Object Oriented Programming with C++ Laboratory Manual

2025-26

Practical Component	
PART – A **FIXED SET OF EXPERIMENTS**	
1	You are building a feature for an e-commerce platform that displays product prices. Write a C++ program that accepts a list of product prices and displays them sorted both in ascending (low to high) and descending (high to low) order, allowing customers to choose how they want to view the prices." (Use basic programming concepts like looping constructs for menu driven programming, arrays and sorting logic etc.)
2	As part of a student management system, you are tasked with designing a module to generate student performance reports. Create a C++ program using classes that captures a student's name, roll number, and marks in three subjects. The program should then calculate and display the student's total score along with their details. (Use C++ concepts like classes, constructors and methods etc.)
3	You are developing a contact management application where each contact contains details such as name, phone number, and email address. To ensure safe and accurate duplication of contact records (e.g., for backup or editing), implement a C++ class Contact that: a. Includes a copy constructor to properly copy all data members from one contact object to another. b. Also includes a destructor to manage clean-up when a contact object is destroyed (e.g., to release memory or display a message indicating the object is being deleted). Demonstrate the creation of an original contact and a copy using the copy constructor, display their contents, and show how the destructor is automatically invoked when objects go out of scope. (Use C++ concepts like classes, constructors, copy constructor, destructor etc.)
4	Develop a loan eligibility evaluation system for a bank. The system has two classes: Customer – contains private data members such as name, account balance, and credit score. Loan – is responsible for evaluating whether the customer is eligible for a loan based on specific criteria (e.g., balance ≥ ₹50,000 and credit score ≥ 750 out of 1000). Since the Loan class should not directly access the private members of the Customer class, implement a friend function named checkEligibility() that can access the private data of Customer and evaluate the eligibility. (Use C++ concepts like classes, constructor, Friend functions etc.)
5	You are developing a banking application that enables customers to manage their accounts efficiently. As part of the system implement the following functionality: a. Implement function overloading to perform addition of both integer-based amounts (e.g., ₹500 + ₹200) and floating-point values (e.g., ₹1000.75 + ₹499.25), ensuring accuracy across transaction types like deposits and interest calculations. b. Use operator overloading to redefine the unary minus (-) operator, allowing quick reversal of transaction values — for example, converting a credit transaction into a debit during a refund or correction. (Use concepts like classes, function overloading and operator overloading etc.)
6	You are designing a modular calculator system where arithmetic functionalities are separated into logical units. One class provides addition functionality, another provides subtraction, and a third class inherits from both to perform a full arithmetic operation on two numbers. Implement a C++ program using multiple inheritances to demonstrate this system. (Use concepts like classes, constructors, multiple inheritances etc.)

7	"You're working on a scientific computation module where different values—such as alpha, beta, and gamma—represent coefficients in a formula. These values come from different sources: a base class initializes alpha and beta, while the derived class introduces gamma. Write a C++ program using constructors in derived classes to properly initialize and display all three values, showcasing inheritance and constructor chaining." (Use concepts like classes, constructors, inheritance and constructor chaining etc.)

PART – B

OPEN ENDED EXPERIMENTS

Students should develop a program for the given scenario by the course teacher on the following concepts.

1. File operation in C++
2. Built-in & User Defined Exception Handling

Textbooks:

1. Herbert schildt, The Complete Reference C++, 4 th edition, TMH, 2005.

References:

1. Balagurusamy E, Object Oriented Programming with C++, Tata McGraw Hill
2. Education Pvt.Ltd., Sixth Edition 2016.
3. Bhave , " Object Oriented Programming With C++", Pearson Education , 2004.
4. A K Sharma , "Object Oriented Programming with C++", Pearson Education, 2014.

