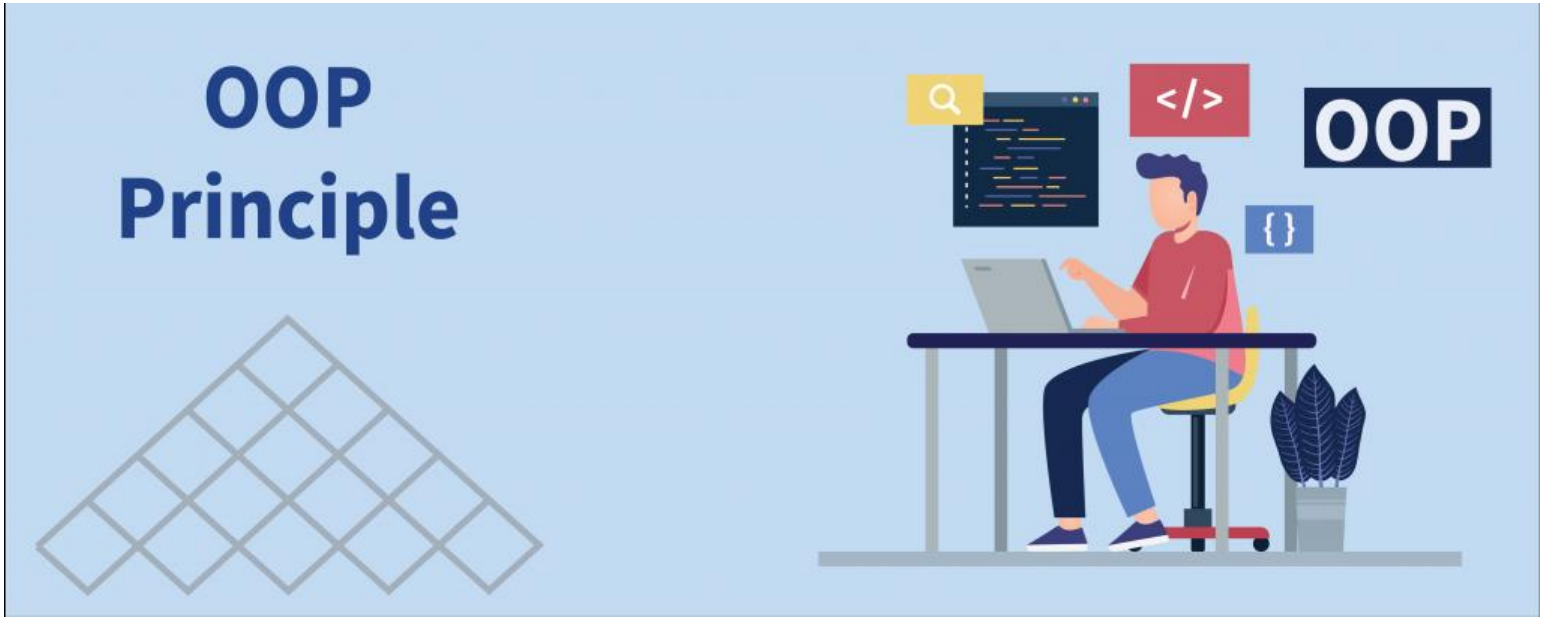


LỚP NÂNG CAO

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

NỘI DUNG

- Con trỏ đối tượng
- Con trỏ this và biến thực thể
- Static và biến lớp
- Hàm tạo sao chép
- Nạp chồng toán tử
- Hàm tạo chuyển kiểu
- Kết tập, hợp thành

CON TRỎ ĐỐI TƯỢNG

- Con trỏ đối tượng
- Truy cập thành viên qua con trỏ
- Cấp phát động và giải phóng đối tượng

KHAI BÁO CON TRỎ, CẤP PHÁT ĐỐI TƯỢNG

- Có thể khai báo con trỏ trỏ đến đối tượng:

```
Rectangle *rPtr;
```

- Khi con trỏ cấp phát động với toán tử new, đối tượng được tạo, muốn bằng hàm tạo nào thì truyền đối số tương ứng:

```
Rectangle *r2 = new Rectangle;
Rectangle *r3 = new Rectangle(10);
Rectangle *r1 = new Rectangle(10, 20);
```

