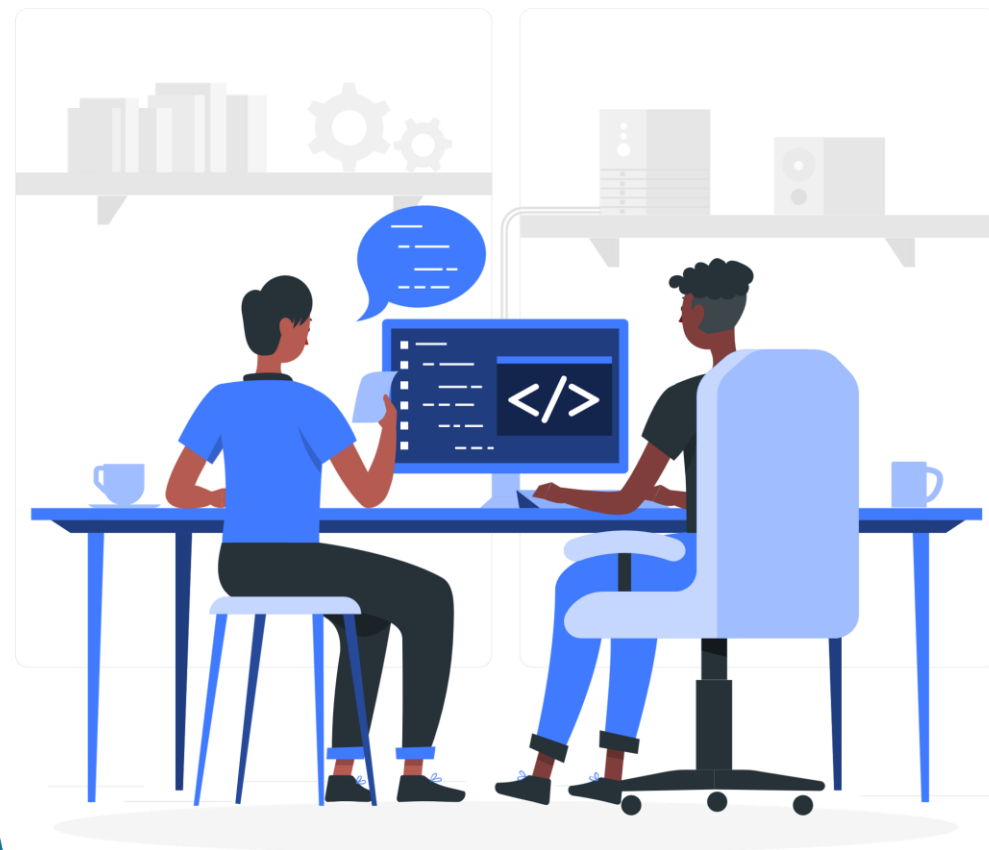




I. KẾ THỪA (INHERITANCE)



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.



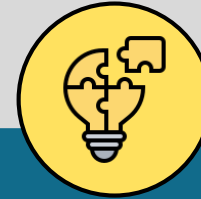
Vậy để có thể tránh được việc dư thừa code và tái sử dụng phần mềm thì hướng giải quyết là gì ?

1. Đặt vấn đề:

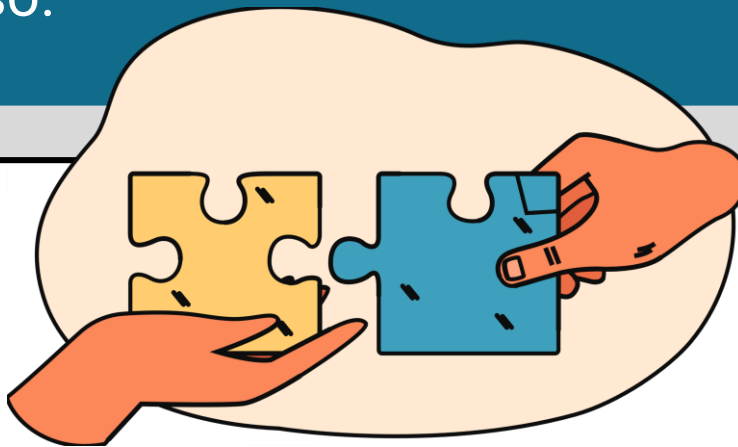
GIẢI PHÁP



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.



```
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 print() 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 !

