**TASK 03**

**Q1:**

Yes, a friend function can be used to overload an operator that modifies the invoking object. The += operator is an example of such an operator, as it modifies the left-hand operand.

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

MyClass(int val) : value(val) {}

// Friend function to overload +=

friend MyClass& operator+=(MyClass& obj, int add) {

obj.value += add;

return obj;

}

int getValue() const { return value; }

};

int main() {

MyClass obj(10);

obj += 5;

cout << "Value after += operation: " << obj.getValue() << endl;

return 0;

}



**Q2:**

Yes, a friend function can be used to overload an operator when one of the operands is a primitive data type. For example, overloading the + operator to allow addition between an object and an integer is possible using a friend function.

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

MyClass(int val) : value(val) {}

friend MyClass operator+(const MyClass& obj, int add) {

return MyClass(obj.value + add);

}

int getValue() const { return value; }

};

int main() {

MyClass obj(10);

MyClass result = obj + 5;

cout << "Value after + operation: " << result.getValue() << endl;

return 0;

}



**Q3:**

No, a friend function cannot access private and protected members of a class without using an object of that class. The friend function does not have direct access to the class's members without an instance of the class.

To access private or protected members, the friend function must use an object of the class. The object provides the context for accessing the members.

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

MyClass(int val) : value(val) {}

friend void accessPrivateMember();

};

void accessPrivateMember() {

MyClass obj(10);

cout << "Accessing private member without object: " << obj.value << endl;

}

int main() {

accessPrivateMember();

return 0;

}

