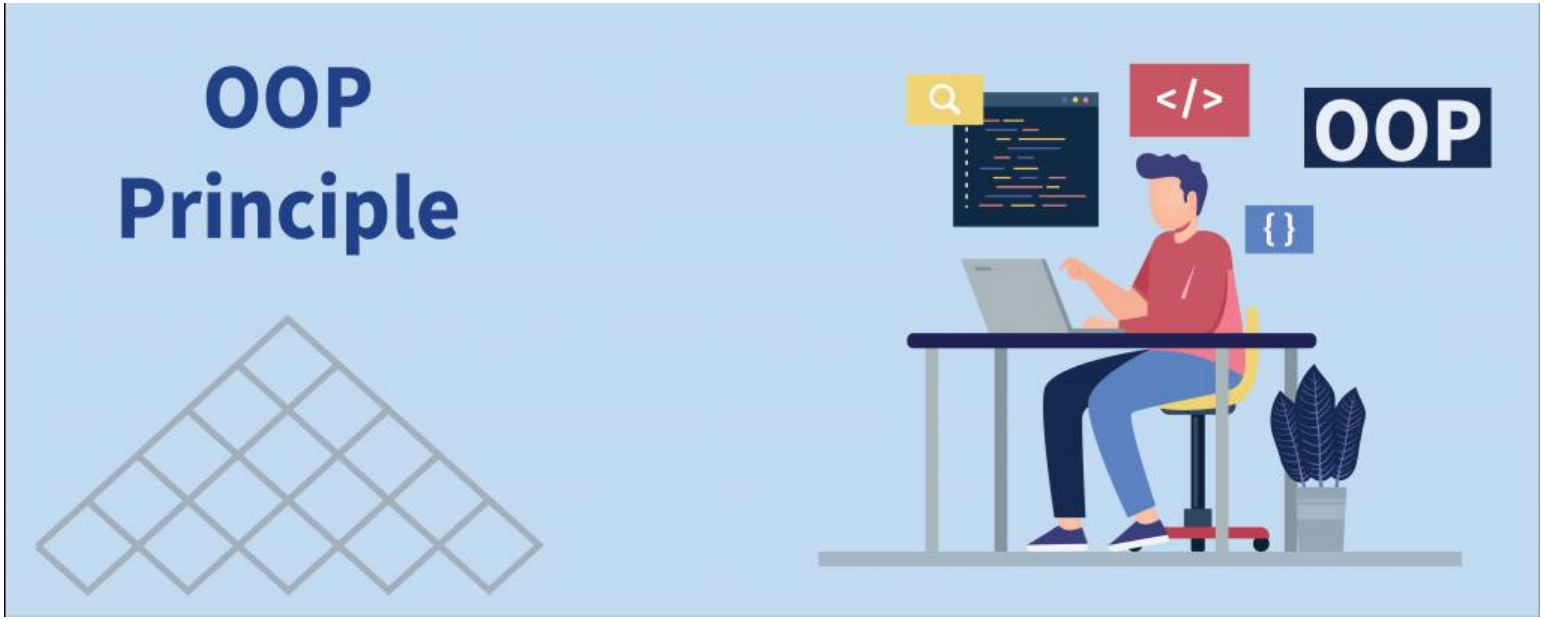


LỚP NÂNG CAO (TIẾP)

Ths. Đinh Thị Thúy – AI LAB/TLU
Thuydt@thanglong.edu.vn – PAI006

NỘI DUNG

- Nạp chồng toán tử nâng cao
- Hàm tạo chuyển kiểu
- Kết tập, hợp thành
- Tách rời giao diện và cài đặt

NẠP CHỒNG TOÁN TỬ (NÂNG CAO)

- Hàm bạn bè và nạp chồng toán tử
- Toán tử nhập, xuất
- Toán tử []
- Toán tử chuyển kiểu
- Hàm tạo chuyển kiểu

VD: HÀM BẠN BÈ

- Khi một lớp A coi một hàm F nằm bên ngoài A là bạn bè, A tin tưởng cho F quyền truy cập mọi thành viên kể cả private của mình
- Khi lớp A coi một lớp B là bạn bè, mọi hàm của B đều là bạn của A
- Tình bạn luôn chỉ được trao, không thể tự nhận. Chỉ A mới được quyền công bố ai là bạn mình.
- Khai báo bạn bè đặt trong khai báo lớp của A
  - friend <nguyên mẫu hàm F>; // hàm F thành bạn của A
  - friend class <tên lớp B>; // lớp B thành bạn của A
- Tình bạn trong C++ không 2 chiều. A là bạn B chưa chắc B là bạn A ☹

