PART A

1. In your own words, define encapsulation. Include an example of how it can prevent unintended changes to data.

Encapsulation is a concept in programming where you bundle data and the methods that operate on that data into a single unit, typically a class, and control access to that data from the outside world. It also restricts direct access to some of the class's components, which helps prevent unintended or harmful changes to the data. Example showing the use of encapsulation in the Vehicle class with:

```
// Base class: Vehicle
class Vehicle {
protected:
    double speed; // Protected member variable to store the speed of the vehicle

public:
    // Method to accelerate the vehicle by a certain amount
    void accelerate(double amount) {
        if (amount > 0) { // Only allow positive acceleration values
            speed += amount; // Increase the speed by the given amount
        }
    }
};

// Derived class: Car inherits from Vehicle
class Car : public Vehicle {
public:
    // Method to boost the car's speed
    void boostSpeed(double amount) {
        accelerate(amount);  // Accessing protected method from the base class to increase speed
    }
};
```

2. Visibility Modifiers: Compare public, private, and protected access. For each, list one benefit and one potential drawback (in terms of code flexibility, safety, or maintainability). Name a scenario in which you might intentionally use protected members instead of private members.

Public Access

Benefit: Allows unrestricted access, providing flexibility.

Drawback: Risks unintended modifications, compromising safety.

Private Access

Benefit: Ensures data safety by limiting access to within the class.

Drawback: Reduces flexibility, requiring getter/setter methods to interact with data.

Protected Access

Benefit: Allows derived classes to access and modify data, promoting maintainability in inheritance.

Drawback: Less strict encapsulation, as subclasses can still access members.

Scenario for Protected

Use protected when you want derived classes to access and modify specific internal data directly, such as in a class hierarchy like Vehicle, where subclasses (like Car, Truck, or Motorcycle) need to modify shared data like color, model, or engineType.

3.Impact on Maintenance:Explain why encapsulation can reduce debugging complexity when maintaining a large codebase.Provide a brief example (no code needed, just a scenario) of how code could break if internal data is made public.

Encapsulation hides internal data and allows access only through controlled methods. This reduces the risk of unintended changes and makes it easier to track bugs, as data can only be modified in specific ways.

If the Vehicle class has a public data member, like speed, it could lead to issues:

Problem: If the speed is public, any part of the code can modify it directly, even setting an invalid value (setting speed to a negative value).

Impact: A method that expects speed to always be a positive number (like accelerate()) could fail or produce incorrect behavior because of unexpected values.

4.Real-World Analogy:Think of a real-life object or system. How would you describe its “public interface” vs. its “private implementation”? Why is it helpful to keep the private side hidden?

Real-World Analogy: Class Dates

Public Interface: The calendar we see and interact with, showing days, months, and years. We can pick a date to schedule events without worrying about the details.

Private Implementation: The complex rules behind dates, like leap years, month lengths, and time zone calculations, which determine how dates are managed and calculated.

Why Keep the Private Side Hidden?

Simplicity: We don't need to understand the rules; we just use the calendar.

Accuracy: Hiding complexity ensures dates are handled correctly without errors.

Safety: It prevents users from accidentally altering important date calculations.

PART B

1. Class Skeleton:Outline a small class C++ that represents a BankAccount.Include:Two private data members (e.g., balance, accountNumber).One or two public methods that allow interaction with the balance (e.g., deposit or withdraw).

```
class BankAccount {
private:
    double balance;          // Private data member for balance
    int accountNumber;       // Private data member for account number

public:
    // Method to deposit money into the account
    void deposit(double amount) {

    }

    // Method to withdraw money from the account
    void withdraw(double amount) {

    }
};
```

2. Encapsulation Justification:For each private data member, explain in 1–2 sentences why it should be kept private.For each public method, explain how it enforces constraints or validations before modifying any private data.

Private Data Members :

balance:Keeping balance private ensures it cannot be changed directly from outside the class, protecting the integrity of the account's financial data.

accountNumber:Keeping accountNumber private ensures that it cannot be accessed or modified externally, preserving security and uniqueness of each account.

Public Methods :

deposit(double amount):This method ensures that only positive amounts are deposited, preventing invalid or negative deposits.

withdraw(double amount):This method ensures the withdrawal amount is positive and does not exceed the available balance, preventing overdraws.

3.Briefly show how you would document the class or methods so other developers understand they must not directly manipulate the balance.

```
class BankAccount {
private:
    double balance;         // Private data member for balance
    int accountNumber;      // Private data member for account number

public:
    // Method to deposit money into the account
    void deposit(double amount) {

    }

    // Method to withdraw money from the account
    void withdraw(double amount) {

    }
};
```

Class Documentation: Explains the encapsulation of the balance and accountNumber, and why direct manipulation is not allowed.

Method Documentation: Clearly states that the deposit and withdraw methods are the only allowed ways to modify the balance, ensuring validation is performed before making any changes.

PART C

1. Pros and Cons:List two benefits of hiding internal data behind methods, and one potential limitation or overhead introduced by this design approach.Encapsulation vs. Other ConceptsHow does encapsulation differ from abstraction?Why might we consider both “encapsulation” and “abstraction” to be forms of “information hiding”?Testing Encapsulated Classes:If the data is private, how can we still unit test the class thoroughly? Propose a short strategy that ensures valid testing without exposing private data.

Pros and Cons of Hiding Internal Data Behind Methods

Benefits:It ensures data integrity by allowing modifications only through validated methods, preventing invalid data.It simplifies usage by hiding complex implementation details, providing a cleaner interface.

Limitation:It can introduce minor performance overhead due to the additional method calls needed to access or modify data.

Encapsulation vs. Abstraction: Encapsulation hides the internal state of an object, ensuring controlled access, while abstraction hides the complex implementation details, exposing only essential functionality. Both are forms of information hiding, as they restrict external access to inner workings, ensuring safer and simpler usage.

Testing Encapsulated Classes:To test a class with private data, focus on testing its public methods, ensuring they behave correctly under various conditions without needing access to the private data directly.

PART D

1. This part is optional and can be completed for extra credit or deeper understanding.

Encapsulation in Various Languages:Compare how two programming languages (e.g., Java vs. C++) handle visibility modifiers. Do they differ in how they define public, private, or protected?

Encapsulation in Large-Scale Systems:Find a short online reference or article discussing how large companies ensure critical data is protected within their codebase. Summarize the key points in a paragraph.

In Java:

Public: Accessible everywhere.

Private: Accessible only within the same class.

Protected: Accessible within the same package and also in subclasses

Default :"Package-private," accessible only within the same package.

Java enforces encapsulation strictly via these keywords, and there’s no way to bypass them without reflection.

In C++:

Public: Accessible everywhere.

Private: Accessible only within the same class.

Protected: Accessible in the class and its subclasses

Differences: C++ doesn’t have a package-level default like Java; everything is either public, private, or protected. Also, C++ allows "friends" (functions) to access private/protected members, breaking encapsulation explicitly if declared, which Java doesn’t permit natively.

Encapsulation in Large-Scale Systems: Companies like Google protect their data by using strict rules for how code is written. They limit who can access sensitive data using visibility controls like private or protected in languages such as C++ and Java. They also do regular code reviews and use tools to catch any mistakes where private data might be accessed inappropriately. Their code is organized into separate parts, and APIs are used to control how different parts of the code communicate with each other. To further protect critical data, only certain teams are allowed to modify it through access control lists (ACLs). This system helps prevent errors, keeps the data secure, and ensures the code stays manageable as it grows.