TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

Đề cương bài giảng

Lập trình hướng đối tượng

Ngành : Công nghệ thông tin
Trình độ đào tạo: Đại học
Người biên soạn : Khoa CNTT

(Lưu hành nội bộ)

Hà Nội – 5 / 2021

Bài 3: Phương thức khởi tạo, phương thức hủy, con trỏ this và quan hệ kế thừa

PHẦN 1 – PHƯƠNG THỨC KHỞI TẠO/PHƯƠNG THỨC HỦY/CON TRỎ THIS

1.1 Phương thức khởi tạo

Là phương thức đặc biệt của lớp. Phương thức này sẽ được tự động gọi tới khi ta khai báo đối tượng. Phương thức khởi tạo dùng để khởi tạo các giá trị của các thuộc tính, cấp phát bộ nhớ cho các biến con trỏ ngay khi đối tượng được sinh ra.

Cách định nghĩa phương thức khởi tạo:

<Tên phương thức khởi tạo> (<danh sách các đối nếu có>)

{

<Nội dung của phương thức>;

}

VD:

```
class Phanso
{
private:
     int Tuso, Mauso;
public:
     //hàm tạo cho lớp Phanso
     Phanso()
     {
          Tuso = 0;
          Mauso = 1;
     }
};
```

Đặc điểm của phương thức khởi tạo:

- Mỗi lớp có thể có 1 hoặc nhiều phương thức khởi tạo.
- Tên phương thức khởi tạo luôn trùng với tên lớp.
- Phương thức khởi tạo không có kiểu trả về, không có giá trị trả về.

Hàm tạo được chia làm hai loại:

- **Hàm tạo không đối:** để khởi tạo các giá trị mặc định cho các thuộc tính của đối tượng. Ví dụ trên là một hàm tạo không đối, khởi gán các giá trị mặc định cho hai thuộc tính Tuso và Mauso.

- **Hàm tạo có đối:** để khởi gán các giá trị bất kỳ cho các thuộc tính. Các giá trị của các đối sẽ được gán vào các thuộc tính.

VD:

```
class Phanso
{
private:
     int Tuso, Mauso;
public:
     //hàm tạo cho lớp Phanso
     Phanso()
     {
          Tuso = 0;
          Mauso = 1;
     }
     Phanso(int T, int M)
     {
          Tuso = T;
          Mauso = M;
     }
};
```

Nếu một lớp chỉ có hàm tạo có đối, thì khi khai báo đối tượng thuộc lớp, ta phải truyền các giá trị vào các đối tương ứng.

VD1: Trong ví dụ trên, ta có thể khai báo:

Phanso P1, P2(2, 3);

P1 là một đối tượng thuộc lớp Phanso với các giá trị của thuộc tính Tuso = 0 và Mauso = 1; đối tượng P2 được đang có giá trị của các thuộc tính Tuso = 2 và Mauso = 3.

Nhận xét:

- Nếu một lớp có nhiều phương thức khởi tạo thì phương thức khởi tạo không đối là phương thức khởi tạo mặc định.

- Nếu lớp Phanso chỉ có phương thức khởi tạo có đối thì khai báo P1 sẽ báo lỗi.

- Nếu lớp Phanso chỉ có phương thức khởi tạo không đối thì khai báo P2(2, 3) sẽ báo lỗi.

VD2: Viết chương trình định nghĩa lớp Phanso với các thuộc tính Tuso, Mauso và các phương thức:

- Phương thức khởi tạo không đối khởi gán các giá trị mặc định Tuso = 0 và Mauso = 1.
- Phương thức khởi tạo có đối khởi gán các giá trị bất kỳ cho Tuso va Mauso.
- Phương thức xuất: in ra màn hình phân số dưới dạng Tử số/ Mẫu số.

```
class Phanso
   {
      int Tuso, Mauso;
   public:
     Phanso()
     {
          Tuso = 0;
          Mauso = 1;
     }
     Phanso(int T, int M)
     {
          Tuso = T;
          Mauso = M;
     }
     void xuat()
     {
      cout<<"Phan so : "<<Tuso<<"/"<<Mauso<<endl;
     }
  };
     void main()
     {
          Phanso P1, P2(2,5);
          P1.xuat();
          P2.xuat();
          getch();
     }
```

Nếu một lớp có sử dụng đối tượng thuộc lớp khác làm tham số thì hàm tạo của lớp sẽ được gọi trước hàm tạo của đối tượng là tham số.

1.2 Phương thức hủy

Phương thức huỷ là phương thức đặc biệt của một lớp. Phương thức này sẽ được tự động gọi tới mỗi khi kết thúc chu trình sống của một đối tượng thuộc lớp đó, ví dụ: giải phóng bộ nhớ

Phương thức huỷ được xây dựng theo mẫu:

```
<~> <Tên Lớp> (<Danh sách các đối nếu có>)
{
        Thân phương thức huỷ;
}
```

Đặc điểm của phương thức huỷ:

- Tên của phương thức huỷ trùng với tên lớp nhưng có thêm ký tự "~" phía trước: VD phương thức huỷ của lớp Hanghoa là:

```
~Hanghoa()
{
        Thân của phương thức huỷ;
}
```

- Mỗi lớp chỉ có 1 phương thức hủy, không có đối.
- Phương thức huỷ không có kiểu trả về và không có giá trị trả về.
- Phương thức huỷ thường được sử dụng khi trong lớp có sử dụng con trỏ và có cấp phát bộ nhớ. Bộ nhớ được cấp phát cho các thuộc tính cần phải được giải phóng khi kết thúc chu trình sống của đối tượng. Công việc này được thực hiện bởi hàm huỷ.