VD: HÀM BẠN BÈ LÀ HÀM TOÀN CỤC

- Công bố nguyên mẫu, hoặc công bố và định nghĩa hàm bạn luôn

```
class aClass
{
    int x;
    friend void fSet(aClass &, int);
};
void fSet(aClass & c, int a)
{
    c.x = a;
}
```

```
class aClass
{
    int x;
    friend void fSet(aClass & c, int a)
    {
        c.x = a;
    }
};
```

VD: HÀM BẠN BÈ LÀ HÀM THÀNH VIÊN

- Thứ tự: Khai trước lớp kết bạn, khai nguyên mẫu của hàm bạn, khai báo lớp kết bạn kèm công bố hàm là bạn, định nghĩa cho hàm bạn

```
class aClass;                          // khai báo trước lớp kết bạn
class OtherClass
{
public:
    void fSet(aClass &, int);          // chỉ có thể để dạng nguyên mẫu
};
class aClass                           // khai báo đầy đủ của lớp kết bạn
{
    int x;
    friend void OtherClass::fSet(aClass &, int); // công bố hàm bạn
};

void OtherClass::fSet(aClass & c, int a)     // định nghĩa hàm bạn
{
    c.x = a;
}
```

VD: LỚP BẠN BÈ

- Thứ tự: Khai báo lớp kết bạn, khai báo lớp bạn
- Hạn chế kết lớp bạn bè, chỉ nên kết bạn với hàm thật cần thiết

```
class aClass
{
    int x;
    friend class frClass;
};
class frClass
{
public:
    void fSet(aClass & c,int a){ c.x = a; }
    int fGet(aClass c){ return c.x; }
};
```

HÀM BẠN BÈ VÀ NẠP CHỒNG TOÁN TỬ

- Mọi toán tử đều có thể nạp chồng theo cách là hàm bạn bè của lớp

friend <kiểu trả về> operator <toán tử> (<toán hạng trái>, <toán hạng phải>)

  - Không còn con trỏ this nữa, nên phải nêu tường minh toán hạng trái như tham số đầu tiên trong danh sách
  - Toán tử một ngôi (++, --) có 1 tham số thực duy nhất, dạng hậu tố thêm vào sau đó một tham số giả kiểu int.

- Nạp chồng như hàm bạn bè là lựa chọn duy nhất với các toán tử mà toán hạng trái không phải đối tượng lớp
  - Toán tử nhập xuất: toán hạng trái luôn kiểu ifstream hay ofstream

VD: VIẾT TOÁN TỬ NHƯ HÀM BẠN BÈ

- Viết lại các toán tử từng viết sau nhưng lần này như hàm bạn bè:
- Lớp Cat,
  - Toán tử + 2 conmèo cho ra một con mèo có tên, tuổi trùng con thứ nhất và cân bằng tổng cân 2 mèo
  - Toán tử + một con mèo với số thực trả về một con mèo có tên tuổi trùng con mèo kia và cân bằng số cân mèo đó cộng với số thực.
  - Toán tử so sánh (==) một con mèo với một con mèo khác.

TOÁN TỬ NHẬP >> VÀ XUẤT <<

- Là các toán tử kết hợp được với cin, cout để nhập vào hay in ra một đối tượng
- Toán hạng trái ở đây là cin (đối tượng istream) và cout (đối tượng ostream) nên chỉ có thể nạp chồng các toán tử << và >> này như hàm bạn bè (??)

```
friend ostream & operator << (ostream & out, <toán hạng phải>) ; // toán tử <<
friend istream & operator >> (istream & in, <toán hạng phải>);      // toán tử >>
```

  - Tham số và kiểu trả về đều phải là tham chiếu
  - Coi tham số như thể cout, cin để nhập/xuất các dữ liệu của toán hạng phải
  - Kết hàm nhớ return chính tham số này
  - Với toán tử nhập (>>), toán hạng phải bắt buộc để dạng tham chiếu thường (??)

