



28TECH
Become A Better Developer

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG





Hướng đối tượng - Object Oriented Programming hay OOP là một mô hình lập trình quan trọng, được ứng dụng rộng rãi trong phát triển phần mềm. OOP tập trung xoay quanh dữ liệu và đối tượng thay vì tập trung vào thủ tục và hàm như các mô hình lập trình truyền thống.



1. Class và Object:



Hai khái niệm quan trọng của lập trình hướng đối tượng là : **Lớp (Class)** và **Đối tượng (Object)**.



Mục tiêu của OOP đó là cố gắng đưa các thực thể, đối tượng trong thực tế vào phần mềm.



Ví dụ phần mềm của bạn cần quản lý những đối tượng như Người, Xe, Sách thì khi đó các đối tượng này cũng được xây dựng trong phần mềm của bạn dưới dạng các lớp.



Có thể hiểu đơn giản, lớp chính là bản mô phỏng của một đối tượng trong thực tế.



1. Class và Object:

Lớp có nhiệm vụ phác họa, mô phỏng những thông tin chung còn khái niệm về đối tượng của lớp lại là một thực thể cụ thể của lớp đó.



Ví dụ: Bạn cần xây dựng một lớp là Person để mô phỏng con người thì khi đó một đối tượng của lớp Person là “Nguyễn Văn Tèo” chính là một thực thể, một đối tượng của lớp Person.

2. Thuộc tính và phương thức:



Để mô tả thông tin của một lớp, bạn cần bổ sung **các thuộc tính (Attribute)** và **các phương thức (Method)**.

VD: Lớp Person cần

Thông tin

- Tên
- Ngày sinh
- Số điện thoại
- Địa chỉ

Đây là các **thuộc tính** cần bổ sung cho lớp.



Hành động

- Đi lại
- Ăn uống
- Giao tiếp

Đây là các **phương thức** cần bổ sung cho lớp.



Person

Tên
Ngày sinh
Địa chỉ
Số điện thoại

Đi lại()
Ăn uống ()
Giao tiếp()



3. Xây dựng lớp:

CÚ PHÁP

```
class class_name{  
    access_specifier1:  
        member1, member2...  
    access_specifier2:  
        member1, member2...  
};
```




Trong class sẽ chứa các member có thể là thuộc tính hoặc phương thức. Khi khai báo các member này thì bạn phải chỉ ra `access_specifier` cho các thành phần này.



Các `access_specifier` (tạm dịch là quyền truy cập): `public`, `protected`, `private`.



3. Xây dựng lớp:



Đối với các member là thuộc tính bạn để quyền truy cập là private để đảm bảo tính chất đóng gói của OOP (Encapsulation). Khi quyền truy cập là private thì các thuộc tính này chỉ có thể truy cập bên trong phạm vi của lớp.

Đối với các member là phương thức bạn để quyền truy cập là public.

Quyền truy cập protected sẽ được giải thích rõ hơn ở phần kế thừa. Thời điểm hiện tại các bạn chỉ cần chú ý cho mình sử dụng public cho phương thức và private cho thuộc tính.

```
class Student{  
    private:  
        string id;  
        string name;  
        string birth;  
        double gpa;  
    public:  
        void greet(){  
            cout << "Hello !";  
        }  
        void in(){  
            cout << "Information";  
        }  
};
```


4. Khai báo đối tượng:



Để khai báo một đối tượng của lớp các bạn sử dụng tên lớp như kiểu dữ liệu. Ngoài ra bạn cũng có thể khai báo mảng đối tượng, vector đối tượng.



Để truy cập các phương thức và thuộc tính của đối tượng ta sử dụng **toán tử '.'**

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         void greet(){
7             cout << "Hello !";
8         }
9         void in(){
10            cout << "Information";
11        }
12 };
13
14 int main(){
15     Student s;
16     s.greet();
17     s.in();
18 }
```

OUTPUT

```
Hello !
Information
```



4. Khai báo đối tượng:



Bạn không thể truy cập vào các member private của lớp.

```
1 class Student{
2     private:
3         string id, name, gpa;
4         double gpa;
5     public:
6         void greet(){
7             cout << "Hello !";
8         }
9         void in(){
10            cout << "Information";
11        }
12 };
13
14 int main(){
15     Student s;
16     cout << "Ten sinh vien :" << s.name << endl;
17 }
```

OUTPUT

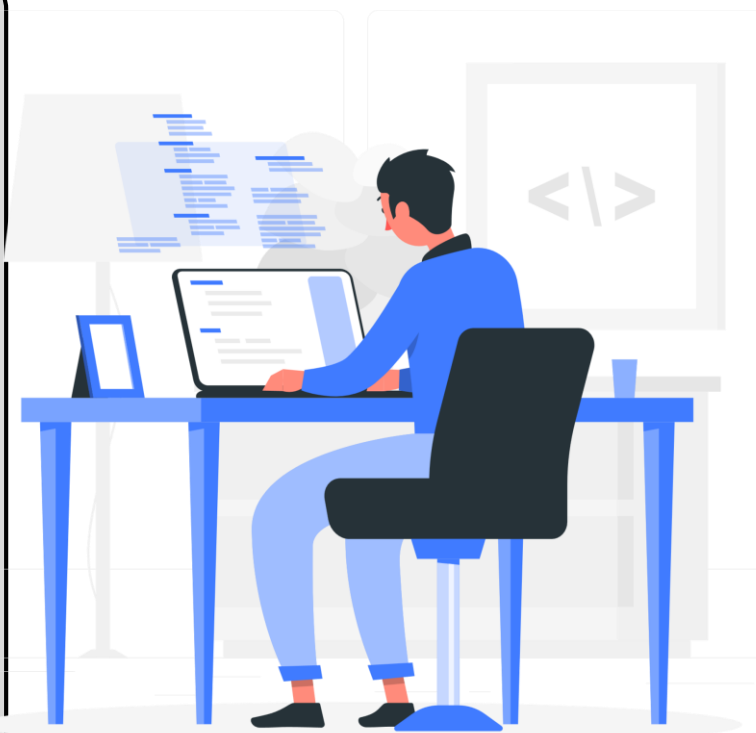
Error : std::string Student::name' is private



5. Hàm tạo và hàm hủy:

Hàm tạo (Constructor)

- Là hàm được gọi mặc định khi bạn khai báo một đối tượng của 1 lớp nào đó, kể cả khi bạn không xây dựng hàm này thì hàm này vẫn tồn tại.
- Tuy nhiên bạn có thể tự xây dựng hàm tạo để nhanh chóng khởi tạo thông tin cho các thuộc tính của đối tượng bằng cách nạp chồng hàm tạo.
- Hàm tạo không có kiểu trả về, có tên trùng với tên lớp, tùy theo tham số bạn truyền vào cho đối tượng khi khai báo thì hàm tạo phù hợp sẽ được gọi.



5. Hàm tạo và hàm hủy:

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         Student(){
7             cout << "Day la ham tao mac dinh !\n";
8             id = "28tech"; name = "28tech";
9             birth = "2804";
10            gpa = 1.0;
11        }
12        Student(string msv, string ten, string ngaysinh, double diem){
13            cout << "Day la ham tao day du tham so !\n";
14            id = msv; name = ten; birth = ngaysinh;
15            gpa = diem;
16        }
17        void in(){
18            cout << id << ' ' << name << ' ' << birth << ' ' << gpa << endl;
19        }
20    };
21
22 int main(){
23     Student s;
24     s.in();
25     Student t("CNTT1", "Nguyen Van Teo", "23/12/2004", 3.4);
26     t.in();
27 }
```

OUTPUT

Day la ham tao mac dinh !
28tech 28tech 2804 1
Day la ham tao day du tham so !
CNTT1 Nguyen Van Teo 23/12/2004 3.4

5. Hàm tạo và hàm hủy:

Hàm tạo (Constructor)

Là hàm được gọi khi một đối tượng của lớp kết thúc quá trình hoạt động. Hàm hủy tương tự như hàm tạo nó có tên trùng với tên lớp, không có kiểu trả về, tuy nhiên có thêm dấu ~ trước tên hàm.

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         Student(){
7             cout << "Ham tao mac dinh duoc goi !\n";
8         }
9         ~Student(){
10            cout << "Ham huy duoc goi !\n";
11        }
12 };
13
14 void solve(){
15     Student tmp;
16 }
17
18 int main(){
19     Student s;
20     solve();
21 }
```

OUTPUT

```
Ham tao mac dinh duoc goi !
Ham tao mac dinh duoc goi !
Ham huy duoc goi !
Ham huy duoc goi !
```



6. Xây dựng phương thức bên ngoài class:



Để xây dựng các phương thức của lớp, bạn có thể xây dựng trực tiếp trong class hoặc xây dựng bên ngoài class. Khi xây dựng các phương thức này ở bên ngoài class cần khai báo thêm toán tử phạm vi.

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         void in();
7 };
8
9 void Student::in(){
10     cout << id << ' ' << name << ' ' << birth << ' ' << gpa << endl;
11 }
12
13
14 int main(){
15     Student s;
16     s.in();
17 }
```



7. Con trỏ this, getter() và setter():



Bạn có thể sử dụng **con trỏ this** trước tên các thuộc tính hoặc phương thức để đảm bảo sự rõ ràng khi có những tham số trùng tên với các thuộc tính.

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         Student(string id, string name, string birth, double gpa){
7             this->id = id;
8             this->name = name;
9             this->birth = birth;
10            this->gpa = gpa;
11        }
12    };
```



7. Con trỏ this, getter() và setter():



Getter(): Khi làm việc với các đối tượng, đôi khi bạn muốn lấy ra các thuộc tính của lớp nhưng không thể truy cập trực tiếp vào các thuộc tính này, giải pháp cho vấn đề này là bạn xây dựng phương thức getter() để lấy về thuộc tính mà bạn mong muốn.

```
1 class Student{
2     private:
3         string id, name, birth;
4         double gpa;
5     public:
6         Student(string id, string name, string birth, double gpa){
7             this->id = id;
8             this->name = name;
9             this->birth = birth;
10            this->gpa = gpa;
11        }
12        string getName(){
13            return this->name;
14        }
15    };
16
17 int main(){
18     Student s("123", "Teo", "22/12/2002", 3.7);
19     cout << s.getName() << endl;
20 }
```

OUTPUT

Teo



7. Con trỏ this, getter() và setter():



Setter(): Tương tự như getter(), khi bạn muốn thay đổi thuộc tính của đối tượng bạn phải sử dụng hàm setter để thay đổi vì không thể trực tiếp truy cập vào các thuộc tính này.

```
class Student{
private:
    string id, name, birth;
    double gpa;
public:
    Student(string id, string name, string birth, double gpa){
        this->id = id;
        this->name = name;
        this->birth = birth;
        this->gpa = gpa;
    }
    string getName(){
        return this->name;
    }
    void setName(string name){
        this->name = name;
    }
};

int main(){
    Student s("123", "Teo", "22/12/2002", 3.7);
    s.setName("Ti");
    cout << s.getName() << endl;
}
```



8. Hàm bạn (Friend function):



Đối với các hàm không phải là phương thức của class thì không thể truy cập vào các thuộc tính private của lớp, tuy nhiên bạn có thể khai báo 1 hàm là hàm bạn của lớp. Hàm này không phải là phương thức của lớp nhưng có thể truy cập vào các thuộc tính private của lớp.

```
class SinhVien{
private:
    string name;
    double gpa;
public:
    SinhVien(string name, double gpa){
        this->name = name;
        this->gpa = gpa;
    }
    friend void in(SinhVien x);
};

void in(SinhVien x){
    cout << x.name << ' ' << x.gpa << endl;
}

int main(){
    SinhVien a("Nguyen Van Teo", 2.8);
    in(a);
}
```

OUTPUT

Nguyen Van Teo 2.8



9. Nạp chồng toán tử với lớp số phức:



Nạp chồng **toán tử +** với lớp số phức, các phép toán khác các bạn tự thực hiện.

```
class SoPhuc{
private:
    int thuc, ao;
public:
    SoPhuc(){}
    SoPhuc(int thuc, int ao){
        this->thuc = thuc;
        this->ao = ao;
    }
    SoPhuc operator + (const SoPhuc b){
        SoPhuc res;
        res.thuc = this->thuc + b.thuc;
        res.ao = this->ao + b.ao;
        return res;
    }
    void in(){
        cout << this->thuc << ' ' << this->ao << endl;
    }
};

int main(){
    SoPhuc a(10, 3), b(7, 6);
    SoPhuc c = a + b;
    c.in();
}
```

OUTPUT

17 9



9. Nạp chồng toán tử với lớp số phức:



Nạp chồng **toán tử ==** để so sánh 2 số phức.

```
class SoPhuc{
private:
    int thuc, ao;
public:
    SoPhuc(){}
    SoPhuc(int thuc, int ao){
        this->thuc = thuc;
        this->ao = ao;
    }
    bool operator == (SoPhuc b){
        return this->thuc == b.thuc && this->ao == b.ao;
    }
    void in(){
        cout << this->thuc << ' ' << this->ao << endl;
    }
};

int main(){
    SoPhuc a(10, 3), b(10, 3);
    cout << (a == b) << endl;
}
```

OUTPUT

1



9. Nạp chồng toán tử với lớp số phức:



Cài đặt **toán tử +** bằng hàm bạn.

```
class SoPhuc{
private:
    int thuc, ao;
public:
    SoPhuc(){}
    SoPhuc(int thuc, int ao){
        this->thuc = thuc;
        this->ao = ao;
    }
    friend SoPhuc operator + (SoPhuc a, SoPhuc b){
        SoPhuc res;
        res.thuc = a.thuc + b.thuc;
        res.ao = a.ao + b.ao;
        return res;
    }
    void in(){
        cout << this->thuc << ' ' << this->ao << endl;
    }
};

int main(){
    SoPhuc a(10, 3), b(7, 6);
    SoPhuc c = a + b;
    c.in();
}
```

OUTPUT

17 9



9. Nạp chồng toán tử với lớp số phức:



Nạp chồng toán tử nhập (>>) và xuất (<<) sử dụng hàm bạn:

```
class SoPhuc{
private:
    int thuc, ao;
public:
    SoPhuc(){}
    SoPhuc(int thuc, int ao){
        this->thuc = thuc;
        this->ao = ao;
    }
    friend ostream & operator << (ostream &out, SoPhuc a){
        out << a.thuc << ' ' << a.ao << endl;
        return out;
    }

    friend istream & operator >> (istream &in, SoPhuc &a){
        in >> a.thuc >> a.ao;
        return in;
    }
    void in(){
        cout << this->thuc << ' ' << this->ao << endl;
    }
};

int main(){
    SoPhuc a;
    cin >> a;
    cout << a;
}
```



10. Static keyword:



Biến static: Biến static thuộc về class chứ không thuộc về bất cứ một đối tượng nào của lớp, tức là các đối tượng của lớp sẽ chung biến static này.



Một hàm cũng có thể là hàm static, khi hàm được khai báo là static thì bạn có thể sử dụng hàm này bằng cách sử dụng tên lớp kèm theo toán tử phạm vi

```
class SinhVien{  
private:  
    string name;  
    double gpa;  
    static int cnt;  
public:  
    SinhVien(){  
        ++cnt; // tang cnt moi lan khai bao 1 SV  
    }  
    static int getCnt(){  
        return cnt;  
    }  
};  
  
//khoei tao gia tri cho cnt ben ngoai  
int SinhVien::cnt = 0;  
  
int main(){  
    SinhVien a;  
    cout << a.getCnt() << endl;  
    SinhVien b;  
    cout << b.getCnt() << endl;  
    cout << a.getCnt() << endl;  
    cout << SinhVien::getCnt() << endl;  
}
```

OUTPUT

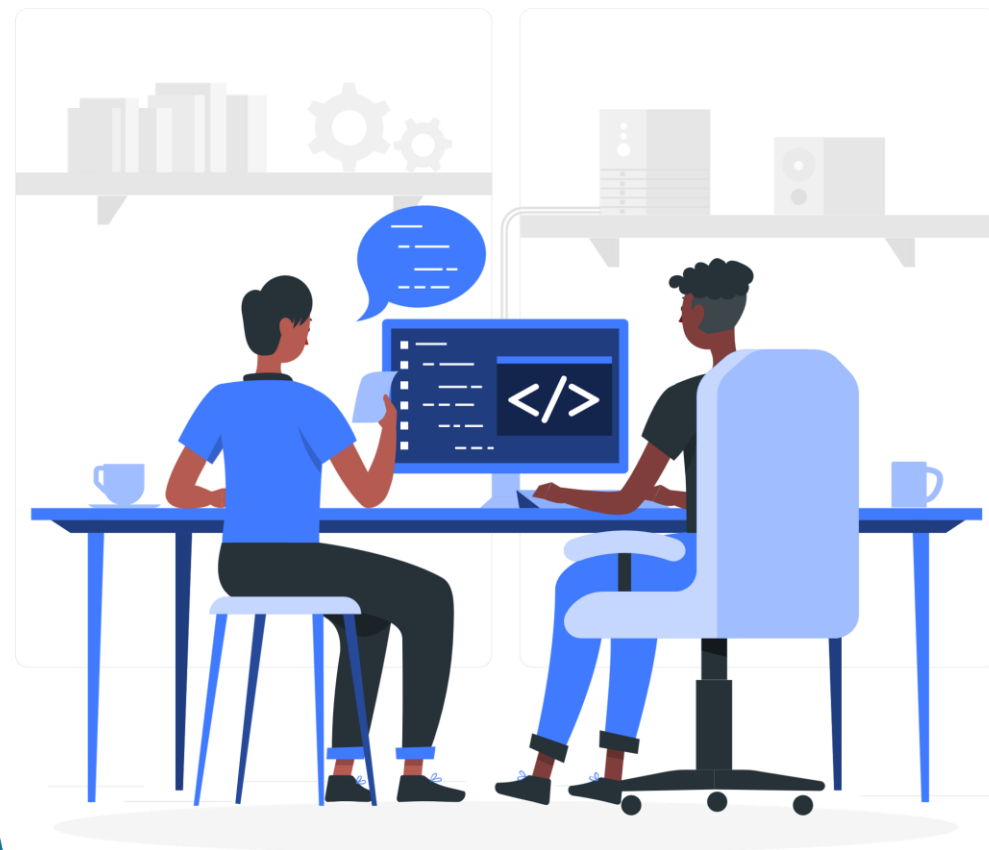
1 2 2 2





28TECH
Become A Better Developer

I. KẾ THỪA (INHERITANCE)



1. Đặt vấn đề:

VẤN ĐỀ

- Giả sử phần mềm của bạn cần quản lý thông tin của những đối tượng Sinh Viên, Giáo Viên, Nhân Viên của một trường đại học.
- Các đối tượng này có những thuộc tính chung ví dụ như tên, tuổi, ngày sinh, địa chỉ.



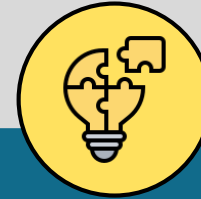
Vậy để có thể tránh được việc dư thừa code và tái sử dụng phần mềm thì hướng giải quyết là gì ?

1. Đặt vấn đề:

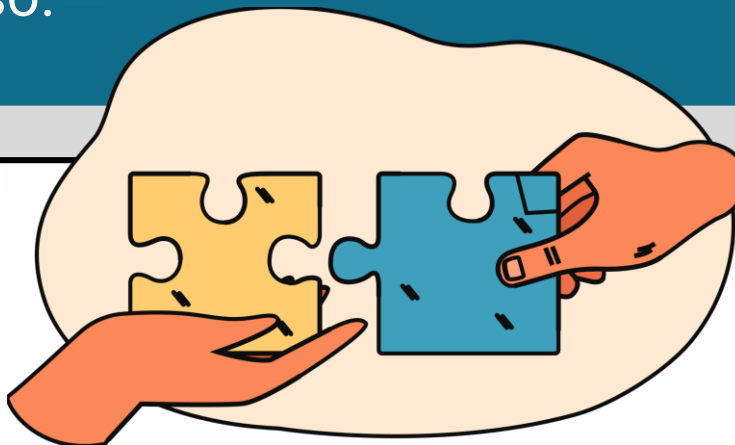
GIẢI PHÁP



Xây dựng một lớp cơ sở (base class) chứa các thuộc tính chung của 3 lớp này, sau đó cho các lớp này kế thừa từ lớp cơ sở.



Khi đó ta cần bổ sung các thuộc tính, phương thức của từng lớp dẫn xuất (derived class).





Lớp trong C++ có thể mở rộng, tạo một lớp mới từ một lớp cũ mà vẫn bảo toàn được những đặc điểm của lớp cũ. Quá trình này gọi là kế thừa, kế thừa liên quan tới các khái niệm như lớp cha (base class hoặc super class), lớp con (derived class hoặc sub class).



2. Cú pháp kế thừa:

CÚ PHÁP

```
class derived_class : access_specifier base_class{};
```

EXAMPLE

```
class Person{  
private:  
    string name, birth, address;  
};  
  
class Student : public Person{  
private:  
    string className;  
    double gpa;  
};
```

2. Cú pháp kế thừa:

— **Chú ý** về access specifier của các thuộc tính trong lớp:

Access	Public	Protected	Private
Member trong cùng 1 class	Yes	Yes	Yes
Member của lớp con	Yes	Yes	No
Không phải member của lớp hay lớp con	Yes	No	No



Giải thích:

- Thuộc tính address trong lớp Person có access specifier là protected nên lớp con là lớp Student có thể truy cập vào thuộc tính address này.
- Tuy nhiên trong lớp Student không thể truy cập vào 2 thuộc tính là name, birth của lớp cha vì 2 thuộc tính này là private, tuy nhiên các bạn cần nhớ rằng mặc dù không thể truy cập nhưng trong lớp Student vẫn có có thuộc tính name và birth.

EXAMPLE

```
class Person{  
private:  
    string name, birth;  
protected:  
    string address = "Hai Duong";  
};
```

```
class Student : public Person{  
private:  
    string className;  
    double gpa;  
public:  
    void in(){  
        cout << address << endl; // OK  
        // cout << name << endl;  
        // cout << birth << endl;  
    }  
};
```

OUTPUT

Hai Duong



3. Các access mode:

Các access mode

Khi kế thừa lớp con cũng kế thừa lớp cha theo 3 access mode: **public, private, protected**.

- **Public:** Nếu access mode là public thì access specifier của các thuộc tính lớp con sẽ giống hệt lớp cha.
- **Private:** Mọi phương thức, thuộc tính của lớp cha sẽ trở thành private của lớp con.
- **Protected:** Các member là public ở lớp cha sẽ trở thành protected ở lớp con.
- **Chú ý:** Thuộc tính hay phương thức private của lớp cha sẽ luôn là private ở lớp con.

4. Hàm khởi tạo và hàm hủy trong kế thừa:



Khi một đối tượng của lớp con được gọi, ban đầu hàm khởi tạo của lớp cha sẽ được gọi trước sau đó mới tới hàm tạo của lớp con.



Ngược lại với hàm hủy, khi một đối tượng của lớp con được hủy thì hàm hủy của lớp con sẽ được gọi trước hàm hủy của lớp cha.

```
class Person{
private:
    string name, birth;
public:
    Person(){
        cout << "Ham khoi tao lop cha\n";
    }
    ~Person(){
        cout << "Ham huy cua lop cha\n";
    }
};
```

```
class Student : private Person{
private:
    string className;
    double gpa;
public:
    Student(){
        cout << "Ham tao cua lop con\n";
    }
    ~Student(){
        cout << "Ham huy cua lop cha\n";
    }
};

int main(){
    Student s;
}
```

EXAMPLE

OUTPUT

```
Ham khoi tao lop cha
Ham tao cua lop con
Ham huy cua lop cha
Ham huy cua lop cha
```



4. Hàm khởi tạo và hàm hủy trong kế thừa:



Để gọi các hàm của lớp cha ở lớp con ta sử dụng tên của lớp cha, kèm theo toán tử phạm vi.

EXAMPLE

```
class Person{
private:
    string name, birth;
public:
    Person(string name, string birth){
        this->name = name;
        this->birth = birth;
    }
    void in(){
        cout << name << ' ' << birth << ' ';
    }
};

class Student : private Person{
private:
    double gpa;
public:
    Student(string name, string birth, double gpa) : Person(name, birth){
        this->gpa = gpa;
    }
    void in(){
        Person::in();
        cout << fixed << setprecision(2) << gpa << endl;
    }
};

int main(){
    Student s("Luong Van Huy", "22/07/2002", 3.4);
    s.in();
}
```



5. Ghi đè hàm:



Giả sử có một hàm cùng tên, kiểu trả về, danh sách tham số ở cả lớp cha và lớp con, khi đó ta đã **ghi đè (Function overriding)** hàm này.



Ví dụ 1: Ghi đè hàm in():

```
class BaseClass{
public:
    void in(){
        cout << "Ham in cua lop cha !\n";
    }
};

class DerivedClass : public BaseClass{
public:
    void in(){
        cout << "Ham in cua lop con !\n";
    }
};

int main(){
    DerivedClass d;
    d.in();
}
```

OUTPUT
Ham in cua lop con !



5. Ghi đè hàm:



Ví dụ 2: Không ghi đè hàm in():