Như vậy: những lớp không cần giải phóng bộ nhớ khi kết thúc chu trình sống của đối tượng thì không nhất thiết phải có hàm huỷ.

1.3. Con trỏ đối tượng.

a. Khai báo và sử dụng con trỏ đối tượng

Con trỏ đối tượng là biến con trỏ dùng để chứa địa chỉ của đối tượng thuộc lớp (biến đối tượng có kiểu là một lớp nào đó).

Cách khai báo: tương tự như khai báo biến con trỏ trong C++, tuy nhiên, con trỏ phải có cùng kiểu lớp với đối tượng mà nó sẽ chứa địa chỉ. Cú pháp khai báo:

<Tên lớp> * <Tên con trỏ>;

Trong đó:

<Tên lớp>: Trùng với tên lớp của đối tượng mà con trỏ sẽ trỏ tới.

<Tên con trỏ>: Tuỳ ý đặt theo quy ước đặt tên trong C++.

VD: xét khai báo sau: **Nguoi a, *p;**

Dòng lệnh trên khai báo đối tượng a và một con trỏ đối tượng p thuộc lớp Nguoi. Con trỏ p có thể dùng để chứa địa chỉ của đối tượng a. Khi đó ta có thể cho con trỏ p trỏ tới a bằng cách gán: **p =&a;**

Khi con trỏ p đã trỏ tới a, ta có thể dùng p để truy cập các phương thức của đối tượng a. Khi đó ta viết:

<Tên con trỏ> → <Tên phương thức>;

Như vậy hai cách viết sau là tương đương:

Đối tượng a thuộc lớp Nguoi	Con trỏ p chứa địa chỉ của a
`a. Hoten`	`p -> Hoten`
`a. Ngaysinh`	`p -> Ngaysinh`
`a. Nhap()`	`p -> Nhap()`
….	….

1.4. Con trỏ this

Xét một ví dụ xây dựng lớp HCN (hình chữ nhật) bao gồm các thuộc tính: Dài, Rộng và phương thức Nhap() cho phép nhập chiều dài, rộng của hình chữ nhật.

```
class HCN
{
   double D, R;
   public:
   void Nhap()
   {
      cout<<"Nhập chiều dài hình chữ nhật"; cinn>>D;
      cout<<"Nhập chiều rộng hình chữ nhật "; cin>>R;
   }
};
```

Nhận xét:

- Như đã biết, để truy cập tới các thành phần của một đối tượng, ta cần viết: **<tên đối tượng> . <Tên thành phần>**

- Phương thức Nhap() ở trên đã truy cập tới 2 thuộc tính D và R của các đối tượng thuộc lớp HCN xong lại không viết theo khuôn mẫu trên (không có tên đối tượng trước các tên thành phần mà nó truy cập).

Lý giải hiện tượng này thế nào?

Rõ ràng, theo suy nghĩ thông thường, nếu muốn truy cập các thành phần của đối tượng nào thì ta phải chỉ rõ đối tượng đó bằng cách viết:

<Tên đối tượng> . <Tên thành phần>

Tuy nhiên, khi định nghĩa lớp, ta định nghĩa các thành phần cho mọi đối tượng thuộc lớp chứ không cho riêng một đối tượng cụ thể nào. Những truy cập trong nội bộ lớp sẽ mặc định là truy cập tới các thành phần của lớp, do đó không cần chỉ rõ thành phần được truy cập thuộc vào đối tượng nào. Như vậy cách viết trên là hợp lý.

Để điều hợp lý trên không mâu thuẫn với quy ước <tên đối tượng> . <Tên thành phần>, người ta đưa ra khai niệm con trỏ this.

Con trỏ this là một con trỏ đặc biệt, được dùng khi ta định nghĩa lớp, để trỏ tới một đối tượng ảo đại diện cho mọi đối tượng thuộc lớp đang định nghĩa.

Như vậy, thực chất của cách viết trên là:

```
void Nhap()
{
   cout<<"Nhập chiều dài hình chữ nhật";     cin>> this -> D;
   cout<<"Nhập chiều rộng hình chữ nhật ";   cin>> this -> R;
}
```

Con trỏ this có đặc điểm là có thể chỉ rõ, cũng có thể không cần chỉ rõ khi ta truy cập các thành phần. Các truy cập trong khi đang định nghĩa lớp không viết rõ thành phần thuộc đối tượng nào sẽ được chương trình dịch hiểu là truy cập các thành phần của con trỏ this.

