Question 1:

Demonstrates the use of a base class pointer pointing to a derived class object to calculate and display the area of a circle. Implement a base class Shape with a virtual method calculateArea and a derived class Circle that inherits from Shape and overrides the calculateArea method to calculate the area of a circle. Use dynamic memory allocation to create a Circle object and use a base class pointer to call the calculateArea method.

Input Format:

- The program prompts the user to enter the radius of the circle. The user should input a numeric value representing the radius of the circle.
- PI= 3.14159

Output Format:

- After receiving the input radius, the program calculates the area of the circle using the formula for the area of a circle (3.14 * radius^2) and displays the calculated area in the output sentence. The calculated area value will replace [calculated area value] in the output.

Title for Question 1: Display the area of a circle-pointers to object

Solution:

```
#include <iostream>

class Shape {
public:
    virtual double calculateArea() const {
        return 0.0; // Base class placeholder
    }
};

class Circle : public Shape {
private:
    double radius;

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

    double calculateArea() const override {
        return 3.14159 * radius * radius;
    }
};

int main() {
    double radius;
    //std::cout << "Enter the radius of the circle: ";
    std::cin >> radius;

    // Create a base class pointer pointing to a derived class object
    Shape* shapePtr = new Circle(radius);

    // Calculate and display the area using the base class pointer
    double area = shapePtr->calculateArea();
    std::cout << "The area of the circle is: " << area << std::endl;

    // Clean up memory
    delete shapePtr;

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2.5	The area of the circle is: 19.6349
2	0.0	The area of the circle is: 0
3	10.75	The area of the circle is: 363.05
4	7.8	The area of the circle is: 191.134
5	1.25	The area of the circle is: 4.90873
6	100.0	The area of the circle is: 31415.9

White List:

Black List:

Question 2:

Implement a program that performs complex number operations. Implement a ComplexNumber class to represent complex numbers and overload the +, -, and * operators to perform addition, subtraction, and multiplication of complex numbers.

The program should prompt the user to input the real and imaginary parts of two complex numbers and then display the results of addition, subtraction, and multiplication.

Input Format:

- The program prompts the user to enter the real part of complex number 1.
- The program prompts the user to enter the imaginary part of complex number 1.
- The program prompts the user to enter the real part of complex number 2.
- The program prompts the user to enter the imaginary part of complex number 2.

Output Format:

- The program calculates the addition of the two complex numbers and displays the result in the form of "Addition: a + bi" where 'a' is the real part and 'b' is the imaginary part of the result.
- The program calculates the subtraction of the two complex numbers and displays the result in the form of "Subtraction: a + bi" where 'a' is the real part and 'b' is the imaginary part of the result.
- The program calculates the multiplication of the two complex numbers and displays the result in the form of "Multiplication: a + bi" where 'a' is the real part and 'b' is the imaginary part of the result.

Title for Question 2: Performs complex number operations using operator overloading

Solution:

```
#include <iostream>

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

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

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

    // Input for complex number 1
    //std::cout << "Enter real part of complex number 1: ";
    std::cin >> real1;
    //std::cout << "Enter imaginary part of complex number 1: ";
    std::cin >> imag1;

    // Input for complex number 2
   // std::cout << "Enter real part of complex number 2: ";
    std::cin >> real2;
   // std::cout << "Enter imaginary part of complex number 2: ";
    std::cin >> imag2;

    Complex complex1(real1, imag1);
    Complex complex2(real2, imag2);

    // Perform addition
    Complex addition = complex1 + complex2;
    std::cout << "Addition: ";
    addition.display();

    // Perform subtraction
    Complex subtraction = complex1 - complex2;
    std::cout << "Subtraction: ";
    subtraction.display();

    // Perform multiplication
    Complex multiplication = complex1 * complex2;
    std::cout << "Multiplication: ";
    multiplication.display();
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	2 3 4 -1	Addition: 6 + 2i Subtraction: -2 + 4i Multiplication: 11 + 10i
2	5.5 -2.5 3.2 1.8	Addition: 8.7 - 0.7i Subtraction: 2.3 - 4.3i Multiplication: 22.1 + 1.9i
3	0 7 0 -2	Addition: 0 + 5i Subtraction: 0 + 9i Multiplication: 14 + 0i
4	-1 4 -3 2	Addition: -4 + 6i Subtraction: 2 + 2i Multiplication: -5 - 14i
5	2.8 0 -1.2 0	Addition: 1.6 + 0i Subtraction: 4 + 0i Multiplication: -3.36 + 0i
6	1 -1 1 1	Addition: 2 + 0i Subtraction: 0 - 2i Multiplication: 2 + 0i

White List:

Black List:

Question 3:

Program to implement the method overloading with a different number of arguments and different return types.