Activity Based Learning (Suggested Activities in Class)/ Project Based Learning :

- Group Assignment to develop small projects and demonstrate using C++

Lab CIE (30 marks) –Rubrics used for Continuous Evaluation in every lab session
(Part A – 20 Marks + Part B – 10 Marks)

Part A Evaluation – 20 Marks

Parameter	Allocated Marks	LOW	MEDIUM	HIGH
Execution	18	The given program was coded but not executed in the lab session	The given program was coded ,debugged and executed in the lab session but results are not up to the expectation.	The given program was coded, debugged and executed in the lab session appropriately. (with any additions / modifications given by course coordinator)
		0-5 Marks	**6-15 Marks**	**16-18 Marks**
Record writing	09 Attendance: 2 Marks)	he record was not submitted in the lab session	The record was submitted in the lab session but was incomplete (no Algorithm / wrong Algorithm & no Flowchart / wrong Flowchart)	Completed record was submitted in the lab session. (Tracing of code with sample inputs & proper indication of graphs, tables, figures etc)
		0 Marks	**1-5 Marks**	**6 -7 Marks**
Viva-voce	03	The student did not answer any viva questions asked	The student answered few viva questions asked	The student answered all viva questions asked (Tracing for different data / formula deriving etc)
		0 Marks	**1-2 Marks**	**3 Marks**

CO PO Co-relation mapping table:

COs	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PSO 1	PSO 2
CO 1	3	1			2							2	
CO 2	3	1			3							2	
CO 3	3	2			2							2	
CO 4	3	2			2							2	

CO 5	2	3	1		2	2		2			2	2	

CO PO Co-relation mapping justification:

CO No.	Mapped POs/PSOs	Justification for Mapping
CO1 – Apply object-oriented programming constructs to develop structured and efficient solutions for computing problems. *(K3)*	PO1 (3), PO2 (1), PO5 (2), PSO1 (2)	Strongly contributes to **engineering knowledge (PO1)** through application of OOP constructs. Supports **problem analysis (PO2)** at a basic level. Demonstrates **modern tool usage (PO5)** while coding. Links to **PSO1** (programming fundamentals applied to computing problems).
CO2 – Analyze program structure and relationships among components to ensure modularity, extensibility, and reusability in software design. *(K3)*	PO1 (3), PO2 (1), PO5 (3), PSO1 (2)	Strong contribution to **PO1** (engineering knowledge) in applying modular design principles. Minimal mapping to **PO2** since analysis aids problem decomposition. Major contribution to **PO5** (modern tool usage) via use of software tools & debugging. Supports **PSO1** in structured software design.
CO3 – Examine the use of polymorphism, virtual functions, and generic programming techniques to improve flexibility and adaptability of solutions. *(K4)*	PO1 (3), PO2 (2), PO5 (2), PSO1 (2)	**PO1** strongly mapped due to conceptual understanding & coding of advanced OOP. **PO2** moderately mapped as students analyze adaptability of solutions. **PO5** relevant for applying coding environments. Aligns with **PSO1** for designing reusable solutions.
CO4 – Inspect program behavior in terms of error handling, file operations, and data management to ensure robustness and reliability. *(K4)*	PO1 (3), PO2 (2), PO5 (2), PSO1 (2)	**PO1** strongly mapped as it requires knowledge of exception handling and file systems. **PO2** moderately mapped since analysis is needed for error detection. **PO5** mapped through debugging tools, testing. Supports **PSO1** for robust C++ programming.
CO5 – Evaluate problem-solving strategies by implementing object-oriented solutions in C++ to ensure efficiency, scalability, and correctness. *(K5)*	PO1 (2), PO2 (3), PO3 (1), PO5 (2), PO6 (2), PO8 (2), PO10 (2), PO11 (2), PSO1 (2)	**PO1** moderately mapped as it requires engineering knowledge for efficiency. **PO2** strongly mapped due to critical problem analysis. **PO3** slightly mapped as it involves designing small-scale projects. **PO5** linked to using compilers, IDEs. **PO6/PO8/PO10/PO11** weak-to-moderate mapping through social, ethical, communication, and life-long learning aspects (team projects, documentation). **PSO1** supports professional programming practice.

1. You are building a feature for an e-commerce platform that displays product prices. Write a C++ program that accepts a list of product prices and displays them sorted both in ascending (low to high) and descending (high to low) order, allowing customers to choose how they want to view the prices." (Use basic programming concepts like looping constructs for menu driven programming, arrays and sorting logic etc.).

