DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Student Name: Gurnoor Madaan UID:22BCS15406
Branch: BE-CSE Section/Group:620-'B'
Smester:5th Date :19/12/2024

Medium

1) Objective

Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

```
#include <iostream>
#include <cmath> // For M_PI constant
using namespace std;

double area(double radius) {
    return M_PI * radius * radius;
}

double area(double length, double width) {
    return length * width;
}

double area(double base, double height, bool isTriangle) {
    if (isTriangle) {
        return 0.5 * base * height;
    }
    return -1;
}

int main() {
    double radius, length, width, base, height;

    cout << "Enter radius of the circle ";
    cin >> radius;
    cout << "Area of the circle " << area(radius) << endl;

    cout << "Enter length and width of the rectangle ";
    cin >> length >> width;
    cout << "Area of the rectangle" << area(length, width) << endl;

    cout << "Enter base and height of the triangle ";
    cin >> base >> height;
    cout << "Area of the triangle " << area(base, height, true) << endl;

    return 0;
}
```

2)Objective

Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy. Implement overloaded functions to compute salary for:

- Intern (basic stipend).
- Regular employee (base salary + bonuses).
- Manager (base salary + bonuses + performance incentives).

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

double calculateSalary(double stipend) {
    return stipend;
}

double calculateSalary(double baseSalary, double bonuses) {
    return baseSalary + bonuses;
}

double calculateSalary(double baseSalary, double bonuses, double
performanceIncentives) {
    return baseSalary + bonuses + performanceIncentives;
}

int main() {
    double stipend, baseSalary, bonuses, performanceIncentives;

    cout << "Enter stipend for intern ";
    cin >> stipend;
    cout << "Salary of the intern " << calculateSalary(stipend) << endl;

    cout << "Enter base salary and bonuses for regular employee ";
    cin >> baseSalary >> bonuses;
    cout << "Salary of the regular employee " <<
calculateSalary(baseSalary, bonuses) << endl;

    cout << "Enter base salary, bonuses, and performance incentives for
manager ";
    cin >> baseSalary >> bonuses >> performanceIncentives;
    cout << "Salary of the manager " << calculateSalary(baseSalary,
bonuses, performanceIncentives) << endl;

    return 0;
}
```

3)Objective

Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store:
Employee ID.
Employee Name.
Employee Salary.
Provide public methods to set and get these attributes, and a method to display all details of the employee.

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

class Employee {
private:
    int employeeID;
    string employeeName;
    double employeeSalary;

public:

    void setEmployeeID(int id) {
        employeeID = id;
    }

    int getEmployeeID() {
        return employeeID;
    }

    void setEmployeeName(string name) {
        employeeName = name;
    }

    string getEmployeeName() {
        return employeeName;
    }
    void setEmployeeSalary(double salary) {
        employeeSalary = salary;
    }
    double getEmployeeSalary() {
        return employeeSalary;
    }
    void displayEmployeeDetails() {
        cout << "Employee ID: " << employeeID << endl;
        cout << "Employee Name: " << employeeName << endl;
        cout << "Employee Salary: " << employeeSalary << endl;
    }
};

int main() {
    Employee emp;
    emp.setEmployeeID(1);
    emp.setEmployeeName("John Doe");
    emp.setEmployeeSalary(75000);
```

```
    emp.displayEmployeeDetails();

    return 0;
}
```

4) *Objective*

Create a program that demonstrates inheritance by defining:

- A base class Student to store details like Roll Number and Name.
- A derived class Result to store marks for three subjects and calculate the total and percentage.