```
class BaseClass{
public:
    void in(){
        cout << "Ham in cua lop cha !\n";
    }
};

class DerivedClass : public BaseClass{
};

int main(){
    DerivedClass d;
    d.in();
}
```

OUTPUT

Ham in cua lop cha !

— Chú ý: Nếu trong lớp con không có hàm in thì việc bạn gọi hàm in của đối tượng d sẽ tương đương với việc bạn gọi hàm in của lớp cha.



6. Các kiểu kế thừa:

a. Kế thừa nhiều mức - Multilevel Inheritance:



Ví dụ: Lớp B kế thừa lớp A, lớp C lại kế thừa lớp B:

```
class A{
public:
    void print(){
        cout << "Base class";
    }
};

class B : public A{

};

class C : public B{

};

int main(){
    C c;
    c.print();
}
```



6. Các kiểu kế thừa:

b. Đa kế thừa - Multiple Inheritance:



Một lớp có thể cùng kế thừa nhiều lớp khác nhau được gọi là đa kế thừa.

VD: Lớp C vừa kế thừa lớp A và lớp B.

```
class A{
public:
    void print(){
        cout << "A class";
    }
};

class B{
public:
    void greet(){
        cout << "B class";
    }
};

class C : public B, public A{
};

int main(){
    C c;
    c.print();
    c.greet();
}
```

OUTPUT

A class
B class



6. Các kiểu kế thừa:

b. Đa kế thừa - Multiple Inheritance:



Trong trường hợp đa kế thừa, nếu trong ví dụ trên lớp A và B đều có chung 1 hàm tên là printf() thì đối tượng của lớp C muốn sử dụng hàm print() của A hoặc B cần chỉ rõ tên lớp kèm theo toán tử phạm vi trước hàm print().

```
class A{
public:
    void print(){
        cout << "A class";
    }
};

class B{
public:
    void print(){
        cout << "B class";
    }
};

class C : public B, public A{
};

int main(){
    C c;
    c.A::print();
    c.B::print();
}
```

OUTPUT

A class
B class



6. Các kiểu kế thừa:

c. Kế thừa phân cấp - Hierarchical Inheritance:



Khi có nhiều lớp con cùng kế thừa từ một lớp cha được gọi là kế thừa phân cấp. **VD:** Lớp Student, Lecturer, Employee kế thừa từ lớp Person

```
class Person{};

class Student : public Person{};

class Lecturer : public Person{};

class Employee : public Person{};
```





II. ĐA HÌNH (POLYMORPHISM)





Đa hình - Polymorphism trong C++ có thể chia ra thành Compile-time polymorphism hoặc Runtime polymorphism. Trong đó compile-time polymorphism xuất hiện trong tính năng ghi đè hàm và nạp chồng toán tử. Runtime polymorphism xuất hiện khi sử dụng hàm ảo - Virtual function.



1. Con trỏ của lớp cha:



Một con trỏ của lớp cha có thể trỏ tới các đối tượng của lớp con. Tuy nhiên nếu trong lớp con có ghi đè một hàm của lớp cha và ta gọi hàm này thông qua con trỏ của lớp cha thì hàm của lớp cha sẽ được lựa chọn thay vì hàm của lớp con.

```
class Person{
public:
    void greet(){
        cout << "Person hello !\n";
    }
};

class Student : public Person{
public:
    void greet(){
        cout << "Student hello !\n";
    }
};

int main(){
    Student s;
    Person *ptr1 = &s;
    ptr1->greet();
}
```

OUTPUT

Person hello !



2. Hàm ảo - Virtual function:



Trong C ++, chúng ta không thể ghi đè các hàm nếu chúng ta sử dụng một con trỏ của lớp cơ sở để trỏ đến một đối tượng của lớp dẫn xuất.



Sử dụng các hàm ảo trong lớp cơ sở đảm bảo rằng hàm có thể được ghi đè trong những trường hợp này.

```
class Person{
public:
    virtual void greet(){
        cout << "Person hello !\n";
    }
};

class Student : public Person{
public:
    void greet(){
        cout << "Student hello !\n";
    }
};

int main(){
    Student s;
    Person *ptr1 = &s;
    ptr1->greet();
}
```

OUTPUT

Student hello !

