**OOP Assignment 2**

**Question 1:**

You are tasked with building a library management system that handles books, magazines, and other types of reading material. Here's the problem: you have a base class called “**ReadingMaterial**” that defines basic attributes like “title” and “author”. Next, you create two derived classes: “Book” and “Magazine”. These classes extend the “ReadingMaterial” base class but implement their own unique methods for managing specific features—"**Book**” might have readPage() for reading a specific page, while “**Magazine**” could have browseSection() for browsing different sections.

Now, you want to create a third class called “ReferenceMaterial” , which represents materials used for research or academic purposes. This class needs to inherit from both `Book` and `Magazine` because it combines features from both, leading to the Diamond Problem.

Moreover, you also have overlapping attributes and methods that cause variable and function shadowing issues. Both “Book” and “Magazine” have a “pageCount” variable representing the number of pages or sections. Each also has its own method to calculate some metric: calculateReadTime() for “Book” and calculatePageTurnSpeed() for `Magazine`. When ReferenceMaterial inherits from both, it has to decide how to handle these conflicting attributes and methods.

**Task:**

Create four classes: ReadingMaterial , Book , Magazine , and ReferenceMaterial. Use virtual inheritance to resolve the Diamond Problem and implement solutions to handle variable and function shadowing issues.

* The ReadingMaterial class should contain common attributes and methods for all reading materials.
* The Book class should represent reading materials with chapters and pages. Include unique methods and attributes specific to books.
* The Magazine class should represent reading materials with sections and articles. Include unique methods and attributes specific to magazines.
* The ReferenceMaterial class should inherit from both `Book` and `Magazine`, representing a reference guide that combines features of both books and magazines.

Resolve the Diamond Problem, variable shadowing, and function shadowing by ensuring the derived classes work without ambiguity or conflicts.

**Question 2**

From Eye in the Sky to Cargo Transport: Creating a Versatile UAV"

You are working for a UAV manufacturer company that has designed a basic UAV

model with some common features.

UAV Class

**Methods:**

takeOff(), land()

The company now wants to create two variants of the basic model to target different

markets - one for surveillance and the other for cargo delivery. Both variants has

now their own additional properties and behaviours.

**SurvellienceUAV Class**

**Attributes:**

Name, weight, maxAlitude, zoomLevel, recordingCapacity, isNightVision

**Methods:**

captureImage(), recordVideo()

**CargoDeliveryUAV**

**Attributes:**

Name, weight, maxAlitude, cargoArea

**Method:**

deliverCargo()

The company has decided to create a third UAV model that combines the features

of both variants and can perform both surveillance and cargo delivery missions. This

model is called the "Multi-Purpose UAV" and it should inherit from both the

surveillance and cargo delivery variants. This variant has a solar panel attached to it

for high power, as it has to deliver and also do the surveillance task.

**Class MultiPurposeUAV**

**Attributes:**

Name, weight, maxAlitude, solarPanelEfficiency;

**Methods:**

deliverCargoWithSurvellience()

This method will call the methods in the following order:

* takeOff()
* recordVideo()
* land()
* deliverCargo()