- Khi bộ nhớ giải phóng, đối tượng bị huỷ, hàm hủy được gọi:

```
delete r;
```

TRUY CẬP THÀNH VIÊN QUA CON TRỎ

- Để truy cập thành viên qua con trỏ, dùng toán tử (->)
  - ptr-> là viết tắt của (*ptr).
  - Cũng vẫn chỉ truy cập được thành viên là public

```
rPtr = &otherRectangle;
rPtr->setLength(12.5); // (*rPtr).setLength()
cout << rPtr->getLenght() << endl;
rPtr->length; // error
```

VD: CẤP PHÁT ĐỘNG VÀ HÀM TẠO

Lớp Cat đang có 4 hàm tạo (0, 1, 2,3 tham số)

- Thêm vào mỗi hàm tạo này một lệnh in ra thông báo đang ở hàm tạo nào
- Main sinh ra 4 biến con trỏ Cat, cấp phát động bằng 4 loại hàm tạo, sau đó thông qua con trỏ lấy thông tin của từng con mèo in ra, sau cùng cập nhật lại thông tin của 1 con, rồi in ra thông tin mới này.

VD: CẤP PHÁT ĐỘNG VÀ HÀM TẠO

```
class Cat{
    string name;
    int age;
    float weight;
    bool chase() const { return rand() % 2; }
public:
    Cat(){setInfor("Bibi", 1, 2); cout << "Ham tao mac dinh\n"; }
    Cat
    Cat(
    Cat(

    voic

    }
    stri
    int
    floa
    voic
    int
};
int Cat:
    int
    for
        count += chase();
    return count;
}
```

```
int main(){
    Cat *c1 = new Cat,
        *c2 = new Cat("Tom"),
        *c3 = new Cat("Mimi", 2),
        *c4 = new Cat ("Toto", 3, 4);
    cout << c1->getName() << " " << c1->getAge() << " " << c1->getWeight() << endl;
    cout << c2->getName() << " " << c2->getAge() << " " << c2->getWeight() << endl;
    cout << c3->getName() << " " << c3->getAge() << " " << c3->getWeight() << endl;
    cout << c4->getName() << " " << c4->getAge() << " " << c4->getWeight() << endl;
    c1->setInfor("Karl", 4, 5);
    cout << c1->getName() << " " << c1->getAge() << " " << c1->getWeight() << endl;
    return 0;
}
```

VD: GIẢI PHÓNG VÀ HÀM HUỶ

- Thêm vào hàm huỷ lớp Cat một lệnh in ra thông báo đang ở hàm huỷ
- Viết hàm inputCat nhận vào 1 biến Cat, không trả về và không làm gì
- Viết hàm main sinh ra 1 biến Cat để truyền vào gọi hàm inputCat, sau đó sinh ra 1 con trỏ Cat, cấp phát động bằng hàm tạo nào đó, sau đó delete con trỏ này

BIẾN THỰC THỂ VÀ BIẾN TĨNH

- Con trỏ this và biến thực thể
- Từ khoá static và biến tĩnh
- Hàm tĩnh

CON TRỎ THIS VÀ HÀM THÀNH VIÊN HẰNG

- Con trỏ this
  - Là tham số được truyền ngầm vào hàm thành viên
  - Trỏ đến đối tượng đang gọi hàm đó.

```
… func (Cat * , ….) ;              // tham số ngầm
```

- Hàm thành viên const không thay đổi biến thành viên
  - Bản chất là con trỏ this được truyền vào kiểu const

```
… func (const Cat * , ….);      // con trỏ kiểu const
```

CON TRỎ THIS VÀ BIẾN THỰC THỂ

- Biến thực thể:
  - Là biến thành viên mà thuộc về riêng từng đối tượng
  - Gắn với con trỏ this:

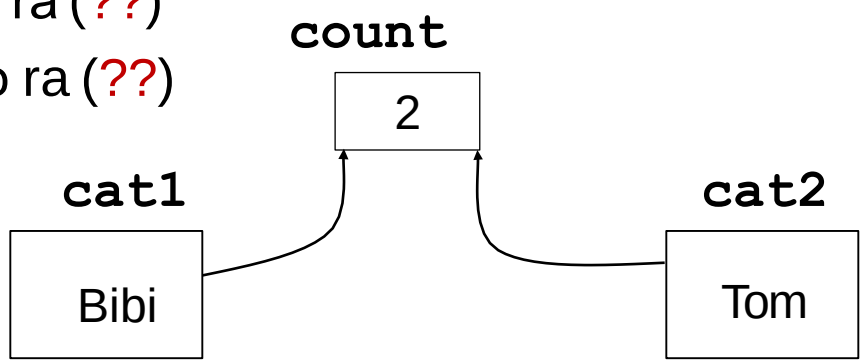
    width = w; // thực chất là this->width = w;
- Sử dụng this:
  - Để truy cập thành viên bị che bởi tham số cùng tên

```
class SomeClass
{
    int num;
public:
    void setNum(int num){ this->num = num; }
};
```

  - *this chính là đối tượng, có thể trả về, truyền hàm, tính toán

TỪ KHOÁ STATIC VÀ BIẾN TĨNH

- Biến tĩnh (biến lớp) lưu thông tin chung cho cả lớp, nên là biến được chia sẻ bởi mọi đối tượng trong lớp
  - VD: count đếm tổng số con mèo từng sinh ra
- Cú pháp: static <kiểu> <tên biến>;
  - Nếu không khởi tạo thì biến tĩnh luôn có giá trị mặc định null
  - Nếu khởi tạo, cần để ngoài khai báo lớp: Cat::catCount = 0;
- Cần được cập nhật để phản ánh sự thay đổi của thông tin chung này
  - Tăng count khi một con mèo sinh ra (??)
- Tồn tại trước cả khi có đối tượng tạo ra (??)
  - Nên truy cập bởi hàm tĩnh (~sau)



VD: BIẾN TĨNH

- Thêm vào lớp Cat một biến count để đếm số mèo đã sinh ra
  - Khai báo count là biến tĩnh
  - Khởi tạo count bằng 0 ngoài khai báo lớp
  - Cập nhật count tại MỌI hàm tạo

HÀM THÀNH VIÊN TĨNH

- Cú pháp: static <khai báo hàm>

  static int getCount() { return count; }
- Giống biến lớp, cũng là hàm chung cho cả lớp thay vì đối tượng,
  - Có thể gọi như hàm thành viên bình thường Cat c;

    cout << c.getCount() << endl;
  - Cũng có thể gọi thông qua lớp, không cần có đối tượng sinh ra cout << Cat::getCount() << endl;
- Không có tham số là con trỏ this
  - không truy cập được biến thực thể, chỉ truy cập biến lớp (biến tĩnh)
  - không bao giờ cần từ khoá const
  - chủ yếu dùng xử lý biến tĩnh

VD: HÀM TĨNH

- Viết hàm getCount để trả về số mèo đã sinh ra của lớp Cat?
  - Tại sao nó nên là hàm tĩnh?

HÀM TẠO SAO CHÉP

- Phép gán theo thành viên
- Hàm tạo sao chép

PHÉP GÁN THEO THÀNH VIÊN (COPY NÔNG)

- Trong OOP, có thể dùng phép bằng với cả các đối tượng:
  - Để khởi gán: tạo 1 đối tượng từ dữ liệu đối tượng khác
  - Để gán: biến 1 đối tượng thành giống đối tượng khác
- Phép bằng sẽ copy giá trị từng biến thành viên của toán hạng phải sang biến thành viên tương ứng ở toán hạng trái

```
Cat c1("Tom", 2, 3), c2 = c1; // khởi gán
Cat c3;
c3 = c2;                      // phép gán
// cả 3 con mèo đều tên Tom, 2 tuổi, 3 kg
```

- Có lúc phép gán theo thành viên là nguy hiểm
  - Khi có biến thành viên là con trỏ (??)