VD: TOÁN TỬ NHẬP >> VÀ XUẤT <<

- Nạp chồng các toán tử sau cho lớp Cat:
  - >> một con mèo: sẽ vớt từ dòng nhập lần lượt một xâu, một số nguyên, một số thực, rồi lưu vào các biến name, age, weight của con mèo đó
  - << một con mèo: sẽ in ra lời giới thiệu nó tên gì, bao tuổi, bao cân
  - >> một con mèo: sẽ vớt từ dòng nhập một xâu lưu vào biến name, còn lại tự gán mèo 1 tuổi và nặng 2 kg.
  - << một con mèo: sẽ in ra tên kèm tuyên bố mèo không có nghĩa vụ phải nói tuổi và cân của nó ở nơi công cộng.
- Nạp chồng các toán tử sau cho lớp Khoảng cách:
  - >> một khoảng cách: sẽ vớt từ dòng nhập 2 số nguyên để gán cho cm, mm
  - << một khoảng cách: sẽ in ra số cm, mm của khoảng cách đó

TOÁN TỬ CHUYỂN KIỂU

- Dùng khi ép đối tượng sang kiểu khác, được gọi tường minh hoặc tự động bởi trình biên dịch trong lúc tính toán, truyền đối số, trả về giá trị

```
Cat tom;
cout << int (tom) << endl; // gọi tường minh toán tử ép kiểu Cat sang int
int x = tom;               // gọi tự động toán tử ép kiểu Cat sang int
                           // nếu chưa viết hàm, sẽ báo lỗi biên dịch.
```

- Cú pháp: operator <tên kiểu muốn chuyển>(){…}
  - Tên hàm là kiểu cần chuyển, không kiểu trả về, không tham số
  - Chỉ có thể viết ở dạng hàm thành viên (??)

VD: TOÁN TỬ CHUYỂN KIỂU

```
class IntVal
{
    int x;
public:
    IntVal(int a = 0){x = a;}
    operator int(){return x;}
};

int main(){
    IntVal obj(15);
    int i;
    i = obj;
    cout << i;
}
```

15

VD: TOÁN TỬ CHUYỂN KIỂU

- Viết các toán tử chuyển kiểu sau cho lớp Cat:
  - Ép một con mèo thành số nguyên chính là tuổi của nó
  - Ép một con mèo thành xâu chính là tên của nó
- Viết một lớp Dog có thành viên là tên, tuổi, hàm tạo 2 tham số,
  - Rồi viết hàm ép kiểu một con mèo thành một con chó cùng tên, tuổi thêm 1.

HÀM TẠO CHUYỂN KIỂU

- Ngược toán tử chuyển kiểu, giúp ép một kiểu khác thành kiểu của đối tượng. Chính là hàm tạo nhận 1 tham số duy nhất là kiểu đó

```
class CClass    // Converting Class – lớp để test hàm tạo chuyển kiểu
{
    int x;
public:
    CClass()                    // hàm tạo mặc định
    CClass(int, int);           // hàm tạo 2 tham số
    CClass(const CClass &);     // hàm tạo sao chép
    CClass(int a);              // hàm tạo chuyển kiểu int thành CClass
    CClass(string s);           // hàm tạo chuyển kiểu string thành CClass
    void print() const {cout << x << endl; }
};
```

- Lớp **string** có một hàm tạo chuyển từ kiểu C-strings sang string:

  string s = “hello”;

HÀM TẠO CHUYỂN KIỂU

- Được gọi tường minh bởi lập trình viên

```
CClass obj1(24);     // chuyển số int thành đối tượng CClass
CClass obj2("hi");   // chuyển string thành đối tượng CClass
CClass(26).print()   // ép số int thành kiểu CClass nên gọi được print
```

- Hay tự động gọi bởi trình biên dịch trong tính toán,
  - Trong phép gán

```
CCClass obj3 = 25;       // ép số int thành đối tượng CClass
CCClass obj4 = "bye";    // ép string thành đối tượng CClass
```

  - truyền tham số, hay trả về giá trị

```
void myFun(CCClass c);

myFun(34);
```

VD: HÀM TẠO CHUYỂN KIỂU

Lớp Cat: Viết thêm các hàm tạo chuyển kiểu sau:

- Ép kiểu string thành một con mèo tên đó, tuổi 1, cân là 2
- Ép kiểu int thành một con mèo tuổi đó, tên Bibi, cân là 2
- Ép kiểu float thành một con mèo cân đó, tên Bibi, tuổi 1

TỔNG KẾT NẠP CHỒNG TOÁN TỬ

- Phần lớn toán tử đều nạp chồng được và có thể mang ý nghĩa hoàn toàn khác toán tử gốc, nhưng không được thay đổi số lượng toán hạng
- Nếu toán hạng trái là đối tượng thì có 2 cách nạp chồng (hàm thành viên hoặc hàm bạn bè), còn khác thì chỉ nạp chồng như hàm bạn bè
- Toán tử chuyển kiểu dùng để ép từ kiểu đối tượng sang kiểu khác
  - Chỉ nạp chồng được như hàm thành viên
- Hàm tạo chuyển kiểu dùng để ép từ kiểu khác sang kiểu đối tượng
- Không thể nạp chồng các toán tử sau:

  ?: . .* sizeof

SỰ KẾT TẬP

- Liên kết, kết tập, hợp thành
- Cài đặt *use a* vs. *has a* vs. *part of*

LIÊN KẾT

- Là quan hệ cấu trúc: đối tượng lớp A biết thông tin đối tượng lớp B
  - VD: Doctor và Patient, Faculty và Department, Course và Teacher
- A chứa biến (biến thường/con trỏ/tham chiếu) kiểu B
- A gọi là lớp bao (owner class), B là lớp bị bao (owned class)
- Bản số của liên kết: có thể 1-1, 1-n (một-nhiều) hay m-n (nhiều-nhiều)
  - Bác sĩ và Bệnh nhân, Khoa và Giáo viên, Hiệu trưởng và Trường

LIÊN KẾT, KẾT TẬP, HỢP THÀNH

A bao chứa B?

A có cần biết B?

A sở hữu B?

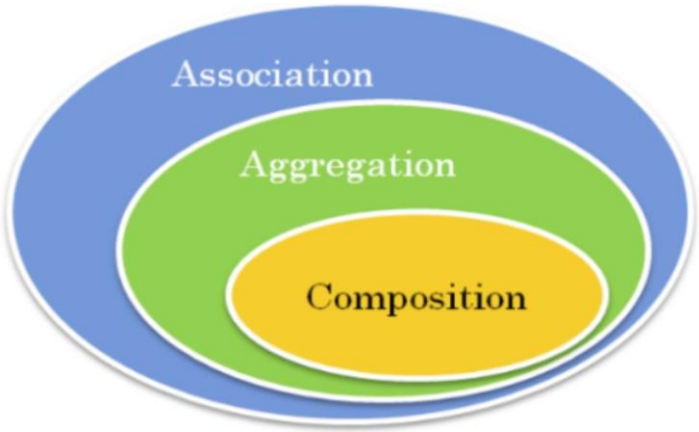
A có cho B liên kết Akhác?

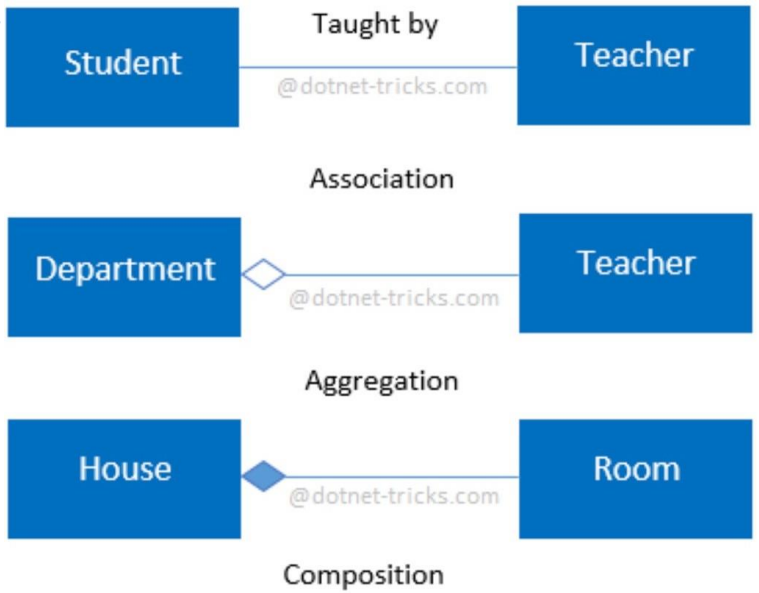
A quản lý tồn tại của B?

B tồn tại trước khi Atạo?
B tồn tại sau khi Achết?