PHẦN 2 – SỰ KẾ THỪA

2.1. Các khái niệm ban đầu.

Nếu lớp A kế thừa trực tiếp từ lớp B, ta gọi A là lớp cơ sở (hay cơ bản) và B là lớp dẫn xuất.

Khi xây dựng xong lớp cơ sở, công việc tiếp theo là xây dựng các lớp dẫn xuất kế thừa các lớp cơ sở đó. Ta có hai loại kế thừa:

- *Kế thừa đơn:* Một lớp dẫn xuất chỉ kế thừa trực tiếp các thành phần từ duy nhất một lớp cơ sở.
- *Kế thừa bội* (đa kế thừa): Một lớp dẫn xuất có thể kế thừa trực tiếp các thành phần từ nhiều lớp cơ sở.

Trong kế thừa, người ta cũng quy định phạm vi kế thừa. Chúng ta có 2 phạm vi kế thừa:

- *Kế thừa public:* Tất cả các thành phần có phạm vi public trong lớp cơ sở sẽ trở thành các thành phần có phạm vi public trong lớp dẫn xuất.
- *Kế thừa private:* Tất cả các thành phần có phạm vi public trong lớp cơ sở sẽ trở thành các thành phần có phạm vi private trong lớp dẫn xuất.

Như vậy, dễ thấy các thành phần private trong lớp cơ sở không thể được kế thừa. Điều này đảm bảo tính riêng tư cho mỗi lớp đối tượng. Mặt khác, không phải tất cả các thành phần có thể kế thừa từ lớp "Cha" xuống lớp "con" lại có thể được lớp "cháu" kế thừa.

Tính kế thừa phân chia các lớp đối tượng khác nhau thành các "thế hệ" mà theo đó, thế hệ sau có thể kế thừa những thuộc tính, phương thức chung nhất của thế hệ trước và hình thành lên cây thứ bậc giống như phả hệ. Việc xây dựng phần mềm hướng đối tượng thực chất là việc xây dựng cây thứ bậc này. Trong cây thứ bậc này, ta dễ dàng thấy:

- Kế thừa trực tiếp: Là việc một lớp B kế thừa các thuộc tính và phương thức của lớp A trong cây thứ bậc mà lớp A lại nằm ngay thế hệ trên gần thế hệ lớp B nhất.

- Kế thừa gián tiếp: Là việc lớp B kế thừa các thuộc tính và phương thức của lớp A trong cây thứ bậc mà lớp B có thể cách lớp A từ hai thế hệ trở lên.

2.2. Xây dựng lớp dẫn xuất.

Để xây dựng các lớp mà lớp này kết thừa các thuộc tính và phương thức của lớp kia, người ta thường tuân theo trình tự sau:

[1]. Xây dựng lớp cơ sở: Lớp này sẽ chứa các thuộc tính và phương thức chung có thể dùng cho nhiều lớp ở thế hệ tiếp theo. Lớp cơ sở ban đầu có thể không kế thừa lớp nào khác. Như vậy, việc xây dựng lớp cơ sở có thể chỉ là việc xây dựng một lớp thông thường.

[2]. Xây dựng lớp dẫn xuất: Ngoài các thuộc tính và phương thức của riêng nó, lớp này còn chứa các thuộc tính và phương thức mà nó kế thừa được của lớp cơ sở trực tiếp của nó.

Lớp dẫn xuất được xây dựng theo mẫu sau:

```
class <tên lớp dẫn xuất> <:> <phạm vi kế thừa> <Tên lớp cơ sở>
{
        //Nội dung lớp dẫn xuất
};
```

- <Phạm vi kết thừa>: có thể là public hoặc private.
- Nếu là kế thừa bội thì các lớp cơ sở được đặt cách nhau bởi dấu phảy.
- Lớp cơ sở phải là lớp đã được xây dựng.

VD1: ta có hai lớp cơ sở A, B. Ta xây dựng lớp dẫn xuất C kế thừa public A và kế thừa private B:

```
class C : public A, private B
{
        //Nội dung lớp C
};
```

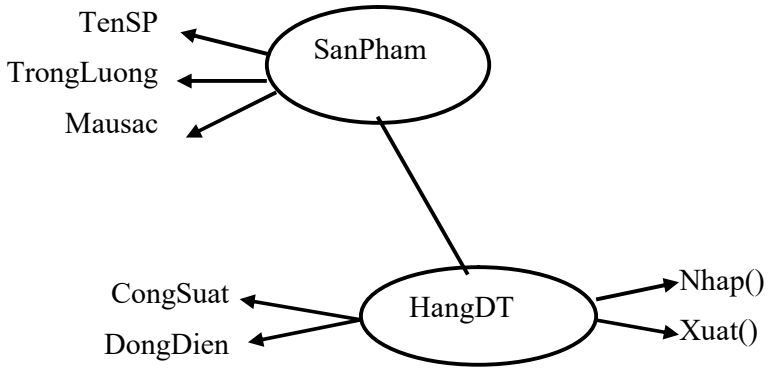
VD2: Xây dựng lớp SanPham gồm các thông tin: Tên sản phẩm, Trọng lượng, Màu sắc. Sau đó, xây dựng lớp dẫn xuất Hàng điện tử ngoài các thông tin của lớp Sản phẩm, lớp Hàng điện tử còn có các thông tin về: Công suất, Dòng điện sử dụng (1 hay 2 chiều) và các phương thức:

Phương thức nhập: nhập các thông tin của hàng điện tử.

Phương thức xuất: xuất các thông tin lên màn hình.

Xây dựng chương trình chính nhập vào một danh sách n hàng điện tử. In danh sách của các hàng điện tử lên màn hình và thông tin của các mặt hàng có trọng lượng thấp nhất.

B1: Thiết kế các lớp:



B2: Tạo khung của lớp

```
class Sanpham
{
public:
     char TenSP[30];
     float TrongLuong;
     char MauSac[20];
};

class HangDT: public Sanpham
{
     float CongSuat;
     char DongDien[20];
public:
     void nhap();
     void xuat();
};
```

B3: Bổ xung các phương thức nhap(), xuat() ở ngoài lớp:

```
void HangDT::nhap()
{
     cout<<"Ten hang "; gets(TenSP);
     cout<<"Trong luong "; cin>>TrongLuong;
     cout<<"Mau sac "; gets(MauSac);
     cout<<"Cong suat "; cin>>CongSuat;
     cout<<"Loai dong dien "; gets(DongDien);
     fflush(stdin);
}


void HangDT::xuat()
{
     cout<<endl;
     cout<<"Ten hang: "<<TenSP<<endl;
     cout<<"Trong luong: "<<TrongLuong<<endl;
     cout<<"Mau sac: "<<MauSac<<endl;
     cout<<"Cong suat: "<<CongSuat<<endl;
     cout<<"Loai dong dien: "<<DongDien<<endl;
}
```

B4: Chương trình chính:

```
void main()
{
     int n,i; HangDT a[100];
     cout<<" Nhap n "; cin>>n;

     for(i=0; i<n; i++)
```

```
        {
             cout<<"Mat hang thu "<<i+1<<endl;
             a[i].nhap();
        }

        cout<<endl<<"thong tin vua nhap la
        "<<endl<<endl;
        for(i=0; i<n; i++) a[i].xuat();

        float Min=a[0].TrongLuong;
        for (i=0; i<n; i++)
             if (a[i].TrongLuong < Min) Min =
        a[i].TrongLuong;

        cout<<endl<<"Mat hang co trong luong nho nhat
        la"<<endl;
        for (i=0; i<n; i++)
             if (a[i].TrongLuong==Min) a[i].xuat();
        getch();
    }
```

2.3.Thiết kế các lớp có sự kế thừa

a. Các thành phần được kế thừa

Trong lớp cơ sở, có những thành phần không bao giờ được kế thừa xuống lớp dẫn xuất.

Các thành phần có thể được kế thừa của lớp cơ sở bao gồm: Các thuộc tính, phương thức Protected, Public.

Các thành phần không được phép kế thừa trong lớp dẫn xuất gồm: Phương thức khởi tạo, phương thức huỷ bỏ, phương thức toán tử, các thuộc tính, phương thức private.

b. Thiết kế các lớp có sự kế thừa

Quá trình thiết kế các lớp dẫn xuất và thừa kế trải qua các bước:

Bước 1: Xác định lớp cơ sở: Ta cần xác định rõ:

- Tên lớp;
- Các thành phần private (là những thành phần không được kế thừa);
- Các thành phần public hoặc protected (các thành phần được kế thừa).

Bước 2: Xác định các lớp dẫn xuất:

- Xác định tập tất cả các thuộc tính, phương thức có thể có của lớp dẫn xuất (gọi là tập A).
- Xác định tập tất cả những thuộc tính, phương thức có thể được kế thừa từ lớp cơ sở (gọi là tập B).

Tập các thuộc tính, phương thức cần xây dựng của lớp cơ sở là A – B.

Bước 3: Vẽ sơ đồ lớp thể hiện sự kế thừa. Sơ đồ được vẽ theo quy ước sau:

- Lớp: được thể hiện bằng đường Elip, trong elip chứa tên lớp.
- Các thuộc tính vẽ bên trái lớp.
- Các phương thức vẽ bên phải lớp.
- Lớp cơ sở vẽ trên lớp dẫn xuất. Liên kết giữa lớp cơ sở và lớp dẫn xuất bằng một đường kẻ thẳng.

Bước 4: Xây dựng khung các lớp: bao gồm các thuộc tính của lớp và nguyên mẫu của các phương thức.

Bước 5: Xây dựng các phương thức cho từng lớp: các phương thức thường được xây dựng bên ngoài lớp để dễ kiểm soát. Xây dựng chương trình chính (nếu cần) để sử dụng các lớp trên.

VD: Thiết kế lớp Person gồm các thông tin: Họ và tên, Ngày sinh, Quê quán. Sau đó, xây dựng lớp dẫn xuất "Kỹ sư" ngoài các thông tin của lớp Person, lớp kỹ sư còn có các thông tin về: Ngành học, Năm tốt nghiệp (int) và các phương thức:

Phương thức nhập: nhập các thông tin của kỹ sư.