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

using namespace std;

class Student {
protected:
    string name;
    int rollNumber;

public:
    Student(string n, int r) : name(n), rollNumber(r) {}

    void displayStudentDetails() const {
        cout << "Roll Number: " << rollNumber << "\nName: " << name <<
endl;
    }
};

class Result : public Student {
private:
    int marks[3];
    int total;
    float percentage;

public:
    Result(string n, int r, int m1, int m2, int m3)
        : Student(n, r) {
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
        total = m1 + m2 + m3;
        percentage = (float)total / 3.0;
    }

    void displayResult() const {
        displayStudentDetails();
        cout << "Marks: " << marks[0] << ", " << marks[1] << ", " <<
marks[2] << endl;
        cout << "Total: " << total << endl;
        cout << "Percentage: " << percentage << "%" << endl;
    }
};

int main() {
    Result student1("John Doe", 101, 85, 90, 88);
```

```
    student1.displayResult();

    return 0;
}
```

5)Objective

Create a program that demonstrates polymorphism by calculating the area of different shapes using a base class Shape and derived classes for Circle, Rectangle, and Triangle. Each derived class should override a virtual function to compute the area of the respective shape.

```
#include <iostream>
#include <cmath>

using namespace std;

class Shape {
public:
    virtual double getArea() const = 0;  // Pure virtual function for
calculating area
    virtual ~Shape() {}  // Virtual destructor
};

class Circle : public Shape {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    double getArea() const override {
        return M_PI * radius * radius;  // Area of the circle: π * r^2
    }
};

class Rectangle : public Shape {
private:
    double length, width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double getArea() const override {
        return length * width;  // Area of the rectangle: length * width
    }
};

class Triangle : public Shape {
private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    double getArea() const override {
```

```
        return 0.5 * base * height;  // Area of the triangle: 0.5 * base *
height
    }
};

int main() {
    Shape* shapes[3];

    shapes[0] = new Circle(5.0);  // Create a circle with radius 5.0
    shapes[1] = new Rectangle(4.0, 6.0);  // Create a rectangle with length
4.0 and width 6.0
    shapes[2] = new Triangle(4.0, 7.0);  // Create a triangle with base 4.0
and height 7.0

    // Calculate and display the areas using polymorphism
    for (int i = 0; i < 3; i++) {
        cout << "Area of shape " << i+1 << ": " << shapes[i]->getArea() <<
endl;
    }

    // Clean up dynamically allocated memory
    for (int i = 0; i < 3; i++) {
        delete shapes[i];
    }

    return 0;
}
```

HARD

1) Objective

Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

```
#include <iostream>
#include <cmath>

using namespace std;

class Shape {
public:
    virtual double area() const = 0;
    virtual ~Shape() {}
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) {
        radius = r;
    }
    double area() const override {
        return M_PI * radius * radius;
    }
};

class Rectangle : public Shape {
private:
    double width, height;
public:
    Rectangle(double w, double h) {
        width = w;
        height = h;
    }
    double area() const override {
        return width * height;
    }
};

class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) {
        base = b;
        height = h;
    }
    double area() const override {
        return 0.5 * base * height;
    }
};
```

```
int main() {
    int choice;
    double area;

    cout << "Select the shape to calculate the area:\n";
    cout << "1. Circle\n";
    cout << "2. Rectangle\n";
    cout << "3. Triangle\n";
    cout << "Enter your choice (1/2/3): ";
    cin >> choice;

    Shape* shape = nullptr;

    switch(choice) {
        case 1: {
            double radius;
            cout << "Enter the radius of the circle: ";
            cin >> radius;
            shape = new Circle(radius);
            break;
        }
        case 2: {
            double width, height;
            cout << "Enter the width of the rectangle: ";
            cin >> width;
            cout << "Enter the height of the rectangle: ";
            cin >> height;
            shape = new Rectangle(width, height);
            break;
        }
        case 3: {
            double base, height;
            cout << "Enter the base of the triangle: ";
            cin >> base;
            cout << "Enter the height of the triangle: ";
            cin >> height;
            shape = new Triangle(base, height);
            break;
        }
        default:
            cout << "Invalid choice\n";
            return 0;
    }

    area = shape->area();
    cout << "The area of the selected shape is: " << area << endl;

    delete shape;

    return 0;
}
```

2) Objective

Implement matrix operations in C++ using function overloading. Write a function operate() that can perform:

- Matrix Addition for matrices of the same dimensions.
- Matrix Multiplication where the number of columns of the first matrix equals the number of rows of the second matrix.