Algorithm for the Program

1. Start
2. Input number of products (n).
3. Declare an array to store n product prices.
4. Read product prices into the array using a loop.
5. Display a menu to the user with options:
    - (1) Sort in ascending order
    - (2) Sort in descending order
    - (3) Exit
6. Use a loop (do-while or while) to repeatedly show the menu until the user chooses to exit.
7. If user chooses option 1:
    - Apply a sorting algorithm (like Bubble Sort or Selection Sort) to arrange prices in ascending order.
    - Display the sorted prices.
8. If user chooses option 2:
    - Apply a sorting algorithm to arrange prices in descending order.
    - Display the sorted prices.
9. If user chooses option 3:
    - Terminate the program.
10. End

Expected Output:

When the program is executed, the user will first be asked to enter the number of products and their respective prices. After input, the program displays a menu that allows the user to choose how they want to view the prices. If the user selects the ascending option, the program will sort and display the prices from the lowest to the highest. If the descending option is chosen, the prices will be shown from the highest to the lowest. The program will continue to display the menu until the user decides to exit. For example, if the entered prices are 500, 200,

750, 300, the ascending option will display 200, 300, 500, 750, while the descending option will display 750, 500, 300, 200. This ensures a user-friendly way for customers to view prices as per their preference.

C++ Program

```
#include <iostream>
using namespace std;

// Function to sort in ascending order
void sortAscending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function to sort in descending order
void sortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                // swap
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function to display the array
void display(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << "  ";
    }
    cout << endl;
}
```

```
int main() {
    int n;
    cout << "Enter the number of products: ";
    cin >> n;

    int prices[100]; // fixed size array
    cout << "Enter the prices of " << n << " products:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> prices[i];
    }

    int choice;
    do {
        cout << "\n---- MENU ----\n";
        cout << "1. View prices in Ascending order\n";
        cout << "2. View prices in Descending order\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            sortAscending(prices, n);
            cout << "Prices in Ascending Order: ";
            display(prices, n);
            break;

        case 2:
            sortDescending(prices, n);
            cout << "Prices in Descending Order: ";
            display(prices, n);
            break;

        case 3:
            cout << "Exiting program. Thank you!" << endl;
            break;

        default:
            cout << "Invalid choice! Please try again." << endl;
        }
    } while (choice != 3);

    return 0;
}
```

```
Sample Output
Enter the number of products: 4
Enter the prices of 4 products:
500
200
750
300

---- MENU ----
1. View prices in Ascending order
2. View prices in Descending order
3. Exit
Enter your choice: 1
Prices in Ascending Order: 200  300  500  750

---- MENU ----
1. View prices in Ascending order
2. View prices in Descending order
3. Exit
Enter your choice: 2
Prices in Descending Order: 750  500  300  200

---- MENU ----
1. View prices in Ascending order
2. View prices in Descending order
3. Exit
Enter your choice: 3
Exiting program. Thank you!
```

2. As part of a student management system, you are tasked with designing a module to generate student performance reports. Create a C++ program using classes that captures a student's name, roll number, and marks in three subjects. The program should then calculate and display the student's total score along with their details.
   (Use C++ concepts like classes, constructors and methods etc.).

Algorithm

1. Start
2. Define a class Student with the following members:
   - Data members: name, rollNo, marks[3], total.
   - Constructor: To initialize name, rollNo, and marks.
   - Method calculateTotal(): Compute sum of three subject marks.
   - Method displayDetails(): Display student details and total marks.
3. In main():
   - Create a Student object.
   - Accept input for name, roll number, and marks.
   - Call methods to calculate and display the report.
4. End

Expected Output:
When the program is executed, the user is prompted to enter a student's name, roll number, and marks in three subjects. After entering these details, the program calculates the total score by adding all three marks. Finally, it generates a performance report displaying the student's name, roll number, individual subject marks, and the computed total. For example, if the input is *Name: Kiran Kumar*, *Roll No: 101*, *Marks: 85*, *90*, *78*, the output will display a neatly formatted report showing the student's details along with the total score of 253. This allows the user to quickly view both the individual marks and the overall performance of the student in a clear and structured way.