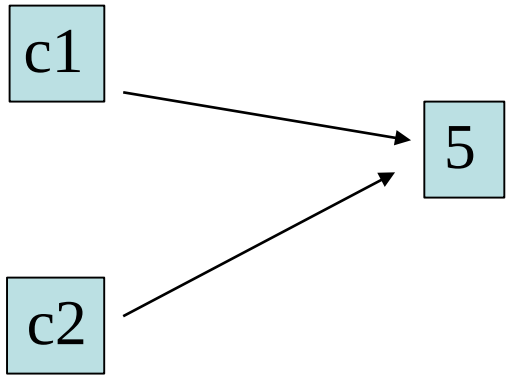
HÀM TẠO SAO CHÉP

- Là hàm tạo gọi khi tạo đối tượng sao chép đối tượng khác
- Nếu không tự viết, C++ sẽ cho một hàm tạo sao chép mặc định, hàm này sẽ sao chép kiểu phép gán theo thành viên
- Hàm sao chép mặc định này đủ tốt cho phần lớn trường hợp.
- Vấn đề chỉ nảy sinh khi đối tượng có con trỏ cấp phát động

```
class CpClass  // Copying pointer Classs – lớp test việc copy con trỏ cấp phát động
{
    int *p;
public:
    CpClass(int v = 0): p(new int) { *p = v; }
    ~CpClass(){ delete p; }
};
```

Hàm tạo sao chép mặc định gây ra sự chia sẻ bộ nhớ

```
CpClass c1(5);
if (true)
{
    CpClass c2=c1;
}
// c1 bị phá hỏng khi c2 hết thời gian
// tồn tại kéo theo hàm huỷ của nó thực thi
```



- Hàm huỷ của một đối tượng xoá mất vùng nhớ của đối tượng khác
- Một đối tượng cập nhật vùng nhớ kéo theo đối tượng khác bị cập nhật.

TỰ VIẾT HÀM TẠO SAO CHÉP

- Nếu lớp có biến cấp phát động, ta CẦN viết hàm tạo sao chép
- Cú pháp: là hàm tạo có tham số là tham chiếu tới đối tượng lớp đó

  Rectangle (const Rectangle & r);

  Cat (const Cat & r);

  - Tham số phải kiểu tham chiếu (??) và const (??)
- Tránh vấn đề chia sẻ vùng nhớ động bằng cách:
  - Con trỏ của đối tượng mới được cấp phát động vùng nhớ riêng
  - Vùng này sao chép dữ liệu từ vùng tương ứng của đối tượng gốc
  - => Dữ liệu giống nhau nhưng nằm ở 2 vùng nhớ khác nhau

VD: TỰ VIẾT HÀM TẠO SAO CHÉP

```
#include <iostream>
using namespace std;
class CpClass  // lớp để test việc copy con trỏ cấp phát động
{
    int *p;
public:
    CpClass(int v = 0): p(new int) { *p = v; }
    ~CpClass(){ delete p; }
    CpClass(const CpClass & r){ // cần tự viết hàm tạo sao chép
        p = new int;            // cấp phát vùng nhớ mới cho con trỏ
        *p = *r.p;              // sao chép dữ liệu thay vì sao chép con trỏ
    }
    void set(int v) { *p = v; }
    int get() const { return *p; }
};
```

```
int main(){
    CpClass c1(5);
    if (true){
        CpClass c2(c1);      // gọi hàm tạo sao chép kiểu truyền đối
        CpClass c3 = c2;     // gọi hàm tạo sao chép kiểu khởi gán

        cout << c1.get() << endl;
        cout << c2.get() << endl;
        cout << c3.get() << endl;

        c1.set(7);
        cout << c1.get() << endl;
        cout << c2.get() << endl;
        cout << c3.get() << endl;
    }
    cout << c1.get() << endl;
    return 0;
}
```

```
5   5   5
```

```
7   5   5
```

```
7
```

