all assessments are equally weighted for simplicity.

b. Grade Assignment

Grades are assigned based on a standard scale:

Score Range	Grade
Average ≥ 90	A
80 ≤ Average < 90	B
70 ≤ Average < 80	C
60 ≤ Average < 70	D
Average < 60	F

---

c. Performance Category

Grade	Performance Category
A, B	Excellent
C	Good
D	Average
F	Needs Improvement

d. Predictive Score Model

A Weighted Average Prediction Model (WAPM) is used.

Given three completed internal test scores S_1, S_2, S_3, the predicted future performance is:

$$\text{Predicted Average} = \frac{S_1 + S_2 + S_3}{3}$$

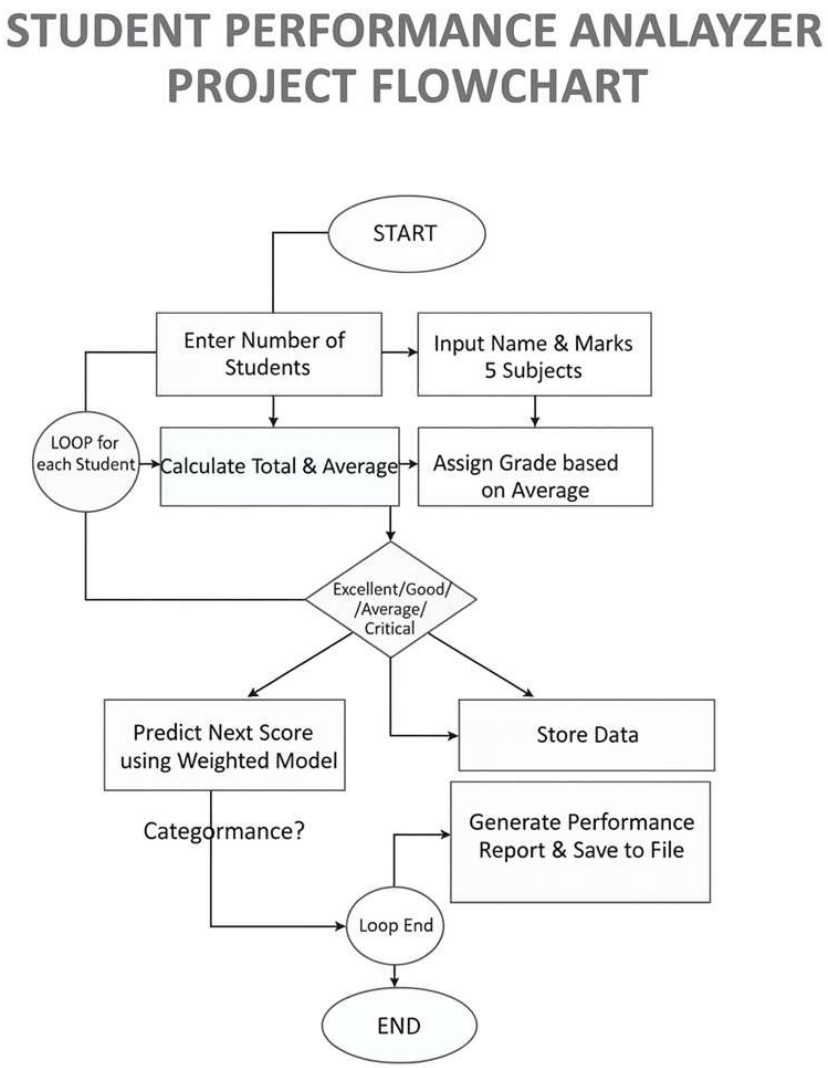
The assumption is that the student will continue performing at the same level in upcoming assessments.

---

8. Flowchart Summary

1. Start
2. Load existing student records (if available)
3. Display menu (Add, View All, Search, Delete, Exit)
4. Accept user choice

5. Execute corresponding operation
6. Return to menu
7. Exit program and save data



---

9. Program Code

Source code includes:

- Required header files (stdio.h, stdlib.h, string.h)

- Student struct definition
- Functions for:
    - Input handling
    - Average calculation
    - Grade assignment
    - Prediction model
    - File storage and retrieval
    - Menu operations

---

10. Testing & Sample Output

Test Scenarios

1. Boundary Value Testing:
   Scores at extremes (0 and 100), grade cut-off values.
2. Persistence Test:
   Add → Save → Restart → Reload data.
3. Error Handling:
   Non-numeric input for score fields.
4. Deletion Test:
   Ensure deleted records do not reappear.

Sample Output (View All**)**

ID	Name	Score 1	Score 2	Score 3
101	Alice Smith	95.00	92.00	98.00
102	Bob Johnson	75.00	81.00	78.00
103	Clara Vue	55.00	62.00	58.00

11. Advantages

- Fast and lightweight (pure C)
- Accurate automated calculations
- Cross-platform portability
- Simple performance prediction
- Data stored permanently using file handling

---

12. Limitations

- CLI interface (no GUI)
- File-based storage is not ideal for very large datasets
- Limited error handling

- Prediction model is simplistic

---

13. Future Enhancements

- GUI using GTK+ or Qt
- Database support (e.g., SQLite)
- Advanced analytics (regression, clustering)
- Printable reports and charts

---

14. Conclusion

The Student Performance Analyzer meets its objectives by providing a functional, efficient, and accurate academic analysis tool. It showcases strong fundamentals in C programming, especially in the areas of data structures, file handling, and algorithm design. Though simple in its current form, it establishes a solid foundation for future enhancements in educational data analytics.