- Thuần liên kết (use-a): A dùng dịch vụ của B, không sở hữu B
  - Môn học chứa biến Giảng viên qua đó có thể biết tên, chức vụ, số đt, email
- Kết tập: A sở hữu B, mỗi A có B khác nhau (dù giá trị có thể bằng nhau)
  - Thuần kết tập (has-a) : B tồn tại độc lập A, chỉ lắp ghép vào A
    - Xe hỏng thì Bánh xe vẫn dùng được, có thể lắp vào xe khác
  - Hợp thành (part of): B tồn tại phụ thuộc A, như máu thịt với A
    - Huỷ Nhà thì Phòng không tồn tại, huỷ Trường thì Khoa không tồn tại

LIÊN KẾT, KẾT TẬP, HỢP THÀNH





- Thuần liên kết (use a) hay được gọi tắt là liên kết. Nói liên kết có lúc là tập to nhất, có lúc chỉ là phần thuần liên kết. Thường hiểu theo nghĩa thuần liên kết.
- Tương tự, thuần kết tập (has a) còn gọi tắt là kết tập.

CÀI ĐẶT LIÊN KẾT, KẾT TẬP, HỢP THÀNH

- Use a: A chứa biến thành viên là tham chiếu/con trỏ kiểu B ***truyền từ ngoài***
- Has a: A chứa biến thành viên là tham chiếu/con trỏ kiểu B ***truyền từ ngoài***
- Part of: A chứa biến kiểu B hay con trỏ kiểu B ***do chính A cấp phát và huỷ***
- Với *use a* và *has a*, tuy con trỏ phức tạp song linh hoạt hơn tham chiếu
  - Được quyền null, được quyền gán lại. VD: Môn học và Giảng viên
  - Tạo biến lớp B, truyền địa chỉ vào hàm tạo/setter của một/nhiều biến lớp A
- Với *part of*, dùng biến thường là đủ tốt, không cần dùng con trỏ
- ***Đi thi cài luôn cài như sự hợp thành trừ phi đề yêu cầu phân biệt rõ***

VD: QUAN HỆ HỢP THÀNH (PART OF)

- Lớp Point mô tả toạ độ trên mặt phẳng

```
class Point{
    int x;
    int y;
public:
    Point (int x = 0, int y = 0){ setAll(x, y); }
    void setAll(int x, int y){
        this->x = x;
        this->y = y;
    }
    friend ostream & operator << (ostream & out, const Point & r) {
        out << r.x << " " << r.y;
        return out;
    }
    ~Point() { cout << "Point " << *this << " bi huy\n"; }
};
```

- Lớp Creature mô tả một sinh vật có tên và toạ độ

```
class Creature{
    string name;
    Point location;
public:
    // nhập tham số Point để khởi tạo vị trí kiểu Point
    Creature(string name, const Point & p)
    : name(name), location(p){}

    // nhập 2 toạ độ để gọi hàm tạo 2 tham số khởi tạo vị trí kiểu Point
    Creature (string name, int x, int y)
    : name(name), location(x,y){}
    void setAll(string s, const Point & p){
        name = s;
        location = p;
    }

    // in ra location dùng đến toán tử nạp chồng <<
    void print() const{
        cout << "Creature " << name << " is at " << location << endl;
    }
    ~Creature(){ cout << "Creature " << name << " bi huy\n"; }
};
```

- Creature không chỉ chịu trách nhiệm việc tạo, mà cả việc huỷ location

```
int main(){
    int n = 3;
    Point p(3,7);
    Creature a[] = {Creature ("Phoenix", 0, 5), // truyền x, y để tạo Point
                    Creature ("Dragon", 5, 7),  // truyền x, y để tạo Point
                    Creature ("Unicorn", p)};   // truyền Point để tạo Point
    for (int i = 0; i < n; i++)
        a[i].print();
    return 0;
}
```

```
Creature Phoenix is at 0 5
Creature Dragon is at 5 7
Creature Unicorn is at 3 7
Creature Unicorn bi huy
Point 3 7 bi huy
Creature Dragon bi huy
Point 5 7 bi huy
Creature Phoenix bi huy
Point 0 5 bi huy
Point 3 7 bi huy
```

VD: QUAN HỆ KẾT TẬP (HAS A)

```
class Person{
    string name;
    int age;
public:
    Person(string name, int age)
    : name(name), age(age){}
    friend istream & operator >> (istream & in, Person & r){
        getline(in, r.name);
        in >> r.age;
        return in;
    }
    friend ostream & operator << (ostream & out, const Person & r){
        out << r.name << " " << r.age;
        return out;
    }
    ~Person(){cout << "Nguoi " << * this << " huy\n"; }
};
```

