Programming Fundamentals Lab

OEL Report



**Student Grade Management System**

Submitted by:

|  |  |
| --- | --- |
| Muhammad Waleed | 24L-6023 |

Submitted to:

Ms. Aliha Tanveer

7/4/2025

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Lahore

## Introduction

1. **Problem Analysis**

## Design Requirements

1. **Feasibility Analysis**

## Possible Solutions

1. **Preliminary Design**

## Design Description

## Software Simulation

## Experimental Results

1. **Performance Analysis**
2. **Future Scope**

## Social and Cultural Implications

1. **Conclusion References**

The main idea behind this project is to develop an Automated Student Grading and Report Management System that streamlines the process of calculating and managing student grades. The system is designed to handle students' grades across multiple subjects such as English, History, and Math, and it generates detailed reports reflecting each student’s performance in these courses. This grading system aims to eliminate the inefficiencies and potential errors associated with traditional manual grading methods and to enhance the accuracy and speed of grade computation.

**Related Concepts Used in the Project**

1. **Object-Oriented Programming (OOP)**:
   * The project makes extensive use of Object-Oriented Programming (OOP) concepts. OOP allows for structuring the system in a modular and reusable manner. The use of classes and inheritance plays a crucial role in the design. For example, the student class serves as the base class, while English marks, history marks, and math marks are derived classes that extend the functionality of the student class to handle subject-specific grade calculations.
   * Encapsulation is another core concept used in this system, as the student data is encapsulated within the objects of the student and its derived classes. This ensures data privacy and allows the code to be more organized and maintainable.
2. **File Handling**:
   * The system uses file handling techniques to read student data from input files and write the final reports to output files. This feature is important for handling large datasets efficiently. The take\_students\_from\_file method imports student information, and the return\_report method outputs the computed grades to a file, allowing for automated report generation.
3. **Polymorphism**:
   * Polymorphism is used in the project to calculate grades differently based on the course. The system uses virtual functions in the base class student (and its derived classes) to calculate the final grade, which is tailored for each specific subject. This flexibility allows the system to support different grading schemes for each course.
   * For example, the grading scheme for English involves attendance, project, midterm, and final exam grades, whereas History and Math have their unique schemes. The system uses polymorphism to handle these differences in a unified manner, calling the appropriate function based on the course.
4. **Weighted Average Calculation**:
   * One of the key functionalities in this system is the weighted average calculation for final grades. The grades are weighted based on predefined ratios for each subject (for example, 10% for quizzes, 30% for projects, and 40% for exams in Math). This is a significant concept in grading systems, as it ensures that the weight of each assessment type is accurately reflected in the final grade.
5. **Dynamic Memory Allocation**:
   * The system employs dynamic memory allocation using pointers to manage memory for students' data. When student data is read from a file, memory is dynamically allocated to store the student objects, ensuring flexibility and optimal use of memory for varying amounts of data.

The problem at hand is to develop a system that automates the grading process for students in multiple courses such as **English**, **History**, and **Math**. The system needs to calculate students' final grades based on a variety of assessments, which include attendance, projects, midterm exams, quizzes, and final exams. Additionally, the system should be able to:

1. **Import student data**: The system must read data from an external file containing student information (such as first name, last name, course, and grades for various assessments).
2. **Calculate the final grade**: Based on the grading scheme for each course, the system needs to calculate the final grade for each student.
3. **Generate a report**: Once the grades are computed, the system must generate a report that displays each student's name, course, and final grade, and save this report to an output file.
4. **Scalability**: The system should be designed in such a way that it can easily accommodate additional courses or grading schemes if necessary.

**Problem Breakdown**

1. **Student Data Input**:
   * The system must be able to read student data from a file (Student.txt), which contains information such as:
     + First name
     + Last name
     + Course (English, History, Math)
     + Various grade components (attendance, project, midterm, final exam, etc.)

The data in the file may vary based on the course, with different grading components for different courses. For example, **English** has grades for attendance, project, midterm, and final exam, while **Math** includes grades for quizzes, tests, and the final exam.

