Problem Statement

University Management System

You are tasked with developing a University Management System that facilitates the organization of students, teachers, courses, and departments within a university. The system should allow the creation, management, and display of information related to these entities.

Requirements:

Person Class:

1. A base class `Person` should represent a generic person with a name.
2. It should have a method `displayInfo()` to display the person's information.

Student Class:

1. Derived from `Person`, the `Student` class should include additional attributes such as `rollNumber` and `grade`.
2. Implement a method `displayGrade()` to display the student's grade.

Teacher Class:

1. Derived from `Person`, the `Teacher` class should include attributes like `title` and `subject`.
2. Implement a method `displayInfo()` to display the teacher's information.

Course Class:

1. The `Course` class should include attributes like `courseCode`, instructor (of type `Person`), and a list of `students` (of type `Student`).
2. Implement methods to add students to a course and display the course information.

Department Class:

1. The `Department` class should include a `departmentCode` and a list of `persons` (of type `Person`), which can include both students and teachers.
2. Implement a method to display department information.

Additional Considerations:

Ensure proper encapsulation and data hiding by using access specifiers appropriately.

Implement a user interface or a test program to demonstrate the functionality of your classes.

Objectives of the Project:

1.Object-Oriented Modeling:

Implement classes to model persons, students, teachers, courses, and departments in a university setting.

2.Inheritance and Polymorphism:

Use class inheritance to share common attributes from a base class (`Person`), and showcase polymorphism by overriding methods in derived classes (`Student`, `Teacher`).

3.Composition:

Demonstrate composition by including objects of one class within another (e.g., `Person` in `Department`, `Student` in `Course`).

4.Dynamic Memory Allocation:

Utilize dynamic memory allocation using pointers and `new` to manage the storage of objects.

5.User Interaction and Information Display:

Provide a console-based interface for user interactions, allowing input of information and displaying details about persons, courses, and departments.

6.Information Display:

Console Output: Display relevant information about persons, courses, and departments through appropriate console outputs.

Description of the outcomes

The C++ Student Management System is a comprehensive console-based application developed to address the multifaceted needs of educational institutions in managing student and teacher information, exam marks, and class details. This project leverages the power of C++ and incorporates a range of object-oriented programming (OOP) principles to achieve a modular and extensible design.

Features

Student Management

Addition and Retrieval of Student Information:

The system facilitates the addition of students with detailed information including name, department, section, admission number, and mobile number.

The ability to retrieve student details based on class, section, and department.

Dynamic Class and Section Handling:

The application dynamically manages class and section creation, ensuring seamless organization of students into different sections based on their departments

Teacher Management

Teacher Information Management:

- Teachers can be added to the system with comprehensive details such as name, department, mobile number, designation, and ID number.
- The authentication system ensures secure access for teachers using their credentials.

Teacher-Specific Functionalities:

- Teachers can log in and perform tasks like adding students, updating marks, and viewing student details.

Mark Entry:

- Teachers have the capability to enter marks for various exams, with the system automatically calculating total marks and percentages for each student.

Class and Section Handling

File Management:

- The application employs file handling to create and manage class and section-specific files for storing student details and exam marks.

Department-Specific Processing:

- Different sections within a department (e.g., A, B, C) are handled separately, ensuring efficient organization and retrieval of information.

Code Structure:

- The project adopts a modular structure with several classes, each dedicated to specific functionalities, promoting code reusability and maintainability. A brief overview of key classes:

viewstudent Class

- Responsible for displaying student details based on the provided class and section.

markview Class

- Manages the viewing of marks, ranking students, and displaying results for a given exam.

marks Class

- Inherits from markview and provides additional functionalities for entering marks for different departments and sections.

sectionprocess Class

- Manages the creation of section-specific files for each department, ensuring organized data storage.

studentprocess Class

- Inherits from sectionprocess and processes student information based on their department, promoting data integrity.

addstudent Class

- Inherits from studentprocess and facilitates the addition of new students, enhancing the system's flexibility.

addteacher Class

- Handles the addition of teachers with relevant details, ensuring comprehensive teacher management.

studentlogin Class

- Manages student logins and provides functionality for viewing exam results securely.

student Class

- Inherits from studentlogin and extends functionality for viewing specific student details and exam marks.

teacherlogin Class

- Manages teacher logins, allowing them to perform various tasks like adding students, updating marks, and viewing details, ensuring data security.

Execution

- The application is designed for ease of execution in a console environment. Users interact with the system by providing inputs based on the displayed prompts.

Conclusion

- The C++ Student Management System project serves as a comprehensive solution for educational institutions, demonstrating a robust implementation of OOP concepts, file handling, and user input processing in C++. Its modular and extensible design makes it suitable for further expansion with additional features.

Future Enhancements

- While the current implementation offers a solid foundation, future enhancements could include:
- Enhanced Authentication: Implementing more robust authentication mechanisms for both students and teachers.

Graphical User Interface (GUI):

- Transitioning to a GUI-based application for improved user experience.

Database Integration:

- Integrating a database system for enhanced data management and scalability.

Additional Reports:

- Generating more detailed reports on student performance, attendance, and overall academic progress.

Notification System:

- Implementing a notification system for teachers and students regarding important announcements and updates.