C++ Program

```
#include <iostream>
#include <string>
using namespace std;
```

```
class Student {
private:
    string name;
    int rollNo;
    int marks[3];
    int total;

public:
    // Constructor to initialize student details
    Student(string n, int r, int m1, int m2, int m3) {
        name = n;
        rollNo = r;
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
        total = 0;
    }

    // Method to calculate total marks
    void calculateTotal() {
        total = marks[0] + marks[1] + marks[2];
    }

    // Method to display details
    void displayDetails() {
        cout << "\n----- Student Performance Report -----\n";
        cout << "Name       : " << name << endl;
        cout << "Roll No    : " << rollNo << endl;
        cout << "Marks      : " << marks[0] << ", " << marks[1] << ",
" << marks[2] << endl;
        cout << "Total Score: " << total << endl;
    }
};

int main() {
    string name;
    int roll, m1, m2, m3;

    cout << "Enter Student Name: ";
    getline(cin, name);
    cout << "Enter Roll Number: ";
    cin >> roll;
    cout << "Enter marks in 3 subjects: ";
    cin >> m1 >> m2 >> m3;

    // Create object and generate report
```

```
    Student s1(name, roll, m1, m2, m3);
    s1.calculateTotal();
    s1.displayDetails();

    return 0;
}
```

```
Sample Output:
Enter Student Name: Kiran Kumar
Enter Roll Number: 101
Enter marks in 3 subjects: 85 90 78

----- Student Performance Report -----
Name        : Kiran Kumar
Roll No     : 101
Marks       : 85, 90, 78
Total Score: 253
```

3. You are developing a contact management application where each contact contains details such as name, phone number, and email address. To ensure safe and accurate duplication of contact records (e.g., for backup or editing), implement a C++ class

Contact that:

a. Includes a copy constructor to properly copy all data members from one contact object to another.
b. Also includes a destructor to manage clean-up when a contact object is destroyed (e.g., to release memory or display a message indicating the object is being deleted).
Demonstrate the creation of an original contact and a copy using the copy constructor, display their contents, and show how the destructor is automatically invoked when objects go out of scope.
(Use C++ concepts like classes, constructors, copy constructor, destructor etc.).

Algorithm:

1. Start
2. Define a class Contact with private data members: name, phone, email.
3. Add:
    - A parameterized constructor to initialize all fields.
    - A copy constructor that copies every data member from an existing Contact object.
    - A destructor that prints a message when the object is destroyed (simulating cleanup).
    - A display() method to print the contact's details.
4. In main():
    - Create an original Contact object using the parameterized constructor.
    - Create a copy object using the copy constructor (e.g., Contact c2 = c1;).
    - Call display() on both to verify identical contents.
    - End of main() lets objects go out of scope, so destructors run automatically.
5. End.

Expected Output:

When the program executes, it first constructs the original contact, printing a constructor message for "Kiran Kumar." Next, a second object is created using the copy constructor, which prints a message indicating the contact was copied. The program then displays both the original and the copied contact, showing identical details for the name, phone number, and email address. Finally, when main() ends and the objects go out of scope, the destructor is automatically invoked for each contact in reverse order of creation, printing messages that each contact object is being deleted. This demonstrates safe duplication via the copy constructor and automatic cleanup via the destructor.