VD: QUAN HỆ KẾT TẬP (HAS A)

- Cài kết tập bằng tham chiếu

```
class Faculty{
    string name;
    Person & dean, & vice;
public:
    Faculty (string name, Person & dean, Person & vice)
    : name(name), dean(dean), vice(vice){}
    void print() const{
        cout << "Khoa        : " << name << endl
             << "Truong khoa: " << dean << endl
             << "Pho khoa   : " << vice << endl;
    }
    ~Faculty(){cout << "Khoa " << name << " huy\n"; }
};
```

VD: QUAN HỆ KẾT TẬP (HAS A)

- Cài kết tập bằng con trỏ

```
class Faculty{
    string name;
    Person * dean, * vice;
public:
    Faculty (string name, Person * dean, Person * vice)
    : name(name), dean(dean), vice(vice){}
    void print() const{
        cout << "Khoa        : " << name << endl
             << "Truong khoa: " << *dean << endl
             << "Pho khoa   : " << *vice << endl;
    }
    ~Faculty(){cout << "Khoa " << name << " huy\n"; }
};
```

VD: QUAN HỆ KẾT TẬP (HAS A)

- Kết tập tham chiếu:

```
int main(){
    Person p1("Prof. John", 60), p2 ("Prof. Klaus", 55);
    Faculty f("Philosophy", p1, p2);
    f.print();
    return 0;
}
```

```
Khoa        : Philosophy
Truong khoa: Prof. John 60
Pho khoa    : Prof. Klaus 55
Khoa Philosophy huy
Nguoi Prof. Klaus 55 huy
Nguoi Prof. John 60 huy
```

- Kết tập con trỏ:

```
int main (){
    Person p1("Prof. John", 60), p2 ("Prof. Klaus", 55);
    Faculty f("Philosophy", &p1, &p2);
    f.print();
    return 0;
}
```

VD: QUAN HỆ KẾT TẬP (HAS A)

- Đối tượng bao không chịu trách nhiệm huỷ đối tượng bị bao

```
int main (){
    Person  * p1 = new Person("Prof. John", 60),
            * p2 = new Person("Prof. Klaus", 55);
    if (true){
        Faculty f("Philosophy", p1, p2);
        f.print();
    }
    cout << "Truoc khi xoa con tro\n";
    delete p1; p1 = 0;
    delete p2; p2 = 0;
    cout << "Sau khi xoa con tro\n";
    return 0;
}
```

```
Khoa        : Philosophy
Truong khoa: Prof. John 60
Pho khoa    : Prof. Klaus 55
Khoa Philosophy huy
Truoc khi xoa con tro
Nguoi Prof. John 60 huy
Nguoi Prof. Klaus 55 huy
Sau khi xoa con tro
```

VD: QUAN HỆ LIÊN KẾT (USE A)

- Quan hệ giữa lớp với giáo trình và giảng viên.

```
class TextBook          // lớp giáo trình
{
private:
    string title ;      // tiêu đề
    string author;      // tác giả
    string publisher;   // nhà xuất bản
public:
    TextBook() { set("", "", ""); }
    TextBook(string textTitle, string auth, string pub)
    { set(textTitle, auth, pub); }
    void set(string textTitle, string auth, string pub)
    { title = textTitle; author = auth; publisher = pub; }
    friend ostream & operator << (ostream & out, const TextBook & r) const
    {
        out << "Tieu de        : " << r.title << endl
            << "Tac gia        : " << r.author << endl
            << "Nha xuat ban   : " << r.publisher << endl;
    }
};
```

```
class Instructor                // lớp giảng viên
{
private:
    string lastName;            // họ
    string firstName;           // tên
    string officeNumber;        // mã giảng viên
public:
    Instructor() { set("", "", ""); }
    Instructor(string lname, string fname, string office)
    { set(lname, fname, office); }
    void set(string lname, string fname, string office)
    { lastName = lname; firstName = fname; officeNumber = office; }
    void print() const
    {
        cout << "Ho              : " << lastName << endl
             << "Ten             : " << firstName << endl
             << "Ma giang vien : " << officeNumber << endl;
    }
    ~Instructor(){ cout << "Giang vien " << officeNumber << " huy\n"; }
};
```

```
class Course
{
private:
    string courseName;          // tên lớp
    TextBook & textbook;        // giáo trình
    Instructor & instructor;    // giảng viên
public:
    // hàm tạo nhận vào đủ thông tin để tạo được lớp, giảng viên, giáo trình
    Course(string course, TextBook & text, Instructor & instr)
    : courseName(course), textbook(text), instructor(instr){}
    void print() const
    {
        cout << "Ten lop: " << courseName << endl << endl;
        cout << "\nThong tin giao trinh:\n"; textbook.print();
        cout << "Thong tin giang vien:\n"; instructor.print();
        cout << endl;
    }
    ~Course(){ cout << "Lop " << courseName << " huy\n"; }
};
```