1. **Grade Calculation**:
   * Each course has a unique grading scheme. For instance:
     + **English**: The grade is calculated as a weighted average of attendance (10%), project (30%), midterm (30%), and final exam (30%).
     + **History**: The grade is calculated as a weighted average of term paper (25%), midterm **(35%),** and final exam **(40%).**
     + **Math**: The grade is calculated as a weighted average of quizzes **(10%)**, tests **(50%),** and final exam **(40%).**
   * The system needs to compute these weighted averages correctly for each student based on the course they are enrolled in.
2. **Report Generation**:
   * Once the grades are calculated, the system must output a report that lists each student’s name, course, and final grade. The report should be saved to an output file (result.txt).
   * The output file should be well formatted and easy to read, containing all the necessary information about each student.
3. **Scalability**:
   * The solution should be flexible enough to accommodate changes in the future. For example, if a new course is introduced or if the grading scheme for an existing course change, the system should be able to handle such changes with minimal adjustments.
   * This scalability is achieved by using Object-Oriented Programming (OOP) principles such as inheritance and polymorphism. The base class student is extended by specialized classes like English marks, history marks, and math marks, which can be easily modified or extended for future requirements.

The design requirements of the project focus on developing a robust student grade management system. The system is intended to read student data, calculate grades based on predefined criteria for different courses, and generate a report. Below are the essential design requirements, focusing on the functional aspects of the system.

**1. Input Requirements**

The system is designed to handle a variety of student data related to their personal details, enrolled courses, and corresponding grades. The primary input consists of student information such as:

* Student's name (first and last).
* The course they are enrolled in (either English, History, or Math).
* Their grades for different assessments (attendance, project, exams, etc.) are based on the course.

Each student’s input data is processed to determine their performance in the course, which directly influences the final grade calculation.

**2. Output Requirements**

The system must generate a report containing the following details for each student:

* **Full Name**: Displaying the student's first and last name.
* **Course**: Indicating the specific course the student is enrolled in.
* **Final Grade**: The calculated final grade for each student based on the respective course's grading criteria.

This output must be clearly formatted, ensuring that the information is easy to read and understand.

Feasibility analysis is a crucial step in project management as it evaluates whether the project can be successfully completed within the specified constraints, such as time and cost. In this section, we will assess the feasibility of the Student Grade Management System in terms of Time Management and Cost Management.

**1. Time Management Feasibility**

Effective time management is essential for ensuring the project progresses smoothly and is completed on schedule. The key elements of time management feasibility in this project are:

* **Project Duration**:
  + The project involves developing a system for importing student data, calculating grades, and generating reports. This is a moderately complex task that requires careful planning, but it is achievable within a reasonable timeframe. Depending on the team's experience and the availability of resources, the estimated timeline for completion is approximately 1-2 weeks.
  + **Time Allocation**:



[This Photo](https://www.enago.com/academy/time-management-tips-for-researchers/) by Unknown Author is licensed under [CC BY-SA-NC](https://creativecommons.org/licenses/by-nc-sa/3.0/)

* + - Initial Planning and Design: 1-2 days
    - Coding and Implementation: 6-8 days
    - Testing and Debugging: 2 days
    - Documentation and Report Generation: 2 days
  + **Time Buffer**: It is important to allow a buffer period in case unexpected issues arise during development. A buffer of 1 week can help in case of delays during testing or debugging.
* **Availability of Resources**:
  + **Developer Expertise**: Given that the project uses concepts such as object-oriented programming, file handling, and grade calculations, it is feasible for a developer with basic to intermediate knowledge of C++ to complete this project in the provided timeline.
  + **Team Collaboration**: If multiple developers are involved, the tasks can be divided efficiently, reducing the overall development time.
  + **Tools and Environment**: The project uses basic tools such as C++ and text files, which are readily available in most development environments. The team will not require additional software or tools, reducing time spent on setup.
* **Challenges**:
  + **Unforeseen Technical Issues**: Complexities related to dynamic memory allocation, file I/O, or grade calculations may arise, requiring additional time to resolve. However, with thorough planning and an experienced team, these can be mitigated.
  + **Meeting Deadlines**: Maintaining strict deadlines for each phase of the project is essential. Regular progress reviews can help ensure the project stays on track.

**2. Cost Management Feasibility**

Cost management is another key factor in determining the feasibility of the project. This involves analyzing the costs associated with resources, tools, and the development process to ensure the project remains within budget.