Input Format:

- Two integers (`intA` and `intB`).

- Two floating-point numbers (`doubleA` and `doubleB`).
- Three integers (`intA`, `intB`, and `intC` for the second addition).
- Three floating-point numbers (`doubleA`, `doubleB`, and `doubleC` for the second addition).

Output Format:

- "Result of adding two integers: " followed by the sum of the first two integers.
- "Result of adding two doubles: " followed by the sum of the first two doubles.
- "Result of adding three integers: " followed by the sum of the second set of integers.
- "Result of adding three doubles: " followed by the sum of the second set of doubles.

Title for Question 3: Different number of arguments and different return types-method overloading

Solution:

```
#include <iostream>

class MathOperations {
public:
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }

    double add(double a, double b, double c) {
        return a + b + c;
    }
};

int main() {
    MathOperations math;

    int intA, intB, intC;
    double doubleA, doubleB, doubleC;

    // Input for two integers
    //std::cout << "Enter two integers:" << std::endl;
   // std::cout << "First integer: ";
    std::cin >> intA;
   // std::cout << "Second integer: ";
    std::cin >> intB;

    int intResult1 = math.add(intA, intB);
    std::cout << "Result of adding two integers: " << intResult1 << std::

    // Input for two doubles
   // std::cout << "Enter two doubles:" << std::endl;
    //std::cout << "First double: ";
    std::cin >> doubleA;
  //  std::cout << "Second double: ";
    std::cin >> doubleB;

    double doubleResult1 = math.add(doubleA, doubleB);
    std::cout << "Result of adding two doubles: " << doubleResult1 << std

    // Input for three integers
  //  std::cout << "Enter three integers:" << std::endl;
  //  std::cout << "First integer: ";
    std::cin >> intA;
  //  std::cout << "Second integer: ";
    std::cin >> intB;
  //  std::cout << "Third integer: ";
    std::cin >> intC;

    int intResult2 = math.add(intA, intB, intC);
    std::cout << "Result of adding three integers: " << intResult2 << std

    // Input for three doubles
    //std::cout << "Enter three doubles:" << std::endl;
    //std::cout << "First double: ";
    std::cin >> doubleA;
   // std::cout << "Second double: ";
    std::cin >> doubleB;
   // std::cout << "Third double: ";
    std::cin >> doubleC;

    double doubleResult2 = math.add(doubleA, doubleB, doubleC);
    std::cout << "Result of adding three doubles: " << doubleResult2 << s

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 5 2.5 3.5 2 3 4 1.2 2.33.4	Result of adding two integers: 10 Result of adding two doubles: 6 Result of adding three integers: 9 Result of adding three doubles: 3.93
2	15 25 5.7 2.8 10 20 30 1.5 2.5 3.5	Result of adding two integers: 40 Result of adding two doubles: 8.5 Result of adding three integers: 60 Result of adding three doubles: 7.5
3	-5 -5 -2.5 -3.5 -2 3 -4 1.2 -2.3 3.4	Result of adding two integers: -10 Result of adding two doubles: -6 Result of adding three integers: -3 Result of adding three doubles: 2.3
4	0 0 0.0 0.0 0 0 0 0.0 0.0 0.0	Result of adding two integers: 0 Result of adding two doubles: 0 Result of adding three integers: 0 Result of adding three doubles: 0
5	100 -50 -10.5 20.5 1 2 3 0.1 0.2 0.3	Result of adding two integers: 50 Result of adding two doubles: 10 Result of adding three integers: 6 Result of adding three doubles: 0.6
6	0 0 0.5 -0.5 10 -20 30 -1.5 2.5 -3.5	Result of adding two integers: 0 Result of adding two doubles: 0 Result of adding three integers: 20 Result of adding three doubles: -2.5

White List:

Black List:

Question 4:

Define a Account class with attributes for account number and balance. Include a virtual function calculateInterest() that returns 0.0 (placeholder).

Input Format:

- Prompt the user to enter the account number.
- Accept the user input for the account number (an integer).
- Prompt the user to enter the balance.
- Accept the user input for the balance (a floating-point number).

Output Format:

Display the account information:

- Print "Account Number: \<account_number\>" where \<account_number\> is the entered account number.
- Print "Balance: $\<balance\>" where \<balance\> is the entered balance.

Calculate and display the interest:

- Calculate the interest using the overridden \`calculateInterest()\` function in the \`SavingsAccount\` class.
- Print "Interest: $\<calculated_interest\>" where \<calculated_interest\> is the calculated interest based on a 5% interest rate.

Title for Question 4: Calculate Interest-overriding

Solution:

```
#include <iostream>

class Account {
protected:
    int accountNumber;
    double balance;

public:
    Account(int accNumber, double bal) : accountNumber(accNumber), balanc

    virtual double calculateInterest() const {
        return 0.0; // Placeholder interest calculation
    }

    void displayAccountInfo() const {
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Balance: $" << balance << std::endl;
    }
};

class SavingsAccount : public Account {
public:
    SavingsAccount(int accNumber, double bal) : Account(accNumber, bal) {

    double calculateInterest() const override {
        return balance * 0.05; // 5% interest rate
    }
};

int main() {
    int accountNumber;
    double balance;

    //std::cout << "Enter account number: ";
    std::cin >> accountNumber;
    //std::cout << "Enter balance: ";
    std::cin >> balance;

    SavingsAccount savingsAccount(accountNumber, balance);
    savingsAccount.displayAccountInfo();

    double interest = savingsAccount.calculateInterest();
    std::cout << "Interest: $" << interest << std::endl;

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	12345 1000.0	Account Number: 12345 Balance: $1000 Interest: $50
2	67890 5000.0	Account Number: 67890 Balance: $5000 Interest: $250
3	98765 15000.0	Account Number: 98765 Balance: $15000 Interest: $750
4	54321 200.0	Account Number: 54321 Balance: $200 Interest: $10
5	11111 0.0	Account Number: 11111 Balance: $0 Interest: $0
6	24680 8000.0	Account Number: 24680 Balance: $8000 Interest: $400

White List:

Black List:

Question 5:

Write a program to perform matrix addition and multiplication. The program should take the number of rows and columns as input from the user for two matrices, fill the matrices with integer elements, and then perform addition and multiplication. Overload the + and * operators to accomplish these tasks.

Constraints:

- The number of rows and columns in each matrix should be between 1 and 20.
- The elements of the matrix are integers.

Title for Question 5: Matrix addition and multiplication using overloading

Solution:

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

class Matrix {
private:
    int** mat;
    int rows, cols;

public:
    Matrix(int rows, int cols) : rows(rows), cols(cols) {
        mat = new int*[rows];
        for (int i = 0; i < rows; ++i) {
            mat[i] = new int[cols];
        }
    }

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

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

    Matrix operator+(const Matrix& other) {
        if (rows != other.rows || cols != other.cols) throw "Addition not
        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] + other.mat[i][j];
            }
        }
        return result;
    }

    Matrix operator*(const Matrix& other) {
        if (cols != other.rows) throw "Multiplication not possible.";
        Matrix result(rows, other.cols);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < other.cols; ++j) {
                result.mat[i][j] = 0;
                for (int k = 0; k < cols; ++k) {
                    result.mat[i][j] += mat[i][k] * other.mat[k][j];
                }
            }
        }
        return result;
    }

    ~Matrix() {
        for (int i = 0; i < rows; ++i) {
            delete[] mat[i];
        }
        delete[] mat;
    }
};

int main() {
    int n1, m1, n2, m2;
    cin >> n1 >> m1;
    Matrix mat1(n1, m1);
    mat1.input();

    cin >> n2 >> m2;
    Matrix mat2(n2, m2);
    mat2.input();

    try {
        Matrix additionResult = mat1 + mat2;
        cout << "Addition Result:" << endl;
        additionResult.display();
    } catch (const char* msg) {
        cout << msg << endl;
    }

    try {
        Matrix multiplicationResult = mat1 * mat2;
        cout << "Multiplication Result:" << endl;
        multiplicationResult.display();
    } catch (const char* msg) {
        cout << msg << endl;
    }

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs

1	3 3 1 2 3 4 5 6 7 8 9 3 3 10 11 12 13 14 15 16 17 18	Addition Result: 11 13 15 17 19 21 23 25 27 Multiplication Result: 84 90 96 201 216 231 318 342 366
2	2 2 1 1 1 1 2 2 1 1 1 1	Addition Result: 2 2 2 2 Multiplication Result: 2 2 2 2
3	2 3 1 2 3 4 5 6 3 2 7 8 9 10 11 12	Addition not possible. Multiplication Result: 58 64 139 154
4	2 2 1 2 3 4 2 3 5 6 7	Addition not possible. Multiplication Result: 5 6 7 15 18 21
5	1 1 2 1 1 3	Addition Result: 5 Multiplication Result: 6
6	2 2 1 2 3 4 2 2 5 6 7 8	Addition Result: 6 8 10 12 Multiplication Result: 19 22 43 50

White List:

Black List: