C.W
Task 1:

You are building a multi-tier banking application with various account types (Savings, Checking, Business, and so on). Some accounts may have specific operations (e.g., overdraft protection, interest calculations, corporate tax rules). The system also has different user roles (e.g., Customer, Teller, Manager) that might control how certain operations are accessed.

Key Requirements & Challenges:

1. Create a base class BankAccount to model common functionality (account number, balance, deposit, withdraw).
2. Derive multiple classes—e.g., SavingsAccount, CheckingAccount, BusinessAccount—each adding specialized behavior:
   - SavingsAccount applies monthly interest.
   - CheckingAccount allows overdraft (but logs an alert if overdraft limit is exceeded).
   - BusinessAccount might withhold corporate tax on every deposit.
3. Introduce user-role classes (Customer, Teller, Manager) that have different privileges:
   - A Customer can only deposit or withdraw from his/her accounts.
   - A Teller can deposit, withdraw, or freeze accounts.
   - A Manager can do all of the above plus override or adjust account limits.
4. Explore inheritance design for these roles (e.g., an Employee base class from which both Teller and Manager derive).
5. Demonstrate **polymorphism** by maintaining a collection of base pointers to different account types and roles, then performing operations in a loop to show dynamic dispatch.

Task 2:

You are building a media library for different file types: Audio, Video, and Image. Certain file types share attributes (e.g., resolution for video and images, or sample rate for audio and video if it's a music video track). You have to build an inheritance hierarchy that might have a diamond shape (e.g., MediaFile as a base, specialized classes branching off, then merging back).

Key Requirements & Challenges:

1. Define a base class MediaFile with properties like file path, size, and basic operations such as play() and stop().
2. Create intermediate classes:
    - VisualMedia (for images and videos)
    - AudioMedia (for audio and videos)
3. Create VideoFile that (in a naive design) might inherit from both VisualMedia and AudioMedia. This leads to a diamond scenario with MediaFile at the top.
4. Use **virtual base classes** to avoid duplicating the MediaFile sub-object in VideoFile. Show how you must call the correct constructor(s) in the derived classes.
5. Demonstrate how **method resolution** works in the presence of multiple inheritance. Show how you handle potential function collisions (e.g., play(), stop()).
6. Test the final hierarchy by storing pointers of type MediaFile* that actually point to VideoFile, ImageFile, or AudioFile objects, then calling virtual methods.

Task 3:

Build a system to manage orders in a restaurant. There are different types of employees—Waiter, Chef, Cashier—and different menus, like FoodMenu and BeverageMenu. You want to demonstrate both interface-style (pure virtual) classes and implementation inheritance.

Key Requirements & Challenges:

1. Define a base class Employee that outlines minimal employee info (ID, name).
2. Create abstract interfaces (pure virtual classes) for behaviors such as:
   - IOrderTaker (someone who can take orders)
   - IOrderPreparer (someone who can prepare orders)
   - IBiller (someone who can generate the bill)
3. Make Waiter implement IOrderTaker, Chef implement IOrderPreparer, and Cashier implement IBiller.
4. Some restaurants might allow a single employee to play multiple roles, which can be shown using **multiple inheritance**—for instance, a Manager might be both an IOrderTaker and an IBiller.
5. Create classes representing menus (FoodMenu, BeverageMenu) that share from a base Menu class but differ in how they calculate total costs (e.g., applying beverage taxes or combos).
6. Use **polymorphism**: store a list of Employee* or IOrderTaker* to handle tasks generically and illustrate dynamic dispatch in action.

H.W

Task - 01:

A school library wants to organize its library system by categorizing books according to their genre.

They need an automated system that will allow them to input the details of the books that are in

their library. To do this, you need to implement a program that contains a base class called Books

that will contain a data member to store the genre of the book. Derive two other classes from the

base class and name them accordingly. Each of these two classes will hold details about a book

from a specific genre of your choice such as Novel, Narrative, Mystery and so on. The derived class

will contain data members to store the title and the author of the book. Display the details of each

book along with their genre.

