TASK # 03

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.

Answer:
Yes, a friend function can be used to overload the += operator that modifies the invoking object.

Implementation:

```
class Wallet {
   int money;

public:
   Wallet(int m = 0) : money(m) {}
   friend Wallet& operator+=(Wallet& left, const Wallet& right);

};

Wallet& operator+=(Wallet& left, const Wallet& right) {
   left.money += right.money;  // Can access private 'money'
   return left;
}
```

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.

Answer:
Yes, a friend function can be used to overload operators when one operand is an object and the other is a primitive data type

Implementation:

```
class Money {
int dollars;

public:
friend Money operator+(const Money& m, int amount);
};

Money operator+(const Money& m, int amount) {
return Money(m.dollars + amount);
}
```

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.

Answer:
A friend function must always use an object to access private/protected members. It cannot access these members directly without an object instance, even though it has friend privileges. Friend functions need either an object instance (for non-static members) or class qualification (for static members)

Implementation:

```
class Box {
private:
    int number;

public:
    Box(int num) : number(num) {}

    friend void peekInside(Box box);
};

void peekInside(Box box) {
    cout << box.number << endl; // will work
    cout << number << endl; // gives error
}
```