VD: TỰ VIẾT HÀM TẠO SAO CHÉP

Lớp Cat toàn biến tĩnh nên hàm sao chép mặc định C++ là đủ tốt, nhưng nếu muốn, ta có thể thử viết hàm tạo sao chép cho lớp Cat

- Sau đó cập nhật lại chương trình đếm số mèo sinh ra, đảm bảo count đếm được kể cả những con mèo sinh ra kiểu sao chép
- Test thử hàm sao chép bằng cách tạo một hàm inputCat nhập vào một con mèo và trả về chính con mèo đó
- Tạo hàm main sinh 1 đối tượng mèo, 1 con mèo cấp phát động, truyền con mèo đầu vào hàm inputCat
- In ra số mèo sinh ra sau từng bước trên

VD: LỚP CÓ CON TRỎ CẤP PHÁT ĐỘNG

Nhớ lại, lớp Number Array có biến con trỏ mảng được cấp phát động.

- Hãy viết hàm setter, getter để thiết lập và trả về giá trị 1 phần tử chỉ số cho trước trong mảng
- Viết hàm độc lập inputArray nhập vào một đối tượng NumberArray và không làm gì cả
- Viết hàm main tạo đối tượng NumberArray với cỡ mảng là 5, thiết lập phần tử chỉ số 2 bằng 7, thử in ra kiểm tra giá trị phần tử đó
- Nhập đối tượng này để gọi hàm inputArray, rồi in lại phần tử chỉ số 2

```
class NumberArray{
    int size;
    double * a;
    int check(int n) const{
        if (n<0) throw "so am\n";
        return n;
    }
public:
    NumberArray(int s = 10): size(check(s)), a(new double [s]){}
    ~NumberArray(){delete [] a;}
    void set(int i, double v) { a[check(i)] = v; }
    double get(int i) const { return a[check(i)]; }
};
void inputArray(NumberArray a){}
int main(){
    NumberArray a(5);
    a.set(2, 7);
    cout << a.get(2) << endl;
    inputArray(a);
    cout << a.get(2) << endl;
    return 0;
}
```

```
Abort trap: 6
```

VD: TỰ VIẾT HÀM TẠO SAO CHÉP

- Chương trình trước minh hoạ hậu quả nếu không viết hàm tạo sao chép khi lớp có biến cấp phát động.
- Để sửa lỗi, hãy viết hàm tạo sao chép cho lớp NumberArray, để việc sao chép biến con trỏ cấp phát động diễn ra hợp lý.
- Lại làm giống hàm main chương trình trước, so sánh kết quả

```
class NumberArray{
    int size;
    double * a;
    int check(int n) const{
        if (n<0) throw "so am\n";
        return n;
    }
public:
    NumberArray(int s = 10): size(check(s)), a(new double [s]){}
    ~NumberArray(){delete [] a;}
    NumberArray(const NumberArray & r){
        delete [] a;
        size = r.size;
        a = new double [size];
        for (int i = 0; i<size; i++) a[i] = r.a[i];
    }
    void set(int i, double v) { a[check(i)] = v; }
    double get(int i) const { return a[check(i)]; }
};
```

```
void inputArray(NumberArray a){}
int main(){
    NumberArray a(5);
    a.set(2, 7);
    cout << a.get(2) << endl;
    inputArray(a);
    cout << a.get(2) << endl;
    return 0;
}
```

```
7
7
```

Tổng kết: Hàm tạo, hàm tạo sao chép, hàm huỷ

- Hàm tạo được gọi khi cần tạo đối tượng lớp đó
  - Khai báo biến đối tượng (??), cấp phát động đối tượng (??),
  - Hàm tạo sao chép:
    - Khi khởi gán
    - Khi truyền đối số đối tượng by value (??) hay trả về đối tượng by value (??)
- Hàm huỷ được gọi khi đối tượng bị phá hủy
  - Hết phạm vi chứa biến (khối lệnh, hàm, chương trình), giải phóng bộ nhớ động chứa biến
  - Kết thúc truyền đối kiểu tham trị, kết thúc trả về kiểu tham trị