Phương thức xuất: xuất các thông tin lên màn hình.

Xây dựng chương trình chính nhập vào một danh sách các kỹ sư. In danh sách của các kỹ sư lên màn hình và thông tin của các kỹ sư tốt nghiệp gần đây nhất (năm tốt nghiệp lớn nhất)

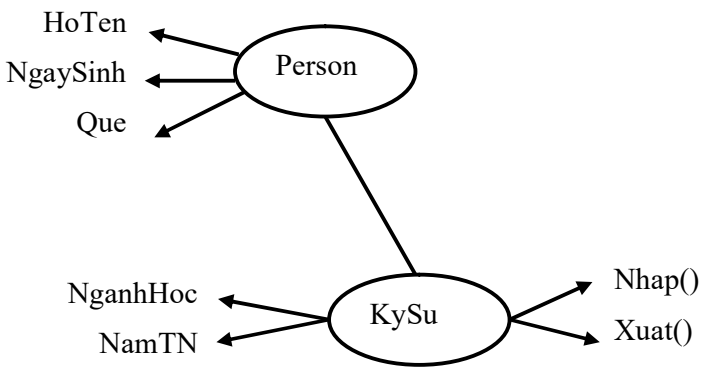
Bước 1: Xác định lớp cơ sở:

Tên lớp	Thành phần private	Thành phần public
Person	Không có	Hoten, NgaySinh, Que

Bước 2: xác định lớp dẫn xuất

Tên lớp	Thuộc tính		Phương thức	
KySu	A	Hoten, NgaySinh, Que, NganhHoc, NamTN	A	Nhap(); Xuat()
	B	Hoten, NgaySinh, Que	B	Không có

Bước 3: Sơ đồ các lớp được thể hiện như sau:



Bước 4: Lập trình tạo khung của các lớp:

```
class Person
{
public:
     char Hoten[30];
     char NgaySinh[8];
     char Que[50];
};
class KySy:public Person
{
 private:
     char NganhHoc[30];
     int NamTN;
 public:
     void nhap();
     void xuat();
 };
```

Bước 5: Viết các phương thức :

```
void KySu::nhap()
{
     cout<<"Nhap Ho ten "; gets(Hoten);
     cout<<"Nhap Ngay Sinh "; gets(NgaySinh);
     cout<<"Nhap Nganh hoc "; gets(NganhHoc);
     cout<<"Nhap nam TN "; cin>>NamTN;
}
void KySu::xuat()
{
     cout<<"Ho ten "<<Hoten<<endl;
     cout<<"Ngay Sinh "<<NgaySinh<<endl;
     cout<<"Nganh hoc "<<NganhHoc<<endl;
     cout<<"nam TN "<<NamTN<<endl;
```

```
}
```

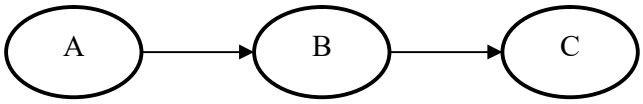
Chương trình chính sử dụng các lớp trên

```
void main()
{
     KySu a;
     a.nhap();
     a.xuat();
     getch();
}
```

2.4. Phương thức khởi tạo, huỷ bỏ và sự kế thừa

a. Thứ tự thực hiện

Giả sử có 3 lớp A, B, C kế thừa nhau theo cây thứ bậc sau:



Tức lớp B kế thừa lớp A, Lớp C lại kế thừa trực tiếp lớp B. Giả sử mỗi lớp đều có các phương thức khởi tạo và phương thức huỷ bỏ riêng.

Trong chương trình chính, nếu ta khai báo một đối tượng thuộc lớp C thì hiển nhiên các phương thức khởi tạo của C sẽ được gọi.

Tuy nhiên, vì lớp C kế thừa lớp B nên để tạo một đối tượng thuộc lớp C, chương trình sẽ phải tạo ra một đối tượng thuộc lớp B để kế thừa.

Vì lớp B lại kế thừa lớp A nên để tạo một đối tượng thuộc lớp B, chương trình sẽ phải tạo ra một đối tượng thuộc lớp A.

Tóm lại khi ta khai báo một đối tượng thuộc lớp C thì chương trình sẽ tạo ra 3 đối tượng thuộc cả 3 lớp trên. Và khi đó, hiển nhiên là cả 3 phương thức khởi tạo của cả 3 lớp sẽ được gọi đến.

Vấn đề đặt ra là: thứ tự thực hiện của các phương thức khởi tạo của cả 3 lớp như thế nào?

Khi ta khai báo một đối tượng thuộc lớp C, trước tiên, phương thức khởi tạo của lớp A sẽ được gọi để tạo ra một đối tượng thuộc lớp A. Tiếp theo, phương thức khởi tạo của lớp B sẽ được gọi để tạo ra đối tượng thuộc lớp B. Và cuối cùng, phương thức khởi tạo của lớp C sẽ được gọi để tạo ra một đối tượng thuộc lớp C.

