Objective

- To understand the concept of operator overloading in C++.
- To implement overloading of unary, binary, and stream operators.
- To practice both member and friend function overloading.

Theory

Operator overloading allows you to redefine the behavior of operators for user-defined types (classes). By overloading operators, you can define how operators such as +, -, ==, <<, etc., work when used with class objects.

It improves code readability, especially when working with abstract data types like vectors, matrices, or complex numbers.

Types of Operator Overloading:

1. **Unary Operator Overloading** – Works on one operand.
2. **Binary Operator Overloading** – Works on two operands.
3. **Friend Function Overloading** – Used when the left operand is not an object of the class.
4. **Stream Insertion (<<) and Extraction (>>) Overloading** – Used for input/output of user-defined types.

Syntax:

◆ 1. Unary Operator Overloading (Prefix)

```
class Sample {

public:

    int value;

    void operator++() {

        ++value;

    }
};
```

2. Binary Operator Overloading (Member Function)

```
class Sample {
public:
    int value;
    Sample operator+(const Sample& obj) {
        Sample temp;
        temp.value = value + obj.value;
        return temp;
    }
};
```

3. Binary Operator Overloading (Friend Function)

```
class Sample {
public:
    int value;
    Sample(int v) : value(v) {}
    friend Sample operator-(Sample a, Sample b);
};

Sample operator-(Sample a, Sample b) {
    return Sample(a.value - b.value);
}
```

4. Stream Insertion and Extraction

```
class Sample {
public:
    int value;
    friend std::ostream& operator<<(std::ostream& out, const Sample& s);
    friend std::istream& operator>>(std::istream& in, Sample& s);
};

std::ostream& operator<<(std::ostream& out, const Sample& s) {
    out << s.value;
    return out;
}

std::istream& operator>>(std::istream& in, Sample& s) {
    in >> s.value;
    return in;
}
```