Question 1: Can a friend function be used to overload an operator that modifies the invoking object?

Problem Statement:
Consider the += operator, which modifies the left-hand operand. **Can a friend function be used to overload this operator?**

- If yes, how should it be implemented?
- If no, what alternative approach should be used?

Justify your answer with supporting C++ code.

1. Yes, a friend function can be used to overload an operator, even though it modifies the invoking object. Friend functions can modify private members of the class they're friends with, therefore a friend function can be used to overload the += operator. The left-hand operand is modified, so it is passed as a non-constant value, and the right operator is not modified, so it is passed as a constant value.

Supporting C++ code: (output: a=8)

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

class Number {
private:
    int value;

public:
    Number(int v = 0) : value(v) {}

    friend Number& operator+=(Number& left, const Number& right);

    int getValue() const {
        return value;
    }
};

Number& operator+=(Number& left, const Number& right) {
    left.value += right.value;
    return left;
}

int main() {
    Number a(5);
    Number b(3);

    a += b;

    cout << "a = " << a.getValue() << endl;
}
```

Question 2: Is it possible to overload an operator using a friend function if one of the operands is a primitive data type?

Problem Statement:
Suppose we want to overload the + operator to allow addition between an object and a primitive type (e.g., object + int).

- **Can a friend function handle this case?**
- If yes, how would it be implemented?
- If no, what limitations exist?

Justify your answer with supporting C++ code.

Yes, a friend function **can** be used to overload operators when one operand is a primitive data type. Object + int can be implemented this way using friend function:

Output: num + 3 = 8

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

class TestClass {
private:
    int value;

public:
    TestClass(int v = 0) : value(v) {}

    friend TestClass operator+(const TestClass & left, int right);

    int getValue() const { return value; }
};

TestClass operator+(const TestClass& left, int right) {
    return TestClass(left.value + right);
}

int main() {
    TestClass num(5);

    TestClass result1 = num + 3;

    cout << "num + 3 = " << result1.getValue() << endl;
}
```

Question 3: Can a friend function access private and protected members of a class without using an object of that class?

Problem Statement:
A friend function is granted access to private and protected members of a class.

- **Does it always need an object to access these members?**
- **Can a friend function access them directly without an object?**
- **Under what conditions might it fail?**

Justify your answer with supporting C++ code.

1. A friend function **must use an object** to access non-static private and protected members of a class. It cannot access these members directly without an object instance. **Only static members** can be accessed without an object

Friend function access will fail when:

1. Trying to access **non-static members** without an object
2. Attempting to access members of **a different class** without friendship
3. When the object provided is **const** and you try to modify non-mutable members

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

class TestClass {
private:
    int var1;
public:
    TestClass(int num) : var1(num) {}

    friend void testFunction (TestClass box);
};

void testFunction (TestClass box) {
    cout << "The variable is: " << box.var1 << endl; // THIS WORKS BECAUSE OF THE OBJECT "BOX"
    cout << "The variable is: " << var1 << endl;  // DOES NOT WORK BECAUSE THERE IS NO OBJECT
}

int main() {
    TestClass obj1(42);
    testFunction (obj1);
}
```