

OBJECT ORIENTED PROGRAMMING

Lab Assignment: Multiple and Hybrid Inheritance; Diamond Problem; Virtual Base Class; Interfaces; MRO; Mixins

24 Sept 2026

1. Design a class **AcademicRecord** that stores a student's roll number and marks obtained in three subjects, and another independent class **SportsRecord** that stores sports scores and player ranking. Create a derived class named **StudentProfile** using multiple inheritance. Write member functions to accept details from standard input and display a comprehensive report sheet, demonstrating proper constructor delegation in multiple inheritance.
2. Implement an educational hierarchy to illustrate the diamond problem. Create a base class **Person** with attributes **name** and **age**. Derive two intermediate classes, **Academician** and **SportsPerson**, each containing relevant performance metrics. Derive a class named **AllRounder** from both intermediate classes without using the **virtual** keyword. Demonstrate the compilation errors that occur due to ambiguity when calling members of **Person**, and explain how the scope resolution operator can temporarily resolve this issue.
3. Refactor the **Person** diamond hierarchy from Question 2 using **virtual** base class inheritance. Write a program to show that only a single instance of **Person** exists inside an **AllRounder** object. Explicitly initialize the virtual base class from the most derived class's constructor, and display the complete flow of constructor and destructor invocations.
4. Model an automated teller machine (ATM) transaction system using pure virtual functions to represent an interface. Define an abstract class **ITransaction** that declares pure virtual methods **executeTransaction()** and **generateReceipt()**. Implement two concrete classes, **CashWithdrawal** and **FundTransfer**, that override these interface methods. Write a driver routine that accepts a pointer to **ITransaction** and demonstrates runtime polymorphism.
5. Investigate how Method Resolution Order (MRO) works in C++ when multiple base classes share identical member function signatures without virtual inheritance. Create two base classes, **Printer** and **Scanner**, each having a method **void executeJob()**. Derive a hybrid device class **AllInOneMachine** from both. Demonstrate why a direct call to **executeJob()** causes compiler ambiguity, and resolve this ambiguity inside the derived class by selectively overriding and delegating calls.
6. Build an order fulfillment and discount computation system using hybrid inheritance. Create a base class **Order** (storing order ID and total amount). Derive a class **ShippedOrder** (adding shipping rates) and a separate class **CustomerTier** (storing discount slabs). Finally, derive **FinalInvoice** from both **ShippedOrder** and **CustomerTier**. Implement member functions to compute the final payable price and print an itemized bill.
7. Implement a mixin pattern using C++ templates (Curiously Recurring Template Pattern or parameterized inheritance) to augment class capabilities without modifying existing class code. Create a base domain class **Account** (storing account balance). Develop a mixin class template **Auditable<T>** that adds auditing functionality (logging timestamps and tracking operations) by deriving from the template type parameter. Instantiate and test **Auditable<Account>**.
8. Implement multiple interfaces in a single C++ class. Create two pure abstract base classes: **ISerializable** (with member function **serialize()**) and **ILoggable** (with member function **logMessage(string)**). Derive a class **SensorDevice** from both interfaces

and provide full concrete definitions for all declared methods. Demonstrate calling these methods using base-interface pointers.

9. Analyze constructor and destructor execution order in complex multiple and virtual inheritance. Construct a hierarchy involving a virtual base class `CoreEngine`, derived intermediate classes `ElectricDrive` and `CombustionDrive`, an auxiliary utility class `DiagnosticUnit`, and a most derived class `HybridVehicle`. Include output statements inside every constructor and destructor to display the precise sequence in which sub-objects are initialized and cleaned up.
10. Create a reusable logging mixin class `TimestampedLog` that adds formatted timestamps to console output. Integrate this mixin alongside a pure virtual interface `INetworkService` to implement an enterprise class `SecureConnectionService`. Ensure the derived class adheres strictly to the interface contract while reusing utility code provided by the mixin.
11. Implement a university payroll system utilizing hybrid inheritance with virtual base classes. A root class `Employee` contains member data `empId` and `basicSalary`. Intermediate classes `TeachingStaff` (adding research allowance) and `AdministrativeStaff` (adding executive allowance) inherit virtually from `Employee`. A derived class `Dean` inherits from both intermediate classes. Write functions to calculate total compensation without duplicating the base `Employee` state.
12. Write a program that implements multiple inheritance where two base classes declare virtual functions with the exact same signature: `virtual void validate() = 0;`. Create base interfaces `IHardwareValidator` and `ISoftwareValidator`, and implement them in a class `SystemDiagnostics`. Demonstrate how C++ handles this scenario and evaluate how the derived class provides an implementation that satisfies both contracts simultaneously.