```
#include <iostream>
#include <vector>

using namespace std;

class Matrix {
private:
    vector<vector<int>> mat;
    int rows, cols;

public:
    Matrix(int r, int c) : rows(r), cols(c) {
        mat.resize(rows, vector<int>(cols));
    }

    void input() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cin >> mat[i][j];
            }
        }
    }

    void display() const {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cout << mat[i][j] << " ";
            }
            cout << endl;
        }
    }

    Matrix operate(const Matrix& m2) {
        if (rows != m2.rows || cols != m2.cols)
            throw invalid_argument("Matrices must have the same
dimensions.");
        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.mat[i][j] = mat[i][j] + m2.mat[i][j];
            }
        }
        return result;
    }

    Matrix operate(const Matrix& m2) const {
        if (cols != m2.rows)
            throw invalid_argument("Invalid dimensions for
multiplication.");
        Matrix result(rows, m2.cols);
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < m2.cols; j++) {
                result.mat[i][j] = 0;
                for (int k = 0; k < cols; k++) {
                    result.mat[i][j] += mat[i][k] * m2.mat[k][j];
                }
            }
        }
        return result;
    }

    int getRows() const { return rows; }
    int getCols() const { return cols; }
};

int main() {
    int rows1, cols1, rows2, cols2;
    cout << "Enter rows and columns for the first matrix: ";
    cin >> rows1 >> cols1;
    Matrix m1(rows1, cols1);
    m1.input();

    cout << "Enter rows and columns for the second matrix: ";
    cin >> rows2 >> cols2;
    Matrix m2(rows2, cols2);
    m2.input();

    try {
        cout << "\nMatrix 1:\n";
        m1.display();
        cout << "\nMatrix 2:\n";
        m2.display();

        if (rows1 == rows2 && cols1 == cols2) {
            Matrix sum = m1.operate(m2);
            cout << "\nMatrix Addition Result:\n";
            sum.display();
        }

        if (cols1 == rows2) {
            Matrix product = m1.operate(m2);
            cout << "\nMatrix Multiplication Result:\n";
            product.display();
        }
    } catch (const invalid_argument& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

3)Objective:

Design a C++ program using polymorphism to calculate the area of different shapes:
A Rectangle (Area = Length × Breadth).
A Circle (Area = π × Radius²).
A Triangle (Area = ½ × Base × Height).
Create a base class Shape with a pure virtual function getArea(). Use derived classes Rectangle, Circle, and Triangle to override this function.

```
#include <iostream>
#include <cmath>

using namespace std;

class Shape {
public:
    virtual double getArea() const = 0;  // Pure virtual function
    virtual ~Shape() {}  // Virtual destructor
};

class Rectangle : public Shape {
private:
    double length, breadth;
public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double getArea() const override {
        return length * breadth;
    }
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double getArea() const override {
        return M_PI * radius * radius;
    }
};

class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    double getArea() const override {
        return 0.5 * base * height;
    }
};

int main() {
    Shape* shape1 = new Rectangle(5.0, 3.0);
    Shape* shape2 = new Circle(4.0);
    Shape* shape3 = new Triangle(6.0, 2.0);
```

```
    cout << "Area of Rectangle: " << shape1->getArea() << endl;
    cout << "Area of Circle: " << shape2->getArea() << endl;
    cout << "Area of Triangle: " << shape3->getArea() << endl;

    delete shape1;
    delete shape2;
    delete shape3;

    return 0;
}
```

4)Objective

Create a C++ program using multiple inheritance to simulate a library system. Design two base classes:

- Book to store book details (title, author, and ISBN).
- Borrower to store borrower details (name, ID, and borrowed book).

Create a derived class Library that inherits from both Book and Borrower. Use this class to track the borrowing and returning of books.