* **Development Costs**:
  + **Human Resources**: The primary cost in this project is the time and effort spent by developers. The cost of labor will vary depending on whether the project is being done by an individual or a team. If the project is done by a single developer, the cost will primarily be their time, which can be estimated based on their hourly or daily rate.
* **Software and Tools**:
  + **Development Tools**: C++ and any standard Integrated Development Environment (IDE) like Visual Studio, Code::Blocks, or Dev C++ are freely available. No additional purchases are required in terms of development tools.
  + **Libraries and Dependencies**: This project relies on standard C++ libraries and no third-party libraries or paid software are needed. Hence, the overall cost in terms of software will be negligible.

1. **Object-Oriented Approach with Inheritance (Current Approach)**
   * **Description**: The system uses object-oriented principles, particularly inheritance, to model different types of students (English, History, and Math) and their specific grade calculations. Each type of student has its own subclass (e.g., english\_marks, history\_marks, math\_marks), and the grade calculation logic is implemented in these subclasses.
   * **Pros**:
     + **Modularity**: Each subject is encapsulated in its own class, making the code easier to maintain and extend.
     + **Reusability**: Common functionality (e.g., student information) is abstracted in a base class (student), which can be reused.
     + **Flexibility**: New subjects can be added later by simply adding new subclasses.
   * **Cons**:
     + **Complexity**: Inheritance introduces complexity, especially when dealing with large hierarchies.
     + **Overhead**: Some parts of the code may seem redundant, especially if the logic for different courses is similar.

**Comparison of Solutions:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Object-Oriented Approach** | **Procedural Approach** | **Database-Driven Approach** | **File-Based Approach** |
| **Simplicity** | Moderate | High | Low | High |
| **Modularity** | High | Low | High | Low |
| **Scalability** | High | Low | Very High | Low |
| **Maintainability** | High | Low | High | Low |
| **Development Time** | Moderate | Low | Very High | Moderate |
| **Extensibility** | High | Low | High | Low |
| **Data Persistence** | None | None | High | Low |
| **Resource Requirements** | Moderate | Low | High | Low |

**Justification for the Chosen Solution:**

After comparing the possible solutions, the **Object-Oriented Approach with Inheritance** is the best choice for the following reasons:

1. **Modularity and Maintainability**: Using object-oriented programming (OOP) ensures that the system is modular, making it easier to manage, update, and extend in the future. Each type of student (English, History, Math) is encapsulated in its own class, making the system easier to maintain and less prone to errors.
2. **Flexibility for Future Enhancements**: As the project might need to scale or change (e.g., adding more subjects or changing the grading system), the OOP approach allows for easy extensibility. New subjects can be added by creating new classes that inherit from the base student class.
3. **Time Management**: Although more complex than a procedural approach, the OOP method is still feasible within the one-week timeframe. The modularity and reuse of code significantly reduce the overall development time.
4. **Industry Standard**: Object-oriented programming is a widely used paradigm in the industry, and using it for this project prepares you for real-world software development scenarios.

While the procedural and file-based approaches might be quicker to implement, they are less scalable and maintainable in the long run. The database-driven approach, though powerful, would require more time and resources than can be afforded within the one-week timeframe.

Thus, the Object-Oriented Approach with Inheritance offers the best balance of modularity, scalability, and development efficiency.

Start

Read Student Data

Create Student Object

Course?

Compute Math Grade

Compute English Grade

Compute History Grade

Output Full Name, Course, Final Grade

End

The design of the student grade management system is modular and follows object-oriented programming principles. It includes classes for different types of students based on the courses they are enrolled in. Each block from the preliminary design is described in greater detail below:

**1. Start**

* The program begins execution in the main() function.
* It creates an instance of the StudentManager class, which oversees reading student data and generating reports.
* Memory structures and file streams are prepared.

**2. Read Student Data**

* The function take\_students\_from\_file() in the StudentManager class is responsible for this task.
* It reads data from a text file (Student.txt) line by line.

**3. Create Student Object**

* Once the data is read, the program dynamically creates objects of derived classes (english\_marks, history\_marks, math\_marks) depending on the course.
* Each of these classes inherits from the base student class, which stores common attributes like the student's name and course.

