Ricky NGUYEN

Hướng dẫn ôn tập

Lập trình hướng đối tượng bằng C++

Dùng cho: ôn thi hết môn, ôn thi tốt nghiệp, ôn thi liên thông cao đẳng - đại học trường ĐHCNHN

Tài liệu của: ..... | ....

Tài liệu bao gồm:

- Phần 1. Cài đặt theo sơ đồ lớp
- Phần 2. Cài đặt bài tập dạng phiếu
- Phần 3. Phương pháp chung cài đặt bài tập HĐT.
- Phần 4. Các dạng bài tập khác.
- Phụ lục. Một số bài tập

Tài liệu sinh viên nên tham khảo:

- C++ và lập trình hướng đối tượng - GS. Phạm Văn Ất
- Lập trình hướng đối tượng – Đoàn Văn Ban

Hà Nội 7-2010

MỤC LỤC

Phần 1. Cài đặt theo sơ đồ lớp 4
Phần 2. Cài đặt bài tập dạng phiếu 32
Phần 3. Phương pháp chung cài đặt bài tập HĐT 39
Phần 4. Các dạng bài tập khác 42
Phụ lục. Một số bài tập 51
Bài tập dạng sơ đồ lớp 51
Bài tập dạng phiếu 58
Bài tập cài đặt phương thức toán tử 62

LỜI NÓI ĐẦU

Tài liệu này không phải là một giáo trình hay một đề cương bài giảng. Nó đơn giản chỉ là bản hướng dẫn các bạn thực hiện các dạng bài tập chính môn học lập trình hướng đối tượng.

Tài liệu được dùng cho việc ôn tập thi hết môn, thi tốt nghiệp môn LTHĐT của hệ cao đẳng và có thể có ích cho việc ôn thi liên thông từ cao đẳng lên đại học của một số bạn.

Không giống như các môn học khác (các dạng bài tập là tương đối độc lập và rõ ràng), với môn học này, khó để có được các phân biệt giữa các dạng bài tập; và cũng thật khó (với cá nhân tác giả) để có được một trình bày thật đơn giản, dễ hiểu cho các bạn. Tuy nhiên, tác giả đã cố gắng trình bày một cách hệ thống để không tạo ra thêm các khó khăn cho người đọc, đặc biệt là những người chưa chuẩn bị tốt cho môn học này.

Do thời gian thực hiện tài liệu hạn hẹp, chắc chắn tài liệu còn nhiều thiếu sót và phần nào chưa đáp ứng được đòi hỏi của người đọc. Tác giả mong nhận được những góp ý, trao đổi của các bạn để phiên bản sau được hoàn thiện hơn.

Ricky NGUYEN

Ngày biên soạn: 12/7/2010; Ngày hoàn thành: 13/7/2010

PHẦN 1

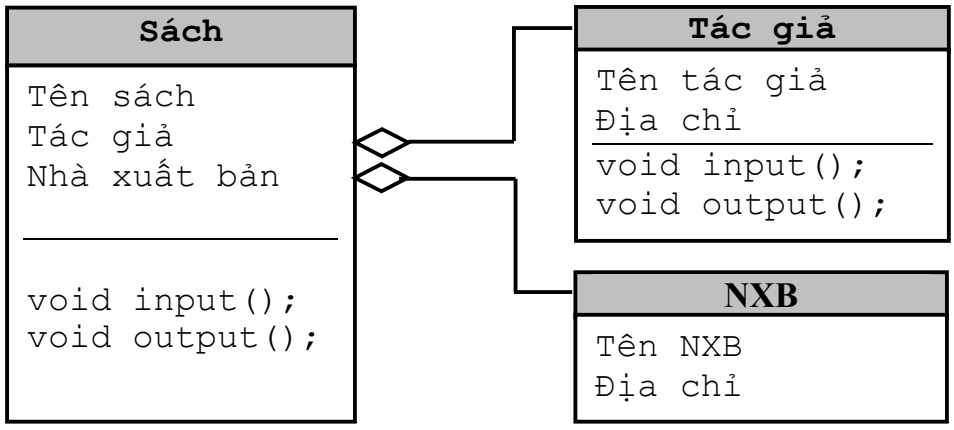
CÀI ĐẶT THEO SƠ ĐỒ LỚP

Đa số các bài tập LTHĐT thường cho dưới dạng sơ đồ lớp. Các bạn cần cài đặt một bài tập theo một sơ đồ lớp cho trước và đáp ứng được các đòi hỏi kèm theo. Muốn vậy, hãy đảm bảo bạn đã nắm được các kiến thức sau đây:

1. Cách đọc và hiểu sơ đồ lớp
2. Cách xác định thứ tự cài đặt các lớp.
3. Cách cài đặt một lớp đơn giản.
4. Cách truy cập thuộc tính riêng tư
5. Sơ đồ 4 bước để hoàn thành cài đặt một bài tập dạng sơ đồ lớp.
6. Một số lưu ý khi cài đặt quan hệ kế thừa

Sau đây sẽ trình bày lần lượt các vấn đề nêu trên. Để dễ dàng nắm được chúng, xin mời theo dõi kèm ví dụ minh họa sau:

Ví dụ minh họa 1: Hãy cài đặt sơ đồ lớp sau:

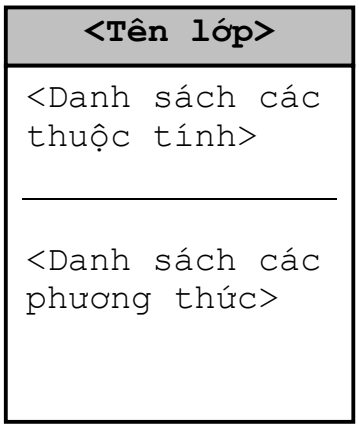


Hãy cài đặt hàm main nhập vào một danh sách gồm n cuốn sách, in ra các sách do nhà xuất bản "Thanh Niên" ấn hành.

---

1. Đọc và hiểu sơ đồ lớp

Một lớp được biểu diễn bằng một hình chữ nhật, trong đó ghi tên lớp, danh sách các thuộc tính và danh sách các phương thức của lớp, như hình sau:



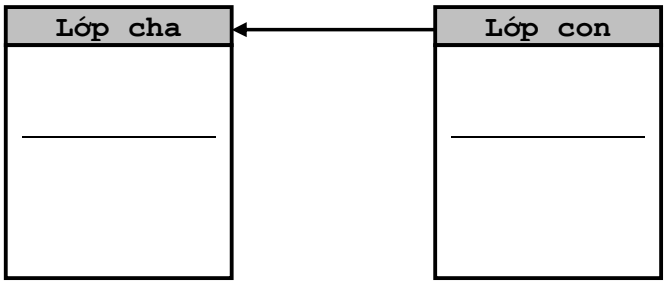
Hình 1. Biểu diễn một lớp

Như vậy, bằng cách nhìn vào sơ đồ lớp, trước hết cần phải xác định: có bao nhiêu lớp cần cài đặt trong bài; với mỗi lớp, cần xác định sơ qua các thuộc tính và phương thức của chúng.

Một sơ đồ lớp không chỉ có duy nhất 1 lớp mà thường chứa nhiều lớp. Các lớp trong sơ đồ không độc lập với nhau mà thường có những mối liên hệ nhất định. Vì vậy, để đọc chính xác một sơ đồ lớp, cần phải nắm vững các mối liên hệ giữa các lớp.

Theo chuẩn UML, hai lớp bất kỳ có thể có nhiều kiểu liên hệ với nhau. Tuy nhiên, để đơn giản, tài liệu này sẽ trình bày 2 kiểu liên hệ chính:

- **Liên hệ cha con:** Nếu hai lớp A và B có liên hệ cha con thì mối liên hệ này được biểu diễn bằng một mũi tên rỗng (sau đây dùng mũi tên đặc !), nối giữa A và B. Chiều của mũi tên chỉ từ lớp con về lớp cha.

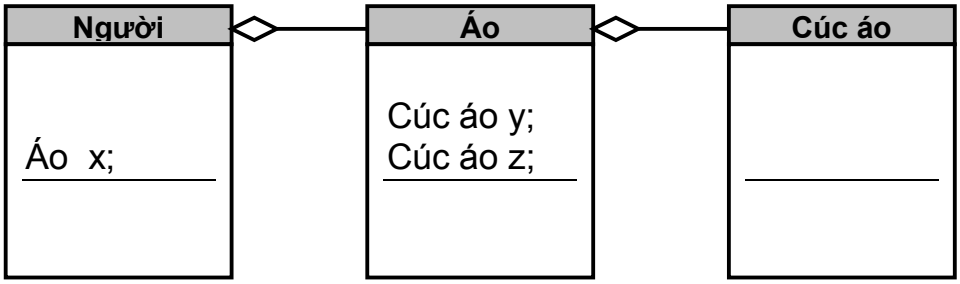


Hình 2. Mũi tên chỉ mối liên hệ cha – con

- **Liên hệ kết tập (hàm chứa):** Bạn là một đối tượng thuộc lớp "Người". Chiếc áo bạn đang mặc chỉ là một thuộc tính của bạn; tuy nhiên nó lại là một đối tượng thuộc lớp áo. Vậy giữa lớp "Áo" và lớp "Người" có quan hệ với nhau như thế nào?

Trên thực tế có rất nhiều mối quan hệ kiểu như vậy. Chẳng hạn với lớp áo, những chiếc cúc gắn trên đó chỉ là các thuộc tính của áo, nhưng nó lại là các đối tượng thuộc lớp "Cúc áo".

Khi đó, ta nói: lớp "Cúc áo" kết tập vào lớp "Áo" và lớp "Áo" kết tập vào lớp "Người". Để biểu diễn mối quan hệ giữa hai lớp A và B trong đó B kết tập vào A, người ta dùng một đường nối giữa A và B, đầu phía A, người ta vẽ một hình thoi rỗng. Vì vậy, có thể biểu diễn mối quan hệ giữa 3 lớp "Người", "Áo", "Cúc áo" theo hình sau:

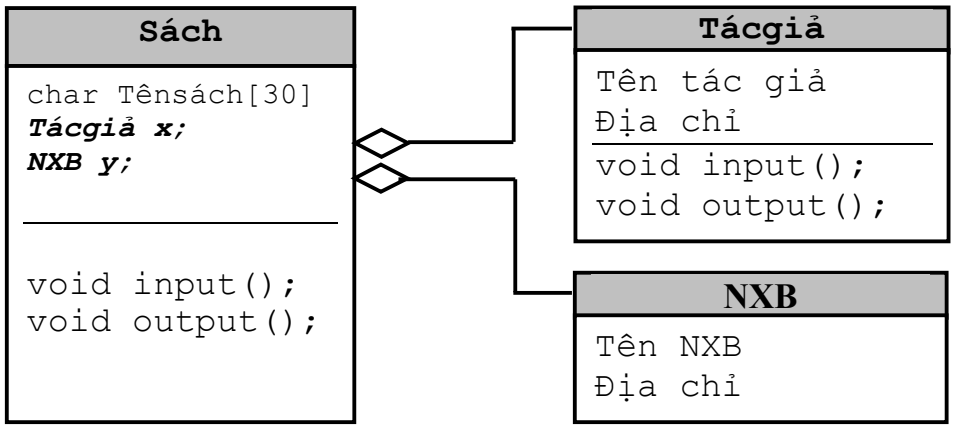


Hình 3. Mối quan hệ kết tập.

Dễ thấy x là một thuộc tính của lớp "Người", tuy nhiên nó lại là một đối tượng thuộc lớp "Áo" (hay còn gọi là áo x); và thuộc tính này đã thể hiện mối quan hệ kết tập của lớp "Áo" vào lớp "Người". Khi đọc sơ đồ lớp, hãy quan tâm tới những thuộc tính kiểu như vậy. Tương tự thuộc tính y, z của lớp "Áo" thể hiện sự kết tập của lớp "Cúc áo" vào lớp "Áo". Những thuộc tính như vậy sẽ không có kiểu nguyên thủy (int, float, char...) mà có kiểu là một tên lớp (x có kiểu Áo; y, z có kiểu Cúc áo).

Trở lại ví dụ chính, dễ thấy sơ đồ gồm 3 lớp: Sách, Tác giả, NXB và hai lớp Tác giả, NXB kết tập vào lớp Sách. Tuy nhiên, trong lớp Sách, các bạn hãy chú ý tới 2 thuộc tính thể hiện sự kết tập này: "Tác giả" (1) , "Nhà xuất bản" (2). Tác giả của một cuốn sách phải là một đối tượng thuộc lớp "Tác giả"; vì vậy ta cần hiểu (1) chính là một đối tượng thuộc lớp "Tác giả"; cách hiểu chính xác phải là: Tác giả x; Tương tự, với thuộc tính (2), ta cần hiểu đó chính là NXB y; với x, y là hai đối tượng mà ta tự đặt tên, thuộc lớp "Tác giả" và lớp "NXB".

Vậy cụ thể, sơ đồ lớp trên được hiểu như sau:



Hình 4. Cách hiểu rõ hơn các thuộc tính kết tập.

2. Xác định thứ tự cài đặt các lớp trong sơ đồ

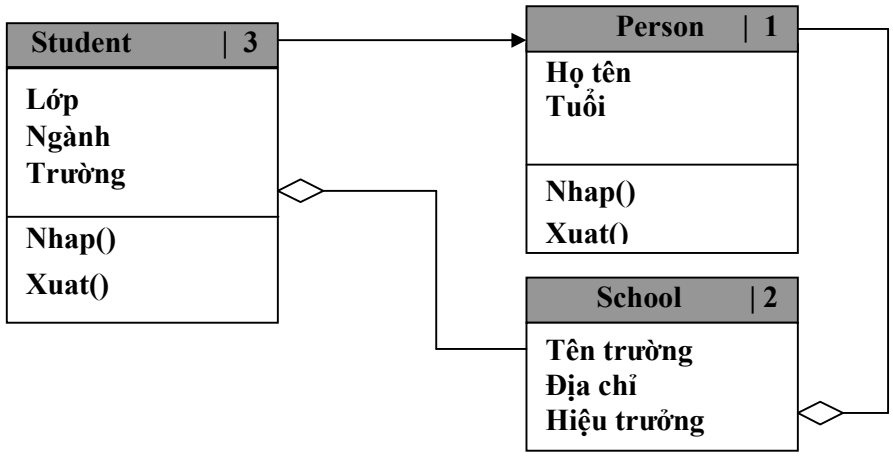
Một việc làm không tốn nhiều thời gian mà lại rất quan trọng khi cài đặt sơ đồ lớp là xác định đúng thứ tự cài đặt các lớp trong một sơ đồ. Việc xác định sai thứ tự cài đặt trong một sơ đồ phức tạp có thể gây ra hậu quả xấu. Để làm được việc này, hãy chú ý tuân thủ quy tắc sau:

[1]. Cha trước, con sau

[2]. Không hình thoi trước, có hình thoi sau.

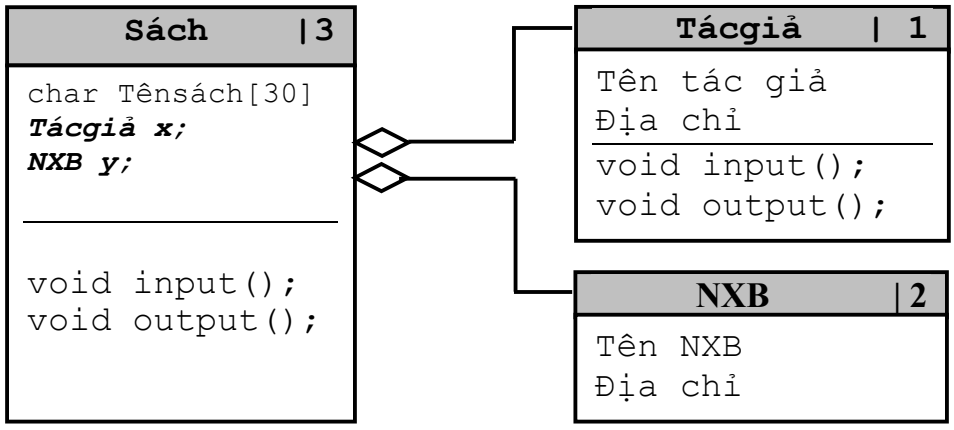
Quy tắc này có nghĩa là: với 2 lớp có quan hệ cha con, phải cài đặt (hoặc khai báo) lớp cha trước, sau đó cài đặt lớp con. Với hai lớp có quan hệ kết tập thể hiện bằng một đường nối với 1 đầu của nó có chứa hình thoi, ta hãy cài đặt lớp ở phía không có hình thoi trước, có hình thoi sau.