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

using namespace std;

class Book {
protected:
    string title, author, isbn;

public:
    Book(string t, string a, string i) : title(t), author(a), isbn(i) {}

    void displayBookDetails() const {
        cout << "Title: " << title << "\nAuthor: " << author << "\nISBN: "
<< isbn << endl;
    }
};

class Borrower {
protected:
    string name;
    int id;
    string borrowedBook;

public:
    Borrower(string n, int i) : name(n), id(i), borrowedBook("") {}

    void borrowBook(const string& bookTitle) {
        borrowedBook = bookTitle;
        cout << name << " has borrowed the book: " << bookTitle << endl;
    }

    void returnBook() {
        if (!borrowedBook.empty()) {
```

```
            cout << name << " has returned the book: " << borrowedBook <<
endl;
            borrowedBook = "";
        } else {
            cout << name << " has no borrowed book to return." << endl;
        }
    }

    void displayBorrowerDetails() const {
        cout << "Borrower Name: " << name << "\nBorrower ID: " << id <<
endl;
        if (!borrowedBook.empty()) {
            cout << "Borrowed Book: " << borrowedBook << endl;
        } else {
            cout << "No borrowed book." << endl;
        }
    }
};

class Library : public Book, public Borrower {
public:
    Library(string t, string a, string i, string n, int id)
        : Book(t, a, i), Borrower(n, id) {}

    void displayLibraryDetails() {
        cout << "\nLibrary Details:\n";
        displayBookDetails();
        displayBorrowerDetails();
    }
};

int main() {
    Library library("The Great Gatsby", "F. Scott Fitzgerald", "978-
0743273565", "John Doe", 101);

    library.displayLibraryDetails();
    library.borrowBook("The Great Gatsby");
    library.displayLibraryDetails();
    library.returnBook();
    library.displayLibraryDetails();

    return 0;
}
```

5)Objective

Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

- SavingsAccount: Interest = Balance × Rate × Time.
- CurrentAccount: No interest, but includes a maintenance fee deduction.

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

using namespace std;

class Account {
protected:
    double balance;
    string accountNumber;

public:
    Account(double b, string accNum) : balance(b), accountNumber(accNum) {}

    virtual void calculateInterest() = 0;  // Pure virtual function

    void displayAccountDetails() const {
        cout << "Account Number: " << accountNumber << "\nBalance: $" <<
balance << endl;
    }

    virtual ~Account() {}  // Virtual destructor
};

class SavingsAccount : public Account {
private:
    double interestRate;  // Annual interest rate (as a percentage)
    double time;          // Time in years

public:
    SavingsAccount(double b, string accNum, double rate, double t)
        : Account(b, accNum), interestRate(rate), time(t) {}

    void calculateInterest() override {
        double interest = balance * interestRate * time / 100;
        balance += interest;
        cout << "Interest added: $" << interest << endl;
    }

    void displayAccountDetails() const override {
        Account::displayAccountDetails();
        cout << "Account Type: Savings\n";
    }
};

class CurrentAccount : public Account {
private:
    double maintenanceFee;  // Monthly maintenance fee
```

```
public:
    CurrentAccount(double b, string accNum, double fee)
        : Account(b, accNum), maintenanceFee(fee) {}

    void calculateInterest() override {
        balance -= maintenanceFee;
        cout << "Maintenance fee deducted: $" << maintenanceFee << endl;
    }

    void displayAccountDetails() const override {
        Account::displayAccountDetails();
        cout << "Account Type: Current\n";
    }
};

int main() {
    Account* acc1 = new SavingsAccount(1000.0, "SA12345", 5.0, 1.0);  // 5%
interest for 1 year
    Account* acc2 = new CurrentAccount(2000.0, "CA67890", 15.0);      //
$15 maintenance fee

    cout << "Savings Account Details:\n";
    acc1->displayAccountDetails();
    acc1->calculateInterest();
    cout << "Updated Balance: $" << acc1->balance << "\n\n";

    cout << "Current Account Details:\n";
    acc2->displayAccountDetails();
    acc2->calculateInterest();
    cout << "Updated Balance: $" << acc2->balance << "\n";

    delete acc1;
    delete acc2;

    return 0;
}
```

Very Hard

1)Objective

Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:
Manager: Add and calculate bonuses based on performance ratings.
Developer: Add and calculate overtime compensation based on extra hours worked.
The program should allow input for both types of employees and display their total earnings.