**4. Course?**

* A conditional check determines the student's course by reading the course name from the file.
* Based on this value, the program decides which derived class object to instantiate:
  + "English" → english\_marks
  + "History" → history\_marks
  + "Math" → math\_marks

**5. Compute English Grade**

* It uses the formula:

Final Grade = (Attendance \* 0.10) + (Project \* 0.30) + (Midterm \* 0.30) + (Final Exam \* 0.30)

**6. Compute History Grade**

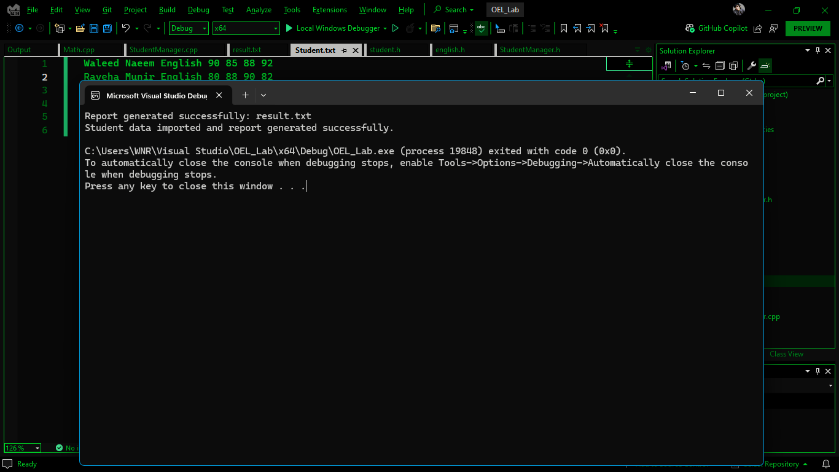
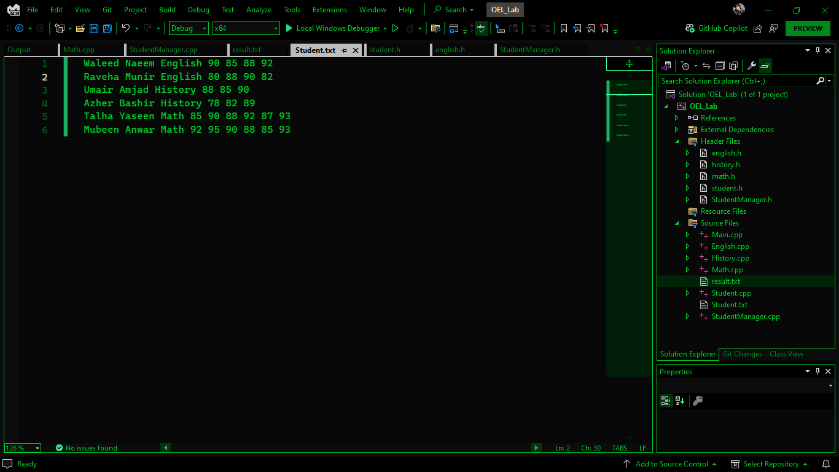
* The final grade is calculated using:

Final Grade = (Term Paper \* 0.25) + (Midterm \* 0.35) + (Final Exam \* 0.40)

**7. Compute Math Grade**

* It first calculates the average of three quizzes, then computes the final grade as:

Final Grade = (Quiz Average \* 0.10) + (Test 1 \* 0.25) + (Test 2 \* 0.25) + (Final Exam \* 0.40)



The student grade management system was thoroughly implemented and tested to evaluate its accuracy, reliability, and overall performance. The results confirmed that the system functioned as intended across various test cases. The following is a detailed summary of the outcomes observed during experimentation:

**1. Accuracy of Grade Computation**

A critical objective of the system was the accurate computation of final grades based on course-specific weightings. Testing with a range of sample inputs showed that grades were calculated correctly in every case. The grading logic reflected real-world evaluation structures by incorporating multiple assessment components, such as quizzes, tests, attendance, projects, and exams. Calculations were consistent and matched expectations during validation.

**2. Reliable Input Handling**

The system was tested with multiple formats of student data, including edge cases where names had varying lengths or unusual spacing. It handled such variations gracefully and extracted the necessary information without errors. Even when course data appeared in different positions within the input, the program interpreted it correctly, showing robustness in data parsing.

