

Himalaya College of Engineering

**Advanced C++ Programming Lab Report**

Lab 5: Operator overloading in C++

**Prepared By :** Monsoon Sapkota (HCE081BEI022)

**Subject :** Object-Oriented Programming (OOP)

**Program :** Bachelor of Electronics, Communication and Information Engineering

**Institution :** Himalaya College of Engineering

**Date :** June 13, 2025

**OBJECTIVE**

* To understand the concept and purpose of operator overloading in C++.
* To implement various types of operator overloading for custom data types.
* To learn how to overload binary, unary, and I/O stream operators.
* To enhance code readability and provide intuitive syntax for user-defined classes.

**BACKGROUND THEORY**

Operator overloading is a powerful feature in C++ that allows operators to be redefined or overloaded for user-defined data types. This means you can use operators like +, -, \*, /, ==, ++, --, etc., with objects of your own classes, giving them special meaning for those classes. The primary purpose of operator overloading is to provide a more natural and intuitive syntax for operations involving user-defined types, making the code more readable and easier to understand.

When an operator is overloaded, it is implemented as a function. This function can be either a member function of the class or a non-member (friend) function. The choice between a member function and a friend function often depends on the type of operator and the number of operands.

**General Syntax for Operator Overloading:**

1. A member function inside a class: class ClassName { public:

// Constructor and data members

ClassName(data\_type var) : variable(var) {} // Overload operator as member function return\_type operator<symbol>(const ClassName& other) {

// Define operator behavior return result;

} private:

data\_type variable;

};

1. Or a non-member (often friend) function: class ClassName { public:

// Constructor and data members

ClassName(data\_type var) : variable(var) {} // Declare friend function for operator overloading friend return\_type operator<symbol>(const ClassName& obj1, const ClassName& obj2); private:

data\_type variable;

};

// Define non-member operator function return\_type operator<symbol>(const ClassName& obj1, const ClassName& obj2) {

// Define operator behavior return result;

}

**Importance of Operator Overloading**

1. **Makes Code Easy to Read:**

Operator overloading lets us use symbols like +, -, or == with objects, making the code look simple and clear.

1. **Hides Internal Details:**

It helps create user-friendly data types where complex details are hidden, and only useful operations are shown.

1. **Saves Time and Effort:**

Once an operator is overloaded in a class, it can be reused easily. This reduces repeated code and makes updates easier.

1. **Supports OOP Concepts:**

Operator overloading follows object-oriented ideas, like **polymorphism**, by letting the same operator work in different ways for different objects.

**Rules of Operator Overloading in C++**

1. Only existing operators can be overloaded; new operator symbols cannot be created.
2. At least one operand must be a user-defined type like a class or structure.
3. Operator precedence and associativity remain unchanged after overloading.
4. The number of operands for an operator cannot be changed.
5. Operators such as =, (), [], and -> must be overloaded as member functions.
6. Friend functions can be used to overload operators needing access to private members.
7. Overloaded operators should behave in a meaningful and expected way.