**OOP Lab 08**

**In Lab Task-03**

***Rayyan Aamir***

***BCS – 2F***

***24K-0687***

**Q1:**

Yes, the friend function can overload the += operator as it has access to the private and protected members of an object of a class and directly modify the left-hand operand if it is passed by a non-constant reference.

Implementation:

1. First, declare the friend function inside the class.

2. Define the function to accept the left-hand operand as a non-constant reference of an object (to modify) and the right-hand operand as a constant reference of an object (to avoid unnecessary copies).

3. Modify the left-hand operand directly without any getters or setters, as it is a friend function.

4. Return the left-hand operand.

**Code:**

#include <iostream>

using namespace std;

class Number{

    int value;

public:

    Number(int v) : value(v) {}

    void display() const {

        cout << "Number: " << value << endl;

    }

    friend Number& operator += (Number &lhs, const Number &rhs);

};

Number& operator += (Number &lhs, const Number &rhs){

    lhs.value += rhs.value;

    return lhs;

}

int main(){

    Number obj1(5), obj2(10);

    obj1.display();

    obj2.display();

    obj1 += obj2;

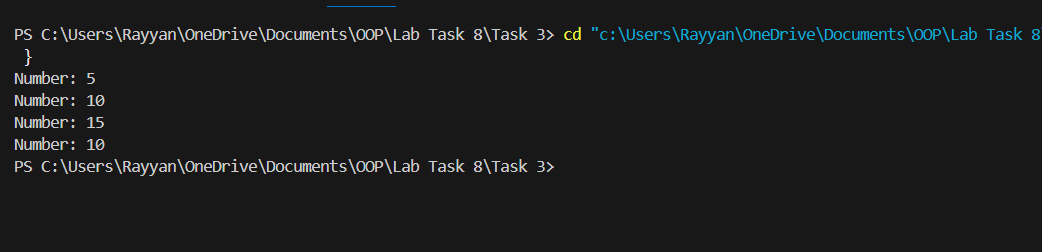
    obj1.display();

    obj2.display();

    return 0;

}

**Sample Output:**



**Q2:**

Yes, a friend function can be used in this scenario. It is particularly useful when a primitive type appears on the right-hand side, such as object + int.

However, if the primitive type is on the left-hand side, like int + object, the friend function must be implemented accordingly to manage that specific order.

**Code:**

#include <iostream>

using namespace std;

class Number{

    int value;

public:

    Number(int v) : value(v) {}

    void display() const {

        cout << "Number: " << value << endl;

    }

    friend Number operator + (const Number &lhs, int rhs);

    friend Number operator + (int lhs, const Number &rhs);

};

Number operator + (const Number &lhs, int rhs){

    return Number(lhs.value + rhs);

}

Number operator + (int lhs, const Number &rhs){

    return Number(lhs + rhs.value);

}

int main(){

    Number obj1(10);

    Number obj2 = obj1 + 30;

    Number obj3 = 15 + obj1;

    obj1.display();

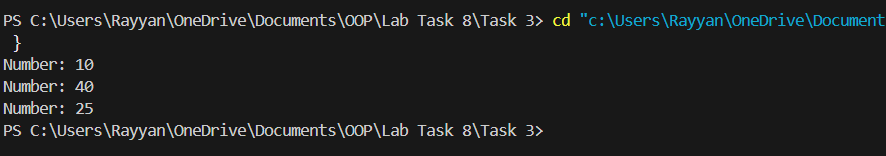
    obj2.display();

    obj3.display();

    return 0;

}

**Sample Output:**



**Q3:**

No, a friend function must require an object to access non-static private or protected members. It cannot access them directly

(like static members) because friendship grants access only through instances of the class.

Does it always need an object to access these members?

Yes, if you need to access non-static private or protected members of an object of a class.

Can a friend function access them directly without an object?

No, unless you need to access static members.

Under what conditions might it fail?

If the member is non-static and the function tries to access it without an object.

**Code:**

#include <iostream>

using namespace std;

class Number{

    int value;

    static int staticMember;

public:

    Number(int v) : value(v) {}

    friend void accessMember(const Number &obj);

    friend void accessStaticMember();

};

void accessMember(const Number &obj){

    cout << "Private data member (Value): " << obj.value << endl;

}

void accessStaticMember(){

    cout << "Static data member: " << Number :: staticMember << endl;

}

int Number :: staticMember = 50;

int main(){

    Number obj(7);

    accessMember(obj);

    accessStaticMember();

    return 0;

}

**Sample Output:**