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

using namespace std;

class Employee {
protected:
    string name;
    int id;
    double salary;

public:
    Employee(string n, int i, double s) : name(n), id(i), salary(s) {}

    virtual double calculateEarnings() const = 0;  // Pure virtual function
to calculate total earnings

    void displayDetails() const {
        cout << "Employee Name: " << name << "\nEmployee ID: " << id <<
"\nSalary: $" << salary << endl;
    }
};

class Manager : public Employee {
private:
    double performanceRating;

public:
    Manager(string n, int i, double s, double rating)
        : Employee(n, i, s), performanceRating(rating) {}

    double calculateEarnings() const override {
        double bonus = 0.0;
        if (performanceRating >= 4.5) {
            bonus = salary * 0.20;  // 20% bonus for high performance
        } else if (performanceRating >= 3.0) {
            bonus = salary * 0.10;  // 10% bonus for average performance
        }
        return salary + bonus;
    }

    void displayDetails() const override {
        Employee::displayDetails();
        cout << "Performance Rating: " << performanceRating << endl;
```

```
    }
};

class Developer : public Employee {
private:
    int extraHoursWorked;

public:
    Developer(string n, int i, double s, int hours)
        : Employee(n, i, s), extraHoursWorked(hours) {}

    double calculateEarnings() const override {
        double overtimePay = extraHoursWorked * 20.0;  // $20 per extra
hour
        return salary + overtimePay;
    }

    void displayDetails() const override {
        Employee::displayDetails();
        cout << "Extra Hours Worked: " << extraHoursWorked << endl;
    }
};

int main() {
    string name;
    int id, extraHours;
    double salary, performanceRating;

    // Input and create Manager object
    cout << "Enter Manager details:\n";
    cout << "Name: ";
    getline(cin, name);
    cout << "ID: ";
    cin >> id;
    cout << "Salary: ";
    cin >> salary;
    cout << "Performance Rating (0.0 to 5.0): ";
    cin >> performanceRating;
    cin.ignore();  // Clear the input buffer

    Manager manager(name, id, salary, performanceRating);

    // Input and create Developer object
    cout << "\nEnter Developer details:\n";
    cout << "Name: ";
    getline(cin, name);
    cout << "ID: ";
    cin >> id;
    cout << "Salary: ";
    cin >> salary;
    cout << "Extra Hours Worked: ";
    cin >> extraHours;
    cin.ignore();  // Clear the input buffer

    Developer developer(name, id, salary, extraHours);

    // Display details and earnings
    cout << "\n--- Manager Details ---\n";
    manager.displayDetails();
```

```
    cout << "Total Earnings: $" << manager.calculateEarnings() << endl;

    cout << "\n--- Developer Details ---\n";
    developer.displayDetails();
    cout << "Total Earnings: $" << developer.calculateEarnings() << endl;

