LAB SESSION 13

Operator Overloading in C++

Objective

The objective of this lab is to understand and apply the concept of operator overloading in C++. By the end of this lab, you should be able to overload operators to perform custom operations on objects of user-defined classes.

Introduction

Operator overloading allows the redefinition of the behavior of operators for user-defined types. This makes it possible to use operators like +, -, *, and others with class objects, making the code more intuitive and easier to understand.

Theory

Operator Overloading

Operator overloading allows you to define how operators work with class objects. The operator keyword is used to define an operator function.

Syntax:

returnType operator operatorSymbol(arguments);

Example

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

class Complex {
private:
   float real;
   float imag;
public:
   Complex() : real(0), imag(0) {}
   Complex(float r, float i) : real(r), imag(i) {}

   // Overloading the '+' operator
   Complex operator + (const Complex& obj) {
      Complex temp;
      temp.real = real + obj.real;
      temp.imag = imag + obj.imag;
      return temp;
   }
```

```
    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(3.5, 2.5), c2(1.5, 4.5);
    Complex c3 = c1 + c2;
    c3.display();
    return 0;
}
```

In this example, the + operator is overloaded to add two Complex objects.

Conclusion

Operator overloading provides a powerful way to define custom operations for user-defined types. By overloading operators, you can make your code more intuitive and easier to read, while still performing complex operations on class objects.

Exercise:

1. Overload the * operator to multiply two Matrix objects (define a simple Matrix class with a 2D array).