- Cài đặt *use a* giống như *has a.* Nhưng khi sử dụng, do không có tính sở hữu nên lớp bị bao được thuộc nhiều lớp bao

```
int main()
{
    TextBook book1 ("Starting Out with C++", "Coronel", "Cengage");
    TextBook book2 ("Database Systems", "Gaddis", "Addison-Wesley");
    Instructor prof ("Kramer", "Shawn", "RH3010");
    Course course1("Intro to Computer Science", book1, prof);
    Course course2("Intro to Database Systems", book2, prof);

    course1.print();
    course2.print();
    return 0;
}
```

```
Thong tin giao trinh:
Tieu de      : Database Systems
Tac gia      : Gaddis
Nha xuat ban : Addison-Wesley
Thong tin giang vien:
Ho           : Kramer
Ten          : Shawn
Ma giang vien : RH3010

Lop Intro to Database Systems huy
Lop Intro to Computer Science huy
Giang vien RH3010 huy
Giao trinh Database Systems huy
Giao trinh Starting Out with C++ huy
```

VD: LIÊN KẾT, KẾT TẬP, HỢP THÀNH

- Lớp Room mô tả một phòng có biến tên và diện tích, hàm tạo 2 tham số mặc định tên là phòng khách, diện tích 10m vuông, một hàm trả về diện tích
- Lớp House mô tả một nhà gồm 4 phòng, hàm tạo có tham số và hàm trả về diện tích cả nhà.
  - Xác định quan hệ giữa House và Room là kiểu gì?
  - Cài đặt House, Room, rồi demo bằng tạo ra một ngôi nhà, sau đó in ra diện tích cả nhà

VD: QUAN HỆ HỢP THÀNH (PART OF)

- Lớp Room mô tả một phòng trong nhà

```
class Room{
    string name;
    float area;
public:
    Room(string name = "Phong khach", float area = 10)
    { setInfor(name, area); }
    void setInfor (string n, float a){
        name = n;
        area = a;
    }
    string getName()const{return name;}
    float getArea() const {return area;}
};
#endif
```

```
const int n = 4;
class House{
    Room a[n];
public:
    House (string name[], float area[]){
        for (int i = 0; i < n; i++)
            a[i].setInfor(name[i], area[i]);
    }
    float getArea()const{
        float area = 0;
        for (int i = 0; i<n; i++)
            area +=a[i].getArea();
        return area;
    }
};
int main(){
    string name[] = {"Phong khach", "Phong tam", "Phong ngu", "Phong an"};
    float area[] = {16, 6, 12, 12 };
    House o (name, area);
    cout << "Dien tich nha: " << o.getArea() << endl;
    return 0;
}
```

```
Dien tich nha: 46
```

TÁCH RỜI GIAO DIỆN VÀ CÀI ĐẶT

- Bao gói, tách rời giao diện và cài đặt
- Phân phối mã bằng nhiều file
- Bộ bảo vệ header

BAO GÓI, TÁCH RỜI GIAO DIỆN VÀ CÀI ĐẶT

- Một nguyên lý quan trọng của OOP là ***tính bao gói***, nghĩa gom nhiều thứ với nhau để che giấu thông tin nghĩa
- Cơ chế private, public khiến dữ liệu phải gắn chặt với các hàm truy cập, dữ liệu nhờ thế được che giấu và lớp thể hiện được tính bao gói
- Bao gói sẽ càng hiệu quả khi có thể tách rời giao diện và cài đặt lớp.
- Cần tách mã xây dựng lớp thành nhiều file, một file chứa giao diện, một file chứa cài đặt, và cần có cách để không phân phối file cài đặt mà người dùng vẫn chạy được dịch vụ của lớp.