NẠP CHỒNG TOÁN TỬ

- Toán tử
- Nạp chồng toán tử
  - Toán tử +
  - Toán tử =
  - Toán tử +=
  - Toán tử ==
  - Toán tử tăng, giảm
  - Toán tử nhập, xuất
  - Toán tử []

HÀM CHỒNG (ÔN LẠI)

- Các hàm trùng tên song khác danh sách tham số
- Nạp chồng hàm thường lẫn hàm thành viên (và cả hàm tạo)

  void setCost(double);

  void setCost(char *);
- Toán tử thực chất cũng là một loại hàm với ký hiệu toán tử là tên hàm, còn các toán hạng là các đối số hàm
- => Liệu có thể nạp chồng toán tử?

NẠP CHỒNG TOÁN TỬ

- Là hàm có dạng

  <kiểu trả về> operator<ký hiệu toán tử> (<danh sách tham số>);

  Cat operator+ (Cat, Cat); // nạp chồng để cộng 2 con mèo
- Dù nạp chồng, số lượng toán hạng vẫn giữ như phiên bản gốc.
  - vd cộng 2 con mèo; cộng 1 con mèo và 1 số nguyên, ..
- Có thể viết như hàm thành viên, hoặc hàm bạn bè (~học sau).
- Nếu là hàm thành viên, đối tượng gọi hàm luôn là toán hạng trái ngầm
  - *this chính là toán hạng trái ngầm
  - Toán tử 2 ngôi do đó chỉ cần truyền vào toán hạng phải, chính là tham số duy nhất

    Cat operator+(Cat); // cộng 2 mèo và trả về 1 mèo mới

HÀM THÀNH VIÊN TOÁN TỬ

<kiểu trả về> operator<toán tử> (<kiểu của toán hạng phải>);

1. Trả về cái gì: một con mèo? một số? void? …
2. Đâu là toán hạng phải: một con mèo, một số, không có?
3. Làm sao từ toán hạng trái và phải tạo ra thứ để trả về?

```
Cat operator+(Cat);          // 2 con mèo cộng nhau, ra một con mèo
Cat operator+(int);          // con mèo cộng số nguyên, ra một con mèo
int operator+(Cat);          // 2 con mèo cộng nhau, ra một số nguyên
Sphinx operator+(Woman);     // con mèo cộng phụ nữ, ra một nhân sư
```

GỌI TOÁN TỬ NẠP CHỒNG

- Có thể gọi như một hàm thành viên lẫn theo cách phổ thông :

```
OpClass a, b, s;
s = a.operator+(b);   // gọi như hàm thành viên
OpClass a, b, s;
s = a + b;            // theo cách phổ thông
```

VD: NẠP CHỒNG PHÉP CỘNG

- Viết toán tử cộng 2 con mèo, trả về một con mèo có tên và tuổi giống con mèo toán hạng trái (con mèo đầu tiên) còn cân nặng bằng tổng cân của 2 con mèo
- Viết hàm main thử sinh 2 con mèo, cộng chúng rồi in ra tên, tuổi, cân nặng của con mèo kết quả

VD: NẠP CHỒNG PHÉP CỘNG

- Viết toán tử cộng một con mèo và một số nguyên trả về con mèo cùng tên, cùng tuổi, có số cân tăng lên đúng số nguyên đó
- Viết hàm main thử sinh 1 con mèo, cộng nó với một số nguyên rồi in ra tên, tuổi, cân nặng của con mèo kết quả

VD: NẠP CHỒNG PHÉP CỘNG, TRỪ

- Viết toán tử cộng hai con mèo và trả về tổng cân của chúng
- Viết toán tử trừ 2 con mèo trả về hiệu tuổi của chúng
- Viết hàm main minh hoạ toán tử cộng, trừ trên