Ví dụ với sơ đồ sau, thứ tự cài đặt được xác định như sau (số thứ tự được đánh kèm với mỗi lớp):



Hình 5. Thứ tự cài đặt của các lớp trong sơ đồ

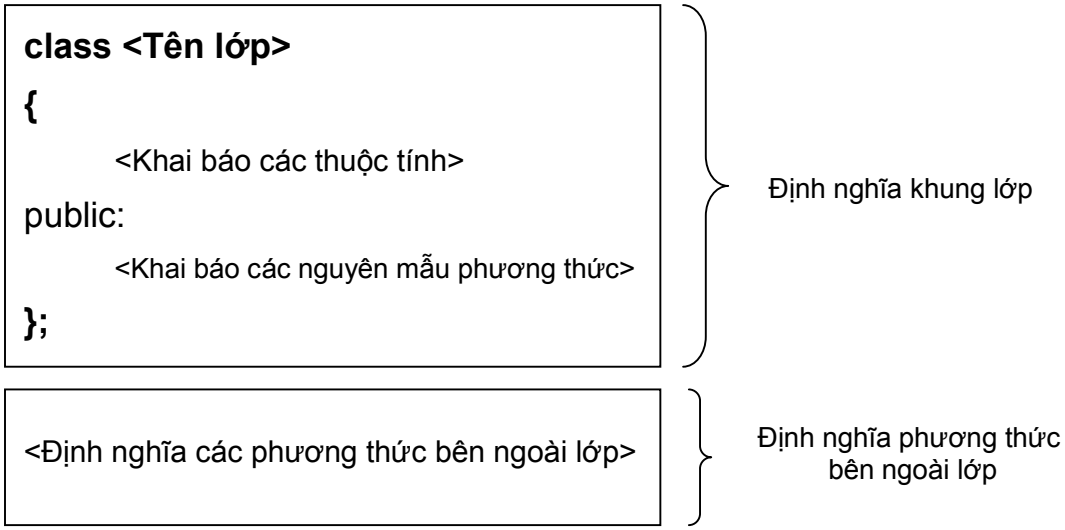
Như vậy, dễ dàng xác định được thứ tự cài đặt cho các lớp trong ví dụ minh họa. Trong sơ đồ này, vì lớp "Tác giả" và lớp "NXB" không có liên hệ trực tiếp gì với nhau nên có thể cài đặt lớp nào trước đều được:



Hình 6. Thứ tự cài đặt các lớp của ví dụ minh họa

3. Cài đặt một lớp đơn giản

Một lớp thường được cài đặt theo cách sau (để không mất tập trung, tác giả chỉ trình bày 1 cách phổ biến).



Hình 7. Định nghĩa một lớp

[1]. Như vậy ta chia công việc này thành 2 phần: định nghĩa khung lớp và định nghĩa các phương thức ngoài lớp. Với khung lớp, mỗi thuộc tính là một biến. Biến này có thể có kiểu nguyên thủy (int, float, char...) hoặc kiểu lớp (nếu nó thể hiện quan hệ kết tập – xin xem phần trên).

[2]. Trước phần <Khai báo các nguyên mẫu phương thức> ta đặt từ khóa "public:". Từ khóa này nhằm xác định phạm vi truy cập "công cộng" cho các phương thức của lớp. Mỗi thuộc tính hay phương thức ta đều có thể chỉ định 1 phạm vi truy cập cho chúng với 3 mức độ truy cập là **private** (riêng tư – chỉ được truy cập trong nội bộ lớp), **public** (công cộng – được truy cập ở mọi nơi) và **protected** (được bảo vệ - chỉ được truy cập trong nội bộ lớp và các lớp con kế thừa lớp này).

Cách đặt phạm vi truy cập: ta viết

<Tên phạm vi truy cập> :

Phạm vi này sẽ ảnh hưởng tới tất cả các thuộc tính, phương thức đứng sau nó cho tới khi gặp một phạm vi truy cập khác. Các thuộc tính không được chỉ định phạm vi truy cập thì mặc định có phạm vi truy cập **private** (riêng tư).

Mỗi phương thức là một hàm được viết theo quy tắc viết hàm của C++ như sau:

```
<Kiểu trả về> <Tên phương thức> ([Danh sách các đối])
{
        //Thân phương thức
}
```

[3]. Với sơ đồ lớp trong ví dụ minh họa, tất cả các phương thức của các lớp đều có kiểu "void" và tên phương thức đã cho trước (input, output), vì vậy một phương thức input() có thể được viết như sau:

```
void input( )
{
        //Thân phương thức
}
```

Phương thức này không có giá trị trả về (nếu vậy thì ta viết kiểu trả về là void); phương thức cũng không có đối vào (ta để trống nhưng không thể thiếu các dấu mở đóng ngoặc).

[4]. Nguyên mẫu của phương thức chính là dòng đầu tiên của phương thức, kèm theo dấu chấm phẩy. Ví dụ với phương thức input ở trên thì nguyên mẫu của nó là: **void input();**

[5]. Khi định nghĩa phương thức ngoài lớp thì cần thêm **<Tên lớp>::** vào trước tên phương thức để xác định rõ là phương thức này thuộc lớp nào. Ví dụ với phương thức input của lớp Sách, ta viết:

```
void Sach::input()
{
        //Thân phương thức
}
```

Với phương thức input của lớp "Tác giả", ta viết:

```
void Tacgia::input()
{
        //Thân phương thức
}
```

Vậy ta có thể cài đặt cho lớp "Tác giả" như sau:

Khung lớp: **chú ý kết thúc khung lớp có dấu ";"**

```
class Tacgia
{
        char TenTg[30];
        char Diachi[50];
public:
        void input();
        void output();
};
```

Cài đặt phương thức ngoài lớp:

```
void Tacgia::input()
{
        //Thân phương thức input
}
```

```
void Tacgia::output()
{
        //Thân phương thức output
}
```

Tương tự lớp NXB, do lớp này không có phương thức nên việc cài đặt chúng hết sức đơn giản (chỉ có khung lớp).

```
class NXB
{
        char TenNXB[30];
        char Diachi[50];
};
```

Và lớp Sách được cài đặt với khung lớp như sau:

```
class Sach
{
        char Tensach[30];
        Tacgia  x;
        NXB  y;
public:
        void input();
        void output();
};
```

và các phương thức ngoài lớp

```
void Sach::input()
{
        //Thân phương thức input
}
void Sach::output()
{
        //Thân phương thức output
}
```

Về nội dung của các phương thức (thân phương thức) thường rất đơn giản. Với ví dụ trên ta chỉ việc tiến hành nhập, xuất các thuộc tính của lớp tương ứng. Với thuộc tính kiểu xâu, ta dùng lệnh gets(Thuộc tính); để nhập và chú ý làm sạch

bộ đệm bàn phím sau khi nhập bằng lệnh fflush(stdin); để xuất, ta dùng lệnh cout<< "nội dung cần xuất"; và dùng endl để xuống dòng. Bây giờ, hãy bắt đầu với lớp "Tác giả":

```
void Tacgia::input()
{
        cout<< "Tên tác giả:"; gets(TenTg); fflush(stdin);
        cout<< "Địa chỉ:"; gets(Diachi); fflush(stdin);
}
void Tacgia::output()
{
        cout<< "Tên tác giả: "<< TenTg<<endl;
        cout<< "Địa chỉ: "<<Diachi<<endl;
}
```

Nhưng với lớp Sách, do có quan hệ kết tập của 2 lớp "Tác giả" và "NXB" vào nên khi nhập, xuất ta cần xác định rõ lớp Sách này có bao nhiêu thuộc tính. Hãy nhìn vào sơ đồ lớp, dễ thấy lớp Sách có 3 thuộc tính chính gồm: Tên sách, x, y; nhưng nếu phân tích chi tiết thì x là một đối tượng thuộc lớp Tác giả nên nó có hai thuộc tính là x.TenTg và x.Diachi. Tương tự y cũng có 2 thuộc tính là y.TenNXB và y.Diachi. Vì vậy, một cách chi tiết, lớp Sách phải có 5 thuộc tính.

Nhưng vì x thuộc lớp "Tác giả" nên nó có cả hai phương thức input() và output() của lớp "Tác giả". Phương thức này có nhiệm vụ nhập, xuất cho các thuộc tính x.TenTg và x.Diachi. Vì vậy, để nhập hai thuộc tính này ta chỉ cần gọi phương thức x.input(); . Việc sử dụng phương thức của đối tượng x để nhập các thuộc tính cho nó được gọi là "sử dụng lại code" hay reuse trong quan hệ kết tập. Cách sử dụng lại code này được viết đơn giản:

<Tên đối tượng>. <Phương thức>;

Tuy nhiên, với thuộc tính (đối tượng) y; ta lại không có được sự tiện lợi như vậy. Đơn giản vì y thuộc lớp NXB, mà lớp này lại không có các phương thức input(), output(). Do vậy, trong lớp Sách này, ta không thể gọi phương thức để nhập, xuất cho hai thuộc tính y.TenNXB và y.Diachi mà cần nhập, xuất trực tiếp chúng.

Vậy hai phương thức input() và output() của lớp Sách được viết như sau:

```
void Sach::input()
{
        cout<< "Tên sách:"; gets(Tensach); fflush(stdin);
        x.input();          //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:"; gets(y.TenNXB); fflush(stdin);
        cout<< "Địa chỉ NXB:"; gets(y.Diachi); fflush(stdin);
}
```

```
void Sach::output()
{
        cout<< "Tên sách: "<<Tensach<<endl;
        x.output();          //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:" <<y.TenNXB<<endl;
        cout<< "Địa chỉ NXB:" <<y.Diachi<<endl;
}
```

Vậy toàn bộ code cho sơ đồ lớp trong ví dụ minh họa được viết lại là:

```
class Tacgia                                                    1
{
        char TenTg[30];
        char Diachi[50];
public:
        void input();
        void output();
};

void Tacgia::input()
{
        cout<< "Tên tác giả:"; gets(TenTg); fflush(stdin);
        cout<< "Địa chỉ:"; gets(Diachi); fflush(stdin);
}
void Tacgia::output()
{
        cout<< "Tên tác giả: "<< TenTg<<endl;
        cout<< "Địa chỉ: "<<Diachi<<endl;
}
```

```
class NXB                                                       2
{
        char TenNXB[30];
        char Diachi[50];
};
```

```
class Sach                                                      3
{
        char Tensach[30];
        Tacgia  x;
        NXB  y;
public:
        void input();
        void output();
};

void Sach::input()
{
        cout<< "Tên sách:"; gets(Tensach); fflush(stdin);
        x.input();        //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:"; gets(y.TenNXB); fflush(stdin);
        cout<< "Địa chỉ NXB:"; gets(y.Diachi); fflush(stdin);
}
void Sach::output()
{
        cout<< "Tên sách: "<<Tensach<<endl;
        x.output();       //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:" <<y.TenNXB<<endl;
        cout<< "Địa chỉ NXB:" <<y.Diachi<<endl;
}
```

Việc cài đặt sơ đồ lớp trong ví dụ minh họa 1, về cơ bản đã hoàn thành. Tuy nhiên đoạn code trên đã mắc phải một lỗi cơ bản. Nếu copy đoạn code trên vào C++ và soát lỗi ta sẽ gặp 4 thông báo lỗi giống nhau. (phần in đậm và gạch

chân trong phương thức Sach::input() và Sach::output(). Lỗi này được hiểu đơn giản như sau:

Do hai phương thức Sach::input() và Sach::output() là thuộc lớp Sach, nhưng trong thân của nó lại truy cập tới hai thuộc tính TenNXB và Diachi của lớp NXB. Hai thuộc tính này có phạm vi truy cập là riêng tư (private) nên không thể được truy cập từ ngoài lớp NXB. Để khắc phục lỗi này, xin xem phần sau: cách truy cập thuộc tính riêng tư.

4. Cách truy cập thuộc tính riêng tư

[1]. Chuyển thuộc tính riêng tư thành công cộng:

Như trên, lỗi của 2 phương thức Sach::input() và Sach::output() dễ dàng khắc phục bằng cách đơn giản là đặt phạm vi truy cập công cộng (public) cho hai thuộc tính TenNXB và Diachi trong lớp NXB. Như vậy lớp NXB được viết lại là:

```
class NXB
{
public:
        char TenNXB[30];
        char Diachi[50];
};
```

Nhưng như vậy, về bản chất không phải là cách để truy cập thuộc tính riêng tư, mặt khác, các thuộc tính cũng hiếm khi được đặt phạm vi truy cập public để đảm bảo an toàn và tính riêng tư về dữ liệu của đối tượng.

[2]. Bổ sung các phương thức Set/ Get cho mỗi thuộc tính:

Chẳng hạn khi muốn truy cập thuộc tính TenNXB của lớp NXB từ lớp Sách, ta bổ sung phương thức GetTenNXB() trả về TenNXB và phương thức SetTenNXB(char * Ten) để gán Ten vào TenNXB. Lớp NXB được viết lại là:

```
class NXB
{
        char TenNXB[30];
        char Diachi[50];
public:
        char * GetTenNXB()
        { return TenNXB;}

        void SetTenNXB(char* Ten)
        {strcpy(TenNXB, Ten);}
};
```

và khi truy cập thuộc tính TenNXB, ta không truy cập trực tiếp thuộc tính này mà truy cập gián tiếp qua hai phương thức Set/ Get ở trên. Ví dụ với phương thức Sach::output(), nếu muốn xuất TenNXB, thay vì viết cout<<y.TenNXB<<endl; ta viết: cout<<y.GetTenNXB()<<endl;

Thực chất của cách này là truy cập gián tiếp vào thuộc tính riêng tư thông qua hai phương thức Set/ Get là hai phương thức công cộng mà ta bổ sung vào. Cách này không được tác giả trình bày chi tiết do không khuyến khích, đề nghị đọc thêm tài liệu nếu cần.

[3]. Lớp bạn, hàm bạn

Để lớp Sách có thể truy cập được thuộc tính riêng tư của lớp NXB, cách đơn giản nhất là cho lớp Sách làm bạn với lớp NXB. Nếu lớp A là bạn của lớp B thì A có toàn quyền truy cập thuộc tính riêng tư của B. Điều ngược lại không đúng.

Vậy làm thế nào để lớp Sách là bạn của lớp NXB? rất đơn giản, trong lớp NXB, ta khai báo: **friend class Sach;**

Vậy lớp NXB được viết lại là:

```
class NXB
{
        char TenNXB[30];
        char Diachi[50];
        friend class Sach;
};
```

Vậy toàn bộ đoạn mã cài đặt sơ đồ lớp của ví dụ minh họa đã hoàn tất nhờ một thao tác đơn giản là đặt lớp Sách là bạn của lớp NXB.

Vậy còn hàm bạn? chúng ta hãy xét tiếp yêu cầu của ví dụ minh họa. Với ví dụ này, sau khi cài đặt xong sơ đồ lớp, việc tiếp theo là: *"Hãy cài đặt hàm main nhập vào một danh sách gồm n cuốn sách, in ra các sách do nhà xuất bản "Thanh Niên" ấn hành."*

Toàn bộ công việc này được giải quyết trong hàm main. Sau khi định nghĩa lớp, việc tiếp theo là viết hàm main để sản sinh các đối tượng thuộc các lớp vừa định nghĩa và sử dụng các đối tượng này để hoàn thành các công việc yêu cầu của đề bài.

Với ví dụ này, ta chỉ đơn giản là khai báo 1 mảng các đối tượng thuộc lớp Sách và nhập danh sách này. Sau đó duyệt mảng và in ra các sách có NXB là "Thanh Niên". Hãy xem code sau:

```
void main()
{
        Sach a[100]; int n;

        //Nhập danh sách – trước tiên nhập n
        cout<< "n="; cin>>n;
        for(int i=0; i<n; i++)
        a[i].input();

        // Xuất thông tin sách của NXB Thanh Niên
        for( i=0; i<n; i++)
         if (strcmp(a[i].y.TenNXB, "Thanh Niên") == 0)
        a[i].output();
}
```

Đoạn code trên cũng mắc lỗi truy cập tương tự như trên. Vì hàm main nằm ngoài các lớp Sach và NXB nhưng lại truy cập tới thuộc tính y (của lớp Sach) và TenNXB (của lớp NXB - đoạn in đậm và gạch chân) nên việc báo lỗi truy cập là hiển nhiên.

Để khắc phục lỗi này, tương tự, ta sẽ cho hàm main là bạn của lớp Sách và lớp NXB.

Nhưng hàm main, do đặc quyền của nó, nó không bao giờ chịu làm bạn với bất kỳ lớp thông thường nào. Vậy xử lý vấn đề này thế nào?

Đơn giản là ta sẽ tách đoạn code mà trong đó có truy cập thuộc tính riêng tư thành 1 hàm riêng (nằm ngoài hàm main); với ví dụ trên, đoạn code đó là đoạn xuất thông tin sách của NXB Thanh Niên. Vậy hàm main giờ được tách thành 2 hàm như sau:

```
void In(Sach a[100], int n)
{
        for( int i=0; i<n; i++)
        if (strcmp(a[i].y.TenNXB, "Thanh Niên") == 0)
                a[i].output();
}
```

```
void main()
{
        Sach a[100]; int n;

        //Nhập danh sách – trước tiên nhập n
        cout<< "n="; cin>>n;
        for(int i=0; i<n; i++)
        a[i].input();

        // Xuất thông tin sách của NXB Thanh Niên – gọi hàm In
        In(a, n);
}
```

Rõ ràng ta đã đẩy lỗi truy cập từ hàm main sang hàm void In(...). Như vậy hàm main không còn là kẻ gây ra lỗi truy cập nữa, mà lỗi này bị đẩy cho hàm void In(...). Và việc tiếp theo, đơn giản ta cho hàm void In(...) làm bạn với lớp Sách và NXB. Vậy lớp Sách và lớp NXB được viết lại là:

```
class NXB
{
        char TenNXB[30];
        char Diachi[50];
        friend class Sach;
        friend void In(Sach *a, int n);                    // ←
};
```

```
class Sach
{
        char Tensach[30];
        Tacgia  x;
        NXB  y;
public:
        void input();
        void output();
        friend void In(Sach *a, int n);                    // ←
};
```

Vậy để 1 hàm (chẳng hạn hàm void In(...) ở trên) là hàm bạn của 1 lớp (chẳng hạn lớp Sách hoặc NXB) thì đơn giản: Trong thân lớp, ta khai báo**: friend <Nguyên mẫu của hàm>;**

Tuy nhiên, hãy lưu ý 2 vấn đề quan trọng mà có thể gặp sai lầm khi thiết lập hàm bạn cho lớp:

[1]. Nguyên mẫu của hàm không được chứa đối vào là mảng: ví dụ với hàm In ở trên, theo cách hiểu thông thường thì nguyên mẫu của hàm phải là dòng đầu tiên của hàm + dấu ";" tức là: void In(Sach a[100], int n);

Nguyên mẫu này có 2 đối vào là Sach a[100] và int n. Với đối a, nó không hợp lệ do nó là mảng. Vậy ta cần chuyển a thành con trỏ (vì con trỏ lúc này cũng tương đương với mảng) và khai báo hàm bạn sẽ là: **friend void In(Sach *a, int n);**

[2]. Nếu viết: **friend void In(Sach *a, int n);** thì trước đó, ta cần khai báo lớp Sách, nếu không chương trình sẽ báo lỗi chỗ đối vào **Sach * a**, do nó không hiểu Sach là gì (vì chương trình dịch sẽ dịch từ trên xuống và tới đó nó chưa gặp lớp Sach).

Để không bị lỗi này, đơn giản là ta khai báo lớp bạn **friend class Sach;** trước khi khai báo hàm bạn (như trên) và đặt dòng khai báo hàm bạn: **friend void In(Sach *a, int n);** sau dòng khai báo lớp bạn này.

Nhưng trong những trường hợp không có khai báo lớp bạn thì sao? Khi đó ta có thể thêm khai báo nguyên mẫu của lớp Sách trước lớp NXB. Tức là:

```
class Sach; //dòng này khai báo nguyên mẫu lớp Sách
class NXB
{
        char TenNXB[30];
        char Diachi[50];
        friend void In(Sach *a, int n); //vậy có thể để khai báo này lên trên
        friend class Sach;
};
```

Vậy đoạn code hoàn thiện bài ví dụ trên sẽ phải là:

```
class Tacgia
{
        char TenTg[30];
        char Diachi[50];
public:
        void input();
        void output();
};
void Tacgia::input()
{
        cout<< "Tên tác giả:"; gets(TenTg); fflush(stdin);
        cout<< "Địa chỉ:"; gets(Diachi); fflush(stdin);
}
void Tacgia::output()
{
        cout<< "Tên tác giả: "<< TenTg<<endl;
        cout<< "Địa chỉ: "<<Diachi<<endl;
}
```

```
class NXB
{
        char TenNXB[30];
        char Diachi[50];
        friend class Sach;
        friend void In(Sach *a, int n);
};
```

```
class Sach
{
        char Tensach[30];
        Tacgia x;
        NXB y;
public:
        void input(); void output();
        friend void In(Sach *a, int n);
};

void Sach::input()
{
        cout<< "Tên sách:"; gets(Tensach); fflush(stdin);
        x.input();      //dòng này nhập 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:"; gets(y.TenNXB); fflush(stdin);
        cout<< "Địa chỉ NXB:"; gets(y.Diachi); fflush(stdin);
}
```

```
void Sach::output()
{
        cout<< "Tên sách: "<<Tensach<<endl;
        x.output();        //dòng này xuất 2 thuộc tính x.TenTg và x.Diachi
        cout<< "Tên NXB:" <<y.TenNXB<<endl;
        cout<< "Địa chỉ NXB:" <<y.Diachi<<endl;
}
```

```
void In(Sach a[100], int n)
{
        for( int i=0; i<n; i++)
        if (strcmp(a[i].y.TenNXB, "Thanh Niên") == 0)
                a[i].output();
}
```

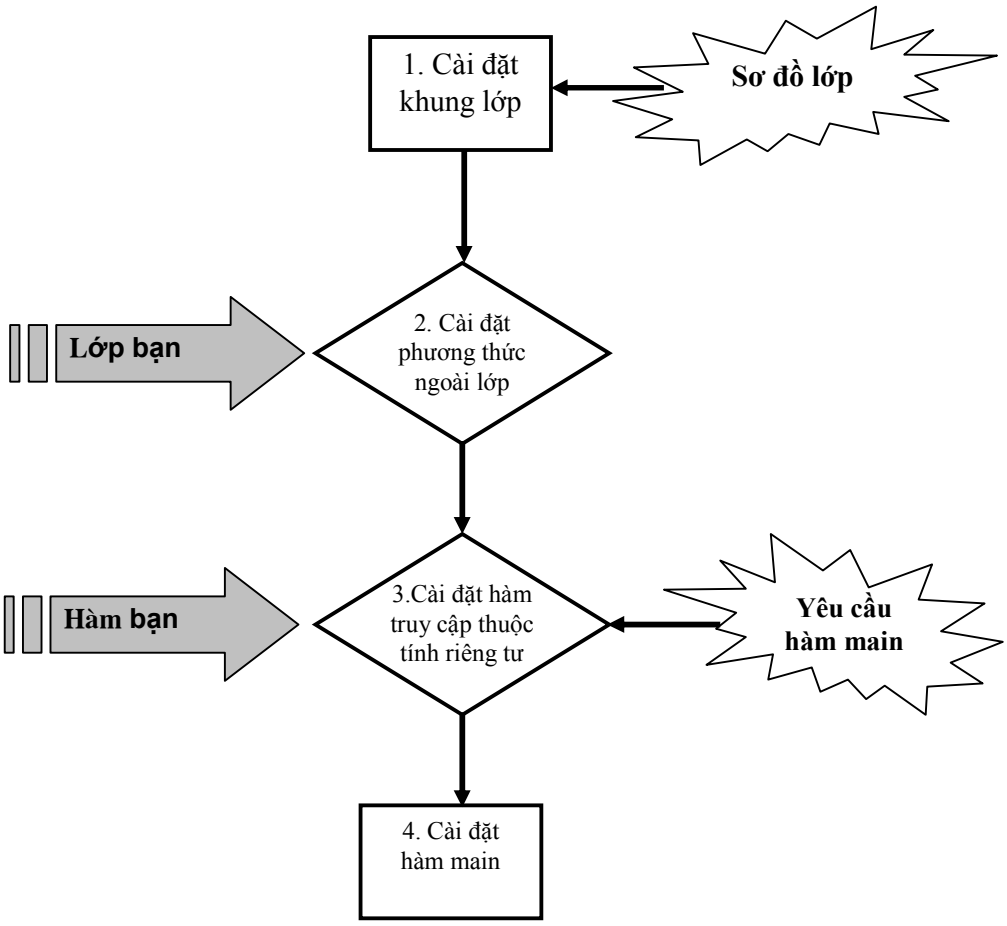
```
void main()
{
        Sach a[100]; int n;

        //Nhập danh sách – trước tiên nhập n
        cout<< "n="; cin>>n;
        for(int i=0; i<n; i++)
        a[i].input();

        // Xuất thông tin sách của NXB Thanh Niên – gọi hàm In
        In(a, n);
}
```

Do trình bày quá chi tiết nên có thể gây khó khăn cho người đọc về tính hệ thống. Rất may là ta không phải lo lắng quá mức về vấn đề đó. Sơ đồ sau đây (tuy chưa hoàn thiện) nhưng nó giúp các bạn hệ thống lại những gì đã được giới thiệu ở phần trên. (xem trang sau)

5. Sơ đồ 4 bước để hoàn thành cài đặt một bài tập dạng sơ đồ lớp

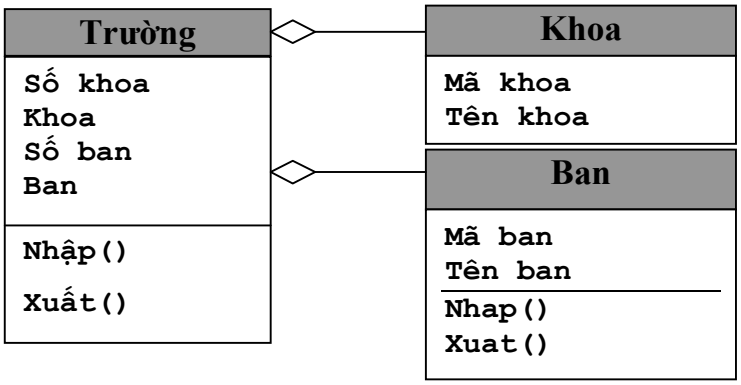


Hình 7. Bốn bước để cài đặt một bài tập dạng sơ đồ lớp.

✍ **Kết tập 1- nhiều:**

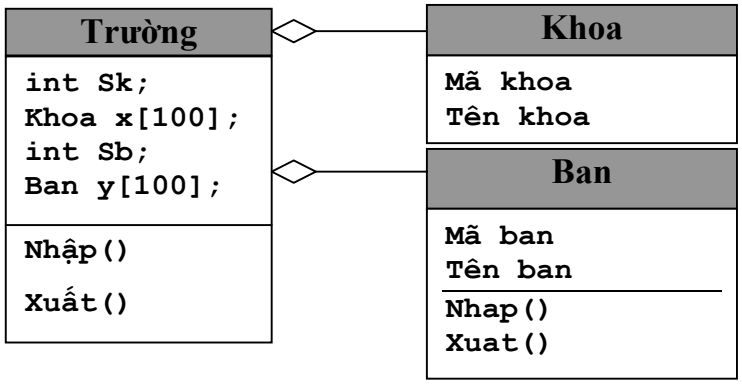
Một số thao tác xử lý tương đối phức tạp có thể cần thiết khi cài đặt các bài tập có quan hệ kết tập 1-nhiều như bài tập sau:

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



Viết chương trình chính nhập vào thông tin của n trường, in ra thông tin của các trường có ít nhất 3 ban hoặc có khoa CNTT.

Theo đề bài, cần hiểu rõ sự kết tập của hai lớp Khoa và Ban vào lớp Trường. Mỗi trường thông thường có nhiều khoa và nhiều ban. Như vậy, ta cần hiểu thuộc tính thể hiện sự kết tập trong lớp Trường (Khoa, Ban) là các mảng đối tượng thuộc lớp Khoa, Ban mà không phải là các đối tượng đơn. Hai thuộc tính Số khoa, Số ban là hai số nguyên thể hiện số khoa, ban thực tế mà mỗi trường có, hay chính là kích thước của 2 mảng đối tượng trên. Vậy sơ đồ được hiểu là:



Hai lớp Khoa, Ban được cài đặt như thông thường mà không gặp phải trở ngại gì:

```
class Khoa
{
   char MaK[30];
   char TenK[30];
};

class Ban
{
  char MaB[30];
  char TenB[30];
public:
  void Nhap();
  void Xuat();
};

void Ban::Nhap()
{   cout<<"Ma ban ="; gets(MaB); fflush(stdin);
    cout<<"Ten ban ="; gets(TenB); fflush(stdin);
}

void Ban::Xuat()
{
    cout<<"Ma ban:"<<MaB<<endl;
    cout<<"Ten ban:"<<TenB<<endl;
}
```

Nhưng lớp Trường, do có 2 mối kết tập 1-nhiều nên khi cài đặt cần chú ý xử lý cho chính xác, đặc biệt là các xử lý hàm bạn, lớp bạn.

```
class Truong
{
   Khoa x[100];
   int Sk;
   Ban y[100];
   int Sb;
public:
   void Nhap();
   void Xuat();
};

void Truong::Nhap()
{
          cout<<"Nhập số khoa:"; cin>>Sk;
          for(int i=0; i<Sk; i++)
          {
             cout<<"Ma khoa:"; gets(x[i].MaK); fflush(stdin);
             cout<<"Ten khoa:"; gets(x[i].TenK); fflush(stdin);
          }

          cout<<"Nhập số ban:"; cin>>Sb;
          for( i=0; i<Sb; i++)   y[i].Nhap();
}

void Truong::Xuat()
{
          for(int i=0; i<Sk; i++)
          {
             cout<<"Ma khoa:"<<x[i].MaK<<endl;
             cout<<"Ten khoa:"<<x[i].TenK<<endl;
          }

          for( i=0; i<Sb; i++)
             y[i].Xuat();
}
```

Nói chung, khi xử lý các thuộc tính kết tập 1-nhiều, vì nó là mảng, nên cần sử dụng các vòng lặp.

Tuy nhiên lớp Truong đã truy cập thuộc tính riêng tư của lớp Khoa nên nó cần là bạn của lớp Khoa. Vậy lớp Khoa được viết lại là:

```
class Khoa
{
   char MaK[30];
   char TenK[30];
   friend class Truong;
};
```

Bây giờ hãy cài đặt chương trình chính. Vì cần in ra thông tin các trường có nhiều hơn 3 ban hoặc có khoa CNTT nên chắc chắn sẽ truy cập tới thuộc tính riêng tư của lớp Truong (thuộc tính Sb) và lớp Khoa (thuộc tính TenK). Vì vậy một hàm In() để thực hiện công việc này là cần thiết:

```
void In(Truong a[100], int n)
{
    for(int i=0; i<n; i++)
    {
        int Check = 0;
        for(int j = 0; j<a[i]. Sk; j++)
        if(strcmp(a[i].x[j].TenK, "CNTT")==0)
        Check=1;

        if (a[i]. Sb >3 | | Check==1)
            a[i].Xuat();
    }
}
```

```
void main()
{
    Truong a[100]; int n;

            cout<<"n="; cin>>n;
            for(int i=0; i<n; i++)
            a[i].Nhap();
    //-----------------------------------------
     In(a, n);
}
```

Rõ ràng là việc xử lý trên các thuộc tính kết tập 1-nhiều luôn phức tạp hơn với nhiều vòng lặp cùng các giải thuật được đưa vào. Điều này có thể gây thêm khó khăn cho những thí sinh chưa nắm vững kiến thức.