Tương tự như vậy, khi kết thúc chu trình sống của đối tượng thuộc lớp C thì cả 3 phương thức huỷ thuộc 3 lớp A, B, C đều được gọi. Tuy nhiên, thứ tự thực hiện sẽ ngược lại: tức phương thức huỷ của C sẽ được gọi đầu tiên, tiếp đó là phương thức huỷ của B và C.

VD: Xét đoạn trình định nghĩa 3 lớp A, B, C ở trên.

```
class A
{public:
     A()
     {
          cout<<" Phuong thuc khoi tao cua lop A";
     }
     ~A()
     {
          cout<<" Phuong thuc huy cua lop A";
     }
};
class B: public A
{public:
     B()
     {
          cout<<" Phuong thuc khoi tao cua lop B";
     }
     ~B()
     {
          cout<<" Phuong thuc huy cua lop B";
     }
};
class C: public B
{public:
     C()
     {
          cout<<" Phuong thuc khoi tao cua lop C";
     }
     ~C()
     {
          cout<<" Phuong thuc huy cua lop C";
     }
};
void main()
{
     C b;
     getch();
}
```

Có thể chạy chương trình này để khẳng định những kết luận trên.

b. Phương thức khởi tạo có đối

Giả sử ta có 2 lớp A, B với lớp B kế thừa trực tiếp từ lớp A. Cả hai lớp A, B đều có phương thức khởi tạo có đối.

Trong chương trình chính, nếu ta khai báo một đối tượng thuộc lớp B thì cần phải truyền giá trị khởi tạo vào cho các thuộc tính.

VD: Lớp B có một thuộc tính và 1 phương thức khởi tạo có đối:

```
class B: public A
{
public:
     int a;
     B(int x)
     {
          a = x;
     }
};
```

Khi đó, muốn khai báo đối tượng thuộc lớp B thì ta cần truyền giá trị khởi tạo cho thuộc tính a, chẳng hạn: **`B dt(5);`** tức đối tượng `dt` thuộc lớp B với tham số khởi tạo là 5 cho thuộc tính a (`dt.` a = 5).

Vấn đề là: vì B kế thừa A nên khi đối tượng thuộc lớp B được tạo ra thì cũng có một đối tượng thuộc lớp A được tạo ra. Nếu lớp A có hàm tạo có đối thì ta cũng phải truyền giá trị cho đối này. Truyền thế nào?

Khi đó, khi định nghĩa phương thức khởi tạo cho B, ta cần định nghĩa thêm các đối để truyền cho đối tượng thuộc lớp A khi đối tượng này được tạo ra. Cú pháp như sau:

B(<Kiểu> <Đối của B>, <Kiểu> <Đối của A>**) : A**(<Đối của A>)

{

Thân phương thức khởi tạo của B;

}

Trong đó:

<Kiểu> <Đối của B>: là danh sách các đối cần thiết để truyền vào cho đối tượng thuộc lớp B.

<Kiểu> <Đối của A>: Là danh sách các đối định nghĩa thêm để khởi gán các gía trị cho đối tượng thuộc A khi nó được tạo ra do ta tạo đối tượng thuộc B.

VD: Lớp A có 1 thuộc tính a và phương thức khởi tạo có đối.

```
class A
{
public:
     int a;
     A(int x)
     {
          a=x;
     }
};
```

Lớp B có 1 thuộc tính b và phương thức khởi tạo có đối. B kế thừa A.

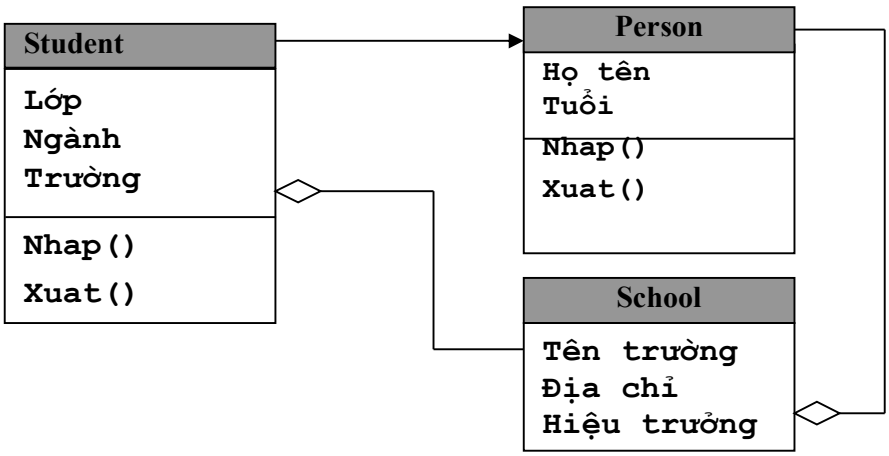
```
class B: public A
```

```
{
public:
int b;
      B(int x, int y):A(x)
      {
           b=y;
      }
      void xuat()
      {
           cout<<" a = " <<a<<endl;
           cout<<" b = "<<b<<endl;
      }
};
void main()
{
      B b(3, 5);
      b.xuat();
      getch();
}
```

2.4. . Một số lưu ý khi cài đặt quan hệ kế thừa

Để ôn lại sơ đồ cài đặt trên (hình 7) và duyệt lại [1, 5], ta xét một sơ đồ trong đó có cả quan hệ kế thừa, cả quan hệ kết tập như ví dụ minh họa số 2 sau:

Ví dụ minh họa 2: Cài đặt các lớp theo sơ đồ sau:



Viết chương trình chính nhập vào danh sách n sinh viên, in ra các sinh viên của trường ĐHCNHN.