C++ Program

```
#include <iostream>
#include <string>
```

```
using namespace std;

class Contact {
private:
    string name;
    string phone;
    string email;

public:
    // Parameterized constructor
    Contact(const string& n, const string& p, const string& e)
        : name(n), phone(p), email(e) {
        cout << "[Constructor] Contact created for: " << name << endl;
    }

    // Copy constructor (safe duplication of all members)
    Contact(const Contact& other)
        : name(other.name), phone(other.phone), email(other.email) {
        cout << "[Copy Constructor] Contact copied for: " << name <<
endl;
    }

    // Destructor (cleanup / message)
    ~Contact() {
        cout << "[Destructor] Contact for " << name << " is being
deleted." << endl;
    }

    // Display method
    void display() const {
        cout << "\n--- Contact ---\n";
        cout << "Name : " << name  << "\n";
        cout << "Phone: " << phone << "\n";
        cout << "Email: " << email << "\n";
    }
};

int main() {
    // Original contact
    Contact c1("Kiran Kumar", "9876543210", "kiran@example.com");

    // Copy using copy constructor
    Contact c2 = c1;   // invokes copy constructor

    // Show both
    cout << "\nDisplaying Original Contact:";
```

```
    c1.display();

    cout << "\nDisplaying Copied Contact:";
    c2.display();

    // Destructors will be called automatically for c2 then c1
(reverse of creation order)
    return 0;
}
```

```
Sample Output:
[Constructor] Contact created for: Kiran Kumar
[Copy Constructor] Contact copied for: Kiran Kumar

Displaying Original Contact:
--- Contact ---
Name : Kiran Kumar
Phone: 9876543210
Email: kiran@example.com

Displaying Copied Contact:
--- Contact ---
Name : Kiran Kumar
Phone: 9876543210
Email: kiran@example.com

[Destructor] Contact for Kiran Kumar is being deleted.
[Destructor] Contact for Kiran Kumar is being deleted.
```

4. Develop a loan eligibility evaluation system for a bank. The system has two classes: Customer – contains private data members such as name, account balance, and credit score. Loan – is responsible for evaluating whether the customer is eligible for a loan

based on specific criteria (e.g., balance ≥ ₹50,000 and credit score ≥ 750 out of 1000). Since the Loan class should not directly access the private members of the Customer class, implement a friend function named checkEligibility() that can access the private data of Customer and evaluate the eligibility.

(Use C++ concepts like classes, constructor, Friend functions etc.)

Algorithm:

1. Start
2. Create a class Customer with private members: name, balance, creditScore.
   - Provide a parameterized constructor to initialize these.
   - Provide a display() method for showing details.
   - Declare a friend function checkEligibility(const Customer&, const Loan&).
3. Create a class Loan with private members: minBalance (default ₹50,000) and minScore (default 750).
   - Provide a parameterized constructor with sensible defaults.
   - Provide a showCriteria() method to display the criteria.
   - Declare the same friend function so it can access Loan's private thresholds, too.
4. Implement the non-member friend function checkEligibility(const Customer&, const Loan&) that:
   - Accesses Customer's balance and creditScore.
   - Accesses Loan's minBalance and minScore.
   - Returns true if both conditions are satisfied, else false.
5. In main():
   - Create a Loan object (with defaults or specified thresholds).
   - Create one or more Customer objects.
   - Call checkEligibility(customer, loan) and print Eligible/Not Eligible with reasons.
6. End.

Expected Output:

When the program runs, it first prints the loan eligibility criteria—minimum account balance of ₹50,000 and a minimum credit score of 750 out of 1000. For each customer, it displays their name, account balance, and credit score. Then, using the checkEligibility() friend function (which can safely access private data from both Customer and Loan), it evaluates whether the customer meets both the balance and credit score thresholds. The program reports "Eligible" if both conditions are satisfied; otherwise, it reports "Not Eligible." This demonstrates proper encapsulation with private members, initialization via constructors, and controlled access through a friend function that

performs the evaluation without exposing internal details directly to the Loan class.

C++ Program:

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Loan; // forward declaration for friend signature

class Customer {
private:
    string name;
    double balance;
    int creditScore; // out of 1000

public:
    Customer(const string& n, double b, int cs)
        : name(n), balance(b), creditScore(cs) {}

    void display() const {
        cout << "Customer: " << name << "\n"
             << "Balance : \u20B9" << fixed << setprecision(2) <<
balance << "\n"
             << "Credit  : " << creditScore << " / 1000\n";
    }