Mặt khác hàm In cần là bạn của lớp Truong và lớp Khoa. Vậy đoạn mã sau với các bổ sung hàm bạn, lớp bạn sẽ là lời giải cho bài tập trên:

```
class Khoa
{
   char MaK[30];
   char TenK[30];
   friend class Truong;
   friend void In(Truong *a, int n);
};
```

```
class Ban
{
  char MaB[30];
  char TenB[30];
public:
  void Nhap();
  void Xuat();
};

void Ban::Nhap()
{   cout<<"Ma ban ="; gets(MaB); fflush(stdin);
    cout<<"Ten ban ="; gets(TenB); fflush(stdin);
}

void Ban::Xuat()
{
    cout<<"Ma ban:"<<MaB<<endl;
    cout<<"Ten ban:"<<TenB<<endl;
}
```

```
class Truong
{
   Khoa x[100];
   int Sk;
   Ban y[100];
   int Sb;
public:
   void Nhap();
   void Xuat();
   friend void In(Truong *a, int n);

};

void Truong::Nhap()
{
            cout<<"Nhập số khoa:"; cin>>Sk;
            for(int i=0; i<Sk; i++)
            {
               cout<<"Ma khoa:"; gets(x[i].MaK); fflush(stdin);
               cout<<"Ten khoa:"; gets(x[i].TenK); fflush(stdin);
            }

            cout<<"Nhập số ban:"; cin>>Sb;
            for( i=0; i<Sb; i++)
               y[i].Nhap();
}

void Truong::Xuat()
{
            for(int i=0; i<Sk; i++)
            {
               cout<<"Ma khoa:"<<x[i].MaK<<endl;
               cout<<"Ten khoa:"<<x[i].TenK<<endl;
            }

            for( i=0; i<Sb; i++)
               y[i].Xuat();
}
```

```
void In(Truong a[100], int n)
{
    for(int i=0; i<n; i++)
    {
         int Check = 0;
         for(int j = 0; j<a[i]. Sk; j++)
         if(strcmp(a[i].x[j].TenK, "CNTT")==0)
         Check=1;

         if (a[i]. Sb >3 || Check==1)
              a[i].Xuat();
    }
}
```

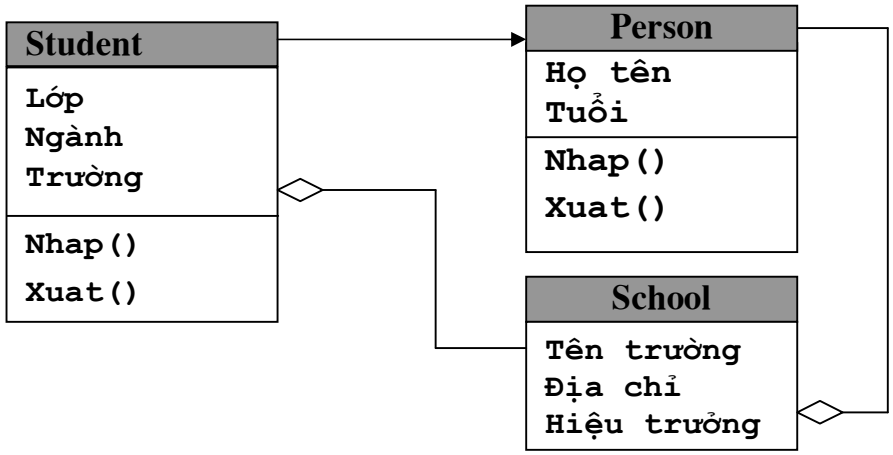
```
void main()
{
     Truong a[100]; int n;
            cout<<"n="; cin>>n;
            for(int i=0; i<n; i++)
            a[i].Nhap();

     In(a, n);
}
```

6. 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 (vì sao? xin xem mục 4. [3]).

```
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);
}
```

PHẦN 2

CÀI ĐẶT BÀI TẬP DẠNG PHIẾU

Việc cài đặt bài tập dạng phiếu về cơ bản giống với việc cài đặt bài tập dạng sơ đồ lớp. Hãy đảm bảo bạn nắm vững cách cài đặt dạng sơ đồ lớp trước khi đọc sang phần 2 này.

Ta xét ví dụ minh họa số 3:

Ví dụ minh họa 3: Viết chương trình đơn giản quản lý việc tạo phiếu kiểm kê tài sản trong các phòng ban của một công ty theo phương pháp hướng đối tượng. Yêu cầu các thuộc tính đều đặt phạm vi truy cập riêng tư và chương trình phải có các chức năng sau:

- **Tạo một phiếu kiểm kê:** cho phép nhập các thông tin chung của phiếu (mã phiếu, ngày kiểm kê); thông tin về nhân viên kiểm kê; thông tin về phòng ban đang kiểm kê; thông tin về các tài sản kiểm kê.
- **In ra phiếu kiểm kê theo mẫu sau:**

PHIẾU KIỂM KÊ TÀI SẢN

Mã phiếu: *PH01.* Ngày kiểm kê: *1/1/2007*
Nhân viên kiểm kê: *Kiều Thị Thanh* Chức vụ: *Kế toán viên*
Kiểm kê tại phòng: *Tổ chức hành chính* Mã phòng: *PTC*
Trưởng phòng: *Hoàng Bích Hảo*

Tên tài sản	**Số lượng**	**Tình trạng**
Máy vi tính	5	Tốt
Máy vi tính	3	Hết khấu hao - hỏng
Bàn làm việc	6	Tốt

Số tài sản đã kiểm kê*: 3.* *Tổng số lượng: 14*

Với dạng này, việc đầu tiên là tiến hành chuyển bài toán về dạng sơ đồ lớp. Khi có sơ đồ lớp, việc cài đặt được tiến hành như PHẦN 1.

Để chuyển bài toán về sơ đồ lớp, ta tiến hành thông qua 4 bước sau:

[1]. Liệt kê các thuộc tính trong phiếu:

Các thuộc tính trong phiếu là những thành phần mà nội dung của chúng có thể khác nhau giữa 2 phiếu. Ví dụ, với 2 phiếu bất kỳ, thì dòng chữ "PHIẾU

KIỂM KÊ TÀI SẢN" là giống nhau nên nó không phải là thuộc tính mà ta quan tâm, nhưng dữ liệu về Mã phiếu, Ngày kiểm kê, ... có thể khác nhau và chúng là các thuộc tính của phiếu.

Vậy ta được danh sách các thuộc tính như danh sách bên:

Stt	Tên thuộc tính	Ký hiệu
1	Mã phiếu	MaPh
2	Ngày kiểm kê	Ngay
3	Nhân viên kk	TenNV
4	Chức vụ	CVNV
5	Phòng kiểm kê	TenP
6	Mã phòng kk	MaP
7	Trưởng phòng	TruongP
8	Tên tài sản	TenTS
9	SoLuong	SLTS
10	Tình trạng	TTTS
11	Tổng số TS	TongTS
12	Tổng số lượng	TongSL

[2]. Loại bỏ thuộc tính suy diễn:

Thuộc tính suy diễn là thuộc tính mà giá trị của nó có thể tính toán được từ giá trị của các thuộc tính khác (xem tài liệu Hướng dẫn ôn tập môn CSDL – cùng tác giả). Trong danh sách này, thuộc tính bị loại là hai thuộc tính số 11 và 12. Vậy phiếu trên chỉ còn 10 thuộc tính.

[3]. Phân nhóm các thuộc tính:

Ta tiến hành phân nhóm các thuộc tính trong danh sách theo nguyên tắc: **các thuộc tính mô tả chung một loại đối tượng sẽ thành một nhóm**. Ví dụ: các thuộc tính TenNV và CVNV đều mô tả thuộc tính của một loại đối tượng là Nhân Viên; các thuộc tính MaP, TenP, TruongP đều mô tả các thuộc tính của một loại đối tượng là Phòng ban....Do vậy, ta được một danh sách đã phân nhóm như danh sách kế bên:

1	Mã phiếu	MaPh
2	Ngày kiểm kê	Ngay
3	Nhân viên kk	TenNV
4	Chức vụ	CVNV
5	Phòng kiểm kê	TenP
6	Mã phòng kk	MaP
7	Trưởng phòng	TruongP
8	Tên tài sản	TenTS
9	SoLuong	SLTS
10	Tình trạng	TTTS

[4]. Hình thành các lớp và vẽ sơ đồ lớp:

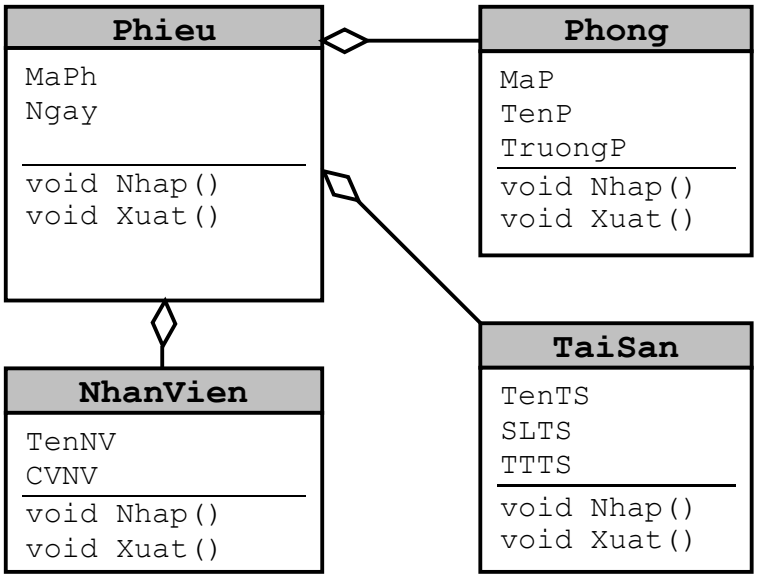
Mỗi nhóm thuộc tính, hãy đưa ra tên loại đối tượng mà nhóm thuộc tính đó mô tả. Ví dụ, với 4 nhóm trên, ta có thể đưa ra 4 danh từ: Phiếu, Nhân Viên, Phòng, Tài sản.

Cần chú ý là bao giờ cũng có 1 nhóm mô tả đối tượng Phiếu, vì vậy luôn luôn có danh từ Phiếu trong mọi bài tập dạng này.

Mỗi danh từ ta vừa xác định sẽ thành 1 lớp. Trong ví dụ minh họa này, ta có 4 lớp: Phiếu, NhanVien, Phong, TaiSan (xem hình dưới đây).

1 2	Mã phiếu Ngày kiểm kê	MaPh Ngay	**Phiếu**
3 4	Nhân viên kk Chức vụ	TenNV CVNV	**Nhân Viên**
5 6 7	Phòng kiểm kê Mã phòng kk Trưởng phòng	TenP MaP TruongP	**Phòng**
8 9 10	Tên tài sản SoLuong Tình trạng	TenTS SLTS TTTS	**Tài sản**

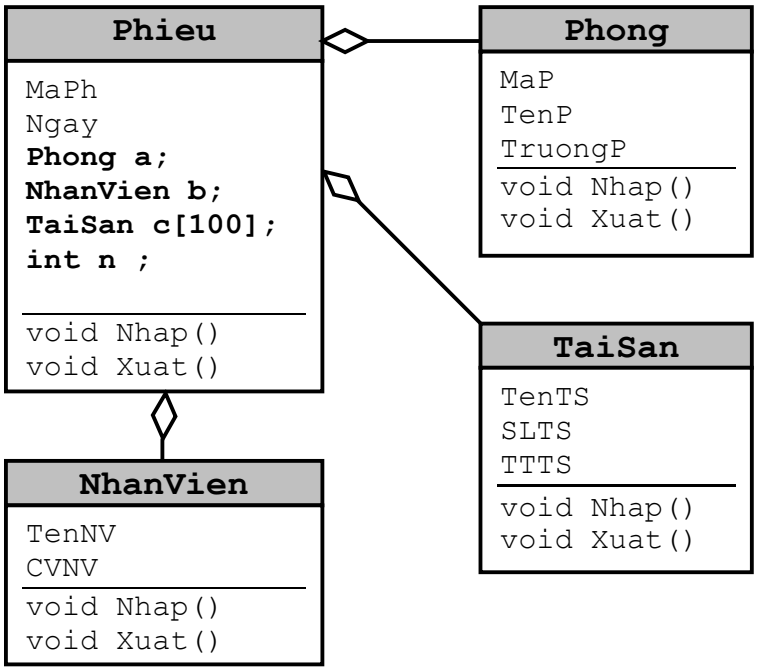
Ta tiến hành vẽ sơ đồ lớp với 4 lớp vừa xác định được. Trong mỗi lớp, hãy chuyển các thuộc tính tương ứng thành thuộc tính của lớp và thêm cho mỗi lớp 2 phương thức Nhap(), Xuat().



Riêng lớp Phiếu (có trong mọi bài dạng này) luôn là lớp chính và các lớp còn lại kết tập vào nó (xem sơ đồ trên). Vì vậy lớp này chúng ta cần phải **bổ sung thêm thuộc tính** cho nó để thể hiện sự kết tập này.

Một phiếu, theo mẫu, ngoài thông tin về Mã phiếu, ngày kiểm kê còn có chứa thông tin của một nhân viên, một phòng ban (kiểm kê) và nhiều tài sản (kiểm kê). Đó chính là 3 sự kết tập của 3 lớp NhanVien, Phong, TaiSan vào lớp Phiếu. Dễ thấy 3 sự kết tập này được chia làm 2 loại: kết tập 1-1 (Phòng-Phiếu;

NhanVien-Phieu) và kết tập 1-nhiều (Phiếu-Tài Sản). Hãy chú ý vào lớp Phieu với các thuộc tính ta sẽ bổ sung để thể hiện 2 loại kết tập này:



Vì Tài sản kết tập vào phiếu theo kiểu 1 – nhiều, nên ta hiểu trong 1 phiếu có chứa nhiều tài sản. Việc này được thể hiện bằng cách khai báo một mảng c nhiều đối tượng thuộc lớp TaiSan (100 đối tượng – TaiSan c[100];) và thêm biến n là "số tài sản thực tế có trong mỗi phiếu".

Việc cài đặt, sau đó, được tiến hành theo sơ đồ 4 bước ở PHẦN 1 (hình 7).

Khi cài đặt các lớp này, trong mỗi phương thức Xuat() của mỗi lớp, ta cần chú ý định dạng sao cho khi xuất dữ liệu được giống như mẫu phiếu mà đầu bài cho. Việc này được thực hiện đơn giản bằng cách chú ý đặt dấu xuống dòng (endl) phù hợp, đúng vị trí trong mỗi lệnh cout.

Sau đây là 4 bước để cài đặt theo sơ đồ hình 7:

Bước 0. Thứ tự cài đặt: Phong, NhanVien, TaiSan, Phieu. (chú ý bao giờ cũng cài đặt lớp Phiếu sau cùng).

Bước 1. Cài đặt các khung lớp:

```
class Phong
{
	char MaP[30];
	char TenP[30];
	char TruongP[30];
public:
	void Nhap();
	void Xuat();
};


class NhanVien
{
	char TenNV[30];
	char CVNV[30];
public:
	void Nhap();
	void Xuat();
};


class TaiSan
{
	char TenTS[30];
	int SLTS;
	char TTTS[30];
public:
	void Nhap();
	void Xuat();
	friend class Phieu;
};


class Phieu
{
	char MaPh[30];
	char Ngay[12];
	Phong a;
	NhanVien b;
	TaiSan c[100];
	int n;
public:
	void Nhap();
	void Xuat();
};
```

B2. Cài đặt các phương thức ngoài lớp:

Chú ý phương thức Phieu::Xuat() do có thống kê tổng số lượng tài sản nên nó truy cập vào thuộc tính SLTS của lớp TaiSan, vì vậy mà lớp Phiếu phải là bạn của lớp TaiSan (xem B1).

```
void Phong::Nhap()
{
      cout<<"Ma phong:"; gets(MaP); fflush(stdin);
      cout<<"Ten phong:"; gets(TenP); fflush(stdin);
      cout<<"Truong phong:"; gets(TruongP); fflush(stdin);
}
void Phong::Xuat()
{
      cout<<"Kiem ke tai phong: "<<TenP;
      cout<<"Ma phong: "<<MaP<<endl;
      cout<<"Truong phong: "<<TruongP<<endl;
}

void NhanVien::Nhap()
{
      cout<<"Ten nhan vien:"; gets(TenNV); fflush(stdin);
      cout<<"Chuc vu:"; gets(CVNV); fflush(stdin);
}
void NhanVien::Xuat()
{
      cout<<"Nhan vien kiem ke: "<<TenNV;
      cout<<"Chuc vu: "<<CVNV<<endl;
}

void TaiSan::Nhap()
{
      cout<<"Ten tai san:"; gets(TenTS); fflush(stdin);
      cout<<"So luong:"; cin>>SLTS;
      cout<<"Tinh trang:"; gets(TTTS); fflush(stdin);
}
void TaiSan::Xuat()
{
      cout<<TenTS<<"     "<<SLTS<<"     "<<TTTS<<endl;
}

void Phieu::Nhap()
{
      cout<<"Ma phieu:"; gets(MaPh); fflush(stdin);
      cout<<"Ngay lap:"; gets(Ngay); fflush(stdin);
      a. Nhap();
      b. Nhap();
      cout<<"Nhap so tai san cua phieu:"; cin>>n;
      for(int i=0; i<n; i++)
            c[i].Nhap();
}
void Phieu::Xuat()
{
      cout<<"        PHHIEU KIEM KE TAI SAN"<<endl;
      cout<<"Ma phieu: "<<MaPh<<" Ngay kiem ke:"<<Ngay<<endl;
      a. Xuat();
      b. Xuat();
      cout<<"Ten ts|  So luong | Tinh trang"<<endl;
      for(int i=0; i<n; i++)
            c[i].Xuat();
      cout<<"Tong so tai san da kiem ke:"<<n;
      int TongSL=0;
      for(i=0; i<n; i++)
            TongSL+=c[i].SLTS;
      cout<<"   Tong so luong:"<<TongSL;
}
```

B3. Cài đặt hàm truy cập thuộc tính riêng tư: không có.

B4. Cài đặt hàm main:

```
void main()
{
	Phieu x;
	x.Nhap();
	x.Xuat();
}
```

PHẦN 3
PHƯƠNG PHÁP CHUNG CÀI ĐẶT BÀI TẬP HĐT

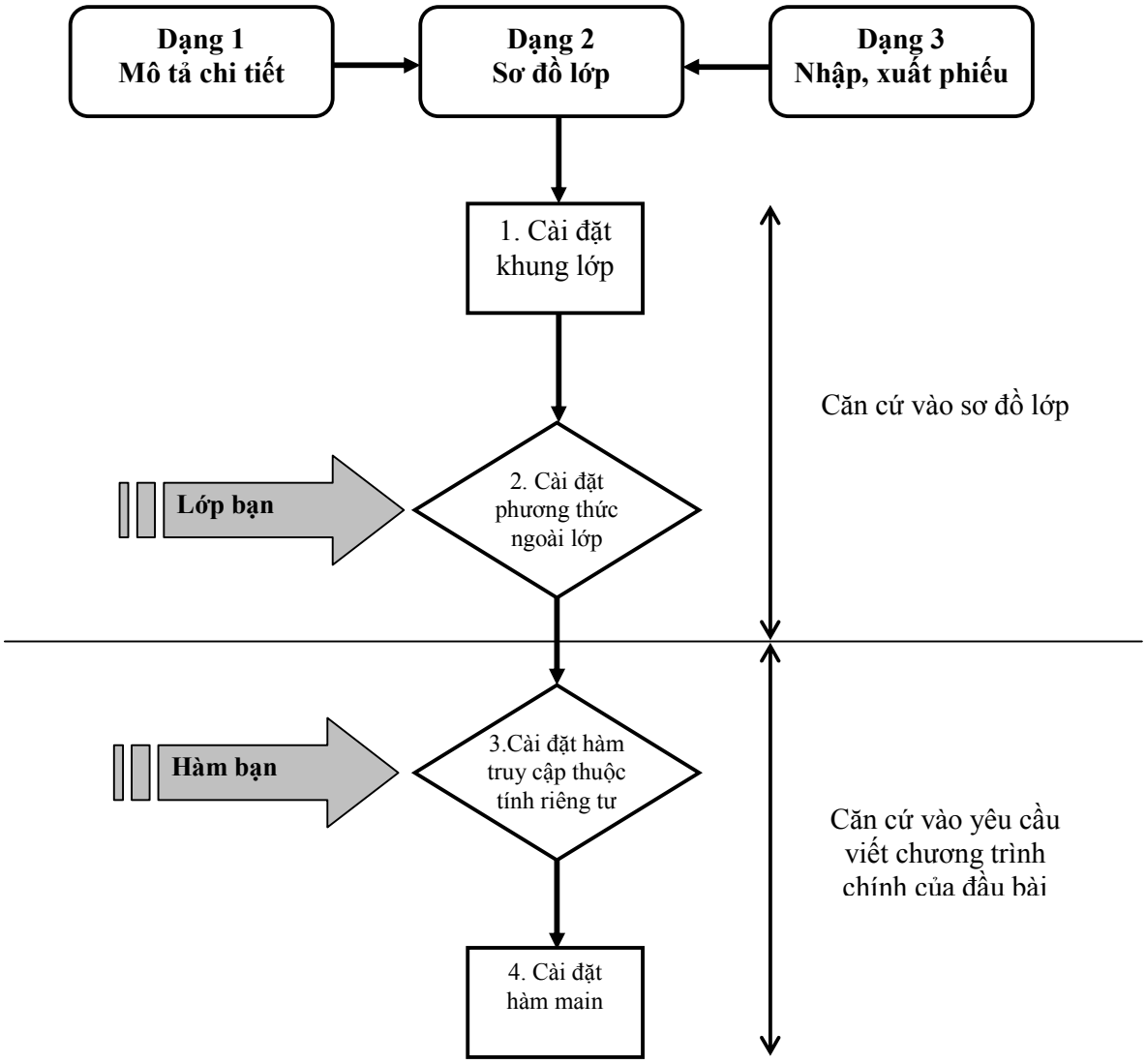
Một đề bài lập trình hướng đối tượng, thông thường được cho ở 3 dạng:

[1]. Dạng mô tả bằng lời: Toàn bộ các lớp, các mối quan hệ giữa các lớp được đề bài mô tả bằng lời một cách chi tiết. Dạng này, các bạn dễ dàng xác định được các lớp của bài và mối quan hệ giữa chúng; các thuộc tính và phương thức trong mỗi lớp. Do vậy, ta dễ dàng vẽ một sơ đồ lớp cho mỗi bài (nếu cần).

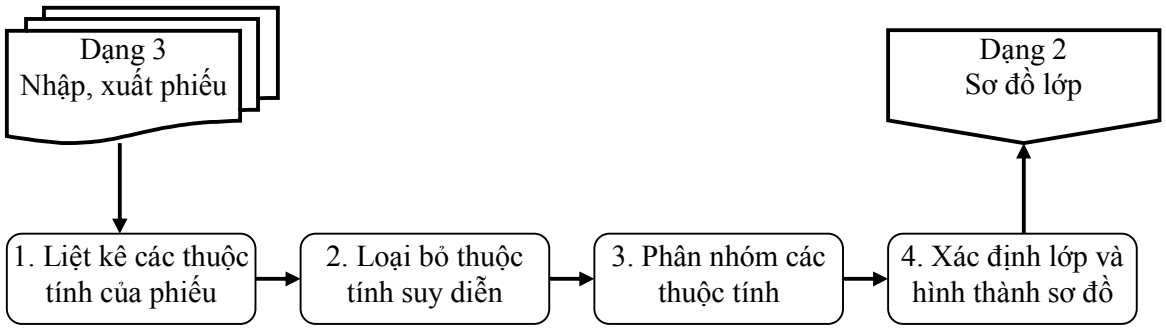
[2]. Dạng sơ đồ lớp: Đề bài yêu cầu cài đặt 1 sơ đồ lớp, trong đó đã thể hiện các lớp, mối quan hệ giữa các lớp, các thuộc tính và phương thức của từng lớp. Như vậy, cần có kiến thức tối thiểu để đọc, hiểu sơ đồ và giải nó theo phương pháp 4 bước ở hình 7 (xem phần 1). Với dạng này, luôn có phần yêu cầu ngay sau sơ đồ lớp. Phần đó, thông thường là yêu cầu các chức năng chính của bài được thực hiện sau khi đã cài đặt sơ đồ lớp.

[3]. Dạng phiếu: Với việc cài đặt các chức năng Nhập, Xuất cho một phiếu bất kỳ, ta dễ dàng chuyển chúng thành sơ đồ lớp và dạng này biến thành dạng [2]. Với dạng này, cần chú ý tới các thuộc tính suy diễn của phiếu. Thông thường, đây là các thuộc tính mang tính thống kê và ta cần tính giá trị cho các thuộc tính này bằng cách thống kê các giá trị của các thuộc tính khác (ví dụ tính tổng số lượng tài sản bằng cách duyệt qua các tài sản và cộng dồn số lượng). Điều này rất dễ bị thí sinh bỏ qua do nó thường không được quan tâm đúng mức.

Vậy một sơ đồ quan trọng trong việc làm bài tập lập trình HĐT được cho dưới đây. Người đọc nên dành thời gian thích đáng để nghiên cứu và ghi nhớ sơ đồ này:



Hình 8. Sơ đồ chung cho việc cài đặt



Hình 9. Chuyển một phiếu thành sơ đồ lớp.

Một số lưu ý:

[1]. Sử dụng lại code do kết tập:

<Đối tượng>.<Phương thức>;

Trong đó, <Đối tượng> là 1 thuộc tính thể hiện sự kết tập của một lớp vào một lớp khác.

[2]. Sử dụng lại code do kế thừa:

<Tên lớp>::<Phương thức>;

Trong đó, <Tên lớp> là tên lớp cha. <Phương thức> chính là phương thức mà lớp con kế thừa được từ lớp cha đó.

[3]. Khai báo hàm bạn: đối vào của hàm bạn không được là mảng thông thường mà phải chuyển thành con trỏ. Kiểu của đối, nếu là 1 lớp thì lớp đó cần phải khai báo trước đó.

[4]. Hãy quan sát:

<Phạm vi truy cập> hoặc <Tên lớp> **:** [Phạm vi kế thừa]

<Tên lớp> **::** <Phương thức>

<Đối tượng> **.** <Phương thức> hoặc <Thuộc tính>

PHẦN 4
CÁC DẠNG BÀI TẬP KHÁC

Ngoài việc nắm vững cách cài đặt các bài tập thông thường, ta cần bổ sung thêm một số kiến thức để cài đặt các lớp có tính chất đặc biệt. Các lớp có thêm phương thức toán tử là những lớp thuộc loại này.

Ở phần này chúng ta sẽ xem xét cách thức để cài đặt phương thức toán tử thông thường, hàm toán tử đặc biệt. Muốn vậy, sau phần này, cần đảm bảo bạn đã có thể trả lời đầy đủ các câu hỏi sau:

[1]. Tại sao phải cài đặt phương thức toán tử?

[2]. Cài đặt phương thức toán tử như thế nào? Tại sao phương thức toán tử 1 ngôi lại không có đối còn phương thức toán tử 2 ngôi lại chỉ có 1 đối?

[3]. Sử dụng các phương thức vừa cài đặt thế nào?

[4]. Hàm toán tử nhập, xuất làm nhiệm vụ gì? cài đặt và sử dụng thế nào?

Bây giờ ta sẽ xem xét lần lượt chúng.

1. Tại sao phải cài đặt phương thức toán tử?

Trước tiên, cần bàn đến toán tử. Như đã biết, toán tử (hay phép toán) được chia làm hai loại: toán tử 1 ngôi (thực hiện trên 1 toán hạng) và toán tử 2 ngôi (thực hiện trên 2 toán hạng).

Trong C++ đã có chứa sẵn rất nhiều toán tử, tức là người ta đã cài đặt sẵn các hàm toán tử để phục vụ việc tính toán. Một số toán tử phổ biến như:

- Toán tử 1 ngôi: phép đổi dấu (-), phép phủ định (!), phép tăng 1 đơn vị (++), phép giảm 1 đơn vị (--)...

- Toán tử 2 ngôi: Cộng (+), Trừ (-), Nhân (*), Chia (/), Đồng dư (%), So sánh (>, <, >=, <=, ==, !=), Logic (&&, | |), Phép gán (=)...

Nếu đã có sẵn, tại sao ta phải cài đặt chúng?

Hãy xét ví dụ sau: giả sử ta có 3 biến nguyên (int a, b, c) hoặc 3 biến thực (x, y, z). Khi đó, các câu lệnh sau là hoàn toàn hợp lệ:

c = a + b; và z = x + y;

Lý do thật đơn giản vì phép cộng hai số nguyên hoặc 2 số thực đã được cài đặt sẵn trong C++. Khi ta viết a + b, máy tính sẽ kiểm tra xem hai toán hạng a, b

thuộc kiểu gì. Khi nó xác định được a, b nguyên, nó sẽ xem toán tử cộng hai số nguyên đã được cài đặt hay chưa. Thật may mắn là toán tử này (+) đã có sẵn và câu lệnh là hợp lệ.

Tương tự các bạn có thể dùng các phép toán: -, *, /, %, ...

Nhưng giả sử ta có 3 biến không phải kiểu nguyên thủy mà là kiểu lớp (tức là 3 đối tượng thuộc lớp nào đó). Chẳng hạn 3 đối tượng thuộc lớp Sinhvien sau: **Sinhvien a, b, c;** Khi đó việc viết: c = a + b; sẽ không còn hợp lệ nữa. Lý do thật đơn giản vì phép cộng (+) trong C++ không định nghĩa cho việc cộng hai toán hạng có kiểu Sinhvien.

Tóm lại: Hầu hết các toán tử có sẵn trong C++ thường chỉ định nghĩa trên 1 loại toán hạng nhất định. Với các toán hạng đặc biệt, các toán tử này chưa được định nghĩa.

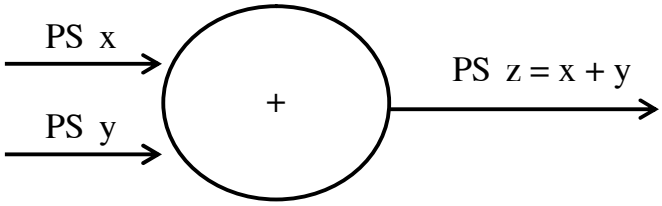
Các toán hạng đặc biệt thông thường là những biến kiểu lớp tức là các đối tượng thuộc 1 lớp nào đó. Một cách phổ biến là các đối tượng thuộc lớp: mảng, ma trận, phân số, số phức, tam thức, đa thức...

Nếu ta có 3 phân số: Phanso a, b, c; thì việc viết: c = a + b; là không hợp lệ. Nhưng trên thực tế ta có thể có nhu cầu cộng 2 phân số. Muốn vậy, trong lớp Phanso cần định nghĩa phương thức toán tử cộng hai phân số. Và khi đó, cách viết c = a + b; sẽ thực hiện việc cộng hai phân số như ta định nghĩa và cách viết đó là hợp lệ.

2. Cài đặt phương thức toán tử như thế nào? Tại sao phương thức toán tử 1 ngôi lại không có đối còn phương thức toán tử 2 ngôi lại chỉ có 1 đối?

Nếu ta muốn cộng hai Phân số, thì trong lớp Phân số ta cần định nghĩa phép cộng này; Nếu ta muốn cộng hai Tam thức, thì trong lớp Tam thức ta cần định nghĩa phép cộng này. Một cách tổng quát, nếu ta muốn cộng hai toán hạng là hai đối tượng thuộc lớp nào, thì trong lớp đó ta cần định nghĩa phép cộng này. Tương tự với các phép toán khác.

Bây giờ hãy xét phép cộng hai Phân số:



Hình 10. Phép cộng hai phân số x và y

Với x, y là hai đối tượng thuộc lớp PS, để thực hiện được phép cộng, trong lớp PS cần cài đặt phương thức toán tử cộng hai phân số.

Một phương thức toán tử được cài đặt theo mẫu sau:

```
<Kiểu trả về> operator <Ký hiệu phép toán>([danh sách các đối])
{
    //Thân phương thức
}
```

Trong đó:

<Kiểu trả về>: là kiểu của giá trị kết quả khi ta thực hiện phép toán. Trong ví dụ cộng hai phân số, kết quả thu được z, và z cũng là 1 phân số (tức z có kiểu PS). Do đó, phép cộng hai phân số có kiểu trả về là PS. Việc xác định kiểu trả về tương đối dễ dàng vì đa số (không phải tất cả) các phương thức toán tử đều trả về 1 kết quả (đầu ra) có kiểu trùng với kiểu của các toán hạng đầu vào.

<Ký hiệu phép toán>: là một trong các ký hiệu toán tử được dùng trong C++, chẳng hạn như: +, -, *, /, %, ++, --, ! ...Hiển nhiên ta có thể dùng một ký hiệu bất kỳ cho phép toán ta định nghĩa miễn là đảm bảo đúng về số ngôi. Ví dụ định nghĩa phép cộng (2 ngôi) thì ta có thể dùng 1 ký hiệu phép toán bất kỳ, miễn là nó 2 ngôi. Tuy nhiên, thật bất thường khi đang định nghĩa phép cộng ta lại dùng ký hiệu (-) hoặc (*)!

[danh sách các đối]: nếu có, thì nó chính là các giá trị đầu vào của phép toán. Nếu là phép toán 1 ngôi, ta có 1 đầu vào; nếu là phép toán 2 ngôi, ta có hai đầu vào (với ví dụ trên, hai đầu vào là 2 phân số x, y). **Vậy với phép toán 1 ngôi ta phải có 1 đối và phép toán 2 ngôi ta cần có 2 đối?**

Tất nhiên là như vậy. Nhưng khi định nghĩa lớp, ta luôn có 1 con trỏ đặc biệt trỏ tới 1 đối tượng ảo đại diện cho lớp gọi là con trỏ this. Con trỏ này luôn là "đối thứ nhất" của các phương thức toán tử trong lớp. Do vậy, với ví dụ trên thì this đóng vai trò như đối vào x, và ta chỉ cần 1 đối thứ 2 là y.

```
PS operator+(PS* this, PS y)
{
}
```

Nhưng con trỏ this có 1 đặc điểm rất đặc biệt là: khi ở trong lớp, nếu ta truy cập các thuộc tính, phương thức của lớp thì mặc định đó là truy cập các thuộc tính, phương thức của con trỏ this và con trỏ this không cần phải viết tường minh. Tức là, con trỏ this có thể ẩn. Vậy phương thức toán tử trên có thể viết là:

```
PS operator+(PS y)
{
}
```

Và ta cần hiểu rằng phép cộng vẫn thực hiện trên 2 đầu vào là this và y. Chỉ có điều this ẩn đi mà thôi. Về hình thức thì phép toán 2 ngôi này có 1 đối vào (y).

Tương tự như vậy phương thức toán tử 1 ngôi có 1 đối vào và nó chính là this. Nhưng vì this ẩn nên rõ ràng về mặt hình thức phương thức toán tử 1 ngôi không có đối.

Để dễ hiểu về this, ta có thể hình dung như sau: khi ta viết a + b thì có 2 đối tượng a, b tham gia vào phép cộng này. Cả a và b đều có phép (+) nhưng ở đây chỉ dùng 1 phép (+) của 1 đối tượng (a chẳng hạn). Nếu dùng phép toán của đối tượng nào thì đối tượng ấy đóng vai trò this. Cách hiểu trên hơi ngây thơ nhưng phần nào lý giải được khái niệm con trỏ this. Để biết rõ hơn, xin nghe bài giảng.

Bây giờ cần quan tâm tới nội dung của phương thức toán tử. Với ví dụ trên, ta cần thực hiện phép cộng hai phân số **this + y** để thu được z và phương thức toán tử trả về z. Mỗi đối tượng đều có hai thuộc tính TS (tử số) và MS (mẫu số). Đối tượng z có thể hình dung như sau (chú ý ký hiệu → tương đương với dấu "."):

$$z = \frac{this \to TS}{this \to MS} + \frac{y.TS}{y.MS}$$

Vậy:

```
z.TS  =  this→TS*y.MS  +  this→MS*y.TS
z.MS  =  this→MS * y.MS
```

Vì this có thể ẩn nên ta có:

```
z.TS  =  TS * y.MS  +  MS * y.TS
z.MS  =  MS * y.MS
```

Vậy phương thức toán tử cộng hai phân số được viết như sau:

```
PS operator+(PS y)
{
    PS z;
    z.TS   =  TS*y.MS   +   MS*y.TS;
    z.MS   =   MS*y.MS;
    return z;
}
```

Thông thường một phương thức toán tử có dạng:

```
<Tên_lớp> operator <phép_toán>( [kiểu đối] [tên đối])
{
        Khai báo biến chứa kết quả (đầu ra) z
        Tính các thuộc tính của z
        return z
}
```

Với định nghĩa như vậy, phép trừ hai phân số như sau:

```
PS operator-(PS y)
{
        PS z;
        z.TS    =   TS*y.MS    -    MS*y.TS;
        z.MS    =   MS*y.MS;
        return z;
}
```

Và phép đổi dấu 1 phân số

```
PS operator-( )
{
        PS z;
        z.TS    =   - TS;
        z.MS    =     MS;
        return z;
}
```

Một cài đặt hoàn chỉnh lớp phân số với các phương thức toán tử +, - hai phân số, đổi dấu 1 phân số như trang sau:

```
class PS
{
      float TS, MS;
public:
      void nhap();
      void xuat();
      PS operator+(PS y) ;      //cộng hai ps
      PS operator-(PS y) ;      //trừ 2 ps
      PS operator-() ;          //đổi dấu 1 ps
};
```

```
void PS::nhap()
{
      cout<<"TS="; cin>>TS;
      cout<<"MS="; cin>>MS;
}
void PS ::xuat()
{
      cout<<TS<<"/"<<MS ;
}
```

```
PS PS ::operator+(PS y)
{
      PS z;
      z.TS = TS*y.MS + MS*y.TS;
      z.MS = MS*y.MS;
      return z;
}
PS PS ::operator-(PS y)
{
      PS z;
      z.TS = TS*y.MS - MS*y.TS;
      z.MS = MS*y.MS;
      return z;
}
PS PS ::operator-()
{
      PS z;
      z.TS = -TS;
      z.MS =  MS;
      return z;
}
```

3. Sử dụng các phương thức vừa cài đặt thế nào?

Khi đã cài đặt phương thức toán tử nào đó cho 1 lớp thì các đối tượng thuộc lớp đó có thể sử dụng phương thức toán tử đó mà không hề bị lỗi toán tử. Tức là ta có thể thực hiện toán tử đó với các toán hạng là các đối tượng thuộc lớp này.

Với ví dụ trên, giả sử ta có hai phân số: PS x, y; ta muốn tính phân số z là tổng của x và y, ta có thể viết: PS z = x+y;

Phép cộng này bình thường sẽ bị lỗi do phép cộng có sẵn trong C++ không thể thực hiện trên 2 đối tượng có kiểu PS. Nhưng vì ta đã cài đặt phương thức toán tử + cho lớp PS nên việc viết như trên là hợp lệ.

Vậy hãy xem đoạn code có sử dụng các phương thức toán tử trên:

```
void main()
{
      PS x, y;
      x.nhap() ; y.nhap() ;
      PS z = x + y ;            //sử dụng phép cộng
      PS t = x - y ;            //sử dụng phép trừ
      z = -z ;                  //sử dụng phép đổi dấu
      z.xuat() ; t.xuat() ;
}
```

4. Toán tử nhập, xuất làm nhiệm vụ gì? cài đặt và sử dụng thế nào?

Toán tử nhập (>>) và xuất (<<) thường được gọi là toán tử đặc biệt. Nhưng trước hết hãy xem nó là gì và dùng để làm gì.

Bây giờ, hãy hình dung ta có 1 biến nguyên a: (int a). Khi đó, việc viết cout<<a; hoặc cin>>a; là hoàn toàn bình thường và quen thuộc.

Nhưng thử hình dung a không phải kiểu int hay float, hay nói chung không phải kiểu nguyên thủy (int, float, char, double, single, long...) mà a là 1 đối tượng thuộc lớp nào đó (lớp PS chẳng hạn: PS a;) khi đó việc viết cout<<a; hoặc cin>>a; lại không hợp lệ! Nguyên nhân là vì các toán tử nhập (>>) và xuất (<<) dùng trong cin và cout không định nghĩa để nhập xuất một đối tượng.

Một cách tổng quát, nếu dùng cin>> và cout<< , thông thường ta không thể nhập, xuất cho 1 biến đối tượng (bất kể nó thuộc lớp nào mà ta định nghĩa). Nhưng nếu ta biến điều không thể đó thành có thể thì thật thuận tiện cho quá trình nhập xuất các đối tượng và có lẽ các phương thức Nhap() hay Xuat() sẽ mất vai trò của nó trong các lớp.

Để làm được điều này, một cách đơn giản, ta sẽ định nghĩa các toán tử nhập (>>) và xuất (<<) cho mỗi lớp. **Khi đã định nghĩa, ta hoàn toàn có thể dùng cin>> và cout<< để nhập, xuất cho các biến đối tượng thuộc lớp đó.**

Việc định nghĩa toán tử này về cơ bản giống như định nghĩa phương thức toán tử thông thường. Nhưng 5 nguyên tắc sau luôn phải chú ý khi định nghĩa toán tử nhập, xuất. Nếu vi phạm bất kỳ nguyên tắc nào, lỗi có thể phát sinh:

[1]. Toán tử nhập, xuất không bao giờ là phương thức của lớp mà chỉ là hàm bạn của lớp.

[2]. Toán tử nhập có kiểu trả về là istream& và toán tử xuất có kiểu trả về là ostream&.

[3]. Luôn có hai đối. Đối thứ nhất có kiểu istream& (với toán tử nhập) và ostream& (với toán tử xuất). Đối thứ hai có kiểu <Tên lớp>&

[4]. Với toán tử nhập, dùng đối thứ nhất thay cho cin để nhập cho đối thứ 2. Với toán tử xuất, dùng đối thứ nhất thay cho cout để xuất đối thứ 2.

[5]. Luôn return <Đối thứ nhất>

Bây giờ hãy xét từng nguyên tắc:

Vì toán tử nhập, xuất là hàm bạn của lớp [1] nên trong thân lớp, ta cần khai báo hàm bạn theo cú pháp: friend <Nguyên mẫu toán tử nhập, xuất>; Với lớp PS ở trên, khai báo này như sau:

```
friend istream& operator>>(istream& x, PS& y);
friend ostream& operator<<(ostream& x, PS& y);
```

Dễ thấy: [2] kiểu trả về của operator>> luôn là istream& và của operator<< luôn là ostream&. Và [3] vì nó là hai hàm bạn chứ không phải là hai phương thức toán tử của lớp nên không có khái niệm con trỏ this trong hai toán tử này. Do đó ta cần định nghĩa đầy đủ 2 đối cho nó. Đối thứ nhất có kiểu istream& cho toán tử nhập (istream& x) và kiểu ostream& cho toán tử xuất (ostream& x). Đối thứ 2 có kiểu <Tên lớp>& chính là tên của lớp ta đang định nghĩa, do đó nó phải có kiểu là PS&.

Các nguyên tắc [4] và [5] giúp ta định nghĩa nội dung toán tử. Xét toán tử nhập. Mục đích của nó là nhập các thuộc tính của đối tượng PS y (tức y.TS và y.MS). Do vậy ta chỉ cần cin>>y.TS; và cin>>y.MS. Tuy nhiên [4] ta cần dùng đối thứ nhất (x) thay cho cin. và như vậy ta cần viết: x>>y.TS; x>>y.MS. Sau khi nhập xong y, ta cần [5] return <Đối thứ nhất>; tức là return x;

```
class PS
{
      float TS, MS;
public:
     friend istream& operator>>(istream& x, PS& y);
     friend ostream& operator<<(ostream& x, PS& y);
};
istream& operator>>(istream& x, PS& y)
{
      cout<<"TS="; x>>y.TS;
      cout<<"MS="; x>>y.MS;
      return x ;
}
ostream& operator<<(ostream& x, PS& y)
{
      x<<y.TS<<"/"<<y.MS ;
      return x ;
}
```

Nói chung, với toán tử nhập, ta có thể dùng cout nhưng không thể dùng cin mà hãy dùng đối thứ nhất (x) thay cho cin; toán tử xuất thì ngược lại, ta luôn dùng x thay cho cout khi xuất các thành phần của y.

Sau khi định nghĩa xong, ta có thể sử dụng cin, cout để nhập xuất các đối tượng thuộc lớp mà ta vừa định nghĩa toán tử nhập xuất.

```
void main()
{
      PS x, y;
      cout<<"Nhập ps x :" ;
      cin>>x ;               //sử dụng toán tử nhập vừa định nghĩa
      cout<<"Nhập ps y :" ;
      cin>>y ;               //sử dụng toán tử nhập vừa định nghĩa
      cout<<"Phân số vừa nhập:" ;
      cout<<x<<endl<<y ;    //sử dụng toán tử xuất vừa định nghĩa
}
```

Ví dụ minh họa: Cài đặt và sử dụng toán tử nhập, xuất cho lớp Hàng gồm các thuộc tính: Mã hàng, Tên hàng, Số lượng, Đơn giá.

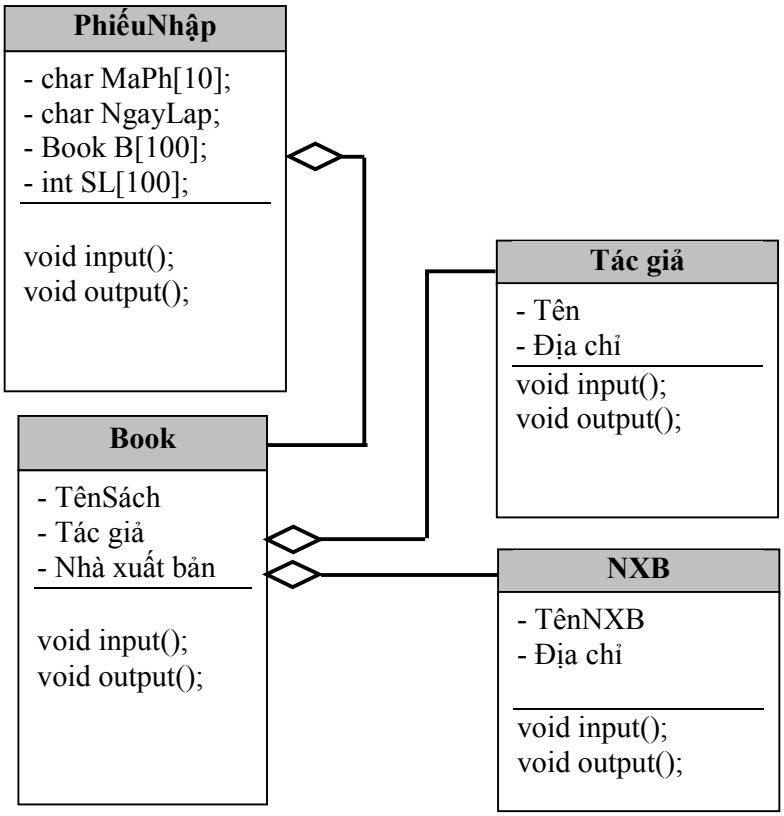
```
class Hang
{
      char MaH[30];
      char TenH[30];
      int SL;
      float DG;
public:
     friend istream& operator>>(istream& x, Hang& y);
     friend ostream& operator<<(ostream& x, Hang& y);
};
istream& operator>>(istream& x, Hang& y)
{
      cout<<"Mã Hang :";   x>>y.MaH;
      cout<<"Tên hàng :";  x>>y.TenH;
       cout<<"Số lượng :";  x>>SL;
      cout<<"Đơn giá :";   x>>DG;
      return x ;
}
ostream& operator<<(ostream& x, Hang& y)
{
      cout<<"Mã Hang :";    x<<y.MaH<<endl;
      cout<<"Tên hàng :";   x<<y.TenH<<endl;
      cout<<"Số lượng :";   x<<SL<<endl;
      cout<<"Đơn giá :";    x<<DG<<endl;
      return x ;
}
void main()
{
      Hang x;
      cout<<"Nhập x"; cin>>x;
      cout<<"Hàng vừa nhập:"<<x;
}
```

Phụ lục

MỘT SỐ BÀI TẬP

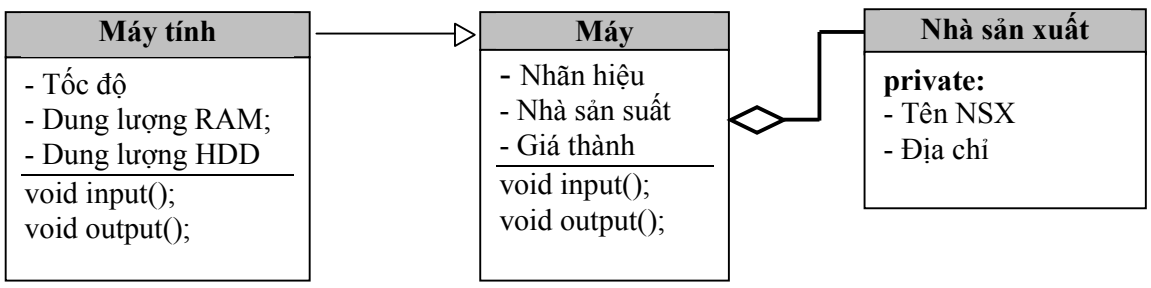
Phần 1. Bài tập dạng sơ đồ lớp

Bài 1.1. Viết các lớp theo sơ đồ sau:



Viết chương trình chính nhập vào thông tin của 1 phiếu nhập. In ra thông tin phiếu vừa nhập.

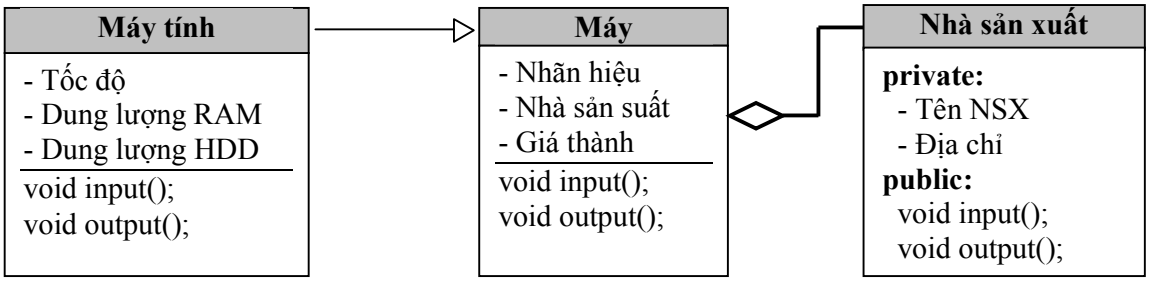
Bài 1.2. Cài đặt các lớp theo biểu đồ sau:



(với input và output là các phương thức nhập, xuất thông tin của các thuộc tính của lớp). Viết chương trình chính nhập vào danh sách n máy tính. In ra thông

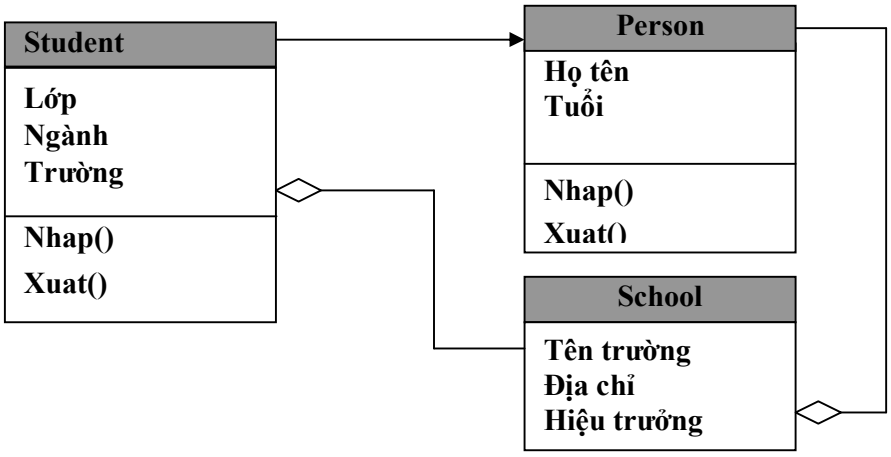
tin của các máy tính của nhà sản xuất IBM. Sắp xếp danh sách các máy tính theo chiều tăng dần của giá thành và in danh sách đã sắp ra màn hình. Xoá mọi máy tính của hãng Intel sản xuất và in danh sách kết quả ra màn hình.

Bài 1.3. Cài đặt các lớp theo biểu đồ sau:



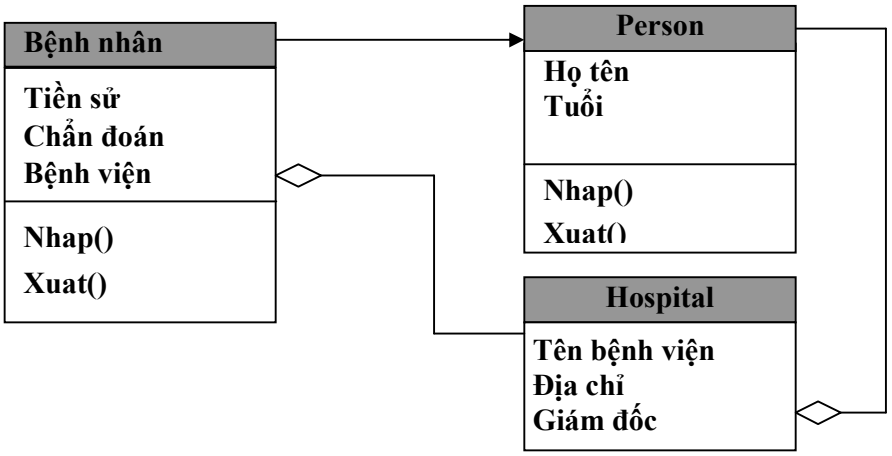
(với input và output là các phương thức nhập, xuất thông tin của các thuộc tính của lớp). Viết chương trình chính nhập vào danh sách n máy tính. In ra thông tin của các máy tính của nhà xản suất Intel. Sắp xếp danh sách các máy tính theo chiều giảm dần của giá thành và in danh sách đã sắp ra màn hình. Cho biết giá thành trung bình của mỗi chiếc máy tính?

Bài 1.4. Hãy cài đặt các lớp theo sơ đồ thiết kế sau



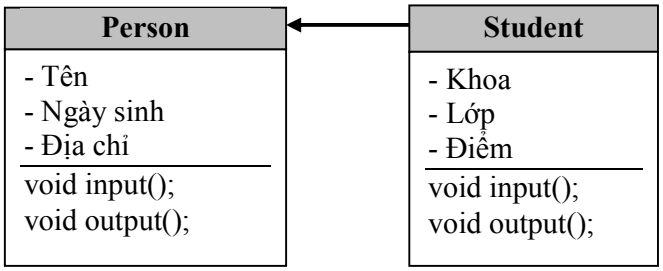
Viết chương trình chính nhập vào danh sách n sinh viên, in ra các sinh viên của trường ĐHCNHN (đã làm trong ví dụ).

Bài 1.5. Hãy cài đặt các lớp theo sơ đồ thiết kế sau



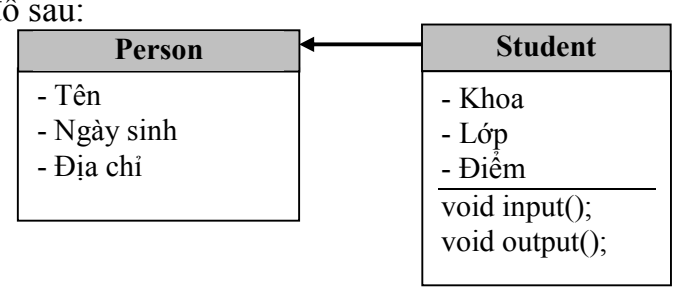
Viết chương trình chính nhập vào danh sách n bệnh nhân, in ra các bệnh nhân của bệnh viện Bạch Mai.

Bài 1.6. Cài đặt lớp theo sơ đồ sau:



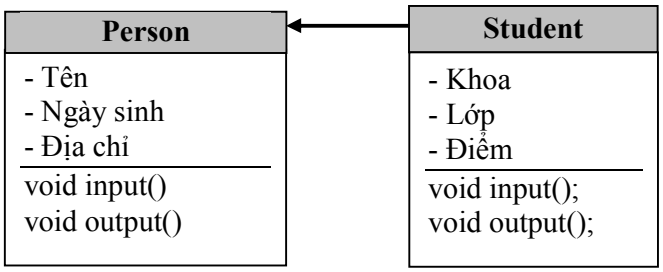
Viết chương trình chính nhập vào thông tin của n sinh viên, in ra các sinh viên khoa CNTT.

Bài 1.7. Cài đặt lớp theo sơ đồ sau:



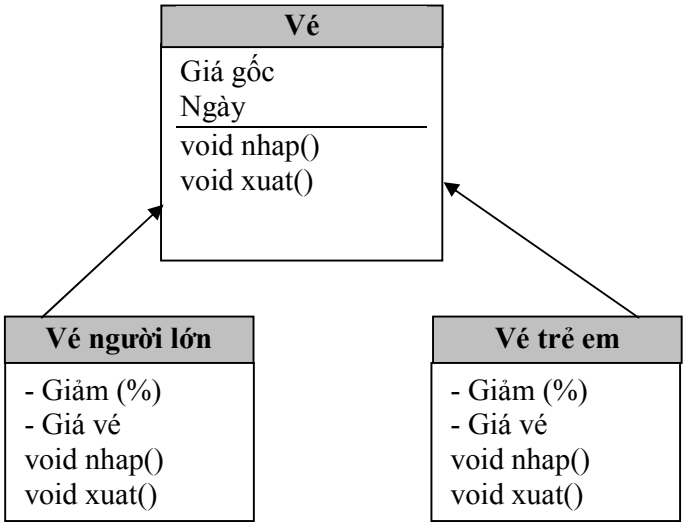
Viết chương trình chính nhập vào thông tin của n sinh viên, in ra các sinh viên khoa CNTT.

Bài 1.8. Cài đặt lớp theo sơ đồ sau:



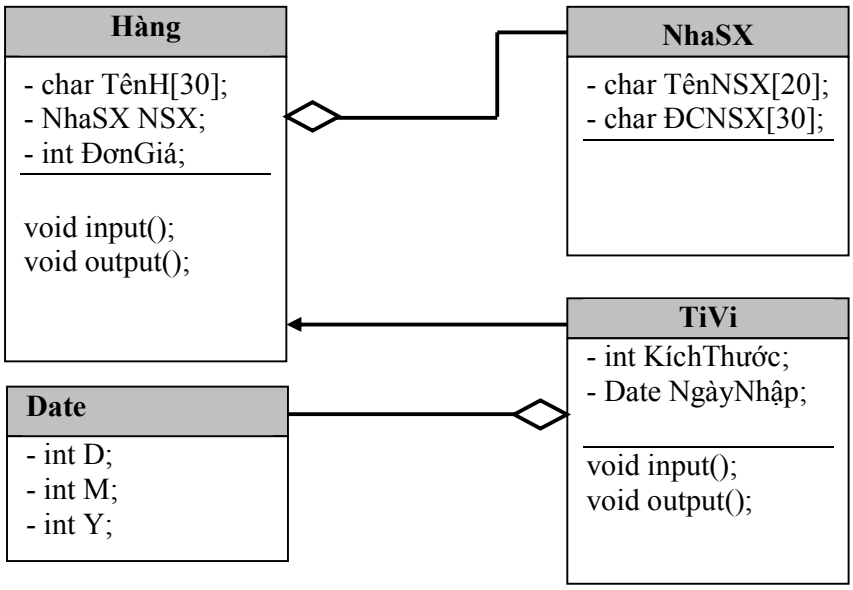
Viết chương trình chính nhập vào thông tin của n sinh viên, sắp xếp danh sách sinh viên theo chiều tăng dần của điểm thi, in ra các sinh viên khoa CNTT.

Bài 1.9. Cài đặt lớp theo sơ đồ sau:



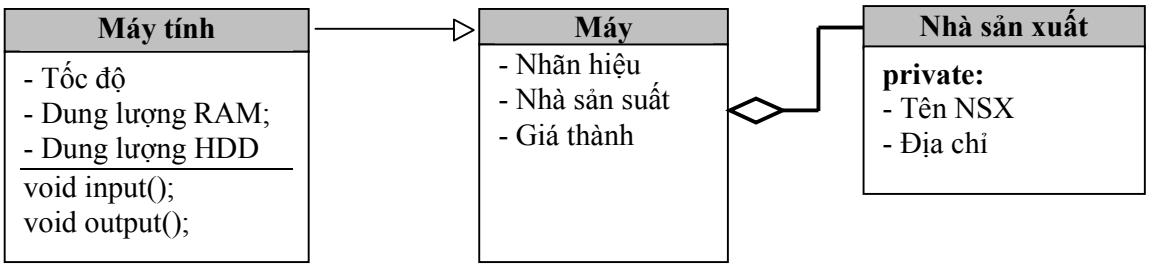
Viết chương trình chính nhập vào 1 vé người lớn và 1 vé trẻ em. In ra thông tin của các vé đó kèm theo giá vé.

Bài 1.10. Xây dựng các lớp theo biểu đồ sau:



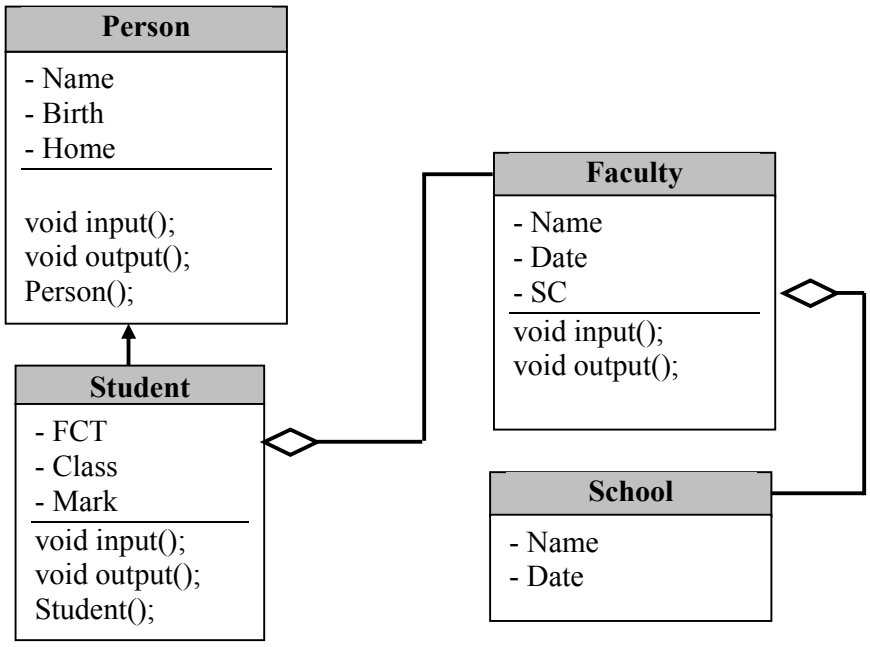
Hãy viết hàm main nhập vào thông tin của n Tivi. Xóa thông tin của các TiVi do hãng LG sản xuất và in kết quả ra màn hình.

Bài 1.11. Cài đặt các lớp theo biểu đồ sau:



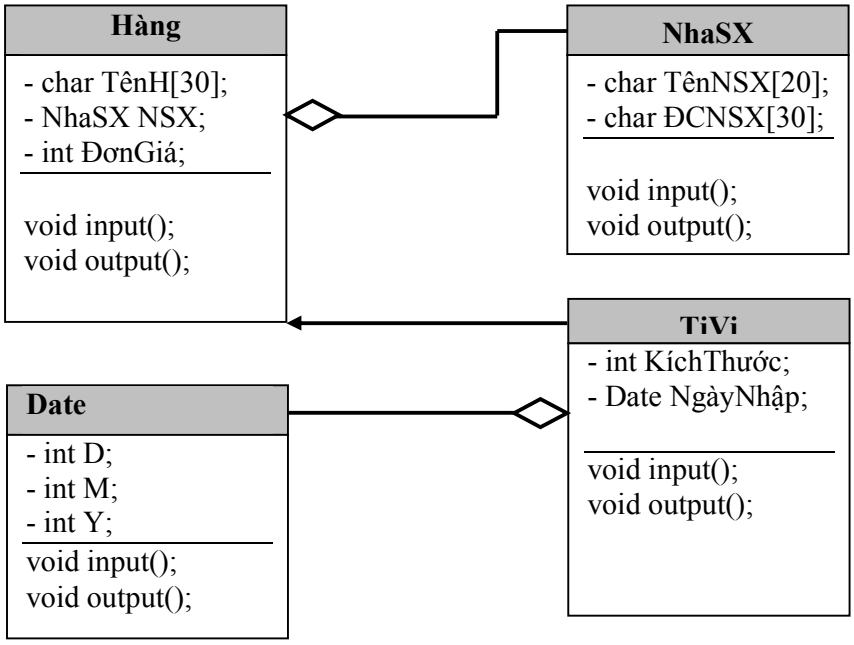
(với input và output là các phương thức nhập, xuất thông tin của các thuộc tính của lớp). Viết chương trình chính nhập vào danh sách n máy tính. Xoá mọi máy tính có giá thành lớn hơn 300 và in danh sách kết quả ra màn hình.

Bài 1.12. Cái đặt lớp theo sơ đồ sau:



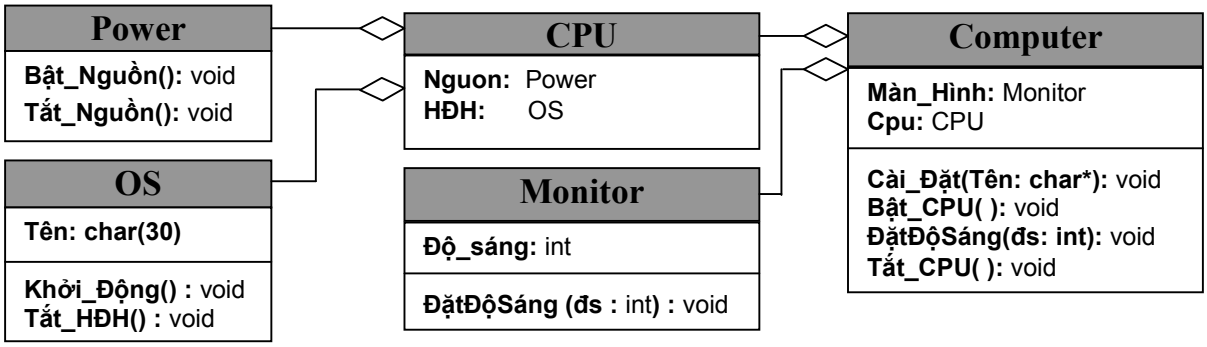
Viết chương trình chính nhập vào một danh sách gồm n sinh viên. Sắp xếp danh sách các sinh viên theo chiều tăng dần của điểm thi (Mark). In ra các sinh viên của trường ĐHCN Hà Nội.

Bài 1.13. Xây dựng các lớp theo biểu đồ sau:



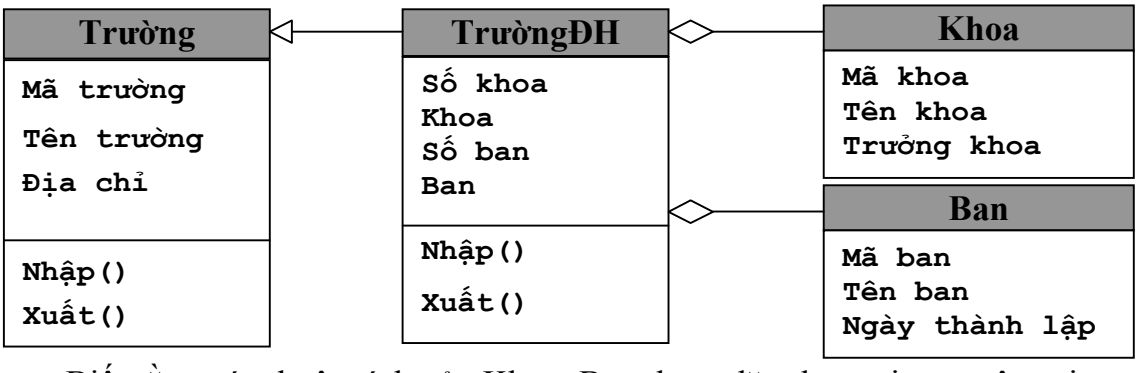
Hãy viết hàm main nhập vào thông tin của n Tivi. Xuất thông tin của các TiVi do hãng LG sản xuất ra màn hình.

Bài 1.14. Viết chương trình mô phỏng hoạt động của một bộ máy vi tính gồm các bộ phận: Nguồn (Power), Hệ điều hành (OS), Màn hình (Monitor), CPU theo sơ đồ sau (nội dung các phương thức thí sinh tự xác định sao cho thỏa mãn yêu cầu trong chương trình chính):



Chương trình chính sinh ra một chiếc máy tính, cài đặt hệ điều hành cho máy tính đó (với tên hệ điều hành được gán là WINXP). Bật CPU của máy (gồm bật nguồn: thông báo nguồn đã bật; khởi động hệ điều hành: thông báo hệ điều hành đã khởi động kèm theo tên hệ điều hành). Đặt độ sáng cho màn hình máy tính với giá trị bất kỳ (có thông báo độ sáng được đặt ra màn hình). Tắt CPU (bao gồm tắt hệ điều hành, tắt nguồn).

Bài 1.15. Hãy cài đặt các lớp theo sơ đồ thiết kế sau:



Biết rằng các thuộc tính của Khoa, Ban được đặt phạm vi truy cập private và mỗi trường có thể có nhiều Khoa, Ban khác nhau. Viết chương trình chính nhập vào danh sách *n* trường đại học (TrườngĐH). Sắp xếp danh sách theo chiều giảm dần của tổng số khoa, ban; In ra các thông tin của các trường có ít nhất 1 Ban thành lập năm 1999-giả thiết ngày thành lập các Ban có định dạng *dd/mm/yyyy*.

Phần 2. Bài tập dạng phiếu

Bài 2.1. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU MUA HÀNG

Mã phiếu: *PH01.* Ngày lập: *1/1/2007*

Tên hàng	Đơn giá	Số lượng	Thành tiền
TiVi	30	2	60
Quạt	1.2	3	3.6
Mobi	5	10	50

Cộng thành tiền: 113.6

Bài 2.2. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU NHẬP HÀNG

Mã phiếu: *PH001.* Ngày lập: *1/1/2007*

Mã NCC: *NCC1* Tên NCC: *LG-Electronic*

Địa chỉ: *Khu công nghiệp Như Quỳnh A*

Tên hàng	Đơn giá	Số lượng	Thành tiền
TiVi	30	2	60
Quạt	1.2	3	3.6
Mobi	5	10	50

Cộng thành tiền: 113.6

Bài 2.3. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU XUẤT HÀNG

Mã phiếu*:* *PH001.* Ngày lập: *1/1/2007*

Mã KH: *KH001* Tên KH: *anh Cường*

Địa chỉ: *23/64 Đặng Văn Ngữ - Hà Nội*

Tên hàng	Đơn giá	Số lượng	Thành tiền
TiVi	30	2	60
Quạt	1.2	3	3.6
Mobi	5	10	50

Cộng thành tiền: 113.6

Bài 2.4. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU BÁO ĐIỂM

Mã sinh viên: *SV001.* Tên sinh viên: *Nguyễn Hải Hà*

Lớp: *Tin 2* Khoá: *52*

Bảng điểm:

Tên môn	Số trình	Điểm
Cơ sở dữ liệu	4	8
Lập trình HĐT	3	7
Hệ điều hành	5	9

Điểm trung bình: *8.17*

Trong đó điểm trung bình = ∑(Số trình * Điểm)/ ∑ (Số trình)

Bài 2.5. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU NHẬP SÁCH

Mã phiếu: *PH001.* Ngày lập: *1/1/2007*

Tổng số lượng nhập về: *300*

Chi tiết:

Tên sách	Tên tác giả	Tên NXB	Số lượng
Cơ sở dữ liệu	Nguyễn Văn Ba	Thống kê	100
Lập trình HĐT	Phạm Văn ất	Thống kê	150
Hệ điều hành	Đỗ Ngọc Tân	GTVT	50

Bài 2.6. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU KHÁM BỆNH

Mã phiếu: *PH01.* Ngày khám: *1/1/2007*

Tên bệnh nhân: *Hoàng Hà* Giới tính: *Nam* Tuổi: *18*

Địa chỉ: *Thái Bình* Tiền sử bệnh: *Viêm mũi dị ứng*

Bác sỹ chẩn đoán: *Đinh Thị Lan* Nơi công tác: *Phòng khám đa khoa BV Bạch Mai*

Mã triệu chứng	Tên triệu chứng
TC005	Nhức đầu váng vất về chiều
TC009	Sốt âm ỷ về đêm
TC010	Bờ dưới khoé mắt bị phù nề

Kết luận: *Viêm xoang cấp.*

Bài 2.7. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU KIỂM KÊ TÀI SẢN

Mã phiếu: *PH01.* Ngày kiểm kê: *1/1/2007*
Nhân viên kiểm kê: *Kiều Thị Thanh* Chức vụ: *Kế toán viên*
Kiểm kê tại phòng: *Tổ chức hành chính* Mã phòng: *PTC*
Trưởng phòng: *Hoàng Bích Hảo*

Tên tài sản	**Số lượng**	**Tình trạng**
Máy vi tính	5	Tốt
Máy vi tính	3	Hết khấu hao - hỏng
Bàn làm việc	6	Tốt

Số tài sản đã kiểm kê*: 3.* *Tổng số lượng: 14*

Bài 2.8. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU NHẬP SÁCH

Mã phiếu: *PH001.* Ngày nhập: *1/2/2000.*
Mã nhà cung cấp: *KH005.* Tên NCC: *Cty sách và thiết bị TH Hà Nội*
Địa chỉ: *2300 Hàng mắm .* Số ĐT: *0912121212*
Thông tin sách nhập:

Mã sách	**Tên sách**	**Giá**	**Số lượng**	**Thành tiền**
S001	*Toán 6*	*12000*	*50*	*600000*
S003	*Văn 6*	*10000*	*30*	*300000*

Tổng thành tiền: *900.000 VNĐ*

Bài 2.9. Viết chương trình cho phép nhập, xuất phiếu sau:

PHIẾU XUẤT SÁCH

Mã phiếu: *PH001.* Ngày xuất: *1/2/2000.*
Mã khách hàng: *KH005.* Tên KH: *Trường tiểu học Minh Khai*
Địa chỉ: *2300 Hàng mắm .* Số ĐT: *0912121212*
Thông tin sách xuất:

Mã sách	**Tên sách**	**Giá**	**Số lượng**	**Thành tiền**
S001	*Toán 6*	*12000*	*50*	*600000*
S003	*Văn 6*	*10000*	*30*	*300000*

Tổng thành tiền: *900.000 VNĐ*

Bài 2.10. Viết chương trình cho phép nhập, xuất phiếu sau:

Phiếu thanh toán cước

Mã KH: *KH001* Tên KH: *Nguyễn Văn A* ĐC: *123 Đội Cấn*

Ngày thanh toán: *1/1/2009* Tại: *Bưu cục Cầu Giấy*

Mã dịch vụ	Tên dịch vụ	Cước	Phụ thu	Tổng
DV001	ADSL	270000	100000	370000
DV002	Home phone	25000	25000	50000
Tổng cộng				420000

Bài 2.11. Viết chương trình cho phép nhập, xuất phiếu sau:

Phiếu nhập sách

Thư viện: TV01-Thư viện trường ĐHQG Hà Nội

Địa chỉ: Xuân thủy – Hà Nội

Ngày nhập: 1/1/2009. Đơn vị cung cấp: Công ty PHAHASA – 123 Tràng tiền

Mã sách	Tên sách	Số trang	Số lượng	Đơn giá
S001	Kỹ thuật lập trình C	235	150	20
S005	C++ và LTHĐT	156	200	25

Bài 2.12. Viết chương trình cho phép nhập, xuất phiếu sau:

VÉ XE KHÁCH

(Liên 2 – giao khách hàng)

Số xe: ……..Số ghế……..

Thời gian chạy:……. h ……..phút. Ngày……tháng…..năm…..

Nơi xuất bến:…….Nơi đến:…….

Phần 3. Bài tập phương thức toán tử

Bài 3.1. Cho hai số phức dạng:

SP1 = a1+ i*b1; SP2 = a2+ i*b2;

Phép cộng, trừ hai số phức được định nghĩa như sau:

SP3 = SP1 + SP2 = (a1+a2) + i*(b1+b2);

SP3 = SP1 - SP2 = (a1-a2) + i*(b1-b2);

Hãy xây dựng lớp số phức với các thuộc tính Thực, ảo và các phương thức: Phương thức khởi tạo: khởi gán phần thực và phần ảo của số phức. Toán tử nhập (>>) và xuất (<<) một số phức. Phương thức toán tử + và - hai số phức. Xây dựng chương trình chính để sử dụng lớp Số phức nói trên.

Bài 3.2. Phép nhân hai phân thức được định nghĩa như sau:

$$\frac{a}{b} x \frac{c}{d} = \frac{ac}{bd}.$$

- Hãy xây dựng một lớp Phân số với các thuộc tính Tử số, Mẫu số và các phương thức: Toán tử nhập (>>) và xuất (<<) đưa phân số ra màn hình (dưới dạng Tử_Số/ Mẫu_số). Toán tử nhân hai phân số (*).

- Viết chương trình chính nhập hai phân số, đưa ra màn hình phân số là tích của hai phân số vừa nhập.

Bài 3.3. Hãy xây dựng lớp phân số với phương thức toán tử trừ hai phân số, cộng hai phân số, chia hai phân số, toán tử nhập (>>), xuất (<<) phân số.

Bài 3.4. Xây dựng lớp ma trận gồm các thuộc tính: float a[100][100] là một mảng hai chiều chứa các phần tử của ma trận; m, n là các thuộc tính chứa kích thước thực tế của ma trận và các phương thức: Toán tử nhập (>>) và xuất (<<) ma trận: nhập các giá trị m, n và ma trận a; xuất ận a lên màn hình. Phương thức toán tử "Đổi dấu ma trận" (-): đổi dấu tất cả các phần tử của ma trận.

Xây dựng chương trình chính trong đó khai báo một đối tượng thuộc lớp ma trận. Nhập các giá trị cho ma trận, đổi dấu ma trận và in ma trận đã đổi dấu ra màn hình.

Bài 3.5. Xây dựng lớp ma trận với phương thức toán tử chuyển vị ma trận (-) (Ma trận A' gọi là chuyển vị của ma trận A nếu A'[i][j] = A[j][i]), phương thức nhập (>>) và xuất (<<) ma trận. Xây dựng chương trình chính minh họa cách sử dụng các phương thức toán tử trên.

Bài 3.6. Ta định nghĩa phương thức toán tử sắp xếp mảng 1 chiều như sau:

- Phương thức -- sắp xếp mảng theo chiều tăng dần.

- Phương thức ++ sắp xếp mảng theo chiều giảm dần.

Hãy định nghĩa một lớp Mảng gồm: thuộc tính a[100] kiểu float, biến kích thước mảng n kiểu nguyên và các phương thức:

- Phương thức Nhập: nhập kích thước mảng và các giá trị cho mảng.

- Phương thức Xuất: Xuất các giá trị của mảng ra màn hình.

- Phương thức toán tử -- và ++ như trên để sắp xếp mảng.

Viết chương trình chính sử dụng lớp trên để nhập vào một mảng n phần tử thực, sau đó sắp xếp mảng theo chiều giảm dần, tăng dần và in các mảng đã sắp lên màn hình.

Bài 3.7. Xây dựng lớp Tam thức bậc hai với các thuộc tính là các hệ số a, b, c thực và các phương thức: Toán tử nhập (>>) và xuất (<<) in tam thức lên màn hình (có dạng $ax^2+bx+c = 0$). Phương thức toán tử "Đổi dấu tam thức": đổi dấu các hệ số a, b, c. Xây dựng toán tử cộng hai tam thức theo định nghĩa:

$$(a_1x^2+b_1x+c_1=0\ ) + (a_2x^2+b_2+c_2=0) = (a_1+a_2)x^2+(b_1+b_2)x+(c_1+c_2)=0$$

Xây dựng chương trình chính khai báo một đối tượng thuộc lớp Tam thức. Khởi gán giá trị cho các hệ số, đảo dấu các hệ số và in tam thức đã đảo dấu ra màn hình.

Cùng toán tử nhập, xuất cho tam thức. Xây dựng chương trình chính để minh họa các sử dụng các toán tử vừa định nghĩa.

Bài 3.8. Một phương trình bậc nhất hai ẩn có dạng ax + by + c = 0. Hãy cài đặt lớp mô tả các phương trình bậc nhất hai ẩn với các phương thức: Phương thức khởi tạo (có đối và không đối). Phương thức toán tử cộng, trừ hai phương trình bậc nhất hai ẩn. Phương thức toán tử nhập (>>), xuất (<<) để nhập/ xuất cho một đối tượng thuộc lớp này.

Viết chương trình chính nhập vào hai phương trình bậc nhất hai ẩn (sử dụng phương thức toán tử nhập). Tính phương trình tổng và hiệu của chúng và in kết quả ra màn hình (bằng phương thức toán tử xuất).

Bài 3.9. Một phương trình bậc hai có dạng $ax^2 + bx + c = 0$. Hãy cài đặt lớp mô tả các phương trình bậc hai với các phương thức: Phương thức khởi tạo (có đối và không đối). Phương thức toán tử cộng, trừ hai phương trình bậc hai, phương thức toán tử nhập (>>), xuất (<<) để nhập/ xuất cho một đối tượng thuộc lớp này.

Viết chương trình chính nhập vào hai phương trình bậc hai (sử dụng phương thức toán tử nhập). Tính phương trình tổng và hiệu của chúng và in kết quả ra màn hình (bằng phương thức toán tử xuất).

Bài 3.10. Phép nhân ma trận a(n×m) với ma trận b(m×p) thu được ma trận c(n×p) với c[i][j] = $\sum_{k=0}^{m-1} a[i][k]*b[k][j]$ $\forall$i$\in$[0, n), j$\in$[0, p). Cài đặt lớp ma trận với: phương thức khởi tạo khởi gán kích thước mặc định và bất kỳ cho ma trận, phương thức toán tử nhân 2 ma trận (nếu hai ma trận không thể nhân, kết quả trả về một ma trận (0×0)); hàm toán tử nhập, xuất ma trận.

Viết chương trình chính nhập vào 2 ma trận bất kỳ (dùng toán tử nhập). Tính ma trận là tích của 2 ma trận trên và in kết quả ra màn hình (dùng toán tử xuất)

Bài 3.11. Phép nhân hai mảng số nguyên được thực hiện bằng cách nhân các phần tử có chỉ số tương ứng của hai mảng. Ví dụ: nhân mảng {1,3,4} với mảng {1, 2, 5, 6} thu được mảng {1,6,20,6}. Hãy cài đặt lớp mảng số nguyên với các phương thức khởi tạo (không đối và có đối); phương thức toán tử nhân hai mảng; phương thức toán tử giảm giá trị của các phần tử mảng đi 1 đơn vị; hàm toán tử nhập (>>), xuất (<<) mảng.

Viết chương trình chính nhập vào hai mảng số nguyên (sử dụng toán tử nhập). Tính tích của hai mảng và in kết quả ra màn hình; giảm giá trị các phần tử của mảng vừa tính được đi 3 đơn vị. Xuất mảng vừa giảm giá trị ra màn hình (sử dụng toán tử xuất).