PHÂN PHỐI MÃ BẰNG NHIỀU FILE

- Đặt mọi khai báo của lớp trong một tệp, gọi là tệp đặc tả. Đặt tên tệp là **ClassName.h,** ví dụ **Rectangle.h**
- Đặt cài đặt của các hàm thành viên trong tệp **ClassName.cpp**, ví dụ **Rectangle.cpp**. Trong tệp cài đặt này phải **#include** tệp đặc tả
- Mọi chương trình muốn sử dụng lớp phải **#include** tệp đặc tả, rồi dịch và liên kết với bản dịch của tệp cài đặt.
- Phân phối tệp đặc tả kèm bản biên dịch của tệp cài đặt (ko đọc được) thì người khác có thể dùng dịch vụ lớp dù không thấy cài đặt

VD: PHÂN PHỐI MÃ BẰNG NHIỀU FILE

- Đặt khai báo lớp trong tệp đặc tả **Rectangle.h**

```
class Rectangle
{
    double width;
    double length;
public:
    Rectangle();
    Rectangle(double, double);
    bool setWidth(double);
    bool setLength(double);
    double getLength() const;
    double getWidth() const;
    double getArea() const;
};
```

VD: PHÂN PHỐI MÃ BẰNG NHIỀU FILE

- Đặt cài đặt hàm thành viên trong tệp **Rectangle.cpp**.

```
#include "Rectangle.h" // cần có tệp đặc tả

Rectangle::Rectangle()
: width (0), length(0){}

Rectangle::Rectangle(double w, double len){
    if (w >= 0 && len >= 0){
        width = w;
        length = len;
    }
}

bool Rectangle::setLength(double len)
{
    if (len < 0)
        return false;
    length = len;
    return true;
}
```

```
bool Rectangle::setWidth(double w)
{
    if (w < 0)
        return false;
    width = w;
    return true;
}

double Rectangle::getLength() const
{ return length; }

double Rectangle::getWidth() const
{ return width; }

double Rectangle::getArea() const
{ return length * width; }
```

VD: PHÂN PHỐI MÃ BẰNG NHIỀU FILE

- Chương trình dùng lớp **#include** tệp đặc tả, rồi dịch, liên kết với bản dịch của tệp cài đặt.

```
class Rectangle
{
    double width;
    double length;
public:
    Rectangle();
    Rectangle(double, double);
    bool setWidth(double);
    bool setLength(double);
    double getLength() const;
    double getWidth() const;
    double getArea() const;
};
```

```
#include <iostream>
#include "Rectangle.h"
using namespace std;

int main()
{
    double width, length;
    cin  >> width >> length;
    Rectangle box(width, length);
    cout << box.getLength() << " "
        << box.getWidth() << " "
        << box.getArea() << endl;
    return 0;
}
```

BỘ BẢO VỆ HEADER

- Việc include thực chất là copy phần khai báo lớp
- Nếu tệp C include đặc tả lớp A và B, mà tệp B từng include A?? Error
- Giải pháp: Dùng bộ bảo vệ header

```
#ifndef <nhãn lớp>     // thường toàn viết hoa, vd RECTANGLE_H
#define <nhãn lớp>
// đặt mã tệp đặc tả ở đây
#endif
```

- C++ chỉ đọc phần nằm giữa #ifndef và #endif nếu chưa từng định nghĩa nhãn lớp, do đó có include 2 lần mã đặc tả thì lần sau cũng bị bỏ qua.

VD: BỘ BẢO VỆ HEADER

```
// Minh hoạ bộ bảo vệ header
// nếu nhãn RECTANGLE_H chưa định nghĩa mới đọc mã từ #ifndef đến #endif
#ifndef RECTANGLE_H
#define RECTANGLE_H    //định nghĩa nhãn RECTANGLE_H

class Rectangle
{
    double width;
    double length;
public:
    Rectangle();
    Rectangle(double, double);
    bool setWidth(double);
    bool setLength(double);
    double getLength() const;
    double getWidth() const;
    double getArea() const;
};

#endif
```

VD: PHÂN PHỐI MÃ BẰNG NHIỀU FILE

- Lấy lại lớp Cat đã viết đến giờ, tách rời giao diện và cài đặt bằng cách phân phối Cat như nhiều file và thử biên dịch chúng

TỔNG KẾT

- Hàm bạn bè, lớp bạn bè
- Nạp chồng toán tử: nhập xuất, truy xuất mảng, ép kiểu, hàm tạo chuyển kiểu
- Liên kết, kết tập, hợp thành
- Tách rời giao diện và cài đặt: phân phối mã như nhiều file

Keep calm and carry on