2.5友元

友元

- 类的私有成员只能通过该类的成员函数访问（封装性、数据隐藏机制）
- **友元**可实现在类外访问私有成员。
  - 友元函数（friend function）
  - 友元类（friendclass）

建议：不用

- 友元的安全逻辑：A说，B是他的友元.....
  - A可以访问B的私有成员？？？
  - B可以访问A的私有成员。

```
friend ostream& operator<<(ostream& os,
                           const CPoint& point) {
        os << "(" << point.x << ", " << point.y << ")";
        return os;
}
```

```
//----------A.hpp----------
class B;
class A {
private:
    double x;
public:
    A(double x=0);

    friend B;
    friend void absA(A& objA);
    friend ostream& operator<<(ostream& os,
                               const A& objA) {
        //申明友元函数并重载，只能这么写。
        os << "A{x = " << objA.x << "}";
        return os;
    }
};
//----------A.cpp----------
A::A(double x) {
    this->x = x;
}
```

```
//----------myFun.hpp----------
void absA(A& objA);
//----------myFun.cpp----------
void absA(A& objA) {
    if (objA.x < 0) objA.x = -objA.x;
}
```

```
//----------B.hpp----------
#include "A.hpp"
class B {
public:
    double getX(A& objA);
    void setX(A& objA, double x = 0);
};
//----------B.cpp----------
double B::getX(A& objA) {
    return objA.x;
}
void B::setX(A& objA, double x) {
    objA.x = x;
}
```

```
void eg3_12() {
    A objA1(-5);
    B objB1;

    cout << objA1 << endl;
    absA(objA1);
    cout << objA1 << endl;
    objB1.setX(objA1, 100);
    cout << "A.x = " << objB1.getX(objA1) << endl;
}
```

```
A{x = -5}
A{x = 5}
A.x = 100
```

友元：交叉引用问题*

```cpp
// A.hpp
#include "B.hpp"

class A {
// some code...
public:
    friend B;
    // some code...
};

// A.cpp
// some code...
```

```cpp
// A.hpp
class B; // extern class B;

class A {
// some code...
public:
    friend B;
    // some code...
};

// A.cpp
// some code...
```

```cpp
// B.hpp
#include "A.hpp"

class B {
public:
    void funB(A &a);
};

// B.cpp
// some code...
```