Ta làm tuần tự 4 bước theo sơ đồ hình 7 như sau:

B0. Xác định thứ tự cài đặt:

Theo nguyên tắc "Cha trước – con sau" và "Không có hình thoi trước – có hình thoi sau" ta xác định được thứ tự cài đặt là: Person, School, Student.

B1. Cài đặt khung các lớp:

```
class Person
{
public:
        char HT[30];
        int Tuoi;
        void Nhap();   void Xuat() ;
} ;
```

```
class School
{
        char TenTr[30] ;  char Diachi[50] ;
        Person HieuTruong;
};
```

```
class Student:public Person
{
        char Lop[30];  char Nganh[50];
        School Truong;
public:
        void Nhap();   void Xuat();
} ;
```

[1]. Cần chú ý với lớp School, thuộc tính HieuTruong phải là một đối tượng thuộc lớp Person vì thuộc tính này biểu thị quan hệ kết tập của lớp Person vào lớp School. Nếu không chú ý điều này, việc khai báo char HieuTruong[50]; sẽ không được chấp nhận do Hieutruong không thể là 1 thuộc tính có kiểu nguyên thủy mà là thuộc tính có kiểu là 1 lớp (Person). Tương tự với thuộc tính Truong của lớp Student nó biểu thị mối quan hệ kết tập của lớp School vào lớp Student; do vậy nó phải có kiểu là School.

[2]. Một điểm đáng lưu ý khác là với lớp School, ta không có phương thức nhập, xuất nên rất dễ bị lỗi truy cập khi từ 1 lớp bên ngoài hoặc 1 hàm bên ngoài ta truy cập vào các thuộc tính này; do vậy có thể phải xử lý truy cập bằng hàm bạn và lớp bạn. Việc này sẽ được lưu ý ở bước sau.

[3]. Để cài đặt lớp cha (Person), về cơ bản giống với 1 lớp thông thường. Nhưng cần lưu ý: những thành phần nào muốn di truyền xuống lớp con ta cần đặt phạm vi truy cập là public hoặc protected. Để đơn giản, tốt hơn hết nên đặt phạm vi truy cập public ngay khi bắt đầu khai báo các thuộc tính. Nếu không chú ý điều này, một lỗi nghiêm trọng sẽ xảy ra và để lại hậu quả rất lớn cho toàn bộ bài tập.

[4]. Lớp Student là lớp con, kế thừa từ lớp Person. Vậy khi cài đặt, ta chú ý tới sự kế thừa này. Nếu muốn lớp thể hiện sự kế thừa trong cài đặt, ta viết:

- Kế thừa public: (1)

class <Tên lớp con> : public <Tên lớp cha>

```
{
        ...
};
```

- Hoặc kế thừa private: (2)

```
class <Tên lớp con> : private <Tên lớp cha>
{
        ...
};
```

Nhưng để thuận lợi cho truy cập, ta thường sử dụng (1). Do vậy với lớp Student, ta viết:

```
class Student:public Person
{
        ....
};
```

B2. Cài đặt phương thức ngoài lớp:

- Với phương thức nhập, xuất của lớp Person, ta cài như bình thường

```
void Person::Nhap()
{
        cout<<"Họ tên:"; gets(HT); fflush(stdin);
        cout<<"Tuoi:";    cin>>Tuoi;
}
void Person::Xuat()
{
        cout<<"Họ tên:"<<HT<<endl;
        cout<<"Tuổi:"<<Tuoi<<endl;
}
```

- Nhưng với phương thức nhập, xuất của lớp Student, ta cần chú ý xác định đầy đủ các thuộc tính của Student và chú ý tới việc sử dụng lại code.

Trước tiên, ta cần hiểu lớp Student có các thuộc tính: Lop, Nganh, Truong.TrenTr, Truong.Diachi. Ngoài ra nó còn kế thừa được hai thuộc tính HT và Tuoi của lớp Person. Lớp này cũng có 2 phương thức Nhap(), Xuat() của bản thân nó và 2 phương thức Nhap(), Xuat() mà nó kế thừa được từ Person. Vì vậy rất dễ nhầm lẫn giữa các phương thức Nhap(), Xuat() này. Để phân biệt, ta cần chú ý: hai phương thức Nhap(), Xuat() mà nó kế thừa được từ lớp Person được viết là **Person::Nhap()** và **Person::Xuat().**

```
void Student::Nhap()
{
        cout<<"Lớp:"; gets(Lop); fflush(stdin);
        cout<<"Ngành:"; gets(Nganh); fflush(stdin);
        cout<<"Tên trường:";    gets(Truong.TenTr) ; fflush(stdin) ;
        cout<<"Địa chỉ trường:";    gets(Truong.Diachi) ; fflush(stdin) ;
        Person::Nhap();
}

void Student::Xuat()
{
        cout<<"Lớp:"<<Lop<<endl;
        cout<<"Ngành:"<<Nganh<<endl;
        cout<<"Tên trường:"<<Truong.TenTr<<endl ;
        cout<<"Địa chỉ trường:"<<Truong.Diachi<<endl;
        Person::Xuat();
}
```

