Subject:- C++
LAB Assignment - 6

1) Write a C++ program to overload unary minus operator using member functions.

```
#include <iostream>

class Point {
private:
    double x;  // x-coordinate
    double y;  // y-coordinate

public:
    // Constructor to initialize the point
    Point(double xCoord = 0.0, double yCoord = 0.0) : x(xCoord), y(yCoord) {}

    // Overload the unary - operator
    Point operator-() const {
        return Point(-x, -y);  // Negate both coordinates
    }

    // Function to display the point
    void display() const {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main() {
    Point p(3.0, 4.0);  // Create a Point (3, 4)

    std::cout << "Original point: ";
```

```
    p.display();  // Display the original point

    Point negatedPoint = -p;  // Use the overloaded - operator

    std::cout << "Negated point: ";
    negatedPoint.display();  // Display the negated point

    return 0;
}
```

Output:

```
Original Value: 10
After applying unary minus Value: -10
```

2) Write a C++ program to overload unary minus operator using the friend function.

```
#include <iostream>

class Vector {
private:
    double x;  // x-component
    double y;  // y-component
    double z;  // z-component

public:
    // Constructor to initialize the vector
    Vector(double xCoord = 0.0, double yCoord = 0.0, double zCoord = 0.0)
        : x(xCoord), y(yCoord), z(zCoord) {}

    // Friend function to overload the unary - operator
    friend Vector operator-(const Vector& v) {
        return Vector(-v.x, -v.y, -v.z);  // Negate all components
    }
```

```
    // Function to display the vector
    void display() const {
        std::cout << "(" << x << ", " << y << ", " << z << ")" << std::endl;
    }
};

int main() {
    Vector v(1.0, -2.0, 3.0);  // Create a Vector (1, -2, 3)

    std::cout << "Original vector: ";
    v.display();  // Display the original vector

    Vector negatedVector = -v;  // Use the overloaded - operator

    std::cout << "Negated vector: ";
    negatedVector.display();  // Display the negated vector

    return 0;
}
```

Output:

```
Original Value: 10
After applying unary minus Value: -10
```

3) Write a C++ program to overload binary operator (+) using member function.

```
#include <iostream>

class Matrix {
private:
    double elements[2][2];  // 2x2 matrix

public:
    // Constructor to initialize the matrix
    Matrix(double a11 = 0, double a12 = 0, double a21 = 0, double a22 = 0) {
        elements[0][0] = a11;
        elements[0][1] = a12;
        elements[1][0] = a21;
        elements[1][1] = a22;
    }

    // Overload the + operator
    Matrix operator+(const Matrix& other) {
        Matrix result;
        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                result.elements[i][j] = this->elements[i][j] + other.elements[i][j];
            }
        }
        return result;
    }

    // Function to display the matrix
    void display() const {
        for (int i = 0; i < 2; ++i) {
            for (int j = 0; j < 2; ++j) {
                std::cout << elements[i][j] << " ";
            }
            std::cout << std::endl;
```

```
        }
    }
};

int main() {
    // Create two matrices
    Matrix mat1(1, 2, 3, 4);  // Matrix 1
    Matrix mat2(5, 6, 7, 8);  // Matrix 2

    // Display the original matrices
    std::cout << "Matrix 1:\n";
    mat1.display();

    std::cout << "Matrix 2:\n";
    mat2.display();

    // Add the two matrices using the overloaded + operator
    Matrix result = mat1 + mat2;

    // Display the result
    std::cout << "Result of addition:\n";
    result.display();

    return 0;
}
```

Output:

```
First Value: 10
Second Value: 20
Sum Value: 30
```

4) Write a C++ program to overload binary operator(+)

using the friend function.

```
#include <iostream>

class Complex {
private:
    double real;  // Real part
    double imag;  // Imaginary part

public:
    // Constructor to initialize complex numbers
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // Friend function to overload the + operator
    friend Complex operator+(const Complex& c1, const Complex& c2)
{
        return Complex(c1.real + c2.real, c1.imag + c2.imag);
    }

    // Function to display the complex number
    void display() const {
        std::cout << real << " + " << imag << "i" << std::endl;
    }
};

int main() {
    Complex num1(3.5, 2.5);  // Create a Complex number (3.5 + 2.5i)
    Complex num2(1.5, 4.5);  // Create another Complex number (1.5 +
4.5i)

    // Use the overloaded + operator
    Complex result = num1 + num2;
```

```
    std::cout << "Result of addition: ";
    result.display();  // Display the result

    return 0;
}
```

Output:

```
First Value: 10
Second Value: 20
Sum Value: 30
```

5) Write a C++ program to define a Vector class that handles vectors of size 3. The class should include the following features:

Constructors:

a. A default constructor that initializes all elements of the vector to 0.

b. A parameterized constructor that initializes the vector elements with values provided through an array.

Overloaded Operators:

c. Overload the * operator to allow scalar multiplication with the vector. Ensure that multiplication works from both sides, i.e., scalar * vector and vector * scalar.

Display Function:

d. A function to print the elements of the vector.

```
#include <iostream>

class Vector {
private:
    double elements[3];  // Array to hold the vector components

public:
    // Default constructor that initializes all elements to 0
    Vector() {
        for (int i = 0; i < 3; ++i) {
            elements[i] = 0.0;  // Set each element to 0
        }
    }

    // Parameterized constructor to initialize the vector with an array
    Vector(double arr[3]) {
        for (int i = 0; i < 3; ++i) {
            elements[i] = arr[i];  // Initialize with array values
        }
    }

    // Overload the * operator for scalar multiplication (vector * scalar)
    Vector operator*(double scalar) const {
        Vector result;
        for (int i = 0; i < 3; ++i) {
            result.elements[i] = elements[i] * scalar;
        }
        return result;
    }

    // Friend function to overload the * operator for scalar multiplication (scalar *
vector)
    friend Vector operator*(double scalar, const Vector& vec) {
        return vec * scalar;  // Call the member function for multiplication
    }

    // Function to display the vector
```

```
    void display() const {
        std::cout << "(" << elements[0] << ", " << elements[1] << ", " << elements[2]
<< ")" << std::endl;
    }
};

int main() {
    // Create a default vector
    Vector defaultVector;
    std::cout << "Default vector: ";
    defaultVector.display();  // Display the default vector

    // Create a vector using an array
    double arr[3] = {1.0, 2.0, 3.0};
    Vector specificVector(arr);
    std::cout << "Specific vector: ";
    specificVector.display();  // Display the specific vector

    // Scalar multiplication from the left
    Vector result1 = 2.0 * specificVector;  // 2.0 * Vector
    std::cout << "Scalar (left) multiplication: ";
    result1.display();

    // Scalar multiplication from the right
    Vector result2 = specificVector * 3.0;  // Vector * 3.0
    std::cout << "Scalar (right) multiplication: ";
    result2.display();

    return 0;
}
```

Output:

```
Default vector: Vector: (0, 0, 0)
Parameterized vector: Vector: (1, 2, 3)
After multiplying by scalar (Vector * scalar): Vector: (2, 4, 6)
After multiplying by scalar (Scalar * vector): Vector: (3, 6, 9)
```