    // Friend function can access private members of Customer
    friend bool checkEligibility(const Customer& c, const Loan& l);
};

class Loan {
private:
    double minBalance;
    int minScore;

public:
    // Defaults as per problem statement: ₹50,000 and 750/1000
    Loan(double mb = 50000.0, int ms = 750)
        : minBalance(mb), minScore(ms) {}

    void showCriteria() const {
        cout << "Eligibility Criteria -> "
             << "Min Balance: \u20B9" << fixed << setprecision(2) <<
minBalance
```

```
            << ", Min Credit Score: " << minScore << "/1000\n";
    }

    // Friend function can access private members of Loan too
    friend bool checkEligibility(const Customer& c, const Loan& l);
};

// Friend function definition (non-member)
bool checkEligibility(const Customer& c, const Loan& l) {
    return (c.balance >= l.minBalance) && (c.creditScore >=
l.minScore);
}

int main() {
    // Create loan product with default criteria
    Loan homeLoan; // Min balance ₹50,000 and min score 750

    // Create customers
    Customer c1("Kiran Kumar", 72000.00, 780);
    Customer c2("Anita Sharma", 30000.00, 820);

    cout << "=== Loan Eligibility Evaluation System ===\n\n";
    homeLoan.showCriteria();
    cout << "\n";

    // Evaluate Customer 1
    c1.display();
    bool ok1 = checkEligibility(c1, homeLoan);
    cout << "Eligibility: " << (ok1 ? "Eligible" : "Not Eligible") <<
"\n\n";

    // Evaluate Customer 2
    c2.display();
    bool ok2 = checkEligibility(c2, homeLoan);
    cout << "Eligibility: " << (ok2 ? "Eligible" : "Not Eligible") <<
"\n";

    return 0;
}
```

Sample Output:

```
=== Loan Eligibility Evaluation System ===
```

```
Eligibility Criteria -> Min Balance: ₹50000.00, Min Credit Score:
750/1000

Customer: Kiran Kumar
Balance : ₹72000.00
Credit  : 780 / 1000
Eligibility: Eligible

Customer: Anita Sharma
Balance : ₹30000.00
Credit  : 820 / 1000
Eligibility: Not Eligible
```

5. You are developing a banking application that enables customers to manage their accounts efficiently. As part of the system implement the following functionality:

a. Implement function overloading to perform addition of both integer-based amounts

(e.g.,
₹500 + ₹200) and floating-point values (e.g., ₹1000.75 + ₹499.25), ensuring accuracy across
transaction types like deposits and interest calculations.

b. Use operator overloading to redefine the unary minus (-) operator, allowing quick reversal of transaction values — for example, converting a credit transaction into a debit during a refund or correction.
(Use concepts like classes, function overloading and operator overloading etc.)

Algorithm:

1. Start
2. Create a helper utility (free functions or a utility class) with function overloading:
   - add(int, int) → returns int
   - add(double, double) → returns double
     This ensures accurate handling for integer transactions (e.g., cash deposits) and floating transactions (e.g., interest).
3. Create a class Transaction with a private member amount (double).
   - Add a parameterized constructor to set the amount.
   - Overload the unary minus operator - to return a new Transaction with negated amount (quick reversal).
   - Add a value() getter and a print() method to display the amount.
4. In main():
   - Demonstrate add(int,int) with values like ₹500 + ₹200.
   - Demonstrate add(double,double) with values like ₹1000.75 + ₹499.25.
   - Create a Transaction (credit) and apply unary - to show reversal (debit).
   - Print results neatly with rupee symbol and two decimals.
5. End.

Expected Output:

When the program runs, it first demonstrates function overloading by adding two integer amounts (₹500 and ₹200) and printing the integer total. It then adds two floating-point amounts (₹1000.75 and ₹499.25), preserving decimals to show precise financial calculations. Next, it demonstrates operator overloading: a Transaction object with a positive amount (credit) is printed, and applying the unary minus operator produces a new transaction with the negative of that amount (debit), effectively showing a quick reversal (useful for refunds or corrections). All values are displayed with the rupee symbol and two decimal places for clarity.

C++ Program:

```
#include <iostream>
#include <iomanip>
using namespace std;

// ---------- Function Overloading ----------
int add(int a, int b) {
    return a + b; // integer-based amounts
}

double add(double a, double b) {
    return a + b; // floating-point amounts (e.g., interest + principal)
}

// ---------- Transaction Class (Unary - Overloading) ----------
class Transaction {
private:
    double amount; // positive = credit, negative = debit

public:
    explicit Transaction(double amt) : amount(amt) {}

    // Unary minus operator: quick reversal (credit <-> debit)
    Transaction operator-() const {
        return Transaction(-amount);
    }

    double value() const { return amount; }

    void print(const string& label = "Transaction") const {
        cout << label << " : \u20B9" << fixed << setprecision(2) <<
amount << '\n';
    }
};

int main() {
    cout << "=== Banking Operations: Overloading Demo ===\n\n";

    // a) Function overloading (int + int)
    int cash1 = 500, cash2 = 200;
    int cashTotal = add(cash1, cash2);
    cout << "Integer Addition -> "
         << "\u20B9" << cash1 << " + \u20B9" << cash2
         << " = \u20B9" << cashTotal << "\n";

    // a) Function overloading (double + double)
    double f1 = 1000.75, f2 = 499.25;
```

```
    double fTotal = add(f1, f2);
    cout << "Floating Addition -> "
         << "\u20B9" << fixed << setprecision(2) << f1
         << " + \u20B9" << f2
         << " = \u20B9" << fTotal << "\n\n";

    // b) Operator overloading (unary minus for quick reversal)
    Transaction credit(1250.00);   // +1250 credit
    credit.print("Original (Credit)");

    Transaction reversed = -credit; // now becomes -1250 (debit)
    reversed.print("Reversed (Debit)");

    return 0;
}
```

```
Sample output:
=== Banking Operations: Overloading Demo ===

Integer Addition -> ₹500 + ₹200 = ₹700
Floating Addition -> ₹1000.75 + ₹499.25 = ₹1500.00

Original (Credit) : ₹1250.00
Reversed (Debit)  : ₹-1250.00
```

6. You are designing a modular calculator system where arithmetic functionalities are separated into logical units. One class provides addition functionality, another provides subtraction, and a third class inherits from both to perform a full arithmetic operation on two numbers. Implement a C++ program using multiple inheritances to

demonstrate this system.
(Use concepts like classes, constructors, multiple inheritances etc.)

Algorithm

1. Start
2. Create class Adder with:
   - Constructor (prints a creation message).
   - Method add(double, double) → returns sum.
3. Create class Subtractor with:
   - Constructor (prints a creation message).
   - Method sub(double, double) → returns difference.
4. Create class Calculator that inherits from both Adder and Subtractor:
   - Data members: a, b.
   - Constructor to initialize a, b (and show message).
   - Method perform() to call add() and sub() from base classes, then print results.
5. In main():
   - Read two numbers (or use sample numbers).
   - Construct Calculator and call perform().
6. End.

Expected Output:
When the program runs, it first initializes the modular units: an Adder and a Subtractor, each announcing readiness via their constructors. The Calculator object is then created with two numbers and, through multiple inheritance, gains access to both addition and subtraction functionalities. Calling perform() displays the sum and the difference of the two numbers, demonstrating how separate arithmetic units can be composed into a single calculator using multiple inheritance, while constructors show the creation sequence of each component.

C++ Program

```
#include <iostream>
#include <iomanip>
using namespace std;

// Unit 1: Addition provider
class Adder {
public:
    Adder() { cout << "[Adder] Ready\n"; }
    double add(double x, double y) const { return x + y; }
};

// Unit 2: Subtraction provider
```

```
class Subtractor {
public:
    Subtractor() { cout << "[Subtractor] Ready\n"; }
    double sub(double x, double y) const { return x - y; }
};

// Full calculator: combines both via multiple inheritance
class Calculator : public Adder, public Subtractor {
private:
    double a, b;

public:
    Calculator(double x, double y) : a(x), b(y) {
        cout << "[Calculator] Created with a=" << a << ", b=" << b <<
"\n";
    }

    void perform() const {
        cout << fixed << setprecision(2);
        double s = add(a, b);   // from Adder
        double d = sub(a, b);   // from Subtractor

        cout << "\n--- Results ---\n";
        cout << "a + b = " << s << "\n";
        cout << "a - b = " << d << "\n";
    }
};

int main() {
    cout << "=== Modular Calculator (Multiple Inheritance) ===\n\n";

    double x = 12.5, y = 7.25;    // sample values; replace with user
input if desired
    // If you want input, uncomment below:
    // cout << "Enter two numbers: ";
    // cin >> x >> y;

    Calculator calc(x, y);
    calc.perform();

    return 0;
}
```

Sample Output:

```
=== Modular Calculator (Multiple Inheritance) ===
```

```
[Adder] Ready
[Subtractor] Ready
[Calculator] Created with a=12.5, b=7.25

--- Results ---
a + b = 19.75
a - b = 5.25
```

7. "You're working on a scientific computation module where different values—such as alpha, beta, and gamma—represent coefficients in a formula. These values come from different sources: a base class initializes alpha and beta, while the derived class introduces gamma. Write a C++ program using constructors in derived classes to

properly initialize and display all three values, showcasing inheritance and constructor chaining."

(Use concepts like classes, constructors, inheritance and constructor chaining etc.)

Algorithm

1. Start
2. Create a base class CoeffBase that holds alpha and beta.
   - Provide a parameterized constructor to initialize alpha and beta.
   - Provide a showBase() method to display them.
3. Create a derived class CoeffModel that inherits from CoeffBase and adds gamma.
   - Use a constructor with an initializer list to call CoeffBase(alpha, beta) and then initialize gamma.
   - Provide a showAll() method to display alpha, beta, and gamma.
4. In main():
   - Create a CoeffModel object with values for alpha, beta, gamma.
   - Call showAll() to verify all values.
   - Observe constructor messages to see constructor chaining (base constructed before derived).
5. End.

Expected Output:

When the program runs, the derived class constructor invokes the base class constructor first, initializing alpha and beta; a message confirms their values. Immediately after, the derived constructor initializes gamma and prints its message. Finally, calling showAll() displays all three coefficients—alpha, beta, and gamma—demonstrating inheritance and constructor chaining where the base portion of the object is constructed before the derived portion.

C++ Program

```
#include <iostream>
#include <iomanip>
using namespace std;

class CoeffBase {
protected:
    double alpha;
    double beta;

public:
    // Parameterized constructor for base coefficients
    CoeffBase(double a, double b) : alpha(a), beta(b) {
```

```
        cout << "[CoeffBase] Constructed with alpha=" << alpha
             << ", beta=" << beta << '\n';
    }

    void showBase() const {
        cout << "alpha = " << alpha << ", beta = " << beta << '\n';
    }
};

class CoeffModel : public CoeffBase {
private:
    double gamma;

public:
    // Derived constructor chains to base to initialize alpha & beta,
    // then initializes gamma.
    CoeffModel(double a, double b, double g)
        : CoeffBase(a, b), gamma(g) {
        cout << "[CoeffModel] Constructed with gamma=" << gamma << '\n';
    }

    void showAll() const {
        cout << fixed << setprecision(2);
        cout << "\n--- Coefficient Set ---\n";
        cout << "alpha = " << alpha << '\n';
        cout << "beta  = " << beta  << '\n';
        cout << "gamma = " << gamma << '\n';
    }
};

int main() {
    cout   <<   "===   Constructor   Chaining:   Base(alpha,beta)   +
Derived(gamma) ===\n\n";

    // Example values (can be replaced with input if needed)
    double a = 1.25, b = 2.50, g = -0.75;

    // Construction order: CoeffBase first, then CoeffModel
    CoeffModel model(a, b, g);

    // Display all coefficients
    model.showAll();

    return 0;
}
```

Sample Output:

```
=== Constructor Chaining: Base(alpha,beta) + Derived(gamma) ===

[CoeffBase] Constructed with alpha=1.25, beta=2.5
[CoeffModel] Constructed with gamma=-0.75

--- Coefficient Set ---
alpha = 1.25
beta  = 2.50
gamma = -0.75
```