    return 0;
}
```

2) *Objective*

Create a C++ program to simulate a vehicle hierarchy using multi-level inheritance. Design a base class Vehicle that stores basic details (brand, model, and mileage). Extend it into the Car class to add attributes like fuel efficiency and speed. Further extend it into ElectricCar to include battery capacity and charging time. Implement methods to calculate:
Fuel Efficiency: Miles per gallon (for Car).
Range: Total distance the electric car can travel with a full charge.

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

using namespace std;

class Vehicle {
protected:
    string brand;
    string model;
    double mileage;

public:
    Vehicle(string b, string m, double mil) : brand(b), model(m),
mileage(mil) {}

    void displayVehicleDetails() const {
        cout << "Brand: " << brand << "\nModel: " << model << "\nMileage: "
<< mileage << " miles" << endl;
    }
};
class Car : public Vehicle {
protected:
    double fuelEfficiency;  // miles per gallon
    double speed;           // in miles per hour

public:
    Car(string b, string m, double mil, double fe, double spd)
        : Vehicle(b, m, mil), fuelEfficiency(fe), speed(spd) {}

    void displayCarDetails() const {
        displayVehicleDetails();
        cout << "Fuel Efficiency: " << fuelEfficiency << " mpg\nSpeed: " <<
speed << " mph" << endl;
    }

    double calculateFuelEfficiency() const {
```

```
        return fuelEfficiency;  // Just return the fuel efficiency as a
simple calculation
    }
};
class ElectricCar : public Car {
private:
    double batteryCapacity;  // in kWh
    double chargingTime;     // in hours
public:
    ElectricCar(string b, string m, double mil, double fe, double spd,
double batteryCap, double chargeTime)
        : Car(b, m, mil, fe, spd), batteryCapacity(batteryCap),
chargingTime(chargeTime) {}

    void displayElectricCarDetails() const {
        displayCarDetails();
        cout << "Battery Capacity: " << batteryCapacity << " kWh\nCharging
Time: " << chargingTime << " hours" << endl;
    }

    double calculateRange() const {
        double range = batteryCapacity * 4;  // Example: 1 kWh provides 4
miles of range
        return range;  // Calculate range based on battery capacity
    }
};
int main() {
    string brand, model;
    double mileage, fuelEfficiency, speed, batteryCapacity, chargingTime;

    // Input details for an Electric Car
    cout << "Enter Electric Car details:\n";
    cout << "Brand: ";
    getline(cin, brand);
    cout << "Model: ";
    getline(cin, model);
    cout << "Mileage: ";
    cin >> mileage;
    cout << "Fuel Efficiency (mpg): ";
    cin >> fuelEfficiency;
    cout << "Speed (mph): ";
    cin >> speed;
    cout << "Battery Capacity (kWh): ";
    cin >> batteryCapacity;
    cout << "Charging Time (hours): ";
    cin >> chargingTime;

    ElectricCar electricCar(brand, model, mileage, fuelEfficiency, speed,
batteryCapacity, chargingTime);

    // Display details and calculate the range for the Electric Car
    cout << "\n--- Electric Car Details ---\n";
    electricCar.displayElectricCarDetails();
    cout << "Range on a full charge: " << electricCar.calculateRange() << "
miles" << endl;

    return 0;
}
```

3)Objective

Design a C++ program using function overloading to perform arithmetic operations on complex numbers. Define a Complex class with real and imaginary parts. Overload functions to handle the following operations:
Addition: Sum of two complex numbers.
Multiplication: Product of two complex numbers.
Magnitude: Calculate the magnitude of a single complex number.
The program should allow the user to select an operation, input complex numbers, and display results in the format a + bi or a - bi (where b is the imaginary part).

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

class Complex {
private:
    double real;
    double imag;

public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imag + other.imag);
    }

    Complex operator*(const Complex& other) const {
        double realPart = real * other.real - imag * other.imag;
        double imagPart = real * other.imag + imag * other.real;
        return Complex(realPart, imagPart);
    }

    double magnitude() const {
        return sqrt(real * real + imag * imag);
    }

    void display() const {
        if (imag >= 0)
            cout << real << " + " << imag << "i" << endl;
        else
            cout << real << " - " << -imag << "i" << endl;
    }
};

int main() {
    double real1, imag1, real2, imag2;
    int choice;

    cout << "Enter the first complex number:\n";
    cout << "Real part: ";
    cin >> real1;
    cout << "Imaginary part: ";
```

```
    cin >> imag1;

    cout << "\nEnter the second complex number:\n";
    cout << "Real part: ";
    cin >> real2;
    cout << "Imaginary part: ";
    cin >> imag2;

    Complex c1(real1, imag1), c2(real2, imag2);

    cout << "\nSelect operation:\n";
    cout << "1. Add complex numbers\n";
    cout << "2. Multiply complex numbers\n";
    cout << "3. Calculate magnitude of the first complex number\n";
    cout << "4. Calculate magnitude of the second complex number\n";
    cout << "Enter choice (1/2/3/4): ";
    cin >> choice;

    switch(choice) {
        case 1: {
            Complex result = c1 + c2;
            cout << "Sum: ";
            result.display();
            break;
        }
        case 2: {
            Complex result = c1 * c2;
            cout << "Product: ";
            result.display();
            break;
        }
        case 3: {
            double mag = c1.magnitude();
            cout << "Magnitude of the first complex number: " << fixed <<
setprecision(2) << mag << endl;
            break;
        }
        case 4: {
            double mag = c2.magnitude();
            cout << "Magnitude of the second complex number: " << fixed <<
setprecision(2) << mag << endl;
            break;
        }
        default:
            cout << "Invalid choice.\n";
    }

    return 0;
}
```

4)Objective

Create a C++ program that uses polymorphism to calculate the area of various shapes. Define a base class Shape with a virtual method calculateArea(). Extend this base class into the following derived classes:
Rectangle: Calculates the area based on length and width.
Circle: Calculates the area based on the radius.
Triangle: Calculates the area using base and height.
The program should use dynamic polymorphism to handle these shapes and display the area of each.