Hãy chú ý: Lệnh Person::Nhap() sẽ gọi tới phương thức Nhap() của lớp Person mà lớp Student này kế thừa được. Vì nó kế thừa được nên nó có quyền gọi. Và phương thức này có nhiệm vụ nhập, xuất cho hai thuộc tính HT và Tuoi mà lớp Student kế thừa được.

Với thuộc tính Truong, ta cần nhập hai thuộc tính Truong.TenTr và Truong.Diachi. Nhưng vì lớp School không có phương thức nhập nên ta cần nhập "thủ công", và việc này đã vô tình truy cập thuộc tính riêng tư của lớp School. Do vậy, ta cần đặt lớp Student này là bạn của lớp School. Vậy khung lớp School được sửa lại bằng cách thêm khai báo lớp bạn:

```
class School
{
        char TenTr[30] ;
        char Diachi[50] ;
        Person HieuTruong;
        friend class Student;
};
```

B3. Cài đặt hàm truy cập thuộc tính riêng tư (nếu có):

Từ bước này, ta phải đọc yêu cầu của đầu bài. Yêu cầu này được cho kèm với sơ đồ lớp: *"Viết chương trình chính nhập vào danh sách n sinh viên, in ra các sinh viên của trường ĐHCNHN".*

Việc nhập vào danh sách n sinh viên, ta chỉ việc sử dụng phương thức Nhap() của lớp Student. Nhưng việc in ra các sinh viên của trường ĐHCNHN, ta cần kiểm tra thuộc tính Truong.TenTr xem có phải là "ĐHCNHN" hay không. Việc này sẽ truy cập vào

thuộc tính Truong (của lớp Student) và TenTr (của lớp School). Do vậy, cần xây dựng một hàm In(...) để làm nhiệm vụ này. Hiển nhiên hàm In này sẽ phải là bạn của lớp Student và lớp School.

```
void In(Student a[100], int n)
{
        for(int i=0; i<n; i++)
        if (strcmp(a[i].Truong.TenTr, "ĐHCNHN")==0)
        a[i].Xuat();
}
-----------------------------------------------------
void main()
{
        Student a[100]; int n;
        cout<<" Nhập n="; cin>>n;
        for(int i=0; i<n; i++)
        a[i].Nhap();
        ------------------------------------
        In(a, n);
}
```

Để khai báo hàm In là bạn của lớp Student và School, hai khung lớp này được khai báo lại như sau: (đây cũng là toàn bộ lời giải cho ví dụ minh họa số 2).

```
class Person
{
public:
        char HT[30];
        int Tuoi;
        void Nhap();  void Xuat() ;
} ;
-----------------------------------------------------
class School
{
        char TenTr[30] ;
        char Diachi[50] ;
        Person HieuTruong;
        friend class Student;                    //lớp bạn
        friend void In(Student *a, int n);       // hàm bạn
};
-----------------------------------------------------
class Student:public Person
{
        char Lop[30];
        char Nganh[50];
        School Truong;
public:
        void Nhap();   void Xuat();
        friend void In(Student *a, int n);       //hàm bạn
} ;
```

Và các phần code còn lại:

```
void Person::Nhap()
{
        cout<<"Họ tên:"; gets(HT); fflush(stdin);
        cout<<"Tuoi:";    cin>>Tuoi;
}

void Person::Xuat()
{
        cout<<"Họ tên:"<<HT<<endl;
        cout<<"Tuổi:"<<Tuoi<<endl;
}
```

```
void Student::Nhap()
{
        cout<<"Lớp:"; gets(Lop); fflush(stdin);
        cout<<"Ngành:"; gets(Nganh); fflush(stdin);
        cout<<"Tên trường:";    gets(Truong.TenTr) ; fflush(stdin) ;
        cout<<"Địa chỉ trường:";    gets(Truong.Diachi) ; fflush(stdin) ;
        Person::Nhap();
}
void Student::Xuat()
{
        cout<<"Lớp:"<<Lop<<endl;
        cout<<"Ngành:"<<Nganh<<endl;
        cout<<"Tên trường:"<<Truong.TenTr<<endl ;
        cout<<"Địa chỉ trường:"<<Truong.Diachi<<endl;
        Person::Xuat();
}
```

```
void In(Student a[100], int n)
{
        for(int i=0; i<n; i++)
        if (strcmp(a[i].Truong.TenTr, "ĐHCNHN")==0)
        a[i].Xuat();
}
```

```
void main()
{
        Student a[100]; int n;
        cout<<" Nhập n="; cin>>n;
        for(int i=0; i<n; i++)
        a[i].Nhap();
        ---------------------------------------
        In(a, n);
}
```