In main, create objects of SurveillanceUAV, CargoDeliveryUAV, and

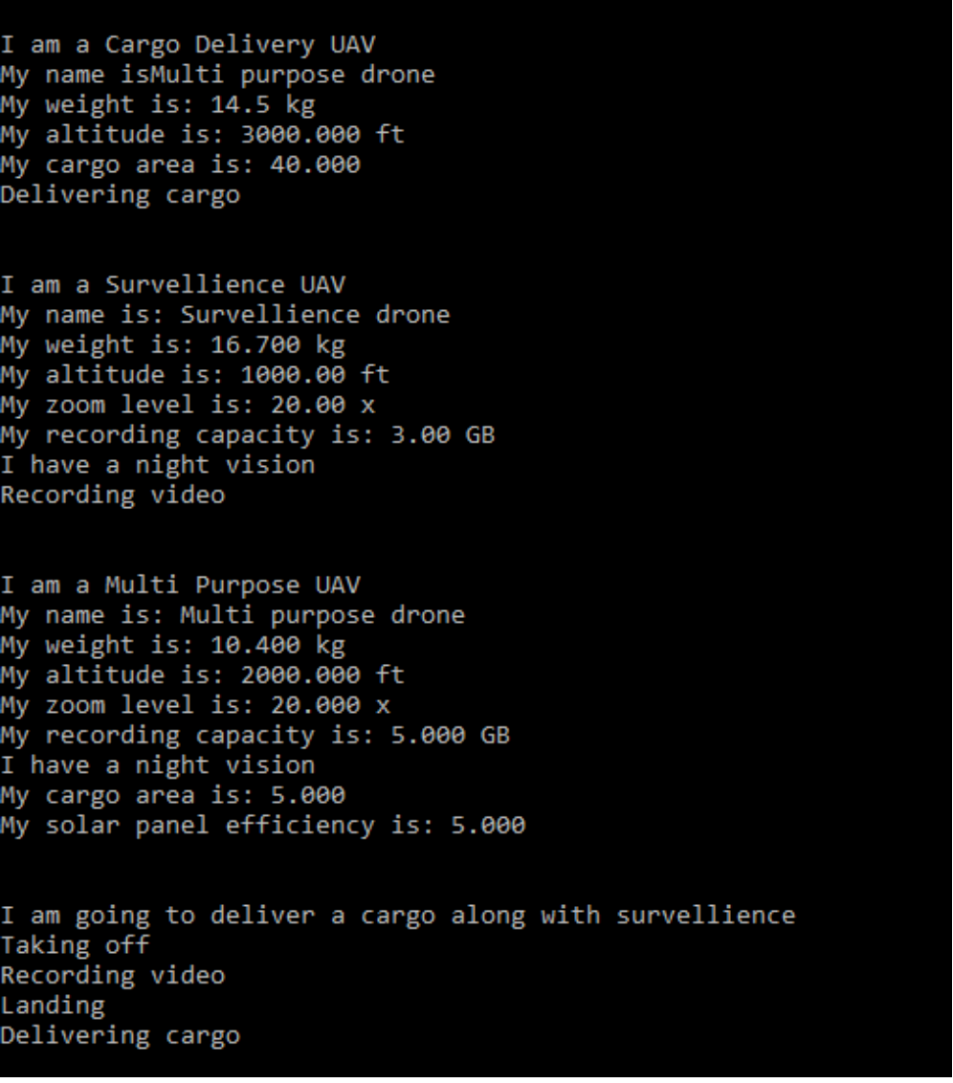
MultiPurposeUAV. Call the display() method of the objects. Plus call the

deliverCargo() with CargoUAV object, recordVideo() with SurveillanceUAV

object, and deliverCargoWithSurveillance() with multi functional UAV object.

Note: Resolve the diamond problem for land() and takeOff() methods in this task.

**Sample Output:**



**Question 3**

A game framework features a class hierarchy for various board types, but suffers from variable and function shadowing issues, leading to ambiguity in attribute access and method behavior. The use of mutable in const methods also raises concerns about board state integrity.

Resolve these issues while maintaining the class structure and const-correctness and rewrite the code below.

**Code:**

#include <iostream>

#include <array>

using namespace std;

// Base class representing a generic game board with fixed size

class GameBoard {

public:

    const array<array<int, 4>, 4> dimensions; // Fixed 4x4 array for board dimensions

    GameBoard() : dimensions({{ {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0} }}) {}

    void display() const {

        cout << "Base GameBoard Display:" << endl;

        for (const auto& row : dimensions) {

            for (const auto& cell : row) {

                cout << cell << " ";

            }

            cout << endl;

        }

    }

};

// Derived class for a puzzle board

class PuzzleBoard : public GameBoard {

public:

    array<array<int, 4>, 4> dimensions; // Shadowing the base class variable

    PuzzleBoard() : GameBoard(), dimensions({{ {1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1} }}) {}

    void display() const {

        cout << "PuzzleBoard Display:" << endl;

        // Modifying a mutable copy of the board despite the base class being const

        mutable auto modifiableBoard = dimensions;

        modifiableBoard[0][0] = 2; // Example modification for display

        for (const auto& row : modifiableBoard) {

            for (const auto& cell : row) {

                cout << cell << " ";

            }

            cout << endl;

        }

    }

};

// Derived class for a battle board

class BattleBoard : public GameBoard {

public:

    array<array<int, 4>, 4> dimensions; // Shadowing the base class variable

    BattleBoard() : GameBoard(), dimensions({{ {2, 2, 2, 2}, {2, 2, 2, 2}, {2, 2, 2, 2}, {2, 2, 2, 2} }}) {}

    void display() const {

        cout << "BattleBoard Display:" << endl;

        mutable auto modifiableBoard = dimensions;

        modifiableBoard[1][1] = 3; // Example modification for display

        for (const auto& row : modifiableBoard) {

            for (const auto& cell : row) {

                cout << cell << " ";

            }

            cout << endl;

        }

    }

};

int main() {

    GameBoard baseBoard;

    baseBoard.display(); // Displaying the base game board

    PuzzleBoard puzzleBoard;

    puzzleBoard.display(); // Displaying the puzzle board

    BattleBoard battleBoard;

    battleBoard.display(); // Displaying the battle board

    return 0; }