Task - 02:

A vehicle company is deciding to hire a programmer to develop a system that will allow the

company to enter the details of the vehicles sold by them. As a programmer, you need to implement

a program that contains a base class called Vehicles that contains a data member to store the price

of the vehicles. Derive two other classes named as Car and Motorcycle.

1. The Car class will contain data members to store details that include seating capacity,

number of doors and fuel type (diesel or petrol).

2. The Motorcycle class will contain data members to store details such as the number of

cylinders, the number of gears and the number of wheels.

Derive another subclass named as Audi of Car and Yamaha of Motorcycle.

3. The Audi class will contain a data member to store the model type.
4. The Yamaha class will contain a data member to store the make – type.

Display the details of an Audi car (price, seating capacity, number of doors, fuel type, model type)

and the details of the Yamaha motorcycle (price, number of cylinders, number of gears, number
of

wheels, make – type).

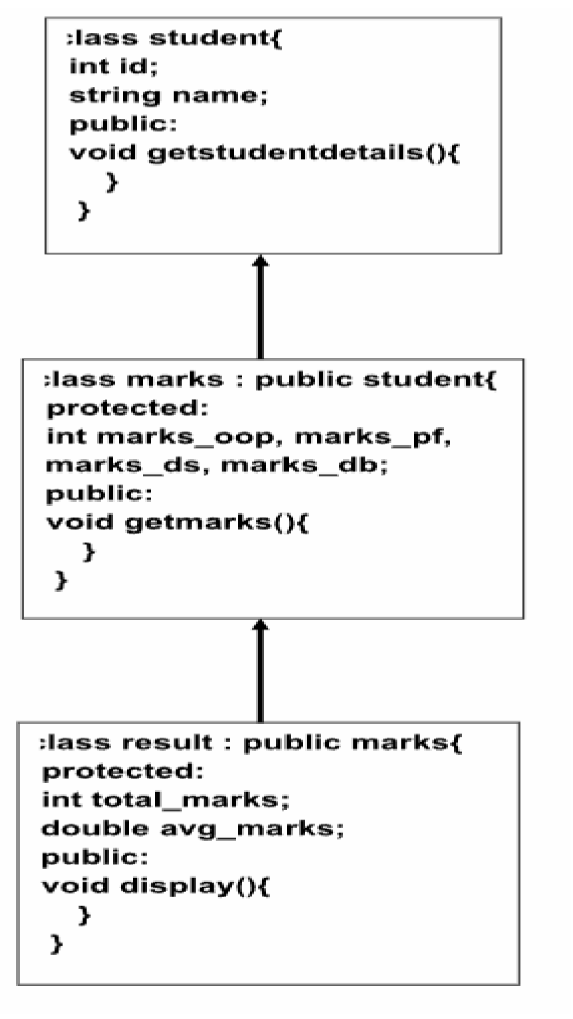
Task 03:

Task - 03:

A university is deciding to upgrade its system. To upgrade, you need to implement the following scenario as shown in the figure:

Note the following:

1. The class student has a function that displays all the information about the student.
2. Class marks is derived from class student and has a function that displays all the marks obtained in the courses by the students.
3. Class result is derived from class marks. This class has a function that calculates the total
4. marks and then calculates the average marks. It then displays both the total and the average marks.
   In the main function you are required to do the following:
5. Create an object of the result class.
6. Then display the student details, the marks obtained in each courses and the totaland the average marks.



```
:lass student{
int id;
string name;
public:
void getstudentdetails(){
   }
 }
```

```
:lass marks : public student{
protected:
int marks_oop, marks_pf,
marks_ds, marks_db;
public:
void getmarks(){
   }
 }
```

```
:lass result : public marks{
protected:
int total_marks;
double avg_marks;
public:
void display(){
   }
 }
```