NẠP CHỒNG PHÉP GÁN

- Nhớ lại, C++ cung cấp sẵn một phép gán mặc định, song chỉ là gán theo thành viên, sẽ gây ra vấn đề dùng chung vùng nhớ nếu đối tượng có biến thành viên cấp phát động.
- Do đó trong trường hợp lớp có biến cấp phát động, ta cũng nên tự viết phép gán, sao cho chỉ gán dữ liệu trên vùng nhớ chứ không gán địa chỉ vùng nhớ
- Phép gán này nên trả về tham chiếu tới đối tượng bị gán, giống như phép gán phổ thông:

```
a = b = c; // b = c trả về biến b, gán tiếp cho a
++(a = b); // a = b trả về biến a, được ++
```

VD: NẠP CHỒNG CÁC KIỂU PHÉP GÁN

Với lớp Cat, hãy viết thêm các toán tử sau:

- Phép = giúp gán một con mèo bằng y hệt một con mèo khác
- Phép = giúp gán một con mèo bằng tuổi một con mèo khác
- Phép += một con mèo với một số thực giúp con mèo đó béo lên chừng ấy kg
- Phép += một con mèo với một số nguyên giúp con mèo đó tăng lên chừng ấy tuổi

VD: NẠP CHỒNG PHÉP GÁN

- Thêm toán tử gán cho lớp sau, minh hoạ trong main

```
#include <iostream>
using namespace std;
class CpClass
{
    int *p;
public:
    CpClass(int v = 0): p(new int) { *p = v; }
    ~CpClass(){ delete p; }
    CpClass(const CpClass & r){
        p = new int;                // cấp phát vùng nhớ mới
        *p = *r.p;                  // sao chép dữ liệu
    }
    void set(int v) { *p = v; }
    int get() const { return *p; }
};
```

VD: NẠP CHỒNG PHÉP GÁN

```
class CpClass
{
    int *p;
public:
    CpClass(int v = 0): p(new int) { *p = v; }
    ~CpClass(){ delete p; }
    CpClass(const CpClass & r){
        p = new int;                    // cấp phát vùng nhớ mới
        *p = *r.p;                      // sao chép dữ liệu
    }
    CpClass & operator=(const CpClass & r){ // trả về kiểu tham chiếu
        *p = *(r.p);                        // sao chép dữ liệu
        return *this;                       // trả về toán hạng trái
    }
    void set(int v) { *p = v; }
    int get() const { return *p; }
};
```

VD: NẠP CHỒNG PHÉP GÁN

```
int main(){
    CpClass c1(5);
    CpClass c2(3);
    cout << c1.get() << endl;
    cout << c2.get() << endl;

    c2 = c1;
    cout << c1.get() << endl;
    cout << c2.get() << endl;
    c1.set(7);
    cout << c2.get() << endl;
    return 0;
}
```

CÁC TOÁN TỬ CƠ BẢN KHÁC

- Toán tử ++, --:
  - Dạng tiền tố: trả về tham chiếu, danh sách tham số rỗng, thay đổi đối tượng rồi trả về đối tượng mới
  - Dạng hậu tố: trả về tham trị, có tham số giả kiểu **int**, tạo biến tạm lưu đối tượng cũ, thay đổi đối tượng rồi trả về biến tạm
- Toán tử quan hệ (==, !=, <, <=, >, >=)
  - Trả về kiểu bool
  - Có thể viết thông qua nhau (ví dụ != là phủ định của ==)

VD: NẠP CHỒNG CÁC TOÁN TỬ KHÁC

- Thêm vào lớp Cat các toán tử sau:
  - ++ dạng tiền tố và hậu tố giúp mèo tăng lên 1 kg
  - -- dạng tiền tố và hậu tố giúp mèo giảm 1 kg
  - ++ dạng tiền tố và hậu tố giúp mèo tăng cân gấp đôi
  - == so sánh 2 con mèo có bằng tuổi
  - == so sánh 2 con mèo có y hệt tên, tuổi, cân nặng

TỔNG KẾT

- Con trỏ đối tượng, cấp phát và giải phóng đối tượng
- Biến thực thể, biến tĩnh, hàm tĩnh
- Phép gán mặc định, hàm tạo sao chép
- Nạp chồng toán tử: toán học, quan hệ, dạng tiền tố, hậu tố

Keep calm and carry on