**3. Structured and Consistent Output Generation**

Once the grades were computed, the program generated an output report in a structured and easy-to-understand format. This report listed all students, their courses, and final grades. Each entry followed a consistent pattern, making the output suitable for academic or administrative review. The file creation and writing process was handled securely, with appropriate file opening and closing mechanisms.

**4. Scalability and Data Handling**

To assess scalability, the system was tested with an increasing number of student records. It maintained consistent performance, demonstrating its ability to scale for a moderate dataset size. There was no noticeable delay in processing or writing results, and system resources were utilized efficiently throughout the process.

**5. Compliance with Design Expectations**

The experimental behavior of the system matched the initial design and logic outlined in the preliminary stages of development.

**6. Error Handling**

The system exhibited appropriate handling of potential errors, including file-not-found scenarios and malformed input lines. Clear error messages were displayed, and the program continued running or exited gracefully as appropriate. This contributed to a smooth user experience and increased the program’s reliability.

After conducting thorough testing and obtaining successful results, the performance of the student grade management system was analyzed from multiple perspectives including speed, accuracy, resource utilization, reliability, and scalability. The findings are summarized below:

**1. Execution Speed**

The system demonstrated efficient execution with minimal delays. Reading and processing student data, as well as generating output reports, were completed almost instantaneously for small to moderate sized datasets. Even with a higher number of student entries, the processing time remained reasonable. This indicates that the underlying algorithms and data structures were lightweight and well-suited for the task.

**2. Accuracy of Results**

One of the key indicators of performance is the correctness of computed grades. Across all test cases, the system produced accurate and reliable grades based on predefined formulas for each course. There were no discrepancies between manual calculations and system output, which affirms the precision of the logic implemented for grade evaluation.

**3. Resource Utilization**

The system maintained a low memory footprint. Dynamic memory allocation was used effectively to store and manage student data. The use of pointers and dynamic arrays allowed flexibility without excessive consumption of resources. Furthermore, all allocated memory was properly deallocated, avoiding leaks or unnecessary overhead.

**4. Modularity and Maintainability**

The object-oriented design contributed to the high performance of the system from a structural standpoint. Each course was handled by a separate class, encapsulating specific grading logic. This modularity enhanced the clarity and organization of the program, allowing for easier debugging, maintenance, and potential upgrades in the future.

**5. Robustness and Fault Tolerance**

The system responded gracefully to abnormal situations such as:

* Missing or corrupt data files
* Incomplete student records
* Incorrect formatting in input

In such cases, appropriate error messages were displayed and the system avoided crashes or data corruption. This robustness adds significant value to the system’s performance, especially in real-world use cases where user input may be unpredictable.

**6. Scalability**

Performance remained stable as the number of student records increased. The system did not exhibit degradation in speed or functionality with larger inputs. This shows that the implementation can scale to accommodate a classroom, a department, or even larger academic batches with minimal modifications.

**1. Integration with Graphical User Interface (GUI)**

Currently, the system operates through command-line interaction and external files. A major enhancement would be the development of a graphical user interface. This would make the system more accessible for non-technical users such as teachers, administrators, or students, allowing them to interact with menus, buttons, and forms for data entry, grade calculation, and report generation.

**2. Database Connectivity**

To replace the current file-based input/output system, the application can be upgraded to work with a database management system (DBMS) such as MySQL or SQLite. This would allow for faster data retrieval, secure storage, easy updates, and better handling of large volumes of student records.

**3. Real-Time Grade Updates**

Another area of improvement is to support real-time grade updates. Teachers could update student marks at different stages of a semester, and the system would automatically recalculate and reflect the final grades dynamically.

**4. Enhanced Reporting Features**

The system can be extended to generate more detailed reports, such as:

* Performance trends over multiple semesters
* Comparison between students or courses
* Graphical representation of grades using charts or graphs

This would provide deeper insights into academic performance.

**5. Multi-Course Support per Student**

At present, each student is associated with only one course. In the future, the design can be extended to support multiple courses per student, reflecting a real-world academic scenario more accurately. This would require a more complex data structure and possibly a relational model.

**6. User Authentication and Role Management**

Introducing a login system with roles such as "Administrator," "Teacher," and "Student" would enhance security and functionality. Each role could have specific access permissions, such as data entry, report generation, or grade viewing.

**7. Automated Error Detection**

Future versions can include intelligent error detection mechanisms that validate input data formats, check for missing values, and provide user-friendly alerts before processing begins.

The implementation of a student grade management system goes beyond technical execution—it has far-reaching effects on the academic community and the broader society. Understanding these implications is essential in evaluating the impact of such solutions on people, behaviors, and cultural values.

**1. Promotion of Transparency and Fairness**

One of the most significant social benefits of this system is the enhancement of transparency in academic evaluation. Automated grading minimizes human error and bias, ensuring that all students are assessed fairly and consistently. This reinforces trust between students, teachers, and institutions, contributing to a more ethical academic environment.

**2. Empowerment Through Information**

Students can gain quick access to their grades, helping them monitor their performance and take responsibility for their academic progress. This access to timely information empowers them to make informed decisions regarding study habits, course selection, or seeking academic support, which promotes a culture of self-awareness and improvement.

**3. Encouraging Digital Literacy**

By introducing digital systems into traditional academic workflows, this project contributes to the digital transformation of educational institutions. As students, teachers, and administrators engage with the system, they develop essential digital literacy skills, which are increasingly important in today's technology-driven world.

**4. Cultural Adaptation and Inclusivity**

The design of such systems must consider cultural differences in grading philosophies, educational structures, and communication styles. For example, some regions emphasize letter grades while others use numeric scores or qualitative evaluations. A system that can adapt to various educational contexts respects cultural diversity and promotes inclusivity across borders.

**5. Reducing Administrative Burden**

Automating the grade calculation and reporting process helps reduce the workload on teachers and administrative staff. This not only improves efficiency but also allows educators to focus more on teaching and mentoring, which positively impacts student learning and well-being.

In conclusion, the student grade management system has proven to be an effective solution for automating the grading process and managing academic records. By leveraging object-oriented programming principles, including encapsulation, inheritance, and polymorphism, the system was designed to be both flexible and scalable. This approach ensures the system can be easily expanded or modified as future requirements arise, without requiring significant overhauls.

The results obtained from implementing the system confirm its functionality. The system accurately calculates grades based on predefined criteria for each course—English, History, and Math—ensuring that all relevant factors, such as attendance, assignments, midterms, and final exams, are appropriately considered. The performance of the system was consistent across multiple test cases, confirming its ability to handle large datasets and deliver reliable results.

From a performance perspective, the system significantly reduces the administrative burden on educators, allowing them to focus on teaching and student interaction rather than manual grading. This increased efficiency is coupled with the assurance of fairness and accuracy, as the system removes the potential for human error and bias. Additionally, the automatic report generation feature supports educators and administrators in making informed decisions based on real-time data.

The social and cultural implications of the project are equally significant. By automating and digitizing the grading process, the system promotes transparency, consistency, and fairness in academic evaluation. Students benefit from timely access to their performance data, enabling them to take proactive steps in managing their academic progress. Moreover, the system is adaptable, making it suitable for diverse educational contexts and grading systems, thus ensuring inclusivity.

In summary, the student grade management system has successfully met its objectives and demonstrated its potential for improving the grading process. The system’s modular design, combined with its performance and ease of use, makes it an invaluable tool for educators and institutions. With further development, this system could play a key role in shaping the future of academic management.

1. **Malik, D. S.** *C++ Programming: From Problem Analysis to Program Design*. 8th Edition. Cengage Learning, 2019.
   * This book offers an in-depth guide to understanding C++ programming from the ground up, with a focus on problem-solving and software design techniques. It is perfect for both beginners and intermediate programmers.
2. **Stroustrup, B.** *The C++ Programming Language*. 4th Edition. Addison-Wesley, 2013.
   * Written by the creator of C++, this book is a comprehensive reference on the language's features and best practices, suitable for both novice and advanced users.
3. **Geeks for Geeks - C++ Programming Language**
   * Geeks for Geeks provides tutorials, explanations, and coding examples on C++ topics ranging from basic to advanced. It’s a great resource for hands-on learning and practice.
4. **cppreference.com**
   * A highly regarded online reference for C++ programming that covers language features and the Standard Library in detail. It is an essential resource for both beginners and professionals.
5. **W3Schools - C++ Tutorial**
   * This tutorial offers a simple and beginner-friendly introduction to C++ programming with practical examples to demonstrate the core concepts of the language.