```
#include <iostream>
#include <cmath>

using namespace std;

class Shape {
public:
    virtual double calculateArea() const = 0;
    virtual ~Shape() {}
};

class Rectangle : public Shape {
private:
    double length, width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double calculateArea() const override {
        return length * width;
    }
};

class Circle : public Shape {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    double calculateArea() const override {
        return M_PI * radius * radius;
    }
};

class Triangle : public Shape {
private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    double calculateArea() const override {
        return 0.5 * base * height;
    }
};
```

```
int main() {
    Shape* shape;
    int choice;
    double area;

    cout << "Select shape type:\n";
    cout << "1. Rectangle\n";
    cout << "2. Circle\n";
    cout << "3. Triangle\n";
    cout << "Enter choice (1/2/3): ";
    cin >> choice;

    switch(choice) {
        case 1: {
            double length, width;
            cout << "Enter length and width of the rectangle: ";
            cin >> length >> width;
            shape = new Rectangle(length, width);
            break;
        }
        case 2: {
            double radius;
            cout << "Enter radius of the circle: ";
            cin >> radius;
            shape = new Circle(radius);
            break;
        }
        case 3: {
            double base, height;
            cout << "Enter base and height of the triangle: ";
            cin >> base >> height;
            shape = new Triangle(base, height);
            break;
        }
        default:
            cout << "Invalid choice.\n";
            return 0;
    }

    area = shape->calculateArea();
    cout << "Area: " << area << endl;

    delete shape;
    return 0;
}
```

5)Objective

Create a C++ program that demonstrates function overloading to calculate the area of different geometric shapes. Implement three overloaded functions named calculateArea that compute the area for the following shapes:
Circle: Accepts the radius.
Rectangle: Accepts the length and breadth.
Triangle: Accepts the base and height.
Additionally, use a menu-driven program to let the user choose the type of shape and input the respective parameters. Perform necessary validations on the input values.

```
#include <iostream>
#include <cmath>
#include <limits>

using namespace std;

double calculateArea(double radius) {
    return M_PI * radius * radius;
}

double calculateArea(double length, double breadth) {
    return length * breadth;
}

double calculateArea(double base, double height) {
    return 0.5 * base * height;
}

bool isValidInput(double value) {
    if(value <= 0) {
        cout << "Error: Value must be positive.\n";
        return false;
    }
    return true;
}

int main() {
    int choice;
    double area;

    while(true) {
        cout << "Select shape type:\n";
        cout << "1. Circle\n";
        cout << "2. Rectangle\n";
        cout << "3. Triangle\n";
        cout << "4. Exit\n";
        cout << "Enter choice (1/2/3/4): ";
        cin >> choice;

        if(choice == 4) {
            cout << "Exiting program.\n";
            break;
        }
```

```
        switch(choice) {
            case 1: {
                double radius;
                cout << "Enter radius of the circle: ";
                cin >> radius;
                if(isValidInput(radius)) {
                    area = calculateArea(radius);
                    cout << "Area of Circle: " << area << endl;
                }
                break;
            }
            case 2: {
                double length, breadth;
                cout << "Enter length and breadth of the rectangle: ";
                cin >> length >> breadth;
                if(isValidInput(length) && isValidInput(breadth)) {
                    area = calculateArea(length, breadth);
                    cout << "Area of Rectangle: " << area << endl;
                }
                break;
            }
            case 3: {
                double base, height;
                cout << "Enter base and height of the triangle: ";
                cin >> base >> height;
                if(isValidInput(base) && isValidInput(height)) {
                    area = calculateArea(base, height);
                    cout << "Area of Triangle: " << area << endl;
                }
                break;
            }
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```